GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao[1] Mohammed Alser[1] Mohammad Sadrosadati[1] Can Firtina[1] Akanksha Baranwal[1]
Damla Senol Cali[2] Aditya Manglik[1] Nour Almadhoun Alserr[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*Bionano Genomics*

Nanopore sequencing is a widely-used high-throughput genome sequencing technology that can sequence long fragments of a genome into raw electrical signals at low cost. Nanopore sequencing requires two computationally-costly processing steps for accurate downstream genome analysis. The first step, basecalling, translates the raw electrical signals into nucleotide bases (i.e., A, C, G, T). The second step, read mapping, finds the correct location of a read in a reference genome. In existing genome analysis pipelines, basecalling and read mapping are executed separately. We observe in this work that such separate execution of the two most time-consuming steps inherently leads to (1) significant data movement and (2) redundant computations on the data, slowing down the genome analysis pipeline.

This paper proposes GenPIP, an in-memory genome analysis accelerator that tightly integrates basecalling and read mapping. GenPIP improves the performance of the genome analysis pipeline with two key mechanisms: (1) in-memory fine-grained collaborative execution of the major genome analysis steps in parallel; (2) a new technique for early-rejection of low-quality and unmapped reads to timely stop the execution of genome analysis for such reads, reducing inefficient computation. Our experiments show that, for the execution of the genome analysis pipeline, GenPIP provides 41.6× *(*8.4×*) speedup and* 32.8× *(*20.8×*) energy savings with negligible accuracy loss compared to the state-of-the-art software genome analysis tools executed on a state-of-the-art CPU (GPU). Compared to a design that combines state-of-the-art in-memory basecalling and read mapping accelerators, GenPIP provides* 1.39× *speedup and* 1.37× *energy savings.*

1. Introduction

Long read genome sequencing technologies [1–4] have significantly advanced the development of several genomic fields, such as personalized medicine [5–11], forensic science [12,13], evolutionary biology [14–19], and investigation of infectious disease outbreaks, especially during the COVID-19 pandemic [20–30]. Oxford Nanopore Technology (ONT) [4] is one of the most widely-used long-read sequencing technologies. ONT provides portable sequencing devices connected to a computer via a USB interface [1, 31–33]. ONT devices generate long subsequences (called *long reads*) based on the organism's DNA sequence [1, 4, 31, 33–50]. Each read usually has a length ranging from a few hundreds to millions of base pairs [31] (i.e., A, C, G, T nucleotide bases) but with a high sequencing error rate (10% to 15% [38, 39, 41, 45, 46]).

ONT devices sequence a genome by detecting the fluctuations in electrical signals when bases of a DNA sequence pass through a nanoscale hole, called a nanopore [33]. A computational-processing step, called *basecalling* [51, 52], translates these raw electrical signals into a sequence of nucleotide bases (i.e., a read). Basecalling is executed either inside the sequencing device [51, 52] or using an external computer connected to the sequencing device via external connection links (e.g., USB, Ethernet) [33]. A translated read is associated with a quality score for each base to reflect the accuracy of the translation. After basecalling, reads are sent to a separate device to perform further analysis [53]. First, *read quality control* detects and filters out low-quality reads (i.e., reads whose average quality score is lower than a threshold, indicating the accuracy of basecalling translation is low) to avoid further computation on unreliable reads. After read quality control, a computationally-costly *read mapping* step identifies potential matching locations of reads against a reference genome (i.e., a high-quality representative sequence of a species) [54–58].

In the entire genome analysis pipeline, basecalling and read mapping are two of the most time-consuming steps because they rely on computationally-intensive algorithms [50, 54, 56, 57, 59–74]. Basecalling commonly uses a deep neural network (DNN) to ensure high accuracy [51, 52]. Read mapping depends on dynamic programming (DP)-based algorithms [75, 76] to find the potential matching locations in the reference genome where the read can be aligned. Basecalling and read mapping are executed separately in different devices [50]. This decoupled execution of basecalling and read mapping causes three main issues. First, the data movement from the basecalling device to the read mapping device becomes a bottleneck. Second, accelerators have been designed separately for basecalling [50, 63, 77, 78] and read mapping [50, 54, 56, 62, 79–84] to reduce the computational bottleneck, which exacerbates the data movement bottleneck. Third, large execution time and energy consumption overheads ensue due to the fact that a significant portion of the basecalling output is *not* used in the subsequent analyses because of either low quality (10-20% of the examined dataset [85]) and/or being different from the reference genome that is used for mapping (30-70% of the examined datasets [86, 87]).

Our goal is to provide effective in-memory acceleration of the entire genome analysis pipeline while minimizing data movement and useless computation. To this end, we propose GenPIP, a fast and energy-efficient in-memory acceleration system for the Genome analysis PIPeline, which we envision to be best implemented inside the sequencing machine. The **key idea** of GenPIP is to tightly integrate the two key steps of genome analysis (i.e., basecalling and read mapping) inside main memory to (1) minimize data movement by eliminating the need to

store intermediate results and (2) minimize useless computation in the genome analysis pipeline that leads to unused outputs by providing timely feedback from read quality control and read mapping steps to the basecalling step. To realize our key idea, we design an in-memory processing architecture for GenPIP, equipped with two key techniques: (1) a chunk-based pipeline that provides fine-grained collaboration of basecalling and read mapping steps by processing reads at chunk granularity (i.e., a subsequence of a read, e.g., 300 bases) and (2) an early-rejection technique that predicts which reads will not be useful downstream by analyzing multiple chunks of the read, and then stops the execution of basecalling and read mapping for such reads.

We compare GenPIP with (1) the state-of-the-art software genome analysis tools on CPUs and GPUs and (2) a combination of state-of-the-art in-memory basecalling [63] and read mapping [88] accelerators. Our experimental results show that GenPIP provides 41.6× (8.4×) speedup and 32.8× (20.8×) energy savings with negligible accuracy loss, over the state-of-the-art software tools executed on a state-of-the-art CPU (GPU). Compared to the combination of prior in-memory accelerators, GenPIP delivers 1.39× speedup and 1.37× energy savings.

We make the following contributions:

- We observe that the combined acceleration of multiple steps of the genome analysis pipeline is critical due to (1) large data movement between multiple genome analysis steps and (2) significant unnecessary computation due to low-quality and unmapped reads.
- We propose GenPIP as the *first* in-memory accelerator for the genome analysis pipeline, including basecalling, read quality control, and read mapping steps.
- We introduce two key mechanisms that GenPIP employs: (1) fine-grained collaboration of two critical steps (i.e., basecalling and read mapping) using a chunk-based pipeline, and (2) timely prediction of low-quality and unmapped reads to stop the execution of basecalling and read mapping for such reads.
- We evaluate GenPIP and demonstrate that it provides significant performance and energy benefits over the state-of-the-art software and in-memory acceleration approaches.

2. Background and Motivation

In this section, we first introduce the current genome analysis pipeline and its conventional execution environment. Second, we elaborate on the shortcomings of current accelerator designs by studying the performance and energy bottlenecks in the genome analysis pipeline. Third, based on our analysis, we describe our key goal in this work.

2.1. Nanopore Genome Analysis Pipeline

Oxford Nanopore Technology (ONT) [4] is a widely-used sequencing technology as it provides portable sequencing devices and offers much higher sequencing speed than prior sequencing technologies [1, 31–33, 50]. An ONT device generates long subsequences of DNA (called *long reads*) by detecting the changes in electrical current signals when a DNA sequence passes through the device's nanopore [33] (called the *data acquisition and sequencing* step in genome sequencing). The genome analysis pipeline executes after genome sequencing to identify and analyze genomic features. Figure 1 shows an example of the ONT-based (i.e., nanopore) genome sequencing and analysis pipeline. The pipeline includes two key steps, *basecalling* (❶) and *read mapping* (❸), and a highly-recommended but optional step, *read quality control* [50] (❷).

The first key step, *basecalling*, receives the measured electrical current signals from an ONT device and then translates these signals into nucleotide bases (i.e., A, C, G, T). State-of-the-art basecallers (e.g., Bonito [51]) use deep neural networks (DNNs) that provide high accuracy for translating electrical signals into bases [4, 51, 52, 63, 89–91]. Basecallers report a *read quality score* (i.e., the accuracy of the translation) for each base along with the translated base. The basecaller first splits a long read in electrical-signal format (e.g., millions of signals) into multiple smaller *chunks* (e.g., thousands of signals per chunk) and then basecalls these chunks. After basecalling all the chunks, the basecaller reassembles them back into a long read.

The second key step, *read mapping*, maps the basecalled read to a reference genome (i.e., a high-quality representative genome sequence of a species). Minimap2 [92] is a state-of-the-art read mapping tool that performs read mapping mainly in four phases. The first is a preprocessing step called *indexing* (ⓐ, shown at the bottom right of Figure 1), which enables efficient queries to quickly find matches between the subsequences of a reference genome and reads. Indexing is performed offline and *only* once for each reference genome. In the indexing step, Minimap2 generates minimizers [93, 94] (i.e., representative subsequences) from the reference genome, and inserts them into a key-value hash table, where minimizers are the keys and their locations in the reference genome are the values. Second, Minimap2 performs *seeding* (ⓑ) to generate minimizers from a basecalled read and query the generated minimizers in the hash table to quickly find matching regions between the reference genome and the read sequence [55]. Third, Minimap2 executes *chaining* (ⓒ) [92] to identify the *candidate regions* in the reference genome that have a high similarity with the read based on the matching minimizers and distances between the read and the reference genome. Chaining is a dynamic programming (DP) approach [95, 96] that assigns a *chaining score* for a chain of matching minimizers based on the distances (i.e., gaps) between these minimizers. As the chaining score increases, the similarity between the corresponding region in the reference genome and read sequence increases. Fourth, Minimap2 performs *sequence alignment* (ⓓ) to quantitatively identify the similarity between the read and each candidate region in the reference genome. Sequence alignment calculates an *alignment score* to quantitatively represent the difference between the two sequences. To calculate the alignment score, sequence alignment uses a computationally-expensive DP algorithm [50, 58] that performs approximate string matching between two sequences.

Read quality control (RQC) is a highly-recommended [50] but optional step that takes place after basecalling but before read mapping. RQC filters out low-quality reads generated by the basecaller to (1) improve the overall accuracy of the

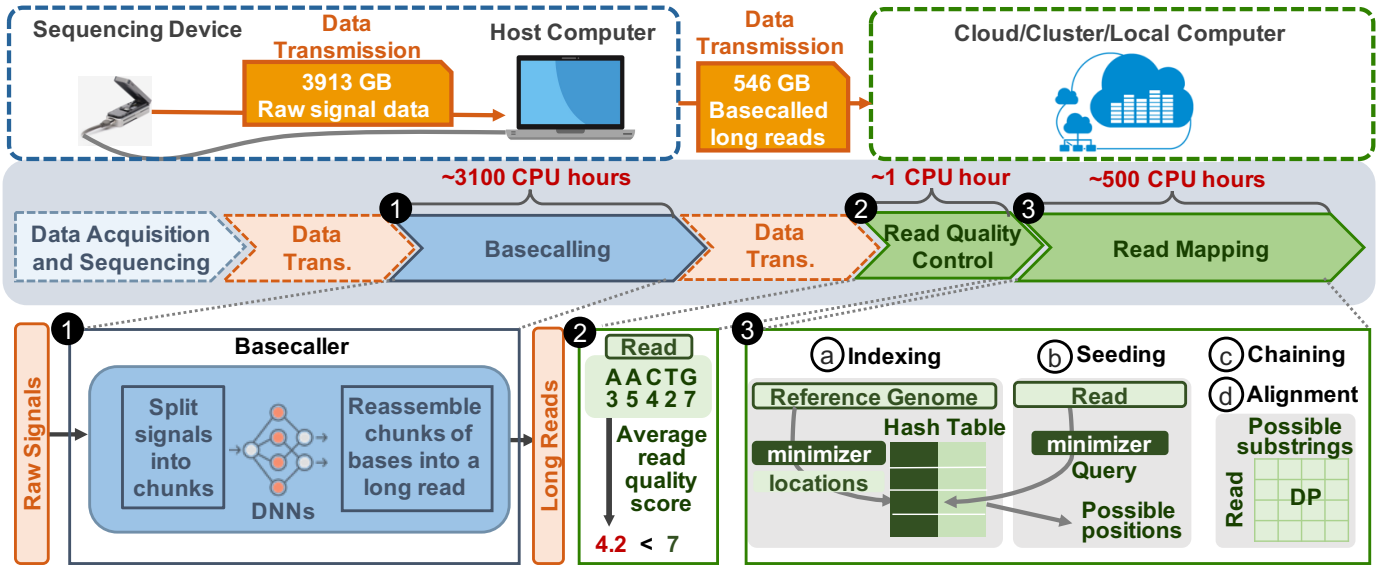

Figure 1: The genome sequencing and analysis pipeline. The basecalling step (❶) and the read mapping step (❸) are the two most time-consuming steps in the genome analysis pipeline. The read quality control step (❷) is a highly-recommended but optional step to reduce the workload of read mapping by eliminating unnecessary computation. Dataset sizes and processing times are from [85].

entire genome analysis pipeline, and (2) reduce the computation and memory overheads associated with processing such low-quality reads in later steps (e.g., read mapping). First, RQC calculates the average quality score of a read by summing the quality scores of each of the read's bases and then dividing this sum by the number of bases. Second, RQC uses a threshold to categorize the reads into low-quality and high-quality groups, and filters out the low-quality reads before performing read mapping. For example, several prior works consider a read with an average quality score of less than *7* as a low-quality read that is not useful in further analysis steps [97, 98].

In the genome analysis pipeline, basecalling and read mapping steps are usually executed on different machines [99]. The computer used for basecalling is usually located in the *wet lab* with the sequencing device [53]. Later analysis steps are executed on completely separate (and usually physically distant) machines located in the *dry lab*. In the entire pipeline, basecalling and read mapping are two of the most time-consuming computational steps [5, 6, 100]. In the real system study shown in [85] and pictorially demonstrated in Figure 1 (middle), the basecalling step takes $\sim$ 3100 CPU hours and the read mapping step takes $\sim$ 500 CPU hours. This motivates system designers to accelerate two key computational steps, basecalling and read mapping. Next, we describe the shortcomings of prior accelerator designs in the context of the entire genome analysis pipeline.

2.2. State-of-the-art Solutions

Several works propose hardware accelerators for basecalling [63,77,78] or read mapping [54,56–58,62,65–68,71,79–83]. Among these accelerators, non-volatile memory (NVM)-based processing in memory (PIM) accelerators offer high performance and efficiency since NVM-based PIM provides in-situ and highly-parallel computation support for matrix-vector multiplications (MVM) [101–111] and string matching operations [112–131], two major operations used in the genome analysis pipeline. MVM is the main operation in DNN-based basecallers [51, 52] and string matching is the main operation in read mapping [92].

NVM-based PIM Array for MVM Operations. Helix [63] is the state-of-the-art basecalling accelerator that exploits an NVM-based PIM array designed for efficiently performing MVM operations. Figure 2 shows the basic structure of an NVM-based PIM array designed for the MVM operation [132]. The NVM-based PIM array performs in-situ MVM operation by applying 1) voltages (V represents the input vector) on the wordlines of the array that stores the matrix (M) and 2) sensing the output vector (O) on the bitlines. In the MVM operation ($O = V \times M$), the PIM array uses the resistance (R) of each NVM cell to represent the corresponding element of matrix M ($M_{i,j} = 1/R_{i,j}$). Based on Kirchhoff's Law, the currents sensed on the bitlines represent O. Using the PIM array, an MVM operation can be performed inside the NVM array in nearly a single NVM read cycle if the matrix fits in the PIM array.

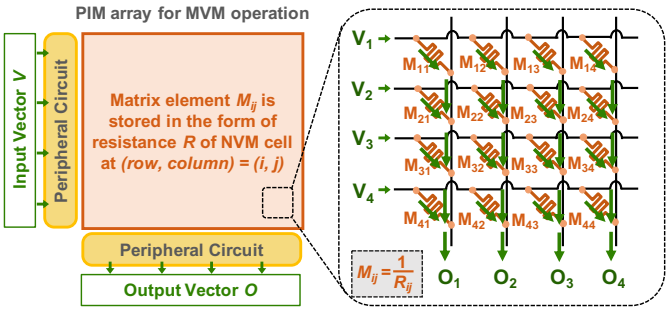

Figure 2: The basic structure of an NVM-based PIM array designed for computing an MVM operation.

NVM-based PIM Array for String Matching Operations. Content addressable memory (CAM) is often leveraged for accelerating string matching operations. PARC [88] is the state-of-the-art work that accelerates the computationally-expensive chaining step in read mapping using an NVM-based CAM. Figure 3 shows an example NVM-based CAM used for string matching. The CAM array consists of $m \times n$ CAM cells that house m reference strings, each of which is n-*bit* long. Each CAM cell stores *one* bit and has two programmable resistors (R_l and R_r) and two transistors (M_l and M_r) (Figure 3❶). To store 1 (or 0) in a CAM cell, R_l and R_r are programmed to high and low (or low and high) resistance, respectively (Figure 3ⓐⓑ).

The NVM-based CAM array is able to query the existence of an n-*bit* string in parallel across all m rows. First, the CAM array precharges the matchline signals to *high* voltage (❷). Second, each bit in the input string and its complement drive the gate voltages of M_l and M_r transistors of the CAM cells in the corresponding column, respectively (❸). Third, each CAM cell compares its stored bit to the corresponding bit in the input string. If these two bits are different, the pull down network in the CAM cell is turned on and the matchline becomes "0". Otherwise, the matchline keeps its precharged *high* voltage. We elaborate on this operation using an example. Assume that the bit stored in a CAM cell is "1", which means R_l and R_r are high and low resistance (ⓐ), respectively. Having "1" in the corresponding bit of the input string implies that transistors M_l and M_r are *on* and *off*, respectively. Hence, none of the pull down circuits are active in this CAM cell since 1) the left circuit cannot drain current due to the high resistance value of R_l, and 2) the right circuit cannot also, due to the off transistor M_r. As a result, matchline keeps its *high* voltage indicating that it is a match in this CAM cell. However, having "0" in the corresponding bit of the input string turns on the right pull down circuit and discharges the matchline signal (R_r is low resistance and M_r is on). Fourth, if all bits of the input string match with all corresponding CAM cells in a row, the matchline will remain high, indicating an *exact match* between the input string and the reference string stored in the CAM array (❹).

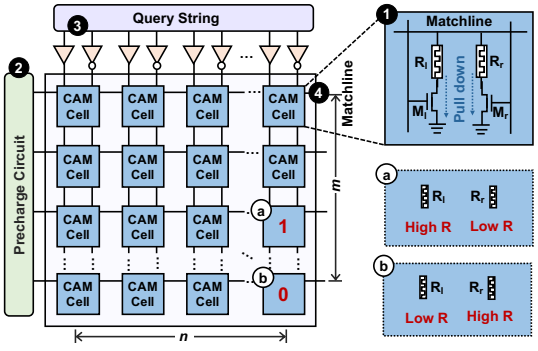

Figure 3: An example NVM-based CAM array for string matching.

2.3. Limitations of the State-of-the-art Accelerators

Although state-of-the-art works accelerate the basecalling and read mapping steps separately, no prior system is designed to support multiple key steps of the genome analysis pipeline in a single accelerator design. This leads to two important limitations: (1) Different steps in the genome analysis pipeline are separated from each other and executed in different devices, which results in in a large amount of data movement between the steps. (2) A considerable portion of computation done in the genome analysis pipeline is likely to be useless due to low-quality or unmapped reads. Next, we describe each of these limitations in more detail.

First, executing the genome analysis steps separately from each other generates a large amount of data movement between the machine that performs basecalling, and the machines perform read quality control and read mapping, as shown in the example in Figure 1. Such data movement introduces two main issues: 1) A large amount of intermediate data (e.g., 3913 GB raw signal dataset and 546 GB basecalled reads, based on real datasets analyzed in [85]) needs to be stored in large memory or storage structures. 2) Transferring data between different machines that execute the different steps is both time-consuming and energy hungry, and it significantly bottlenecks both the performance and energy efficiency of the entire genome analysis pipeline. When machines use state-of-the-art accelerators (Section 2.2), data movement between different machines becomes an even larger bottleneck as computation time reduces with fast yet separate accelerators.

Second, a considerable amount of useless data that flows through the genome analysis pipeline wastes computation and memory resources. Even though read quality control (Section 2.1) filters out the low-quality reads, these reads have already been processed by the expensive basecalling step (because basecalling happens earlier in a separate machine). To quantitatively show the amount of low-quality reads, we perform a descriptive statistical analysis on the Escherichia coli (i.e., E. coli) genome dataset [87]. We make a key observation that a large number of reads (20.5% in [87]) are basecalled but eventually discarded, including very long reads. Besides the low-quality reads, some high-quality reads cannot be mapped (called *unmapped reads*) to the reference genome due to high dissimilarity [92]. To quantitatively show the amount of unmapped reads, we map E. coli reads [87] to the reference genome and find that 10% of all reads are unmapped. Thus, a total of 30.5% of all reads in the E. coli dataset are useless. Such a large amount of useless reads motivates us to reject such reads as soon as possible (ideally even before they go through basecalling) to reduce the computation and memory overheads caused by them.

2.4. Potential Benefits

We would like to quantitatively demonstrate the potential benefits of overcoming the two limitations we identify in prior works [54, 56–58, 62, 63, 65–68, 71, 77–83, 88]. To this end, we devise a study to compare performance of the following four systems using the E. coli dataset we describe in Section 2.3:[1]

System A. **Current practice.** This system separately executes the state-of-the-art open-source basecalling and read mapping software, Bonito [51] and Minimap2 [92]. Each software is executed on a separate machine, a state-of-the-art GPU

[1] We provide our methodology in Section 5

machine [133] for Bonito [51] and a state-of-the-art CPU server [134] for Minimap2 [92]. Reads whose average quality score is less than 7 are discarded after basecalling but before read mapping.

System B. **State-of-the-art accelerators.** The basecalling and read mapping steps are executed in separate NVM-based PIM accelerators, Helix [63] and PARC [88]. The read quality control step is executed in a state-of-the-art CPU [134].

System C. **Accelerators with no data movement overhead.** This system is an idealized version of System B where we ideally eliminate all data movement between separate NVM-based accelerators and the CPU. We demonstrate this ideal system to show the potential benefit of eliminating data movement between separate accelerators and CPUs executing different parts of the genome analysis pipeline. We assume there is *no* data movement between these NVM-based PIM accelerators and the CPU by removing the latency of data movement in our analysis.

System D. **No data movement and no useless reads.** This system is an even more ideal version of System C. Here, we ideally eliminate useless and unmapped reads even before they are basecalled. As such, useless and unmapped reads do not have any overhead in the pipeline.

Figure 4 shows the speedup of using Systems B, C, and D normalized to the performance of System A. We make two observations. First, both System A and System B are bottlenecked by data movement and useless reads. System C and System D provide 2.23× and 3.28× speedup over System B, respectively, by eliminating these bottlenecks. Second, there is a significant potential (as System C and System D show) to accelerate the current practice (System A) by tightly integrating the basecalling and read mapping accelerators to reduce both data movement and useless computation due to useless reads.

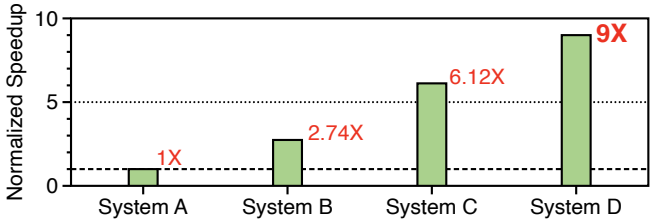

Figure 4: Performance comparison between four different systems.

2.5. Our Goal

We aim to 1) reduce the data movement in the genome analysis pipeline effectively and 2) avoid processing useless reads as early as possible in the genome analysis pipeline. To this end, we tightly integrate the computation of basecalling, read quality control, and read mapping steps inside the memory. Doing so provides two major opportunities for optimizing the genome analysis pipeline holistically: (1) Consuming intermediate data items as soon as they are generated. This eliminates the need for both storing intermediate data items in main memory and storage and transferring them via slow and energy-hungry interconnects. (2) Providing timely feedback from read quality control and read mapping steps to terminate basecalling as soon as possible when the read is determined to be useless (i.e., low-quality or unmapped).

3. GenPIP: Overview

In this section, we present GenPIP, a fast and energy-efficient in-memory system for holistically accelerating the genome analysis pipeline. We envision GenPIP to be best implemented inside the sequencing machine. The key idea of GenPIP is to tightly integrate the two key steps of genome analysis (basecalling and read mapping) inside main memory to (1) minimize data movement by eliminating the need to store intermediate results and (2) minimize useless computation in the basecalling step that leads to unused outputs by providing timely feedback from read quality control and read mapping steps to the basecalling step. GenPIP is equipped with two key techniques: (1) chunk-based pipeline (CP) and (2) early rejection technique (ER). CP is a chunk-based pipeline that provides fine-grained collaboration of basecalling, read quality control, and read mapping steps by processing reads at chunk granularity (i.e., a subsequence of a read, e.g., 300 bases). GenPIP applies ER on top of CP to predict reads that will not be useful downstream by sampling the quality of a number of chunks in each read and stop the execution of CP for low-quality or unmapped reads. ER includes two sub-techniques: (1) rejection based on *quality score*, which executes after basecalling but before read mapping, and (2) rejection based on *chaining score*, which executes during read mapping. The rest of this section explains GenPIP's two key mechanisms (CP and ER)

3.1. Chunk-based Pipeline (CP)

CP processes reads at the granularity of a chunk (i.e., a subsequence of a read, e.g., 300 bases, instead of the complete read sequence, e.g., 90k bases) to increase parallelism and the utilization of computation resources by overlapping the execution of different steps in the genome analysis pipeline. Figure 5 compares CP to the conventional pipeline. In the conventional pipeline (Figure 5(a)), basecalling is executed at the granularity of a chunk, while subsequent read quality control and read mapping steps are executed at the granularity of a read (i.e., assembled by basecalling of tens to hundreds of chunks). We observe that most of the computations performed in the read quality control and read mapping steps do *not* require the information of an entire basecalled read. For example, once a chunk is basecalled, we can calculate its average quality score, perform seeding (query the minimizers of this chunk), and perform chaining with the possible locations in this chunk. While this chunk is going through quality control and read mapping, the next chunk is basecalled. In other words, a significant part of read quality control and read mapping steps can be performed concurrently with basecalling. After the last chunk goes through read quality control and read mapping, CP merges the results of all chunks in a read and outputs the read as the input to the sequence alignment step.

We illustrate the CP mechanism using an example, assuming a read of length $2c$ has two chunks, each of which has a length of c. We explain the independent and concurrent execution of read quality control and basecalling in detail. The conventional pipeline calculates the average read quality score (AQS_{read}) by calculating the average value of the quality scores of all bases

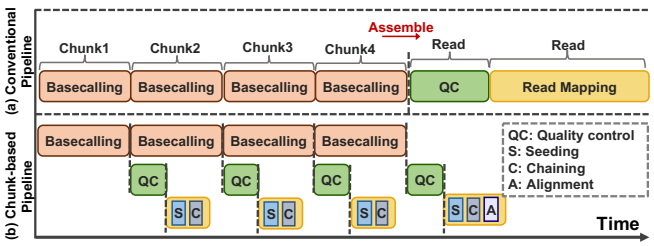

Figure 5: Conventional pipeline (a) vs. the chunk-based pipeline (CP) of GenPIP (b).

in the read (i.e., $q_1, q_2, ..., q_{2c}$) after the *entire* read is basecalled (Equation 1), yet the *SQS* of a chunk (i.e., sum of the quality scores of its bases) (SQS_{first}) can be calculated as soon as *that particular chunk* is basecalled (Equation 2). After basecalling the next chunk, CP calculates the *SQS* of this chunk and merges it with the result of the previous one (Equation 3) to calculate the AQS_{read} of the entire read.

$$AQS_{read} = (q_1 + q_2 + ... + q_c + q_{c+1} + q_{c+2} + ... + q_{2c})/2c \quad (1)$$

$$SQS_{first} = q_1 + q_2 + ... + q_c \quad (2)$$

$$AQS_{read} = (SQS_{first} + q_{c+1} + q_{c+2} + ... + q_{2c})/2c \quad (3)$$

Similarly, we use our example to explain the independent and concurrent execution of the seeding and chaining steps. As soon as the seeding step obtains a set of minimizer hits in the first chunk, the chaining step can work on the output of seeding step while the seeding step can obtain a set of minimizer hits in the second chunk. In the end, the chaining step combines the results from the two chunks.

As we described above, by tightly integrating the basecalling, read quality control, and read mapping steps inside the sequencing machine, we can pipeline the execution of these steps at the granularity of a chunk. Based on this insight, we propose a chunk-based pipeline, called CP, that executes the partial computations of read quality control, seeding, and chaining once a chunk is basecalled. Figure 5(b) shows our CP design. As the figure shows, chunk-based basecalling, read quality control, and a part of read mapping (seeding and chaining) are pipelined. The chunk-based execution flow not only saves time via pipelined execution (by overlapping the execution of several steps), but also reduces the need for storing intermediate data as each pipeline step can quickly consume the small amount of output that is produced by the previous step.

3.2. Early Rejection Based on Chunks (ER)

The goal of ER is to predict and eliminate low-quality and unmapped reads from both basecalling and read mapping steps. Doing so can significantly lower the execution time of the entire genome analysis pipeline. To achieve this goal, the key idea of ER is to use information about several basecalled chunks in a read to predict the quality and usefulness of the read. GenPIP applies the ER technique on top of CP. Figure 6 shows the overview of ER. Instead of basecalling *all* N_{total} (e.g., tens to hundreds) chunks in a read and then checking the average quality score of the *entire* read (as done in conventional systems), ER first checks the average quality score of only *a small number* of (i.e., N_{qs}) chunks basecalled by CP (Figure 6❶❷). If the read fails this chunk-based quality score check, then ER stops basecalling the remaining chunks in the read and discards the read (❸ⓐ). Otherwise, CP basecalls some more chunks (i.e., N_{cm}) in the read (❸ⓑ) and then maps the basecalled chunks so far (i.e., $N_{qs} + N_{cm}$ chunks) (❹). ER checks the chaining score of the $N_{qs} + N_{cm}$ basecalled chunks (❺) (i.e., it predicts the likelihood of mapping the read to the reference genome). If the read fails the chunk-based chaining score check, ER stops basecalling the remaining chunks and discards the read (❻ⓐ). Otherwise, CP basecalls the remaining chunks (i.e., $N_{total} - (N_{qs} + N_{cm})$ chunks) in the read (❻ⓑ) and executes the remaining computation in read mapping (❼).

As such, ER involves two filtering steps: (1) rejection based on the quality score of N_{qs} chunks (❷) and (2) rejection based on the chaining score of $N_{qs} + N_{cm}$ chunks (❺). We describe these two filtering steps in detail.

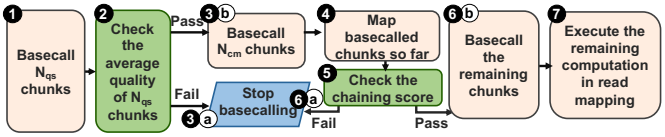

Figure 6: Overview of the early rejection (ER) technique in the genome analysis pipeline (the green boxes ❷❺ are the two early-rejection steps we introduce).

3.2.1. Early Rejection Based on Chunk Quality Scores. The early rejection technique based on the quality score of chunks relies on how accurately it can estimate the quality of the entire read by checking the quality of a small number of (i.e., N_{qs}) sampled chunks. We first investigate whether or not it is possible to accurately estimate the quality of the entire read using a small number of chunks. To this end, we study chunk quality scores from both low-quality reads and high-quality reads in the E. coli [87] dataset (Section 2.3) using a chunk size of 300 bases. As a representative example, Figure 7 shows the chunk quality scores in a low-quality read (Figure 7(a)) and a high-quality read (Figure 7(b)).

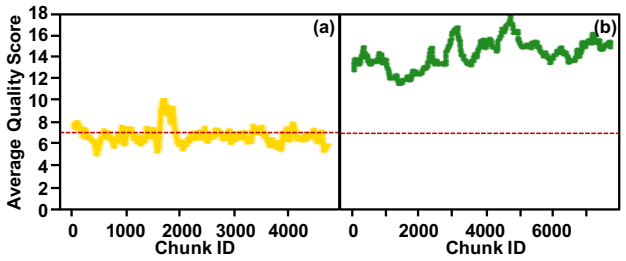

Figure 7: The quality scores of the chunks in two representative reads: (a) a low-quality read and (b) a high-quality read.

We make three key observations: (1) The range of quality scores for the chunks extracted from high-quality reads (e.g., ranging from 11 to 18 in Figure 7(b)) is greatly higher than that from low-quality reads (e.g., ranging from 4 to 10 in Figure 7(a)). (2) A single chunk's quality score is not enough to predict the read quality score because there are many chunks whose quality scores are larger than 7 (which is a common

threshold used to distinguish between low-quality and high-quality reads, as shown in [97, 98]) in a low-quality read. (3) Consecutive chunks' quality scores are usually close to each other, indicating that sampling consecutive chunks may not be representative enough to estimate the quality of an entire read. We conclude that early rejection based on the quality score of chunks should sample a small number of *non-consecutive* chunks to accurately guess whether or not a read is low-quality.

Leveraging our key conclusion, we propose an early rejection technique based on the quality score of chunks, called Quality-Score-based Rejection (*QSR*). *QSR* 1) calculates the average quality score of a set of non-consecutively sampled chunks (i.e., N_{qs} chunks) in a read, and 2) predicts the entire read as either low quality or high quality, by comparing the calculated average score of the N_{qs} chunks to the quality score threshold (θ_{qs}).

Algorithm 1 illustrates the procedure of the *QSR* technique. *QSR* 1) samples N_{qs} chunks that are evenly distributed in a read, 2) calculates the sum of the quality scores of these sampled chunks (lines 1-3), and 3) uses the average quality score of these sampled chunks (line 4) to predict whether or not the quality score of the read is higher than the quality score threshold, θ_{qs} (lines 5-9).

We determine the number of sampled chunks (N_{qs}) by a one-time preprocessing of the chunk quality scores of the reads of a species (see Section 6.3 for more detail). For example, we experimentally observe that sampling only two chunks is enough for accurate read quality prediction in E. coli [87].

Algorithm 1: Quality-Score-based Rejection (QSR)

```
   Input: the original read: read_original;
   length of the original read: N;
   chunk size: C
   number of chunks needed for QSR: N_qs;
   quality score threshold: θ_qs;
   Output: rejection
1  for i=0;i< N_qs;i++ do
2  |   sum_sample_score += quality score of the chunk located at
   |     ⌊i/(N_qs − 1)⌋ × ⌊N/C⌋ in read_original //sum the quality scores of
   |     evenly-sampled chunks in the read
3  end
4  average_score = sum_sample_score/N_qs;
5  if average_score < θ_qs then
6  |   return rejection = TRUE;
7  else
8  |   return rejection = FALSE;
9  end
```

3.2.2. Early Rejection Based on Chunk Mapping. The key idea of the chunk-mapping-based early rejection technique is that a read probably cannot be mapped to the reference genome if enough consecutive chunks in this read cannot be mapped to the reference genome (i.e., the chaining score of the minimizers in these chunks is lower than a threshold). Unfortunately, mapping short chunks provides too large a list of possible mapping locations. To predict whether or not a read can be mapped to the reference genome, our technique needs larger chunk sizes.

We propose a chunk-mapping-based early rejection mechanism, *CMR*, that is based on three key steps: (1) *CMR* basecalls a number of (N_{cm}) continuous chunks. (2) *CMR* combines the N_{cm} continuous chunks into a larger chunk (e.g., by combining five continuous 300-base chunks to create a larger 1500-base chunk). (3) *CMR* maps the large chunk to the reference genome and checks the chaining score. If the chaining score is lower than a threshold θ_{cm} (indicating that this chunk is significantly different from the reference genome), *CMR* rejects the read and stops basecalling it. We determine the value of N_{cm} via a one-time preprocessing of the reads of a species. For example, we experimentally find that combining five consecutive 300-base chunks can effectively predicts the mapping behavior of the reads in the E. coli dataset [87] (see Section 6.3 for more detail).

4. GenPIP: Architecture & Implementation

In this section, we describe the architecture and implementation of GenPIP. Figure 8 shows the overview of the GenPIP architecture. GenPIP architecture has three modules: the basecalling module (Figure 8ⓐ), the read mapping module (ⓑ), and the GenPIP controller (ⓒ). The basecalling module (ⓐ) has two main units: 1) a PIM basecaller similar to prior work [63] (❶) and 2) a new PIM accelerator for chunk quality score calculation (PIM-CQS ❷). The read mapping module (ⓑ) has three main units: 1) a new PIM accelerator for the seeding step (❸), 2) the read mapping controller (❹), and 3) dynamic programming units for chaining and alignment steps similar to prior work [88]. The GenPIP controller (ⓒ) aims to 1) control the execution of CP and 2) issue early rejection commands using ER. In this section, we first explain how the GenPIP architecture implements CP and CP + ER by providing a detailed walkthrough over GenPIP components (Section 4.1). We then explain the details of GenPIP's new components in Sections 4.2-4.4.

4.1. Detailed Walkthrough

Chunk-Based Pipeline (CP) in the GenPIP Architecture. We first describe the operation of CP without ER. First, the GenPIP controller (ⓒ) receives raw electrical signals from the sequencing machine and stores these signals in the *read queue*. Second, the GenPIP controller sends the raw signals to the PIM basecaller (❶) in the basecalling module (ⓐ) chunk by chunk. Third, the PIM basecaller translates each chunk into nucleotide bases using a deep neural network. For the PIM basecaller, GenPIP uses a similar architecture as Helix [63], the state-of-the-art NVM-based PIM accelerator for basecalling. The PIM basecaller performs the inference of basecaller's neural network via an NVM-based PIM array for matrix-vector computation (as described in Section 2.2) and calculates the quality score for *each base* after the inference. The PIM basecaller stores the basecalled chunks in a global buffer. Fourth, after a chunk is basecalled, PIM-CQS calculates the chunk quality score (CQS) by summing the quality scores of the chunk's bases. Fifth, the basecalling module sends the basecalled chunk together with its CQS to the GenPIP controller.

Sixth, the GenPIP controller stores the basecalled chunk inside the *chunk buffer*, and forwards the chunk to the read mapping module (ⓑ). Seventh, the read mapping module first performs the seeding operation on each basecalled chunk to

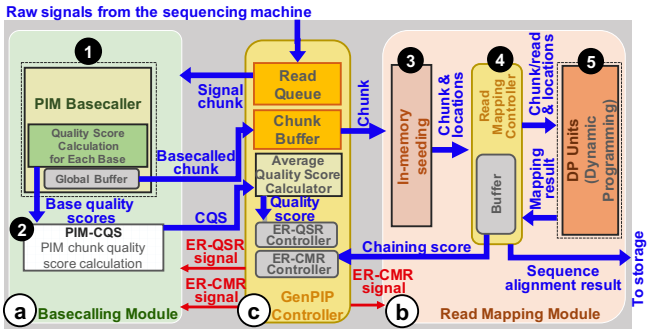

Figure 8: Architecture overview of GenPIP. ⓐ The basecalling module. ⓑ The read mapping module. ⓒ The GenPIP controller.

identify the possible match locations in the reference genome. For the seeding step, we design a new PIM-based accelerator, the in-memory seeding component (❸), to enable fast and energy efficient seeding (see Section 4.4). The seeding component sends a list of the possible match locations to the read mapping controller (❹). Eighth, the read mapping controller sends the chunk and its possible match locations to the DP units to perform chaining (❺). For chaining, GenPIP uses a similar design as PARC [88], the state-of-the-art NVM-based PIM accelerator for chaining. PARC customizes an NVM-based CAM array to implement a DP algorithm used for chaining and alignment. Ninth, the read mapping controller stores the chaining results in its *buffer*. After finishing the chaining step for all chunks in a read, the read mapping controller compares the chaining score of the entire read with a threshold (θ_{cm}). If the chaining score is lower than this threshold, the read mapping controller stops the read mapping for this read. Otherwise, the read mapping controller assembles the entire basecalled read and sends the read to DP units to execute the sequence alignment step. For sequence alignment, GenPIP uses the same hardware units used for chaining while modifying its penalty score calculation, similar to PARC [88]. Tenth, the read mapping controller sends the mapping result to the storage after sequence alignment.

CP Working in Tandem with ER in the GenPIP Architecture. We describe how ER is integrated with CP. As discussed in Section 3.2, ER includes two sub-techniques, ER-*QSR* and ER-*CMR*. GenPIP supports ER-*QSR* using the PIM-CQS unit in the basecalling module (ⓐ❷) and the GenPIP controller (ⓒ). The PIM-CQS unit (1) receives the first N_{qs} basecalled chunks (depending on the dataset; see Section 6.3), (2) calculates the quality score of the basecalled chunks, and (3) sends the chunk quality scores to the GenPIP controller (ⓒ). The GenPIP controller (1) calculates the average quality score of the N_{qs} chunks by using the average quality score calculator unit and (2) compares the average quality score with the threshold (θ_{qs}) by using the ER-*QSR* controller unit. If the quality score is lower than the threshold (i.e., if the read is predicted to be low-quality), the GenPIP controller sends the ER–*QSR* signal to the basecalling module to terminate basecalling on the current read, and starts processing the next read. To support ER-*CMR*, the read mapping controller (❹) (1) combines N_{cm} chunks (depending on the dataset; see Section 6.3) to create a larger chunk, (2) sends the large chunk for chaining, and (3) sends the chaining score of the large chunk to the ER-*CMR* controller inside the GenPIP controller. ER-*CMR* controller compares the chaining score with the threshold (θ_{cm}). If the chaining score is lower than the threshold (i.e., if the read is predicted to be unmapped), the GenPIP controller sends the ER-*CMR* signal to the basecalling module and read mapping module to terminate CP processing for the current read.

4.2. The GenPIP Controller

The GenPIP controller (ⓒ) (1) communicates with the basecalling module (ⓐ) and the read mapping module (ⓑ) to control the chunk-based execution of the genome analysis pipeline, (2) issues early-rejection signals to basecalling and read mapping modules, and (3) merges the quality scores of basecalled chunks. The GenPIP controller has five key units: read queue, chunk buffer, average quality score calculator, ER-*QSR* controller, and ER-*CMR* controller. We explain each unit in more detail.

Read Queue. The GenPIP controller uses the read queue to store raw electrical signals. The sequencing machine enqueues raw electrical signals to this queue, and the controller dequeues them for the basecalling process. GenPIP sizes this buffer as large as needed to store the longest signal (which is around 6 MB [1, 50]). GenPIP can use different memory technologies to build the read queue. However, the memory technology should provide high write endurance, low read/write energy consumption, low read/write latency, and high density. We find that eDRAM [135] is an example memory technology that provides a good tradeoff across these optimization goals. Related work [63, 136] also uses eDRAM-based buffers for the same reason.

Chunk Buffer. The GenPIP controller uses the chunk buffer to store the basecalled chunks. The chunk buffer keeps the basecalled chunks until the end of sequence alignment process for an entire read, unless ER terminates the process of the read. In GenPIP, the chunk buffer is able to house 2.3 million bases, which is the longest read length based on state-of-the-art nanopore sequencing technology [1]. GenPIP uses the eDRAM technology for the chunk buffer, for the same reasons as it does for the read queue.

Average Quality Score (AQS) Calculator. The GenPIP controller uses the *AQS* calculator unit to calculate the average quality score of either an entire read or N_{qs} chunks for ER-*QSR*. The *AQS* calculator unit has a buffer that keeps the sum of the quality scores of the chunks it has received so far. Once the *AQS* unit receives all basecalled chunks for the read, it divides the final calculated sum by the total number of chunks to calculate the average quality score for the entire read.

ER-QSR Controller. This unit receives the average quality score of N_{qs} chunks from the *AQS* calculator unit and compares it with the threshold (θ_{qs}) to predict whether or not the read is low-quality. If so, the GenPIP controller issues ER-*QSR* signal to the basecalling module to stop CP processing for the predicted low-quality read.

ER-CMR Controller. This unit receives the chaining score of a *large* chunk (assembled with N_{cm} chunks) from the read mapping module and compares it with the threshold (θ_{cm}) to predict whether or not the read is unmapped. If so, the GenPIP controller issues the ER-*CMR* signal to stop CP processing for the predicted-unmapped read.

4.3. Timely Early Rejection

This section explains how we implement the ER technique in GenPIP to predict low-quality and unmapped reads in a timely fashion, and stop the execution of the genome analysis pipeline on such reads. We describe the implementation of early rejection based on chunk quality scores (ER-*QSR*) and early rejection based on chunk mapping (ER-*CMR*) in Sections 4.3.1 and 4.3.2, respectively.

4.3.1. ER-QSR Implementation. As described in Section 3.2.1, the goal of *QSR* is to calculate the quality score of a small number of sampled basecalled chunks (N_{qs} chunks) and compare the result with the threshold of *QSR* (θ_{qs}). GenPIP implements this technique partly in the basecalling module (ⓐ) and partly in the GenPIP controller (ⓒ). Inside the basecalling module, we add a new unit, PIM-CQS (❷), to calculate a chunk's quality score by summing the quality scores of its bases. PIM-CQS is an NVM-based PIM array that performs the MVM operation (as described in Section 2.2). We use the PIM-CQS unit to perform the summation of the quality scores of the bases in a chunk by (1) storing the quality scores of bases in a column and (2) inputting an all-1 vector so that a dot product becomes a simple addition. The basecalling module sends the results (i.e., chunk quality scores) to the GenPIP controller. The GenPIP controller calculates the average quality score of the sampled chunks and compares it with the threshold (θ_{qs}). The GenPIP controller sends the ER-*QSR* signal to the basecalling module to terminate basecalling on the current read if the calculated average quality score is lower than θ_{qs}.

4.3.2. ER-CMR Implementation. As described in Section 3.2.2, the goal of *CMR* is to check the chaining score of a larger chunk, which is assembled by combining a small number of consecutive chunks N_{cm} chunks, to estimate whether or not the entire read is unmapped. We implement *CMR* partly inside the read mapping module (ⓑ) and partly inside the GenPIP controller (ⓒ). Inside the read mapping module, the read mapping controller (❹) enqueues the basecalled chunks in its buffer after the seeding step. Once the controller has N_{cm} chunks in the buffer, it assembles a larger chunk and sends the large chunk to the chaining step. The read mapping controller sends the chaining score to the ER-*CMR* controller inside the GenPIP controller. The ER-*CMR* compares the chaining score of the large chunk with the θ_{cm} threshold. If the chaining score is lower than the threshold, the GenPIP controller sends the ER-*CMR* signal to the basecalling module and the read mapping module to terminate the execution of CP on the current predicted-unmapped read.

4.4. In-Memory Seeding

As described in Section 2.1, the seeding component aims to generate query strings (e.g., minimizers) from a basecalled chunk and queries them in the hash table to quickly find matching regions between the reference genome and the chunk. To this end, we design a new in-memory seeding accelerator (Figure 8❸ to speed up the process of seeding so that it can keep up with other components of GenPIP. Figure 9 shows the components of the in-memory seeding accelerator. There are four main components: an eDRAM buffer (Figure 9①), the query string generator (②), ReRAM-based CAM (③) and RAM arrays (④). First, the GenPIP controller writes a basecalled chunk into the seeding unit's eDRAM buffer (①). Second, the seeding unit moves a substring from the chunk to the query string generator (②). Third, the query string generator uses each substring to generate multiple query strings by shifting the substring one base at a time. Fourth, the seeding module queries each query string using the ReRAM-based CAM and RAM arrays. GenPIP stores multiple reference strings inside the CAM array (as the *keys*, ③), and the locations of the reference strings in the reference genome inside the ReRAM-based RAM array (as the *values*, ④). The implementation of ReRAM-based CAM array is similar to what we explain in Section 2.2. If the query string matches one reference string in the ReRAM-based CAM, the ReRAM-based CAM outputs the address (*Addr.*) to access the corresponding values (i.e., the possible match locations) stored inside the ReRAM-based RAM. The seeding unit then reads out the list of possible match locations in the reference genome for that particular reference string and stores the possible locations in the eDRAM buffer. Fifth, the seeding unit outputs the possible match locations of the chunk to the read mapping controller unit.

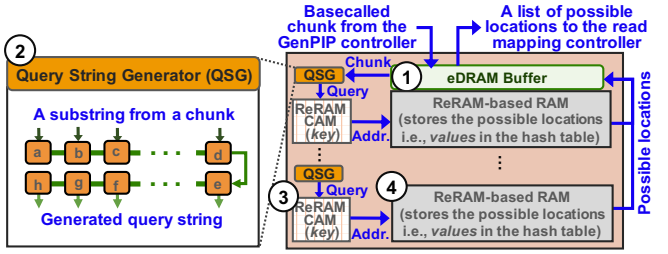

Figure 9: Microarchitecture of the in-memory seeding accelerator.

5. Evaluation Methodology

Performance, Power, and Area Analysis. We implement an in-house simulator to evaluate the performance, energy consumption, and handware area overhead of GenPIP. Since GenPIP includes several components, we embed the latency, power, and area values for each GenPIP component in our simulator. To calculate these values, we use different tools depending on the technology of the component. We use Verilog HDL to implement the logic components in GenPIP. To estimate the area and power consumption of logic components, we synthesize our HDL implementation using the Synopsys Design Compiler [137] with a 32nm process technology node at 1.6 GHz clock frequency. To model the performance, energy, and area

of our ReRAM-based RAM and CAM arrays, we use the state-of-the-art models for non-volatile memories, NVSim [138] and NVSim-CAM [114], respectively. We use CACTI 6.5 [139] to model the performance, energy, and area of the embedded DRAM (eDRAM). For Helix and PARC accelerators, we use the performance, power and area results reported in the original works [63, 88].

Comparison points. Our goal is to 1) compare GenPIP with the state-of-the-art CPU/GPU implementations of software genome analysis tools and state-of-the-art PIM accelerators, and 2) show the benefits of integrating the key mechanisms of GenPIP (CP and ER) to these CPU/GPU implementations and accelerators. To this end, we evaluate the following systems:

- `CPU`: We use the CPU-based state-of-the-art basecaller, Bonito [51], and the CPU-based read mapper, minimap2 [92] executed on an Intel® Xeon® Gold 5118 CPU [134] at 2.3 GHz, with 192 GB DDR4 memory.
- `CPU-CP`: `CPU` integrated with CP (the chunk-based pipeline technique we describe in Section 3.1).
- `CPU-GP`: `CPU` integrated with both CP and ER (GP stands for GenPIP).
- `GPU`: We use the GPU implementation of Bonito [51] as the basecaller executed on an NVIDIA GeForce RTX 2080 Ti GPU and the CPU implementation of minimap2 as the read mapper.
- `GPU-CP`: `GPU` integrated with CP.
- `GPU-GP`: `GPU` integrated with both CP and ER.
- `PIM`: To compare to a single PIM-based accelerator that executes both basecalling and read mapping steps, we connect two state-of-the-art PIM-based accelerators for 1) basecalling, Helix [63], and 2) read mapping, PARC [88]. We *optimistically* make the following three assumptions. (1) There is no latency and energy overhead for data movement between the basecalling and read mapping steps when connecting these accelerators. (2) There are processing elements executing the read quality control step without any performance overhead. (3) There is enough memory to store the intermediate data.
- `GenPIP-CP`: Our GenPIP design *only* equipped with CP.
- `GenPIP-CP-QSR`: Our GenPIP design with both CP and *only QSR* in ER.
- `GenPIP`: The full GenPIP design using both CP and ER (i.e., both *QSR* and *CMR*).

Datasets. Table 1 shows the details of the datasets we use in our evaluations. For all of our experiments, we evaluate GenPIP using the datasets that are representatives of small and large genomes to cover the commonly used genome sizes in genome analysis. As a small genome, we use a publicly available dataset of the Escherichia coli (*E. coli*) genome.[2] As a large genome, we use a human genome dataset of the NA12878 sample. The human dataset can be accessed through ENA [140] or NCBI [141] with accession PRJEB30620. Both *E. coli* and human genomes are sequenced using Oxford Nanopore Technologies (ONT) with R9-based chemistry [47]. This chemistry provides sequencing data with around 80-85% sequencing accuracy [142], which is slightly lower than the most recent chemistry (R10.4) that provides around 95-99% accuracy [143]. We include these less accurate datasets in our experiments to show the robustness and effectiveness of GenPIP in the presence of considerable sequencing inaccuracy.

[2]The *E. coli* dataset is available at: `http://lab.loman.net/2016/07/30/nanopore-r9-data-release/`

Table 1: Details of datasets used in the evaluation.

Dataset Details	*E. coli* [144]	*Human* [145]
Mean read length	9,005.90	5,738.30
Mean read quality	7.9	11.3
Median read length	8,652	6,124
Median read quality	9.3	12.1
Number of reads	58,221	449,212
Total bases	524,330,535	2,577,692,011

6. Results

In this section, we present the experimental results of GenPIP, including 1) the performance of GenPIP compared to the baseline systems (Section 6.1), 2) the energy consumption of GenPIP compared to the baseline systems (Section 6.2), 3) the sensitivity analysis of GenPIP (Section 6.3), and 4) the area and power analysis of the GenPIP architecture (Section 6.4).

6.1. Performance Analysis

To study the effect of GenPIP on accelerating the genome analysis pipeline, we measure the performance of GenPIP and the baseline systems. Figure 10 shows the performance of GenPIP compared to the CPU, GPU, and PIM-based systems that we explain in Section 5 (results are normalized to the performance of the `CPU` system). We use chunk sizes of 300, 400, and 500 bases in the evaluation of two datasets, E. coli and human (300 is the suggested chunk size by state-of-the-art basecallers [6, 51]).

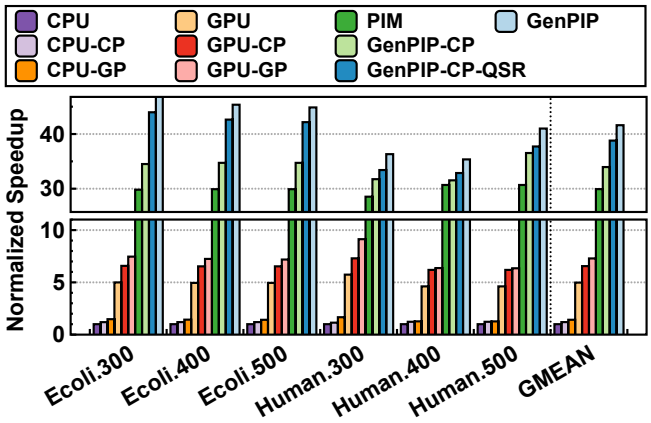

Figure 10: Speedups of various systems normalized to `CPU` (300, 400, and 500 in the x-axis represent the three chunk sizes used in the evaluation).

We make four key observations. First, GenPIP provides 41.6×, 8.4×, and 1.39× speedup compared to the `CPU`, `GPU`, and `PIM` systems, respectively. GenPIP achieves such speedups as it 1) efficiently enables the fine-grained collaboration of the basecalling and the read mapping steps via the CP technique and 2) reduces useless computation via the ER technique. Second, we observe that the `GenPIP-CP`, `GenPIP-CP-QSR`, and `GenPIP`

systems provide 1.16×, 1.32×, and 1.39× speedup compared to the idealized PIM-based accelerator (PIM) that integrates the state-of-the-art basecalling and read mapping accelerators with optimistic assumptions. These speedups identify the main benefits of the key mechanisms of GenPIP that tightly integrate the basecalling and read mapping steps rather than simply connecting two PIM-based accelerators that perform basecalling and read mapping separately (even with idealized assumptions, as we did for PIM). Third, 1) CPU-CP and CPU-GP provide 1.20× and 1.42× speedup compared to CPU, and 2) GPU-CP and GPU-GP provide 1.32× and 1.46× speedup compared to GPU. Implementing the CP and ER techniques significantly improves performance in CPUs and GPUs as these techniques are effective at reducing data movement and useless computation in any system. Fourth, GenPIP's performance benefits do not change significantly with chunk size. We conclude that 1) CP and ER techniques significantly improve the overall performance of genome analysis over the state-of-the-art CPU- and GPU-based approaches, and 2) GenPIP outperforms the optimistic integration of the state-of-the-art PIM-based read mapping and basecalling accelerators.

6.2. Energy Efficiency Analysis

To study the energy efficiency of GenPIP, we measure the energy consumption of GenPIP and the baseline systems. Figure 11 shows the energy savings of each evaluated system normalized to the energy consumption of the CPU system. We make three key observations. First, GenPIP provides 32.8×, 20.8×, and 1.37× energy reduction, compared to CPU, GPU, and PIM systems, respectively. These energy savings are in line with the performance improvements that GenPIP provides by reducing 1) the data movement between the basecalling and read mapping steps and 2) the useless computation due to the low-quality reads and unmapped reads. Second, GenPIP reduces energy by 1.07× and 1.37× than GenPIP-CP-QSR and GenPIP-CP, which shows that filtering based on *both* read quality score and chunk mapping is important to improve the overall energy savings of GenPIP. Third, similar to the performance results, the energy consumption of the evaluated systems is robust to chunk sizes. We conclude that GenPIP is very effective at reducing energy compared to state-of-the-art CPU, GPU, and PIM systems.

6.3. Sensitivity Analysis

In this section, we study the sensitivity of the number of sampled chunks on the effectiveness of ER-*QSR* (Section 6.3.1) and ER-*CMR* (Section 6.3.2). To this end, we calculate two metrics: *rejection ratio* and *false negative ratio*. *Rejection ratio* is ratio of rejected reads (via ER-*QSR* or ER-*CMR*) over all reads. *False negative ratio* is the ratio of *incorrectly* rejected reads over all rejected reads.

6.3.1. Effect of the Number of Sampled Chunks on ER-QSR. To study the effect of the number of sampled chunks on the effectiveness of ER-*QSR*, we calculate the *rejection ratio* and *false negative ratio* metrics while varying the number of sampled chunks from 2 to 6. We identify a rejection as false negative (FN) if ER-*QSR* rejects the read while the average read quality

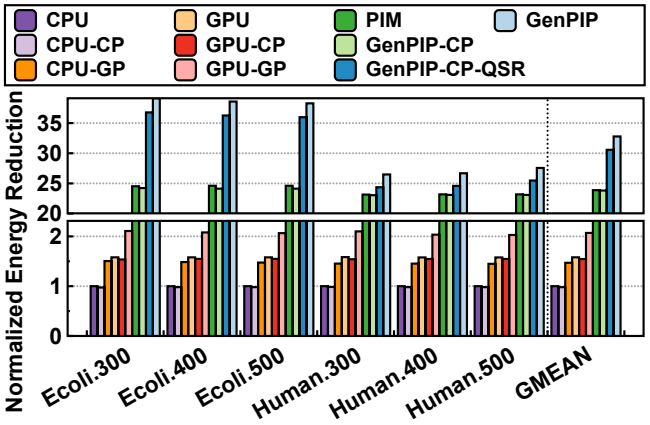

Figure 11: Energy reduction of various systems normalized to CPU (300, 400, and 500 in the x-axis represent the three chunk sizes used in the evaluation).

score of the *entire* read is above the read quality score threshold. Figure 12(a-b) shows the rejection ratio and the false negative ratio for ER-*QSR* using the E. coli and human datasets, respectively. We make three key observations. First, the rejection ratio slightly decreases as the number of sampled chunks increases for both the E. coli and the human datasets. This is because there are many short reads in both datasets that consist of only a few chunks (e.g., 3 chunks); increasing the number of sampled chunks reduces the likelihood of early rejection of such short reads. Second, increasing the number of sampled chunks decreases the false negative ratio for the human dataset but increases the false negative ratio for the E. coli dataset. For the human dataset, increasing the number of sampled chunks provides better read quality prediction accuracy, which leads to a lower false negative ratio. For the E. coli dataset, there are many regions with low-quality chunks although the average quality of reads is high. Using more sampled chunks leads to using more of these low-quality chunks in the read quality prediction, which is the main cause of the false negative predictions in this dataset. Third, the false negative ratio of the human dataset is slightly larger than that of the E. coli dataset. Such large FN ratios are still acceptable since the average alignment score of these incorrectly-rejected reads (14.4) is closer to the average alignment score of low-quality reads (3.9) than the average alignment score of all reads (52.5). Thus, these incorrectly-rejected reads are unlikely to provide high-quality mapping in the read mapping step [92].

Based on our sensitivity analysis, we use two and five sampled chunks for the E. coli and human datasets, respectively, which provides a good balance between achieving a high rejection ratio and achieving a low false negative ratio.

6.3.2. Effect of the Number of Sampled Chunks on ER-CMR. To study the effect of the number of sampled chunks on the effectiveness of ER-*CMR*, we calculate the *rejection ratio* and *false negative ratio* metrics while varying the number of sampled chunks from 1 to 5. We identify a rejection as FN if the read rejected by ER-*CMR* (predicted as unmapped) can be mapped to the reference genome. Figure 13(a-b) shows the rejection ratio and the false negative ratio for ER-*CMR* using

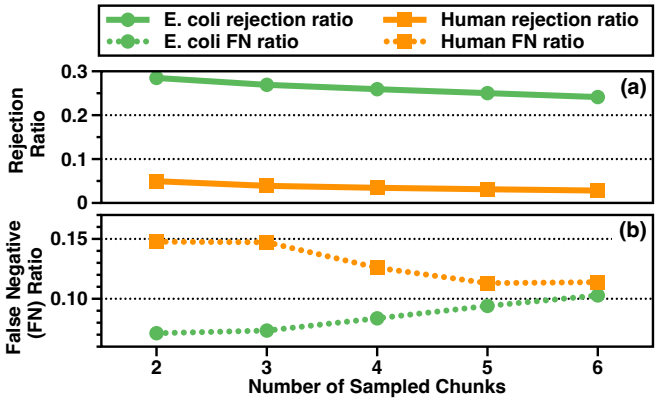

Figure 12: Effect of the number of sampled chunks on ER-QSR's (a) rejection ratio and (b) false negative ratio.

the E.coli and human datasets, respectively. We make two key observations. First, the rejection ratio decreases as the number of sampled chunks increases for both the E. coli and the human datasets. This is due to two main reasons. 1) There are many short reads in both datasets that consist of a few chunks (e.g., 3 chunks). Increasing the number of sampled chunks reduces the likelihood of early rejection of such short reads. 2) Increasing the number of sampled chunks increases the accuracy of ER-*CMR*, which leads to the rejection of fewer reads. Second, the false negative ratio decreases as the number of sampled chunks increases for both the E. coli and the human datasets. This is because using a larger number of sampled chunks results in a larger assembled chunk that is likely more representative of the entire read.

Based on our sensitivity analysis, we use five and three samples for the E. coli and human datasets, respectively, because the false negative ratios they provide are close to zero while the rejection ratios are reasonable (i.e., 6.3% and 5.5%, respectively).

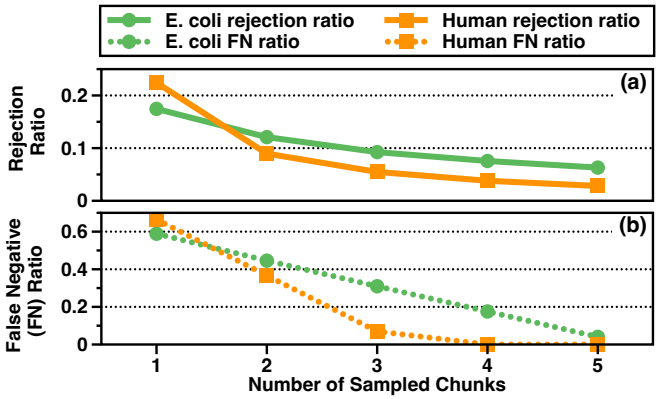

Figure 13: Effect of the number of sampled chunks on ER-CMR's (a) rejection ratio and (b) false negative ratio.

6.4. Area and Power Analysis

To study the area and power overheads of GenPIP, we 1) measure the area and power of the new components designed for GenPIP (Figure 8❷❸❹ⓒ) and 2) use the area and power data of other components as reported in previous works (❶ [63], ❺ [88]). Table 2 shows the area and power breakdown of GenPIP's components in three modules, the basecalling module (ⓐ), the read mapping module (ⓑ), and the GenPIP controller (ⓒ). GenPIP occupies 163.8 mm^2 chip area and consumes 147.2W power at the 32nm technology node. Our analysis shows that the read mapping module is the most expensive module in terms of area and power consumption, as it accounts for 56.9% of the GenPIP total area and 77.8% of the GenPIP total power consumption.

Table 2: Area and power breakdown of GenPIP.

Component	Specification	Power W	Area mm^2
PIM Basecaller ❶	168 Tiles 4MB eDRAM	27.1	49.2
PIM-CQS ❷	SOT-MRAM PIM Array size: 16x1024	0.307	0.0256
Basecalling Module ⓐ Total		**27.4**	**49.2**
Seeding ❸	4096 seeding units ReRAM-based CAM-RAM 8 32×128 CAMs per unit 1 QSG per CAM 8 16KB RAMs per unit 1 4KB eDRAM per unit	28.2	76.68
RMC ❹	4 MB eDRAM	1.346	5.472
DP ❺	1024 units	85	10.9
Read Mapping Module ⓑ Total		**114.5**	**93.1**
Module ⓒ	12 MB eDRAM AQS calculator ER-QSR controller ER-CMR controller	**5.3**	**21.5**
GenPIP Controller Module ⓒ Total			
GenPIP Total		**147.2**	**163.8**

7. Related Work

To our knowledge, GenPIP is the *first* processing-in-memory (PIM) accelerator for the genome analysis pipeline that tightly integrates the two key steps of genome analysis (basecalling and read mapping) to minimize 1) the data movement by eliminating the need to store intermediate results and 2) useless computation due to low-quality and unmapped reads. We have already compared GenPIP extensively to the state-of-the-art CPU-based, GPU-based, and PIM-based systems in Section 6. In this section, we describe other related works in four categories: (1) PIM acceleration of genome analysis, (2) non-PIM acceleration of basecalling, (3) non-PIM acceleration of read mapping, and (4) basecalling-free genome analysis.

PIM Acceleration of Genome Analysis. Previous PIM works focus on the acceleration of either basecalling [63, 77] or read mapping [54, 66, 70, 79, 82–84, 88, 146–158]. For basecalling, previous PIM accelerators [63, 77] accelerate the neural networks of basecallers using non-volatile memory. These accelerators can significantly reduce the performance and energy overheads associated with frequently moving the data for neural networks by implementing these neural networks in-memory [136]. For read mapping, PIM accelerators [54, 66, 70, 79, 82–84, 88, 146–158] accelerate several computationally-costly steps in read mapping (e.g., chaining and sequence alignment). Many of these works provide large-scale in-memory parallelism while reducing the data movement overheads of mapping reads to a reference genome.

These works suffer from two main issues. First, none of these PIM accelerators are designed to accelerate *both* basecalling and read mapping, which requires storing and moving a large amount of data (long reads) after the basecalling step instead of streaming these reads directly to the read mapping step in a pipelined manner that maximizes concurrency. Second, even though read quality control filters out the low-quality reads, these reads have already been processed by the expensive basecalling step (because basecalling happens earlier in a separate PIM accelerator). Compared to these existing approaches, GenPIP effectively and efficiently orchestrates basecalling and read mapping steps to 1) reduce the data movement overhead between the basecalling and read mapping steps and enable fine-grained overlapping between these two steps, and 2) eliminate the redundant computations in both basecalling and read mapping by quickly rejecting low-quality reads and unmapped reads.

Non-PIM Acceleration of Basecalling. SquiggleFilter [159] accelerates the basecalling step by filtering raw electrical signals before basecalling based on their similarity to a *certain* genome (e.g., a viral DNA). SquiggleFilter [159] targets a metagenomics use case, where there are large numbers of reads from different species. Many prior works accelerate the basecalling step by implementing the basecaller using GPUs (e.g., [51, 52, 160–167]) and FPGAs (e.g., [168–171]).

Non-PIM Acceleration of Read Mapping. There are several works that focus on accelerating different steps of read mapping, such as pre-alignment filtering (e.g., [62, 67, 72–74, 172–174]), chaining (e.g., [175, 176]), and sequence alignment (e.g., [57, 61, 65, 68, 71, 177–216]). GenPIP is different from these works as none of the prior read mapping accelerators tightly integrate the basecalling and read mapping steps to reduce the data movement and useless computations in the genome analysis pipeline.

Basecalling-free Genome Analysis. Several works avoid the computationally-costly basecalling step from the genome analysis pipeline by directly mapping raw electrical signals to genomic sequences such as reference genomes (e.g., [159, 217–224]. These works change the representation of the genomic sequence from the base (i.e., DNA character) space into the electrical signal space and perform analysis fully in the signal space. As such, they can accelerate the genome analysis pipeline by reducing or eliminating the need for basecalling for certain use cases (e.g., targeted sequencing [217]). GenPIP is different from these works as it uses the basecalling step and performs genome analysis in the base space, which can be integrated into *any* genome analysis use case.

8. Conclusion

Nanopore genome analysis pipeline has two main computationally-costly processing steps, basecalling and read mapping, which are executed separately on different machines in conventional systems. We observe that the separate execution of these two critical steps results in (1) significant data movement and (2) useless computations on the low-quality and unmapped reads, slowing down the genome analysis pipeline and wasting significant energy. To effectively overcome these two limitations, we propose GenPIP, an in-memory genome analysis accelerator that tightly integrates basecalling and read mapping. GenPIP uses two key mechanisms: (1) a chunk-based pipeline, CP, to collaboratively execute the major genome analysis steps in parallel, and (2) a new early-rejection technique, ER, to timely terminate the analysis on low-quality and unmapped reads. Our experimental results show that GenPIP achieves significant performance improvement and energy savings compared to prior genome analysis accelerators. We envision GenPIP to be best implemented inside the sequencing machine to maximize the efficiency of genome sequence analysis. We hope that our work inspires further rethinking of the construction and acceleration of the genome analysis pipeline in a holistic manner.

Acknowledgments

We thank the anonymous reviewers of MICRO 2022 and ISCA 2022 for their valuable feedback. We thank the SAFARI Research Group members for valuable feedback and the stimulating intellectual environment they provide. We acknowledge the generous gifts provided by our industrial partners, including Google, Huawei, Intel, Microsoft, and VMware. This research was partially supported by the Semiconductor Research Corporation and the ETH Future Computing Laboratory.

References

[1] Shanika L Amarasinghe, Shian Su, Xueyi Dong, Luke Zappia, Matthew E Ritchie, and Quentin Gouil. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, 2020.

[2] Valentine Murigneux, Subash Kumar Rai, Agnelo Furtado, Timothy JC Bruxner, Wei Tian, Ivon Harliwong, Hanmin Wei, Bicheng Yang, Qianyu Ye, Ellis Anderson, et al. Comparison of long-read methods for sequencing and assembly of a plant genome. *GigaScience*, 2020.

[3] Ou Wang, Robert Chin, Xiaofang Cheng, Michelle Ka Yan Wu, Qing Mao, Jingbo Tang, Yuhui Sun, Ellis Anderson, Han K Lam, Dan Chen, et al. Efficient and unique cobarcoding of second-generation sequencing reads from long DNA molecules enabling cost-effective and accurate sequencing, haplotyping, and de Novo assembly. *Genome Research*, 2019.

[4] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Briefings in Bioinformatics*, 2018.

[5] Michelle M Clark, Amber Hildreth, Sergey Batalov, Yan Ding, Shimul Chowdhury, Kelly Watkins, Katarzyna Ellsworth, Brandon Camp, Cyrielle I Kint, Calum Yacoubian, et al. Diagnosis of genetic diseases in seriously ill children by rapid whole-genome sequencing and automated phenotyping and interpretation. *Science Translational Medicine*, 2019.

[6] Lauge Farnaes, Amber Hildreth, Nathaly M Sweeney, Michelle M Clark, Shimul Chowdhury, Shareef Nahas, Julie A Cakici, Wendy Benson, Robert H Kaplan, Richard Kronick, et al. Rapid whole-genome sequencing decreases infant morbidity and cost of hospitalization. *NPJ Genomic Medicine*, 2018.

[7] Euan A Ashley. Towards precision medicine. *Nature Reviews Genetics*, 2016.

[8] Mauricio Flores, Gustavo Glusman, Kristin Brogaard, Nathan D Price, and Leroy Hood. P4 Medicine: How systems medicine will transform the healthcare sector and society. *Personalized Medicine*, 2013.

[9] Lynda Chin, Jannik N Andersen, and P Andrew Futreal. Cancer genomics: from discovery science to personalized medicine. *Nature Medicine*, 2011.

[10] 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. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature Genetics*, 2009.

[11] Geoffrey S Ginsburg and Huntington F Willard. Genomic and personalized medicine: foundations and applications. *Translational Research*, 2009.

[12] Maria Jesus Alvarez-Cubero, Maria Saiz, Belén Martínez-García, Sara M Sayalero, Carmen Entrala, Jose Antonio Lorente, and Luis Javier Martinez-Gonzalez. Next generation sequencing: An application in forensic sciences? *Annals of Human Biology*, 2017.

[13] Claus Børsting and Niels Morling. Next generation sequencing and its applications in forensic genetics. *FSI Genetics*, 2015.

[14] Ana Prohaska, Fernando Racimo, Andrew J Schork, Martin Sikora, Aaron J Stern, Melissa Ilardo, Morten Erik Allentoft, Lasse Folkersen, Alfonso Buil, J Víctor Moreno-Mayar, Thorfinn Korneliussen, Daniel Geschwind, Andrés Ingason, Thomas Werge, Rasmus Nielsen, and Eske Willerslev. Human disease variation in the light of population genomics. *Cell*, 2019.

[15] Hans Ellegren and Nicolas Galtier. Determinants of genetic diversity. *Nature Reviews Genetics*, 2016.

[16] Sean Hoban, Joanna L Kelley, Katie E Lotterhos, Michael F Antolin, Gideon Bradburd, David B Lowry, Mary L Poss, Laura K Reed, Andrew Storfer, and Michael C Whitlock. Finding the genomic basis of local adaptation: pitfalls, practical solutions, and future directions. *The American Naturalist*, 2016.

[17] Hans Ellegren. Genome sequencing and population genomics in non-model organisms. *Trends in Ecology & Evolution*, 2014.

[18] J Romiguier, Philippe Gayral, Marion Ballenghien, Arnaud Bernard, Vincent Cahais, A Chenuil, Ylenia Chiari, R Dernat, L Duret, Nicolas Faivre, et al. Comparative population genomics in animals uncovers the determinants of genetic diversity. *Nature*, 2014.

[19] 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. Great ape genetic diversity and population history. *Nature*, 2013.

[20] 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. 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.

[21] Ramesh Yelagandula, Aleksandr Bykov, Alexander Vogt, Robert Heinen, Ezgi Özkan, Marcus Martin Strobl, Juliane Christina Baar, Kristina Uzunova, Bence Hajdusits, Darja Kordic, et al. Multiplexed detection of SARS-CoV-2 and other respiratory infections in high throughput by SARSeq. *Nature Communications*, 2021.

[22] Fang Wang, Shujia Huang, Rongsui Gao, Yuwen Zhou, Changxiang Lai, Zhichao Li, Wenjie Xian, Xiaobo Qian, Zhiyu Li, Yushan Huang, et al. Initial whole-genome sequencing and analysis of the host genetic contribution to Covid-19 severity and susceptibility. *Cell Discovery*, 2020.

[23] Vlad Nikolayevskyy, Katharina Kranzer, Stefan Niemann, and Francis Drobniewski. Whole genome sequencing of mycobacterium tuberculosis for detection of recent transmission and rtacing outbreaks: A systematic review. *Tuberculosis*, 2016.

[24] Shaofu Qiu, Peng Li, Hongbo Liu, Yong Wang, Nan Liu, Chengyi Li, Shenlong Li, Ming Li, Zhengjie Jiang, Huandong Sun, et al. Whole-genome sequencing for tracing the transmission link between two ard outbreaks caused by a novel hadv serotype 7 variant, China. *Scientific Reports*, 2015.

[25] Carol A Gilchrist, Stephen D Turner, Margaret F Riley, William A Petri, and Erik L Hewlett. Whole-genome sequencing in outbreak analysis. *Clinical Microbiology Reviews*, 2015.

[26] Fernando Meyer, Adrian Fritz, Zhi-Luo Deng, David Koslicki, Till Robin Lesker, Alexey Gurevich, Gary Robertson, Mohammed Alser, Dmitry Antipov, Francesco Beghini, et al. Critical assessment of metagenome interpretation: the second round of challenges. *Nature Methods*, 2022.

[27] Stephen K Gire, Augustine Goba, Kristian G Andersen, Rachel SG Sealfon, Daniel J Park, Lansana Kanneh, Simbirie Jalloh, Mambu Momoh, Mohamed Fullah, Gytis Dudas, et al. Genomic surveillance elucidates ebola virus origin and transmission during the 2014 outbreak. *Science*, 2014.

[28] Mohammed Alser, Jeremie S Kim, Nour Almadhoun Alserr, Stefan W Tell, and Onur Mutlu. COVIDHunter: COVID-19 pandemic wave prediction and mitigation via seasonality aware modeling. *Frontiers in public health*, 2022.

[29] Vien Thi Minh Le and Binh An Diep. Selected insights from application of whole genome sequencing for outbreak investigations. *Current Opinion in Critical Care*, 2013.

[30] Nathan LaPierre, Serghei Mangul, Mohammed Alser, Igor Mandric, Nicholas C Wu, David Koslicki, and Eleazar Eskin. MiCoP: microbial community profiling method for detecting viral and fungal organisms in metagenomic samples. *BMC genomics*, 2019.

[31] Yunhao Wang, Yue Zhao, Audrey Bollas, Yuru Wang, and Kin Fai Au. Nanopore sequencing technology, bioinformatics and applications. *Nature Biotechnology*, 2021.

[32] Bo Segerman. The most frequently used sequencing technologies and assembly methods in different time segments of the bacterial surveillance and refseq genome databases. *Frontiers in Cellular and Infection Microbiology*, 2020.

[33] Miten Jain, Hugh E Olsen, Benedict Paten, and Mark Akeson. The Oxford Nanopore Minion: Delivery of nanopore sequencing to the genomics community. *Genome Biology*, 2016.

[34] Wouter De Coster, Matthias H Weissensteiner, and Fritz J Sedlazeck. Towards population-scale long-read sequencing. *Nature Reviews Genetics*, 2021.

[35] Kishwar Shafin, Trevor Pesout, Ryan Lorig-Roach, Marina Haukness, Hugh E Olsen, Colleen Bosworth, Joel Armstrong, Kristof Tigyi, Nicholas Maurer, Sergey Koren, et al. Nanopore sequencing and the shasta toolkit enable efficient de Novo assembly of eleven human genomes. *Nature Biotechnology*, 2020.

[36] Glennis A Logsdon, Mitchell R Vollger, and Evan E Eichler. Long-read human genome sequencing and its applications. *Nature Reviews Genetics*, 2020.

[37] Alexander Payne, Nadine Holmes, Vardhman Rakyan, and Matthew Loose. BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files. *Bioinformatics*, 2019.

[38] Erwin L van Dijk, Yan Jaszczyszyn, Delphine Naquin, and Claude Thermes. The third revolution in sequencing technology. *Trends in Genetics*, 2018.

[39] Simon Ardui, Adam Ameur, Joris R Vermeesch, and Matthew S Hestand. Single molecule real-time SMRT sequencing comes of age: applications and utilities for medical diagnostics. *Nucleic Acids Research*, 2018.

[40] Wouter De Coster, Svenn D'hert, Darrin T Schultz, Marc Cruts, and Christine Van Broeckhoven. NanoPack: visualizing and processing long-read sequencing data. *Bioinformatics*, 2018.

[41] 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, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 2018.

[42] Franka J Rang, Wigard P Kloosterman, and Jeroen de Ridder. From squiggle to basepair: Computational approaches for improving nanopore sequencing read accuracy. *Genome Biology*, 2018.

[43] Caroline Belser, Benjamin Istace, Erwan Denis, Marion Dubarry, Franc-Christophe Baurens, Cyril Falentin, Mathieu Genete, Wahiba Berrabah, Anne-Marie Chèvre, Régine Delourme, et al. Chromosome-scale assemblies of plant genomes using nanopore long reads and optical maps. *Nature Plants*, 2018.

[44] Martin O Pollard, Deepti Gurdasani, Alexander J Mentzer, Tarryn Porter, and Manjinder S Sandhu. Long Reads: Their purpose and place. *Human Molecular Genetics*, 2018.

[45] Mehdi Kchouk, Jean-Francois Gibrat, and Mourad Elloumi. Generations of sequencing technologies: From first to next generation. *Biology and Medicine*, 2017.

[46] Jason L Weirather, Mariateresa de Cesare, Yunhao Wang, Paolo Piazza, Vittorio Sebastiano, Xiu-Jie Wang, David Buck, and Kin Fai Au. Comprehensive comparison of Pacific Biosciences and Oxford nanopore technologies and their applications to transcriptome analysis. *F1000Research*, 2017.

[47] Miten Jain, John R Tyson, Matthew Loose, Camilla LC Ip, David A Eccles, Justin O'Grady, Sunir Malla, Richard M Leggett, Ola Wallerman, Hans J Jansen, et al. Minion analysis and reference consortium: Phase 2 data release and analysis of R9.0 chemistry. *F1000Research*, 2017.

[48] Francesca Giordano, Louise Aigrain, Michael A Quail, Paul Coupland, James K Bonfield, Robert M Davies, German Tischler, David K Jackson, Thomas M Keane, Jing Li, et al. De novo yeast genome assemblies from MinION, PacBio and MiSeq platforms. *Scientific Reports*, 2017.

[49] James Clarke, Hai-Chen Wu, Lakmal Jayasinghe, Alpesh Patel, Stuart Reid, and Hagan Bayley. Continuous base identification for single-molecule nanopore DNA sequencing. *Nature Nanotechnology*, 2009.

[50] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. From molecules to genomic variations: accelerating genome analysis via intelligent algorithms and architectures. *CSBJ*, 2022.

[51] A Pytorch basecaller for Oxford nanopore reads. `https://github.com/nanoporetech/bonito`.

[52] Oxford Nanopore Technology. Guppy. `https://denbi-nanopore-training-course.readthedocs.io/en/latest/basecalling/basecalling.html`.

[53] Minion. `https://nanoporetech.com/products/minion`.

[54] 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. GenStore: A high-performance in-storage processing system for genome sequence analysis. In *ASPLOS*, 2022.

[55] Can Firtina, Jisung Park, Jeremie S Kim, Mohammed Alser, Damla Senol Cali, Taha Shahroodi, Nika Mansouri Ghiasi, Gagandeep Singh, Konstantinos Kanellopoulos, Can Alkan, et al. BLEND: A fast, memory-efficient, and accurate mechanism to find fuzzy seed matches. *arXiv*, 2021.

[56] Mohammed Alser, Zulal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Accelerating genome analysis: A primer on an ongoing journey. *IEEE Micro*, 2020.

[57] Damla Senol Cali, Gupreet Kalsi, Zulal Bingöl, Lavanya Subramanian, Can Firtina, Jeremie Kim, Rachata Ausavarungnirun, Mohammed Alser, Anant Nori, Juan Luna, et al. GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis. In *MICRO*, 2020.

[58] Mohammed Alser, Jeremy Rotman, Dhrithi Deshpande, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, Harry Taegyun Yang, Victor Xue, Sergey Knyazev, Benjamin D. Singer, Brunilda Balliu, David Koslicki, Pavel Skums, Alex Zelikovsky, Can Alkan, Onur Mutlu, and Serghei Mangul. Technology dictates algorithms:

recent developments in read alignment. *Genome Biology*, 2021.
[59] Jeremie S Kim, Can Firtina, Meryem Banu Cavlak, Damla Senol Cali, Can Alkan, and Onur Mutlu. FastRemap: A tool for quickly remapping reads between genome assemblies. *Bioinformatics*, 2022.
[60] Can Firtina, Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie Kim, Taha Shahroodi, Meryem Banu Cavlak, Joel Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, and Onur Mutlu. ApHMM: Accelerating profile hidden markov models for fast and energy-efficient genome analysis. *arXiv*, 2022.
[61] Damla Senol Cali, Konstantinos Kanellopoulos, Joël Lindegger, Zülal Bingöl, Gurpreet S. Kalsi, Ziyi Zuo, Can Firtina, Meryem Banu Cavlak, Jeremie Kim, Nika Mansouri Ghiasi, Gagandeep Singh, Juan Gómez-Luna, Nour Almadhoun Alserr, Mohammed Alser, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu. SeGraM: A universal hardware accelerator for genomic sequence-to-graph and sequence-to-sequence mapping. In *ISCA*, 2022.
[62] Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Diamantopoulos Diamantopoulos, Juan Gómez-Luna, Henk Corporaal, and Onur Mutlu. FPGA-based near-memory acceleration of modern data-intensive applications. *IEEE Micro*, 2021.
[63] Qian Lou, Sarath Chandra Janga, and Lei Jiang. Helix: Algorithm/architecture co-design for accelerating nanopore genome base-calling. In *PACT*, 2020.
[64] Can Firtina, Jeremie S Kim, Mohammed Alser, Damla Senol Cali, A Ercument Cicek, Can Alkan, and Onur Mutlu. Apollo: A sequencing-technology-independent, scalable and accurate assembly polishing algorithm. *Bioinformatics*, 2020.
[65] Sneha D. Goenka, Yatish Turakhia, Benedict Paten, and Mark Horowitz. SegAlign: A scalable GPU-based whole genome aligner. In *SC*, 2020.
[66] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. PIM-Aligner: A processing-in-MRAM platform for biological sequence alignment. In *DATE*, 2020.
[67] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. SneakySnake: A fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. *Bioinformatics*, 2020.
[68] Anirban Nag, C. N. Ramachandra, Rajeev Balasubramonian, Ryan Stutsman, Edouard Giacomin, Hari Kambalasubramanyam, and Pierre-Emmanuel Gaillardon. GenCache: Leveraging in-Cache operators for efficient sequence alignment. In *MICRO*, 2019.
[69] Jeremie S Kim, Can Firtina, Meryem Banu Cavlak, Damla Senol Cali, Mohammed Alser, Nastaran Hajinazar, Can Alkan, and Onur Mutlu. AirLift: A fast and comprehensive technique for remapping alignments between reference genomes. *arXiv*, 2019.
[70] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC Genomics*, 2018.
[71] Yatish Turakhia, Gill Bejerano, and William J. Dally. Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly. In *ASPLOS*, 2018.
[72] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan. GateKeeper: A new hardware architecture for accelerating pre-alignment in DNA short read mapping. *Bioinformatics*, 2017.
[73] Hongyi Xin, John Greth, John Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. Shifted Hamming Distance: A fast and accurate simd-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics*, 2015.
[74] Hongyi Xin, Donghyuk Lee, Farhad Hormozdiari, Samihan Yedkar, Onur Mutlu, and Can Alkan. Accelerating read mapping with fasthash. *BMC Genomics*, 2013.
[75] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *JMB*, 1981.
[76] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *JMB*, 1970.
[77] Qian Lou and Lei Jiang. Brawl: A spintronics-based portable basecalling-in-memory architecture for nanopore genome sequencing. *CAL*, 2018.
[78] ZhongPan Wu, Karim Hammad, Robinson Mittmann, Sebastian Magierowski, Ebrahim Ghafar-Zadeh, and Xiaoyong Zhong. FPGA-based DNA basecalling hardware acceleration. In *MWSCAS*, 2018.
[79] S Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris H Kim, and Ulya R Karpuzcu. GeNVoM: Read mapping near non-volatile memory. *TCBB*, 2021.
[80] Chirag Jain, Sanchit Misra, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru. Accelerating sequence alignment to graphs. In *IPDPS*, 2019.
[81] Zonghao Feng, Shuang Qiu, Lipeng Wang, and Qiong Luo. Accelerating long read alignment on three processors. In *ICPP*, 2019.
[82] Saransh Gupta, Mohsen Imani, Behnam Khaleghi, Venkatesh Kumar, and Tajana Rosing. RAPID: A reRAM processing in-memory architecture for DNA sequence alignment. In *ISLPED*, 2019.
[83] Farzaneh Zokaee, Hamid R Zarandi, and Lei Jiang. AligneR: A process-in-Memory architecture for short read alignment in ReRAMs. *CAL*, 2018.
[84] Safaa Diab, Amir Nassereldine, Mohammed Alser, Juan Gómez-Luna, Onur Mutlu, and Izzat El Hajj. A framework for high-throughput sequence alignment using real processing-in-memory systems. *arXiv*, 2022.
[85] Rory Bowden, Robert W Davies, Andreas Heger, Alistair T Pagnamenta, Mariateresa de Cesare, Laura E Oikkonen, Duncan Parkes, Colin Freeman, Fatima Dhalla, Smita Y Patel, et al. Sequencing of human genomes with nanopore technology. *Nature Communications*, 2019.
[86] Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul. Metalign: Efficient alignment-based metagenomic profiling via containment min hash. *Genome Biology*, 2020.
[87] Nicholas J. Loman. Nanopore R9 rapid run data release, 2016. `http://lab.loman.net/2016/07/30/nanopore-r9-data-release/`.
[88] Fan Chen, Linghao Song, Yiran Chen, et al. PARC: A processing-in-CAM architecture for genomic long read pairwise alignment using ReRAM. In *ASP-DAC*, 2020.
[89] Oxford Nanopore Technology (ONT). Flappie. `https://github.com/nanoporetech/flappie`.
[90] Oxford Nanopore Technology. Scrappie. `https://github.com/nanoporetech/scrappie`.
[91] Haotian Teng, Minh Duc Cao, Michael B Hall, Tania Duarte, Sheng Wang, and Lachlan J M Coin. Chiron: Translating nanopore raw signal directly into nucleotide sequence using deep learning. *GigaScience*, 2018.
[92] Heng Li. Minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics*, 2018.
[93] Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 2004.
[94] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. In *SIGMOD*, 2003.
[95] Mohammed Alser, Jeremy Rotman, Dhrithi Deshpande, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, Harry Taegyun Yang, Victor Xue, Sergey Knyazev, Benjamin D Singer, et al. Technology dictates algorithms: recent developments in read alignment. *Genome Biology*, 2021.
[96] Sean R Eddy. What is dynamic programming? *Nature Biotechnology*, 2004.
[97] Yoshinori Fukasawa, Luca Ermini, Hai Wang, Karen Carty, and Min-Sin Cheung. LongQC: A quality control tool for third generation sequencing long read data. *G3*, 2020.
[98] Adrien Leger and Tommaso Leonardi. pycoQC, interactive quality control for Oxford nanopore sequencing. *JOSS*, 2019.
[99] Minion Mk1b IT requirements. `https://community.nanoporetech.com/requirements_documents/minion-it-reqs.pdf`.
[100] Stephen F Kingsmore, Audrey Henderson, Mallory J Owen, Michelle M Clark, Christian Hansen, David Dimmock, Christina D Chambers, Laura L Jeliffe-Pawlowski, and Charlotte Hobbs. Measurement of genetic diseases as a cause of mortality in infants receiving whole genome sequencing. *NPJ Genomic Medicine*, 2020.
[101] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, et al. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *ASPLOS*, 2019.
[102] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Sapan Agarwal, Matthew Marinella, Martin Foltin, John Paul Strachan, Dejan Milojicic, Wen-Mei Hwu, and Kaushik Roy. PANTHER: A programmable architecture for neural network training harnessing energy-efficient ReRAM. *IEEE TC*, 2020.
[103] Geng Yuan, Payman Behnam, Zhengang Li, Ali Shafiee, Sheng Lin, Xiaolong Ma, Hang Liu, Xuehai Qian, Mahdi Nazm Bojnordi, Yanzhi Wang, et al. Forms: Fine-grained polarized ReRAM-based in-situ computation for mixed-signal DNN accelerator. *arXiv*, 2021.
[104] Sitao Huang, Aayush Ankit, Plinio Silveira, Rodrigo Antunes, Sai Rahul Chalamalasetti, Izzat El Hajj, Dong Eun Kim, Glaucimar Aguiar, Pedro Bruel, Sergey Serebryakov, et al. Mixed precision quantization for ReRAM-based DNN inference accelerators. In *ASP-DAC*, 2021.
[105] Yi Cai, Yujun Lin, Lixue Xia, Xiaoming Chen, Song Han, Yu Wang, and Huazhong Yang. Long Live Time: Improving lifetime for training-in-memory engines by structured gradient sparsification. In *DAC*, 2018.
[106] Linghao Song, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. GraphR: Accelerating graph processing using ReRAM. In *HPCA*, 2018.
[107] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. FloatPIM: In-memory acceleration of deep neural network training with high precision. In *ISCA*, 2019.
[108] Sparsh Mittal. A survey of ReRAM-based architectures for processing-in-memory and neural networks. *MAKE*, 2019.
[109] Saransh Gupta, Mohsen Imani, Harveen Kaur, and Tajana Simunic Rosing. NNPIM: A processing in-memory architecture for neural network acceleration. *TC*, 2019.
[110] Haiyu Mao, Mingcong Song, Tao Li, Yuting Dai, and Jiwu Shu. LerGAN: A zero-free, low data movement and PIM-based GAN architecture. In *MICRO*, 2018.
[111] Haiyu Mao, Jiwu Shu, Mingcong Song, and Tao Li. LrGAN: A compact and energy efficient PIM-Based architecture for GAN training. *TC*, 2020.
[112] Robert Karam, Ruchir Puri, Swaroop Ghosh, and Swarup Bhunia. Emerging trends in design and applications of memory-based computing and content-addressable memories. *Proceedings of the IEEE*, 2015.
[113] Mohsen Imani, Shruti Patil, and Tajana S Rosing. MASC: Ultra-low energy multiple-access single-charge TCAM for approximate computing. In *DATE*, 2016.
[114] Shuangchen Li, Liu Liu, Peng Gu, Cong Xu, and Yuan Xie. NVSim-CAM: A circuit-level simulator for emerging nonvolatile memory based content-addressable memory. In *ICCAD*, 2016.
[115] Erya Deng, Lorena Anghel, Guillaume Prenat, and Weisheng Zhao. Multi-context non-volatile content addressable memory using magnetic tunnel junctions. In *NANOARCH*, 2016.
[116] Meng-Fan Chang, Chien-Chen Lin, Albert Lee, Yen-Ning Chiang, Chia-Chen Kuo, Geng-Hau Yang, Hsiang-Jen Tsai, Tien-Fu Chen, and Shyh-Shyuan Sheu. A 3T1R nonvolatile TCAM using MLC ReRAM for frequent-Off instant-on filters in IoT

and big-data processing. *JSSC*, 2017.
[117] Hasan Erdem Yantir, Ahmed M Eltawil, and Fadi J Kurdahi. Approximate memristive in-memory computing. *TECS*, 2017.
[118] Xunzhao Yin, Michael Niemier, and X Sharon Hu. Design and benchmarking of ferroelectric FET-based TCAM. In *DATE*, 2017.
[119] Mohsen Imani, Abbas Rahimi, Pietro Mercati, and Tajana Simunic Rosing. Multistage tunable approximate search in resistive associative memory. *TMSCS*, 2017.
[120] Xunzhao Yin, Kai Ni, Dayane Reis, Suman Datta, Michael Niemier, and Xiaobo Sharon Hu. An ultra-dense 2FeFET TCAM design based on a multi-domain FeFET model. *TCAS*, 2018.
[121] Chengzhi Wang, Deming Zhang, Lang Zeng, Erya Deng, Jie Chen, and Weisheng Zhao. A novel MTJ-based non-volatile ternary content-addressable memory for high-speed, low-power, and high-reliable search operation. *TCAS*, 2018.
[122] Krishna Prasad Gnawali, Seyed Nima Mozaffari, and Spyros Tragoudas. Low power spintronic ternary content addressable memory. *TNANO*, 2018.
[123] Payman Behnam, Arjun Pal Chowdhury, and Mahdi Nazm Bojnordi. R-Cache: A highly set-associative in-package cache using memristive arrays. In *ICCD*, 2018.
[124] Ava J Tan, Korok Chatterjee, Jiuren Zhou, Daewoong Kwon, Yu-Hung Liao, Suraj Cheema, Chenming Hu, and Sayeef Salahuddin. Experimental demonstration of a ferroelectric HfO2-based content addressable memory cell. *EDL*, 2019.
[125] Ren Arakawa, Naoya Onizawa, Jean-Philippe Diguet, and Takahiro Hanyu. Multicontext TCAM based selective computing architecture for a low-power NN. In *ICECS*, 2019.
[126] Lei Zhao, Quan Deng, Youtao Zhang, and J. Yang. RFAcc: a 3D ReRAM associative array based random forest accelerator. *SC*, 2019.
[127] Yasmin Halawani, Baker Mohammad, Muath Abu Lebdeh, Mahmoud Al-Qutayri, and Said F Al-Sarawi. ReRAM-based in-Memory computing for search engine and neural network applications. *JETCAS*, 2019.
[128] Can Li, Catherine E Graves, Xia Sheng, Darrin Miller, Martin Foltin, Giacomo Pedretti, and John Paul Strachan. Analog content-addressable memories with memristors. *Nature Communications*, 2020.
[129] Catherine E Graves, Can Li, Xia Sheng, Darrin Miller, Jim Ignowski, Lennie Kiyama, and John Paul Strachan. In-memory computing with memristor content addressable memories for pattern matching. *Advanced Materials*, 2020.
[130] Xunzhao Yin, Chao Li, Qingrong Huang, Li Zhang, Michael Niemier, Xiaobo Sharon Hu, Cheng Zhuo, and Kai Ni. FeCAM: A universal compact digital and analog content addressable memory using ferroelectric. *TED*, 2020.
[131] Nuo Xiu, Yiming Chen, Guodong Yin, Xiaoyang Ma, Huazhong Yang, Sumitha George, and Xueqing Li. Capacitive content-addressable memory: A highly reliable and scalable approach to energy-efficient parallel pattern matching applications. In *GLSVLSI*, 2021.
[132] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. *CAN*, 2016.
[133] NVIDIA RTX GeForce 2080 Ti 11GB, 2018. `https://www.pny.com.tw/jp/upload/download_files/jp_download_list_19b22_wr2i3c5qid.pdf`.
[134] Introduction to Intel architecture, 2016. `https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-introduction-basics-paper.pdf`.
[135] Sparsh Mittal, Jeffrey S Vetter, and Dong Li. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. *TPDS*, 2014.
[136] A. Shafiee, Anirban Nag, N. Muralimanohar, Rajeev Balasubramonian, J. Strachan, Miao Hu, R. Williams, and Vivek Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, 2016.
[137] Pran Kurup and Taher Abbasi. *Logic synthesis using Synopsys®*. Springer Science & Business Media, 2012.
[138] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. NVSIM: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *TCAD*, 2012.
[139] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *MICRO*, 2007.
[140] European Nucleotide Archive (ENA). `https://www.ebi.ac.uk/ena/browser/home`.
[141] National cCenter for Biotechnology Information (NCBI). `https://www.ncbi.nlm.nih.gov/`.
[142] Damla Senol, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Nanopore sequencing technology and tools: cComputational analysis of the current state, bottlenecks and future directions. In *PSB*, 2017.
[143] Mantas Sereika, Rasmus Hansen Kirkegaard, Søren Michael Karst, Thomas Yssing Michaelsen, Emil Aarre Sørensen, Rasmus Dam Wollenberg, and Mads Albertsen. Oxford nanopore R10.4 long-read sequencing enables the generation of near-finished bacterial genomes from pure cultures and metagenomes without short-read or reference polishing. *Nature Methods*, 2022.
[144] Nanopore R9 rapid run data release. `http://lab.loman.net/2016/07/30/nanopore-r9-data-release/`.
[145] European Nucleotide Archive - Sequencing of human genomes with nanopore technology. `https://www.ebi.ac.uk/ena/browser/view/PRJEB30620`.
[146] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. A resistive CAM processing-in-storage architecture for DNA sequence alignment. *IEEE Micro*, 2017.
[147] Roman Kaplan, Leonid Yavits, and Ran Ginosar. RASSA: Resistive pre-alignment accelerator for approximate DNA long read mapping. *IEEE Micro*, 2018.
[148] Farzaneh Zokaee, Mingzhe Zhang, and Lei Jiang. FindeR: Accelerating FM-index-based exact pattern matching in genomic sequences through ReRAM technology. In *PACT*, 2019.
[149] Shaahin Angizi, Wei Zhang, and Deliang Fan. Exploring DNA alignment-in-memory leveraging emerging SOT-MRAM. In *GLSVLSI*, 2020.
[150] Roman Kaplan, Leonid Yavits, and Ran Ginosasr. BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data. In *SYSTOR*, 2020.
[151] Ann Franchesca Laguna, Hasindu Gamaarachchi, Xunzhao Yin, Michael Niemier, Sri Parameswaran, and X Sharon Hu. Seed-and-Vote based in-memory accelerator for DNA read mapping. In *ICCAD*, 2020.
[152] Marcel Khalifa, Rotem Ben-Hur, Ronny Ronen, Orian Leitersdorf, Leonid Yavits, and Shahar Kvatinsky. FiltPIM: In-memory filter for DNA sequencing. In *ICECS*, 2021.
[153] Zamshed I. Chowdhury, Masoud Zabihi, S. Karen Khatamifard, Zhengyang Zhao, Salonik Resch, Meisam Razaviyayn, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. A DNA read alignment accelerator based on computational RAM. *JXCDC*, 2020.
[154] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. RADAR: A 3D-ReRAM based DNA alignment accelerator architecture. In *DAC*, 2018.
[155] Safaa Diab, Amir Nassereldine, Mohammed Alser, Juan Gómez Luna, Onur Mutlu, and Izzat El Hajj. High-throughput pairwise alignment with the wavefront algorithm using processing-in-memory. *arXiv*, 2022.
[156] Taha Shahroodi, Mahdi Zahedi, Can Firtina, Mohammed Alser, Stephan Wong, Onur Mutlu, and Said Hamdioui. Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory. *IEEE Access*, 2022.
[157] Xue-Qi Li, Guang-Ming Tan, and Ning-Hui Sun. PIM-Align: A processing-in-memory architecture for FM-Index search algorithm. *JCST*, 2021.
[158] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. Aligns: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM. In *DAC*, 2019.
[159] Tim Dunn, Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, David Blaauw, Reetuparna Das, and Satish Narayanasamy. SquiggleFilter: An accelerator for portable virus detection. In *MICRO*, 2021.
[160] Zhimeng Xu, Yuting Mai, Denghui Liu, Wenjun He, Xinyuan Lin, Chi Xu, Lei Zhang, Xin Meng, Joseph Mafofo, Walid Abbas Zaher, et al. Fast-bonito: A faster deep learning based basecaller for nanopore sequencing. *Artificial Intelligence in the Life Sciences*, 2021.
[161] Peter Perešíni, Vladimír Boža, Broňa Brejová, and Tomáš Vinař. Nanopore base calling on the edge. *Bioinformatics*, 2021.
[162] Xuan Lv, Zhiguang Chen, Yutong Lu, and Yuedong Yang. An end-to-end Oxford nanopore basecaller using convolution-augmented transformer. In *BIBM*, 2020.
[163] Jingwen Zeng, Hongmin Cai, Hong Peng, Haiyan Wang, Yue Zhang, and Tatsuya Akutsu. Causalcall: Nanopore basecalling using a temporal convolutional network. *Frontiers in Genetics*, 2020.
[164] Yang-Ming Yeh and Yi-Chang Lu. MSRCall: A multi-scale deep neural network to basecall Oxford nanopore sequences. *Bioinformatics*, 2022.
[165] Neng Huang, Fan Nie, Peng Ni, Feng Luo, and Jianxin Wang. SACall: A neural network basecaller for Oxford nanopore sequencing data based on self-attention mechanism. *TCBB*, 2022.
[166] Hiroki Konishi, Rui Yamaguchi, Kiyoshi Yamaguchi, Yoichi Furukawa, and Seiya Imoto. Halcyon: an accurate basecaller exploiting an encoder-decoder model with monotonic attention. *Bioinformatics*, 2021.
[167] Vladimír Boža, Broňa Brejová, and Tomáš Vinař. DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads. *PLOS One*, 2017.
[168] C N Ramachandra, Anirban Nag, Rajeev Balasubramonion, Gurpreet Kalsi, Kamlesh Pillai, and Sreenivas Subramoney. ONT-X: An FPGA approach to real-time portable genomic analysis. In *FCCM*, 2021.
[169] Karim Hammad, Zhongpan Wu, Ebrahim Ghafar-Zadeh, and Sebastian Magierowski. A scalable hardware accelerator for mobile DNA sequencing. *TVLSI*, 2021.
[170] Zhongpan Wu, Karim Hammad, Abel Beyene, Yunus Dawji, Ebrahim Ghafar-Zadeh, and Sebastian Magierowski. An FPGA implementation of a portable DNA sequencing device based on RISC-V. In *Newcas*, 2022.
[171] Zhongpan Wu, Karim Hammad, Ebrahim Ghafar-Zadeh, and Sebastian Magierowski. FPGA-accelerated 3rd generation DNA sequencing. *TBCS*, 2020.
[172] Mohammed Alser, Hasan Hassan, Akash Kumar, Onur Mutlu, and Can Alkan. Shouji: A fast and efficient pre-alignment filter for sequence alignment. *Bioinformatics*, 2019.
[173] Mohammed Alser, Onur Mutlu, and Can Alkan. MAGNET: Understanding and improving the accuracy of genome pre-Alignment filtering. *arXiv*, 2017.
[174] Zülal Bingöl, Mohammed Alser, Onur Mutlu, Ozcan Ozturk, and Can Alkan. GateKeeper-GPU: Fast and accurate pre-alignment filtering in short read mapping. In *IPDPSW*. IEEE, 2021.
[175] Licheng Guo, Jason Lau, Zhenyuan Ruan, Peng Wei, and Jason Cong. Hardware acceleration of long read pairwise overlapping in genome sequencing: a race between FPGA and GPU. In *FCCM*, 2019.
[176] Harisankar Sadasivan, Milos Maric, Eric Dawson, Vishanth Iyer, Johnny Israeli, and Satish Narayanasamy. Accelerating Minimap2 for accurate long read alignment on GPUs. *bioRxiv*, 2022.
[177] Yupeng Chen, Bertil Schmidt, and Douglas L Maskell. A hybrid short read mapping accelerator. *BMC Bioinformatics*, 2013.
[178] S Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris Kim, and Ulya R Karpuzcu. Read mapping near non-volatile memory. *arXiv*, 2017.
[179] Quim Aguado-Puig, Santiago Marco-Sola, Juan Carlos Moure, David Castells-Rufas,

Lluc Alvarez, Antonio Espinosa, and Miquel Moreto. Accelerating edit-distance sequence alignment on GPU using the wavefront algorithm. *IEEE Access*, 2022.
[180] Quim Aguado-Puig, Santiago Marco-Sola, Juan Carlos Moure, Christos Matzoros, David Castells-Rufas, Antonio Espinosa, and Miquel Moreto. WFA-GPU: Gap-affine pairwise alignment using GPUs. *bioRxiv*, 2022.
[181] Abbas Haghi, Santiago Marco-Sola, Lluc Alvarez, Dionysios Diamantopoulos, Christoph Hagleitner, and Miquel Moreto. An FPGA accelerator of the wavefront algorithm for genomics pairwise alignment. In *FPL*, 2021.
[182] Joël Lindegger, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, and Onur Mutlu. Algorithmic improvement and GPU acceleration of the GenASM algorithm. *arXiv*, 2022.
[183] Joël Lindegger, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, Nika Mansouri Ghiasi, and Onur Mutlu. Scrooge: A fast and memory-frugal genomic sequence aligner for CPUs, GPUs, and ASICs. *arXiv*, 2022.
[184] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. GenAx: A genome sequencing accelerator. In *ISCA*, 2018.
[185] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. Race Logic: A hardware acceleration for dynamic programming algorithms. *CAN*, 2014.
[186] Haoyu Cheng, Yong Zhang, and Yun Xu. Bitmapper2: A GPU-accelerated all-mapper based on the sparse Q-gram index. *TCBB*, 2018.
[187] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths. *Computational Biology and Chemistry*, 2018.
[188] Ernst Joachim Houtgast, VladMihai Sima, Koen Bertels, and Zaid AlArs. An efficient GPU-accelerated implementation of genomic short read mapping with BWA-MEM. *CAN*, 2017.
[189] Alberto Zeni, Giulia Guidi, Marquita Ellis, Nan Ding, Marco D Santambrogio, Steven Hofmeyr, Aydın Buluç, Leonid Oliker, and Katherine Yelick. Logan: High-performance GPU-based X-drop long-read alignment. In *IPDPS*, 2020.
[190] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. GASAL2: A GPU accelerated sequence alignment library for high-throughput NGS data. *BMC Bioinformatics*, 2019.
[191] Takahiro Nishimura, Jacir L Bordim, Yasuaki Ito, and Koji Nakano. Accelerating the Smith-waterman algorithm using bitwise parallel bulk computation technique on GPU. In *IPDPSW*, 2017.
[192] Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo. CUDAlign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters. *TPDS*, 2016.
[193] Yongchao Liu and Bertil Schmidt. GSWABE: Faster GPU-accelerated sequence alignment with optimal alignment retrieval for short DNA sequences. *Concurrency and Computation: Practice and Experience*, 2015.
[194] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. CUDASW++ 3.0: Accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. *BMC Bioinformatics*, 2013.
[195] Yongchao Liu, Douglas L Maskell, and Bertil Schmidt. CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes*, 2009.
[196] Yongchao Liu, Bertil Schmidt, and Douglas L Maskell. CUDASW++ 2.0: Enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC Research Notes*, 2010.
[197] Richard Wilton, Tamas Budavari, Ben Langmead, Sarah J Wheelan, Steven L Salzberg, and Alexander S Szalay. Arioc: High-throughput read alignment with GPU-accelerated exploration of the seed-and-extend search space. *PeerJ*, 2015.
[198] Amit Goyal, Hyuk Jung Kwon, Kichan Lee, Reena Garg, Seon Young Yun, Yoon Hee Kim, Sunghoon Lee, and Min Seob Lee. Ultra-fast next generation human genome sequencing data processing using *DRAGEN*^{TM} Bio-IT processor for precision medicine. *OJGen*, 2017.
[199] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. When Spark Meets FPGAs: A case study for next-generation DNA sequencing acceleration. In *HotCloud*, 2016.
[200] Peng Chen, Chao Wang, Xi Li, and Xuehai Zhou. Accelerating the next generation long read mapping with the FPGA-based system. *TCBB*, 2014.
[201] Yen-Lung Chen, Bo-Yi Chang, Chia-Hsiang Yang, and Tzi-Dar Chiueh. A high-throughput FPGA accelerator for short-read mapping of the whole human genome. *TPDS*, 2021.
[202] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, and Reetuparna Das. SeedEx: A genome sequencing accelerator for optimal alignments in subminimal space. In *MICRO*, 2020.
[203] Subho Sankar Banerjee, Mohamed El-Hadedy, Jong Bin Lim, Zbigniew T Kalbarczyk, Deming Chen, Steven S Lumetta, and Ravishankar K Iyer. ASAP: Accelerated short-read alignment on programmable hardware. *TC*, 2019.
[204] Xia Fei, Zou Dan, Lu Lina, Man Xin, and Zhang Chunlei. FPGASW: Accelerating large-scale Smith–Waterman sequence alignment application with backtracking on FPGA linear systolic array. *Interdisciplinary Sciences: Computational Life Sciences*, 2018.
[205] Hasitha Muthumala Waidyasooriya and Masanori Hariyama. Hardware-acceleration of short-read alignment based on the Burrows-wheeler transform. *TPDS*, 2015.
[206] Yu-Ting Chen, Jason Cong, Jie Lei, and Peng Wei. A novel high-throughput acceleration engine for read alignment. In *FCCM*, 2015.
[207] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matias. SWIFOLD: Smith-Waterman implementation on FPGA with OpenCL for long DNA sequences. *BMC Systems Biology*, 2018.
[208] Luyi Li, Jun Lin, and Zhongfeng Wang. PipeBSW: A two-stage pipeline structure for banded Smith-Waterman algorithm on FPGA. In *ISVLSI*, 2021.
[209] Lisa Wu, David Bruns-Smith, Frank A Nothaft, Qijing Huang, Sagar Karandikar, Johnny Le, Andrew Lin, Howard Mao, Brendan Sweeney, Krste Asanović, et al. FPGA accelerated indel realignment in the cloud. In *HPCA*, 2019.
[210] Yiqing Yan, Nimisha Chaturvedi, and Raja Appuswamy. Accel-Align: a fast sequence mapper and aligner based on the seed–embed–extend method. *BMC Bioinformatics*, 2021.
[211] Md. Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. Efficient architecture-aware acceleration of BWA-MEM for multicore systems. In *IPDPS*, 2019.
[212] Jeff Daily. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics*, 2016.
[213] Saurabh Kalikar, Chirag Jain, Md Vasimuddin, and Sanchit Misra. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nature Computational Science*, 2022.
[214] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 2021.
[215] Jordan M. Eizenga and Benedict Paten. Improving the time and space complexity of the WFA algorithm and generalizing its scoring. *bioRxiv*, 2022.
[216] Santiago Marco-Sola, Jordan M. Eizenga, Andrea Guarracino, Benedict Paten, Erik Garrison, and Miquel Moreto. Optimal gap-affine alignment in O(s) space. *bioRxiv*, 2022.
[217] Sam Kovaka, Yunfan Fan, Bohan Ni, Winston Timp, and Michael C Schatz. Targeted nanopore sequencing by real-time mapping of raw electrical signal with uncalled. *Nature Biotechnology*, 2021.
[218] Haowen Zhang, Haoran Li, Chirag Jain, Haoyu Cheng, Kin Fai Au, Heng Li, and Srinivas Aluru. Real-time mapping of nanopore raw signals. *Bioinformatics*, 2021.
[219] Matthew Loose, Sunir Malla, and Michael Stout. Real-time selective sequencing using nanopore technology. *Nature Methods*, 2016.
[220] Harrison S. Edwards, Raga Krishnakumar, Anupama Sinha, Sara W. Bird, Kamlesh D. Patel, and Michael S. Bartsch. Real-Time selective sequencing with RUBRIC: Read Until with basecall and reference-informed criteria. *Scientific Reports*, 2019.
[221] Alexander Payne, Nadine Holmes, Thomas Clarke, Rory Munro, Bisrat J. Debebe, and Matthew Loose. Readfish enables targeted nanopore sequencing of gigabase-sized genomes. *Nature Biotechnology*, 2021.
[222] Nicola De Maio, Charlotte Manser, Rory Munro, Ewan Birney, Matthew Loose, and Nick Goldman. BOSS-RUNS: a flexible and practical dynamic read sampling framework for nanopore sequencing. *bioRxiv*, 2020.
[223] Artem Danilevsky, Avital Luba Polsky, and Noam Shomron. Adaptive sequencing using nanopores and deep learning of mitochondrial DNA. *Briefings in Bioinformatics*, 2022.
[224] Markus Joppich, Margaryta Olenchuk, Julia M. Mayer, Quirin Emslander, Luisa F. Jimenez-Soto, and Ralf Zimmer. SEQU-INTO: Early detection of impurities, contamination and off-targets (ICOs) in long read/MinION sequencing. *CSBJ*, 2020.