

journal homepage: www.elsevier.com/locate/csbj

Review

From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures



Mohammed Alser*, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, Onur Mutlu*

ETH Zurich, Gloriastrasse 35, 8092 Zürich, Switzerland

ARTICLE INFO

Article history:

Received 12 May 2022
 Received in revised form 8 August 2022
 Accepted 8 August 2022
 Available online 18 August 2022

Keywords:

Genome analysis
 Read mapping
 Hardware acceleration
 Hardware/software co-design

ABSTRACT

We now need more than ever to make genome analysis more intelligent. We need to read, analyze, and interpret our genomes not only quickly, but also accurately and efficiently enough to scale the analysis to population level. There currently exist major computational bottlenecks and inefficiencies throughout the entire genome analysis pipeline, because state-of-the-art genome sequencing technologies are still not able to read a genome in its entirety. We describe the ongoing journey in significantly improving the performance, accuracy, and efficiency of genome analysis using intelligent algorithms and hardware architectures. We explain state-of-the-art algorithmic methods and hardware-based acceleration approaches for each step of the genome analysis pipeline and provide experimental evaluations. Algorithmic approaches exploit the structure of the genome as well as the structure of the underlying hardware. Hardware-based acceleration approaches exploit specialized microarchitectures or various execution paradigms (e.g., processing inside or near memory) along with algorithmic changes, leading to new hardware/software co-designed systems. We conclude with a foreshadowing of future challenges, benefits, and research directions triggered by the development of both very low cost yet highly error prone new sequencing technologies and specialized hardware chips for genomics. We hope that these efforts and the challenges we discuss provide a foundation for future work in making genome analysis *more intelligent*.

© 2022 The Authors. Published by Elsevier B.V. on behalf of Research Network of Computational and Structural Biotechnology. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

Contents

1. Introduction	4580
2. Obtaining genomic sequencing data	4582
2.1. Generating sequencing data	4582
2.2. Downloading real sequencing data	4582
2.3. Simulating sequencing data	4583
3. Types of genomic sequencing data	4583
3.1. Short reads	4584
3.2. Ultra-long reads	4584
3.3. Accurate long reads	4584
3.4. Discussion on types of sequencing reads	4584
4. Genome analysis using different types of sequencing reads	4585
5. Basecalling	4586
5.1. Illumina	4586
5.2. ONT	4586
5.3. PacBio	4587
6. Quality control	4587

* Corresponding authors.
 E-mail addresses: alserm@ethz.ch, omutlu@gmail.com (O. Mutlu).

7.	Read mapping	4587
7.1.	Accelerating indexing and seeding	4588
7.1.1.	Sampling seeds	4588
7.1.2.	Improving data structures for seed lookups	4589
7.1.3.	Reducing data movement during indexing	4589
7.2.	Accelerating pre-alignment filtering	4589
7.2.1.	Pigeonhole principle	4590
7.2.2.	Base counting	4590
7.2.3.	q-gram filtering approach	4590
7.2.4.	Sparse dynamic programming	4590
7.3.	Accelerating sequence alignment	4590
7.3.1.	Accurate alignment accelerators	4591
7.3.2.	Alignment accelerators with limited functionality	4591
8.	Variant calling	4592
8.1.	Processing read mapping data	4592
8.2.	Variant classification	4593
8.3.	Generating variant calls	4593
9.	Discussion and future opportunities	4593
	CRedit authorship contribution statement	4594
	Declaration of Competing Interest	4594
	Acknowledgments	4595
	References	4595

1. Introduction

Sequencing genomic molecules stimulates research and development in biomedicine and other life sciences through its rapidly growing presence in clinical medicine [1–5], outbreak tracing [6–10], and understanding of pathogens and urban microbial communities [11–15]. These developments are driven in part by the successful sequencing of the human genome [16] and in part by the introduction of high-throughput sequencing technologies that have dramatically reduced the cost of DNA sequencing [17]. The bioinformatics community has developed a multitude of software tools to leverage increasingly large and complex sequencing datasets [18–20]. These tools have reshaped the landscape of modern biology and become an essential component of life sciences [21]. The increasing dependence of biomedical scientists on these powerful tools creates a critical need for faster and more efficient computational tools. Our understanding of genomic molecules today is affected by the ability of modern computing technology to quickly and accurately determine an individual’s entire genome. In order to computationally analyze an organism’s genome, the DNA molecule must first be converted to digital data in the form of a string over an alphabet of four letters or *base-pairs* (bp), commonly denoted by A, C, G, and T. The four letters in the DNA alphabet correspond to four chemical bases, adenine, cytosine, guanine, and thymine, respectively, which make up a DNA molecule. After more than 7 decades of continuous attempts (since 1945 [22]), there is still no sequencing technology that can read a DNA molecule in its entirety. As a workaround, sequencing machines generate randomly sampled subsequences of the original genome sequence, called *reads* [23]. The resulting reads lack information about their order and corresponding locations in the complete genome. Software tools, collectively known as *genome analysis tools*, are used to reassemble read fragments back into an entire genome sequence and infer genomic variations that make an individual genetically different from another.

There are five key initial steps in a standard genome sequencing and analysis workflow [18], as we show in Fig. 1. The first step is obtaining the genomic data either through sequencing a DNA molecule, downloading real data from publicly available databases, or computer simulation. Sequencing requires the collection and preparation of the sample in the laboratory or on-site. The second

step, known as *basecalling*, is to process the raw sequencing data as each sequencing technology generates different representations of the sequencing data. The basecalling step must convert the raw sequencing data into standard format of sequences of A, C, G, and T in the DNA alphabet. The third step, known as *quality control*, examines the quality of each sequenced base and decides which bases of a read to trim, as sequencing machines introduce different types and rates of sequencing errors leading to inaccurate end results. The fourth step is a process known as *read mapping*, which matches each read sequence to one or more possible locations within the reference genome (i.e., a representative genome sequence for a particular species), based on the similarity between the read and the reference sequence segment at that location. Unfortunately, the bases in a read may not be identical to the bases in the reference genome at the location that the read actually comes from. These differences may be due to (1) sequencing errors (with a rate in the range of 0.1–20 % of the read length, depending on the sequencing technology), and (2) genetic differences that are specific to the individual organism’s DNA and may not exist in the reference genome. A read mapping step must tolerate such differences during similarity check, which makes read mapping even more challenging. The fifth step, known as *variant calling*, aims to identify the possible genetic differences between the reference genome and the sequenced genome. Genetic differences include small variations [24] that are less than 50 bp, such as single nucleotide polymorphisms (SNPs) and small insertions and deletion (indels). Genetic differences can also be larger than 50 bp variations, known as structural variations [25], which are caused by chromosome-scale changes in a genome. For example, insertion of an about 600,000-base long region has been observed in some chromosomes [26].

We define intelligent genome analysis as the ability to read, analyze, and interpret genomes not only quickly, but also accurately and efficiently enough to scale the analysis to population level. Some existing works on accelerating one or more steps in genome analysis sacrifice the optimality of the analysis results in order to reduce execution time (as described in [18,27]). Genome analysis is currently a first-tier diagnostic test for critically ill patients with rare genetic disorders, which necessitates the need for making the analysis much faster while maintaining the same or providing better analysis accuracy for successful clinical practice

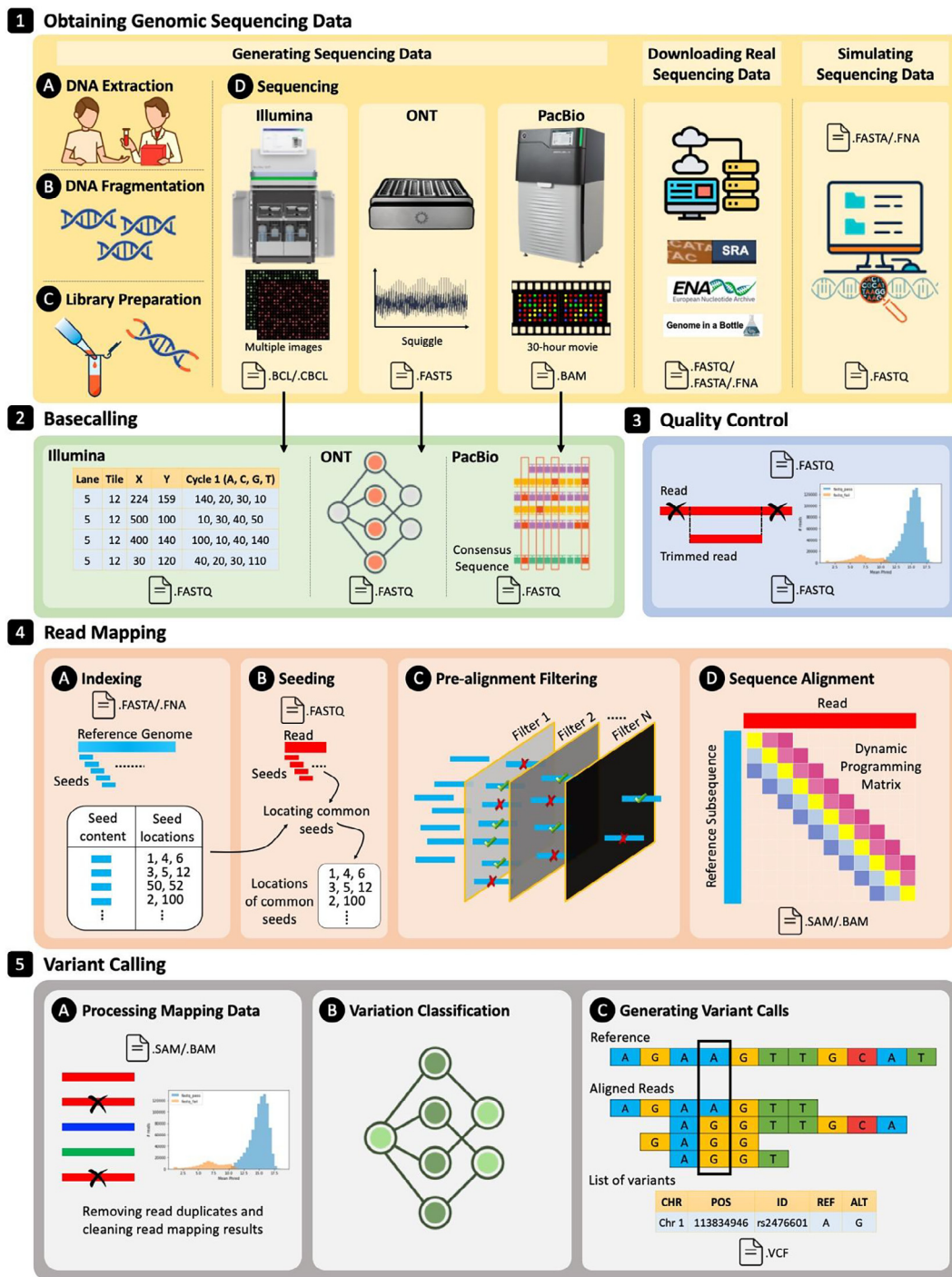


Fig. 1. Overview of a typical genome analysis pipeline. 1) Genomic sequencing data is first obtained through sequencing a new sample, downloading from publicly-available databases, or computer simulation. Sequencing starts with (A) extracting the DNA, (B) fragmenting it, (C) preparing its library, and (D) using sequencing machines for providing raw sequencing data. 2) Raw sequencing data needs to be converted into read sequences of A, C, G, T in the DNA alphabet using basecalling techniques. Basecalling techniques are sequencing technology dependent. 3) To ensure high quality sequencing reads, a quality control step is performed to filter out low quality subsequences of a read or an entire read sequence. 4) Read mapping step is performed to locate each read sequence in a reference genome. Read mapping is four steps: (A) indexing the reference genome, (B) extracting seeds from each read and locating common seeds with the index, (C) pre-alignment filtering dissimilar sequence pairs, and (D) performing sequence alignment for every sequence pair that passes the filtering. 5) Detecting and inferring genomic variations usually consists of three steps: (A) processing the read mapping data for increasing its quality, (B) classifying variations between mapped reads and a reference genome, and (C) identifying genomic variations.

[1,2,28,29]. For example, it is observed that at least 10 % of read sequences simulated from the human reference genome remain unmapped across 14 state-of-the-art aligners [30] due to potential mapping artifacts [31], which demonstrates that accuracy is still an issue even in read mapping that is extensively studied. On the other hand, the vast majority of genome analysis tools are

implemented as software running on general-purpose computers, while sequencing is performed using extremely specialized, high-throughput machines [18]. This introduces two main problems. 1) Modern sequencing machines generate read fragments at an exponentially higher rate than prior sequencing technologies, with their growth far outpacing the growth in computational power in

recent years [32]. 2) Genome analysis generates a large amount of *data movement* between the sequencing machine and the computer performing the analysis and between different components (e.g., compute units and main memory) of computers. The data movement across power-hungry buses is extremely costly in terms of both execution time and energy [33–41]. Increasing the number of CPUs used for genome analysis may decrease the overall analysis time, but significantly increases energy consumption and hardware costs, and potentially worsens the data movement bottleneck (more cores competing for memory access) [42,43]. Cloud computing platforms still suffer from similar issues along with additional concerns due to genomic data protection guidelines in many countries [44–49].

These costs and challenges are a significant barrier to enabling intelligent genome analysis that can keep up with sequencing technologies. As a result, there is a dire need for new computational techniques that can quickly process and analyze a tremendous number of extracted reads in order to drive cutting-edge advances in applications of genome analysis [27,35,36,50,51]. There exists a large body of work trying to tackle this problem by using intelligent algorithms, intelligent hardware accelerators, and intelligent hardware/software co-design [33–35,52–54]. Computer algorithms and hardware architectures are called *intelligent* if they are able to efficiently satisfy three principles, data-centric, data-driven, and architecture/algorithm/data-aware [55]. First, we would like to process genomic data efficiently by minimizing data movement and maximizing the efficiency with which data is handled, i.e., stored, accessed, and processed. Second, we would like to take advantage of the vast amounts of genomic data and metadata to continuously improve decision making (self-optimizing decisions) for many different use cases in science, medicine, and technology. Third, we would like to orchestrate the multiple components across the entire analysis system and adapt algorithms by understanding the structure of the underlying hardware, understanding analysis algorithms, and understanding various properties (i.e., the structure of the genome, type of sequencing data, quality of sequencing data) of each piece of data.

Our goal in this work is to survey a prominent set of these three types of intelligent acceleration efforts for guiding the design of new highly-efficient tools for intelligent genome analysis. *To this end*, we (1) discuss various state-of-the-art mechanisms and techniques that improve the execution time of one or more steps of the genome analysis pipeline using different modern high-performance computing architectures, and (2) highlight the challenges, that system designers and programmers must address to enable the widespread adoption of hardware-accelerated intelligent genome analysis. We comprehensively survey the efforts for accelerating read mapping step using algorithms, hardware/software codesign, and hardware accelerators in [27,56]. We also provide a systematic survey of algorithmic foundations and methodologies across 107 read mapping methods along with rigorous experimental evaluations in [18]. Different from these efforts, in this work we provide a systematic survey of all the five computational steps (obtaining the genomic data, basecalling, quality control, read mapping, and variant calling) of genome analysis with emphasis on the challenges introduced by the three prominent sequencing technologies from Illumina, Oxford Nanopore Technologies (ONT), and Pacific Biosciences (PacBio).

2. Obtaining genomic sequencing data

Genomic sequencing data (reads) can be obtained by 1) sequencing a DNA sample, 2) downloading real sequencing data from publicly available databases, or 3) simulating sequencing data, as we show in Fig. 1.1.

2.1. Generating sequencing data

One of the earliest successful protein sequencing attempts was made in the 1950s by Frederick Sanger who devoted his scientific life to the determination of the chemical structure of biological molecules, especially that of insulin [22,23,57,58]. The first DNA sequencing was successful only after two decades, in 1977, as introduced in two different sequencing methods by Sanger and Coulson [57], and by Maxam and Gilbert [59]. Though there was constant progress in sequencing attempts for different small genomes, we could obtain the first draft of the human genome sequence only in June 2000 as a result of a large international consortium, costing more than USD 3 billion (USD 1 for each DNA base) and more than 10 years of research [16,60]. This sequencing era was referred to as *first generation sequencing*. Since then, DNA sequencing has evolved at a fast pace with increasing sequencing throughput and decreasing cost, which lead to frequent updates and major improvements to the human genome sequence [61–63] and very recently have resulted in a near-complete human genome sequence [64]. These advances reshaped the landscape of modern biology and sequencing became an essential component of biomedical research.

Generating sequencing data includes three key steps: sample collection, preparation (known as *library preparation* [65]), and sequencing (Fig. 1.1). Sample collection and library preparation significantly depend on the protocol, preparation kit, minimum amount of DNA in the sample, and the sequencing machine, which require meeting rigorous requirements by the manufacturer of sequencing machines for the successful sequencing. The sample collection and library preparation steps are performed using non-computational methods in the “wet” laboratory or on-site, especially when the used sequencing machine is portable, prior to performing the actual sequencing. Each sequencing machine has different properties such as sequencing throughput, read length, sequencing error rate, type of raw sequencing data, sequencing machine size, and cost. Sequencing throughput is defined as the number of bases generated by the sequencing machine per second. Reads can have different lengths and they can be categorized into three types: 1) short reads (up to a few hundred bp), 2) ultra-long reads (ranging from hundreds to millions of bp), and 3) accurate long reads (up to a few thousands of bp). Though most of the existing sequencing machines are fundamentally different, they also share common properties, such as requiring library preparation, generating only fragments (i.e., reads) of the DNA sequence, introducing errors in the output sequencing data, and requiring converting the raw sequencing data into sequences of nucleotides (i.e., A, C, G, and T in the DNA alphabet). Most existing sequencing technologies start with DNA fragmentation as part of the library preparation protocol. DNA fragmentation refers to intentionally breaking DNA strands into fragments using, for example, resonance vibration [65,66]. DNA fragmentation helps to exploit a large number of DNA fragments for higher sequencing yield, as sequencing quality usually degrades towards the end of long DNA fragments [67] due to for example the limited lifetime of polymerase enzymes used for sequencing [68].

2.2. Downloading real sequencing data

Research groups, laboratories, and authors of research papers normally release their sequencing data on publicly available repositories to meet requirements of both reproducibility in biomedical and life science research and journals for data sharing [69]. There are currently more than 29 peta (10^{15}) bases of sequencing data publicly available as FASTQ files on the Sequence Read Archive (SRA) database [70], which is doubling in the number of bases every 2 years [71]. Other databases are also available, such as the

European Nucleotide Archive (ENA) [72]. There are also a large number of reference genome sequences, as FASTA files, for more than 108,257 distinct organisms publicly available in the NCBI Reference Sequence Database (RefSeq) database [73], which is doubling in the number of organisms every 3 years [74].

2.3. Simulating sequencing data

Sequencing data can be generated using computer simulation [75–78]. Many computational laboratories use simulated sequencing data, as they lack adequate resources to generate their sequencing data using sequencing machines [19] or lack access to gold standard experimental data when self-assessing a newly developed tool [12,79]. Existing sequencing technologies use different mechanisms and chemistries that result in sequencing data with different characteristics. Read simulators take into account many of these characteristics by modeling them according to each technology. These simulators differ in target sequencing technology, input requirements, and output format. They also have several aspects in common, such as requiring a reference genome, the minimum and maximum read lengths, sequencing error distribution, and type of genetic variations (e.g., substitutions, insertions, or deletions) that make read sequences different from the reference genome sequence. Most read simulators generate different


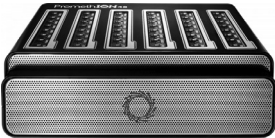

standard file formats, such as FASTQ, FASTA, or BAM, which can be used to locate potential mapping locations of each read in the reference genome.

Examples of read simulators for these three types include ART [80], Mason [78], and ReSeq [81] for short reads, PBSIM2 [76] and NanoSim [77] for ultra-long reads, and PBSIM [82] for accurate long reads (known as HiFi reads). However, the use of read simulations poses several limitations, as simulated data recapitulates the important features of real data and oversimplifies/biases the challenge for tested methods [19]. To avoid such biases, a common practice and more-comprehensive approach is to complement the simulated data with real experimental data. Hence, most journals require making the new sequencing data publicly accessible and explicitly reporting the *accession number* (a unique identifier given to sequencing data) of existing sequencing data used in the study.

3. Types of genomic sequencing data

There currently exist three different sequencing technologies that are widely-used to sequence DNA samples around the globe and in space [83]. Current prominent sequencing technologies and their output data can be categorized into three types: 1) short reads, 2) ultra-long reads, and 3) accurate long reads. In Table 1, we

Table 1
Summary of the main differences between the three types of prominent sequencing technologies. We choose the most capable instrument as a representative of each sequencing technology.

	Short Reads	Ultra-long Reads	Accurate Long Reads
Leading company	Illumina (https://www.illumina.com)	Oxford Nanopore Technologies (https://nanoporetech.com)	Pacific Biosciences (https://www.pacb.com)
Representative instrument	Illumina NovaSeq 6000	ONT PromethION 48	PacBio Sequel IIe
Instrument picture			
Instrument weight	481 kg	28 kg	362 kg
Instrument dimension (W × H × D in cm)	80 × 94.5 × 165.6	59 × 19 × 43	92.7 × 167.6 × 86.4
Instrument cost ¹	3Y	Y	1.6Y
Read length	100–300	100–2 M	10 K–30 K
Read length in a single data file	Fixed	Variable	Modest Variability
Accuracy	99.9 %	90 %–98 %	99.9 % ²
Maximum sequencing throughput per run	6 Tb	14 Tb	35 Gb (160 Gb of raw sequencing data) ³
Number of flow cells operating simultaneously	2	48	1
Hands-on library preparation time	45 min – 2 h	10 min–3 h	6 h
Turnaround library preparation time ⁴	1.5–6.5 h	24 h	24 h
Sequencing run time (including basecalling)	44 h	72 h	30 h

¹ Y can be as low as USD 300,000. The cost does not include consumables (e.g., flow cells and reagents) needed for each sequencing run.

² Consensus accuracy.

³ Per SMRT cell. Sequel IIe can process up to 8 SMRT cells sequentially, slide 25 https://www.pacb.com/wp-content/uploads/HiFi-Sequencing_and_Software_v10.1_Release_Technical_Overview_for_Sequel_II_System_and_Sequel_IIe_System_Users-Customer-Training-01.pdf.

⁴ Includes both hands-on library preparation time and waiting time.

summarize the main differences between these three types of prominent sequencing technologies. We provide more details on the sequencing machine size, cost, throughput, maximum library preparation time, and maximum sequencing time for the most capable instrument for each sequencing technology.

3.1. Short reads

Short read sequencing (sometimes called *second generation sequencing*) technologies, such as Illumina [17,84,85] and Singular Genomics [86], generate subsequences of length 100–300 bp. Illumina is currently the dominant supplier of sequencing instruments. The key advantage of Illumina short reads is the significantly low sequencing error rate (as low as 0.1 % of the read length) introduced by the sequencing machine [87]. Over 6 terabases can be generated in a single sequencing run in about two days using a single instrument (Illumina NovaSeq 6000). Illumina sequencing technology, called *sequencing by synthesis* (SBS), is very similar to that of the first generation sequencing (i.e., Sanger) [88,89]. A key procedural difference in comparison to Sanger sequencing is in the preparation of the sequencing library and the degree of parallelism and throughput during sequencing. Sanger sequencing libraries require multiple steps that require growth in culture and DNA isolation before sequencing [89]. This multistep process can be completed in approximately one week, at which point the processed DNAs are ready for sequencing.

Illumina sequencing requires a hands-on time of less than two hours (or a turnaround time of up to 6.5 h) to prepare the sample for sequencing. Library preparation in Illumina sequencing starts with fragmenting the DNA into short fragments and adding *adapters* (short single strands of synthetic DNA, called oligonucleotides [90]) to both fragment ends. These adapters enable binding each fragment to the flow cell. Fragments can then be amplified to create clusters of up to 1,000 identical copies of each single fragment. Sequencing is then performed base-by-base using a recent technique called *2-channel sequencing* [91]. During each sequencing cycle (3.5–6.75 min [92]), a mixture of two nucleotides, adenine and thymine, labeled with the same fluorescent dye (i.e., green color) is added to each cluster in the flow cell. Images are taken of the light emitted from each DNA cluster using a CMOS sensor. Next, a mixture of two nucleotides, adenine and cytosine, labeled with another fluorescent dye (i.e., red color) is added to each cluster in the flow cell. Another image is taken of the light emitted from each DNA cluster. During basecalling, the combinations of “light observed” and “no light observed” in the two images are interpreted. E.g., if the light is only observed in the first image, it is interpreted as a thymine base. If the light is observed in the second image, it is interpreted as a cytosine base. Clusters with light in both images are flagged as adenine bases, while clusters with no light in both images represent guanine bases. This process is repeated on each nucleotide for the length of the DNA fragment (read). The large number of clusters and the straightforward basecalling process make Illumina sequencing provide the highest throughput of accurate bases compared to other sequencing techniques.

3.2. Ultra-long reads

Ultra-long read (or *nanopore*) sequencing, called *third generation sequencing*, is more recent than Illumina sequencing. The first nanopore sequencing machine, MinION, was introduced in 2014 and made commercially available in 2015. The main concept behind nanopore sequencing was brainstormed much earlier in the 1990s [93]. The first MinION sequencing machine was promising as it was incredibly small in size (smaller than the palm of a hand and weighing only 87 g) compared to existing sequencing machines. However, its sequencing accuracy rate was 65 % (i.e.,

one out of every-three bases is erroneous, which can significantly degrade the accuracy of downstream analyses) [94].

The nanopore sequencing technology requires first preparing the sequencing library by fragmenting the DNA sequence and adding a sequencing adapter and a motor protein at each end of the fragment. The sequencing starts with passing each DNA fragment through a nanoscale protein pore (nanopore in short) that has an electrical current passing through it. The sequencer measures changes to an electrical current as nucleic acids, each with different electrical resistance, are passed through the nanopore. Using the computational basecalling step, the electrical signals are decoded into a specific DNA sequence. The motor protein helps control the translocation speed of the DNA fragment through the nanopore. Over 14 terabases can be generated in a single sequencing run in about 3 days using a single ONT PromethION 48 instrument (Table 1). Recent nanopore machines use dual electrical current sensors (called reader heads) to increase the accuracy of sensing and improve the detection resolution [94,95]. The accuracy of nanopore sequencing technology has been constantly improving from 65 % to above 90 % and can reach 98 % for some reads due to both dual sensing and improved basecalling [96–99]. However, this comes at the cost of computationally-expensive basecalling, as we discuss in Section 5.

3.3. Accurate long reads

The latest sequencing technology, referred to as third or fourth sequencing technology [100], is from Pacific Biosciences (PacBio). It generates high-fidelity (HiFi) reads that are relatively long (10–30 K) and highly accurate (99.9 %) [96,101,102]. The PacBio sequencing requires fragmenting the DNA molecule and adding double-stranded adapters (called SMRTbell) to both ends of the fragment. The DNA fragment has a DNA subsequence binding to its reverse complement sequence. This creates a circular DNA fragment for sequencing. The PacBio sequencing leverages multiple pass circular consensus sequencing (CCS) by sequencing the same circular DNA fragment at least 30 times and then correcting errors by calculating a consensus sequence [102]. Each sequencing pass through the circular DNA fragment produces a *subread*, which is used to calculate the consensus sequence by overlapping all resulting subreads of a single DNA fragment. The PacBio sequencing uses a polymerase that passes through the circular DNA fragment and incorporates fluorescently labeled nucleotides. As a base is held by the polymerase, fluorescent light is produced and recorded by a camera in real-time. The camera provides a movie of up to 30 h of continuous fluorescent light pulses that can be interpreted into bases. The PacBio sequencing provides the least sequencing throughput compared to Illumina and ONT. Over 35 gigabases can be generated in a single sequencing run in about 30 h (Table 1).

3.4. Discussion on types of sequencing reads

Short reads have the advantages of both low sequencing error rate and high sequencing throughput (number of basecalled bases). Another property of short reads that can be considered as an advantage is the equivalent length of all reads stored in the same FASTQ file, which helps in achieving, for example, load balancing between several CPU threads. Repetitive regions in genomes pose challenges for constructing assembly (de novo assembly [103]) using short reads. De novo assembly is an alternative to read mapping, in which it constructs the sequence of a genome from overlapping read sequences without comparison to a reference genome sequence. Both ultra-long reads and accurate long reads in general offer better opportunities for genome assembly and detecting complex structural variant calling.

Ultra-long reads provide two main advantages: 1) providing more contiguous assembly than that of short reads, where each contig can read up to 3 Mb (covering the typical length of bacteria genomes) [104], and 2) its read length is theoretically limited only by the length of the DNA fragment translocating through the pore [105] as it does not require enzyme-based nucleotide incorporation, amplification for cluster generation, nor detection of fluorescence signals [94,106]. However, nanopore sequencing data suffers from high sequencing error rate and some of its computational analysis steps require longer executing time and higher memory footprint compared to both short reads and accurate long reads. For example, performing de novo assembly (using Canu [107]) using ultra-long reads is at least fourfold slower than that when using accurate long reads, which is mainly because of the errors in the raw sequencing data [104]. This also leads to introducing new computational steps with the goal of polishing errors in the assembly [97,99,108]. Such new steps normally increase the computation overhead of analyzing ultra-long reads as the new steps are computationally expensive [108]. Another example of expensive steps for analyzing ultra-long reads is basecalling, which is based on neural networks. For this, ONT PromethION 48 includes 4 A100 GPU boards for accelerating basecalling and coping with its sequencing throughput.

The high accuracy of accurate long reads is a key enabler of the recent improvements in human genome assembly and unlocking complex regions of repetitive DNA [64]. More than 50 % of the regions previously inaccessible with Illumina short reads for the GRCh37 human reference genome are now accessible with HiFi reads (supplemented with ultra-long reads) for the GRCh38 human reference genome [102]. For other applications, such as profiling microbiomes through metagenomics analysis, the use of accurate long reads leads to detecting the same numbers of species as with the use of short reads [109]. However, the sequencing cost and computational expensive basecalling required to generate HiFi reads currently challenge widespread adoption.

This suggests that there is no preferable sequencing data type for all applications and use cases. Each sequencing technology has its own unique advantages and disadvantages. The short reads will continue to be widely used due to their very high accuracy and low cost. With increases in read lengths of Illumina sequencing technology [110], there will be a growing demand for adjusting existing algorithms or introducing new algorithms that leverage new properties of anticipated Illumina sequencing data. With increases in accuracy of long and ultra-long reads, there will be a growing demand for improving execution time of their computational steps (e.g., basecalling) and reducing overall sequencing cost to enable widespread adoption.

4. Genome analysis using different types of sequencing reads

We evaluate the performance of three typical genome analysis pipelines (Fig. 1) for the three prominent sequencing data types, 1) short reads, 2) ultra-long reads, and 3) accurate long reads, as we show in Fig. 2. We report the throughput of each step using a single CPU thread, running on a 2.3 GHz Intel Xeon Gold 5118 CPU with up to 48 threads and 192 GB DDR4 RAM. Note that nanopore basecalling is based on the throughput of Guppy running on a CPU [111]. We report the sequencing throughput using a single flow cell. We calculate the throughput of each step by dividing the number of bases (outputted by sequencing and basecalling or taken by quality control, read mapping, and variant calling) over the total execution time in hours. We provide the data and exact command lines used to run each tool in the GitHub repository of this paper.

We make five key observations based on Fig. 2. 1) Short read sequencing provides the highest throughput per flow cell compared to other long read sequencing technologies. 2) Genome analysis of both ultra-long reads and accurate long reads suffers from long execution time of their basecalling step. This is expected because both technologies use computationally expensive basecalling steps, as we explain in detail in Section 5. 3) Read mapping is the most computationally expensive step, followed by variant calling, in the genome analysis pipeline for short reads. 4) The first major bottleneck in the genome analysis pipelines for both accurate long reads and ultra-long reads is variant calling. Read mapping is the second and the third bottleneck in the genome analysis pipelines for accurate long reads and ultra-long reads, respectively. 5) Sequencing throughput is 341x and 56.8x higher than the throughput of read mapping and variant calling of short reads, respectively. Sequencing throughput is 2.4x and 93.2x (3.8x and 4.8x) higher than the throughput of read mapping and variant calling of ultra-long reads (accurate long reads), respectively.

We conclude that each type of sequencing data imposes different acceleration challenges and creates its own computational bottlenecks. There is also a dire need for developing new computational techniques that can overcome the existing computational bottlenecks and building new hardware architectures that can reduce data movements between different steps of genome analysis and improve overall analysis time and energy efficiency. In the next sections, we survey various state-of-the-art mechanisms and techniques that improve the execution time of one or more steps of the genome analysis pipelines for different types of sequencing data.

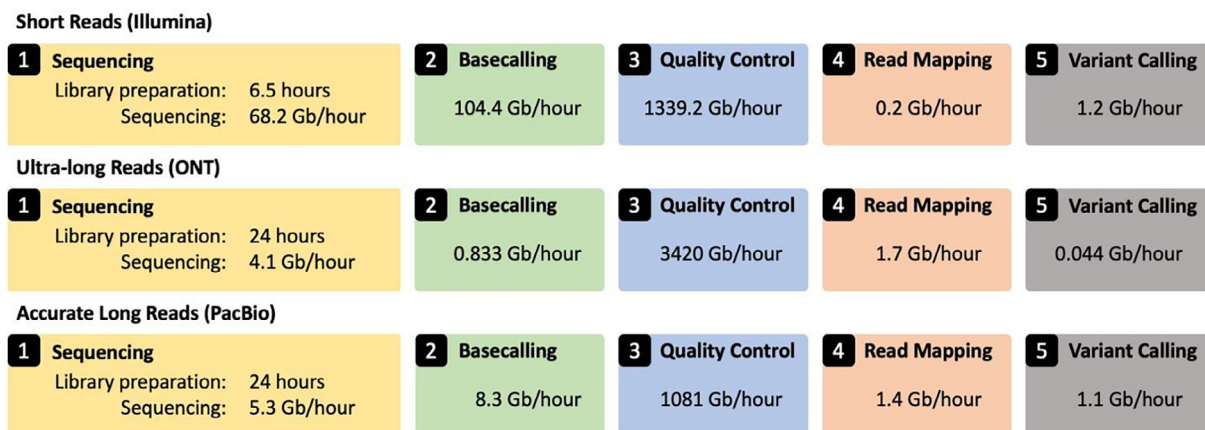


Fig. 2. Performance comparison of the five main steps of genome analysis pipeline.

5. Basecalling

Most existing sequencing machines do not provide read sequences in the DNA alphabets. Instead, they provide a native format that is sequencing technology dependent. Thus, existing sequencing technologies require a *basecalling* step (Fig. 1.2) to convert the native sequencing data format into a standard format that can be understood by all existing genome analysis tools, regardless of the sequencing technology used. Basecalling is the first computational step in the genome sequencing pipeline that converts raw sequencing data (images for Illumina, movie for PacBio, or electric current for ONT) into sequences of nucleotides (i.e., A, C, G, and T in the DNA alphabet). We provide in this section a brief description of basecalling for the three prominent sequencing technologies, Illumina short reads, Oxford Nanopore ultra-long reads, and PacBio HiFi accurate long reads. We summarize the main differences between their basecalling techniques in Table 2.

5.1. Illumina

During each cycle of Illumina sequencing, two images are taken to identify the possible chemical reactions occurring in DNA clusters. The light intensities captured in one or both images directly represent the type of the subject nucleotide, as we explain in Section 3.1. This information is stored as binary data in the CBCL file format. Each sequencing run provides a large number of CBCL files that need to be converted into a single FASTQ file. The basecalling per CBCL file can take up to 0.21 min (50 % of it is spent on reading and writing from/to files) [112], which is significantly less than the time for a single sequencing cycle (3.5–6.75 min [92]).

5.2. ONT

For nanopore sequencing, the conversion of the raw electrical current signal or *squiggle* into a sequence of nucleotides is challenging because: (1) signals have stochastic behavior and low signal-to-noise ratio (SNR) due to thermal noise and the lack of statistically significant current signals triggered by DNA motions, (2) electrical signals can have long dependencies of event data on neighboring nucleotides, and (3) the sensors (reader heads) used cannot measure the changes in electrical current due to a single nucleotide, but rather measuring the effect of multiple nucleotides together. Nanopore raw data are current intensity values measured at 4 kHz, saved in the FAST5 format (a modified HDF5 format). Nanopore basecalling is computationally expensive and its algorithm is quickly evolving. Neural networks have supplanted hidden Markov model (HMM) based basecallers for their better accuracy, and various neural network structures are being tested [96,115].

Another important advantage of nanopore basecalling is its ability to improve read accuracy (by 10 % [116]) by correcting possible sequencing errors.

ONT provides a production basecaller, called *Guppy*, and its development version, *Bonito*. Guppy is optimized for performance and accelerated using modern GPUs. Modern neural network basecallers such as Guppy and Bonito are typically composed of: (1) residual network with convolutional layers, and/or (2) long short-term memory-based recurrent layers. RNN-based models are used to model the temporal dependencies present in basecalling data. Other than Guppy, which is a GPU-based basecaller, most of the existing basecallers are only implemented for CPU execution. Thus, basecallers lack dedicated hardware acceleration implementations, which could greatly reduce the basecalling time. Even Guppy takes 25 h to basecall a 3 Gb human genome on a powerful server-grade GPU [117]. Potential alternatives to further accelerate Guppy and other basecallers are processing-in-memory techniques [33–36,50,52–54,118–120], which consist of placing compute capabilities near or inside memory. PIM techniques are particularly well suited for workloads with memory-bound behavior, as for example RNNs show [39,121]. For that, algorithm/architecture co-designed processing-in-memory (PIM) can accelerate Guppy by 6x [117]. ONT also provides Guppy-lite, a throughput-optimized version of Guppy that provides higher performance at the expense of the basecalling accuracy. Therefore, basecallers need to balance the tradeoff between speed and accuracy depending on the requirements of a particular application [122].

Instead of basecalling followed by analysis in basepair space, several works propose analyzing reads directly in their raw signal space, obviating or alleviating the need for expensive basecalling [123,124]. One motivation for such proposals is the *Read-Until* feature [125] of ONT sequencing machines, which allows to physically eject reads from each nanopore in real time if they are deemed not interesting for the application (e.g., do not belong to target species). Ejecting a DNA segment requires analyzing the partial squiggle signal much earlier than completing sequencing the complete DNA segment, which the computationally expensive basecalling cannot satisfy. SquiggleNet [126] is a deep-learning-based approach that classifies DNA sequences directly from electrical signals. SquiggleNet provides 90 % accuracy with real-time processing, which means that SquiggleNet wrongly identifies, on average, 10 % of the DNA segments as irrelevant. UNCALLED [127] is another approach that segments raw signals into possible k-mers and uses Ferragina-Manzini (FM) index along with a probabilistic model to search for k-mer matches with the target reference genome. UNCALLED achieves an accuracy of 93.7 % with an average detection speed of less than 50 ms per DNA segment. Sigmap [128] has comparable performance to UNCALLED, but it

Table 2

Summary of the main differences between the three types of prominent sequencing technologies. We choose the most capable instrument as a representative of each sequencing technology.

	Short Reads	Ultra-long Reads	Accurate Long Reads
Type of raw sequencing data (before basecalling)	Multiple images of fluorescence intensities for each sequencing cycle	Electrical signal for each DNA segment	Fluorescence traces captured continuously into a 30-hour movie
Input file format for basecalling	BCL or CBCL	FAST5	BAM
Expected size of basecalling input file	One CBCL file of size 350 MB per cycle, lane, and surface	10x the size of the corresponding FASTQ file	Subreads.BAM of size 0.5–1.5 TB
Basecalling algorithm	BCL2FASTQ	Guppy/Bonito (deep neural networks)	CCS
Basecalling time	48 minutes ¹	142 minutes ²	24 hours ³
Number of basecalled bases	83.5 Gb ¹	20 Gb ²	200 Gb of HiFi yield ³

¹ BCL2FASTQ based on SRR2890933, 1.67 billion reads (full 8 lanes) at 50 bp read length [113].

² Using a single V100 GPU, adapted from [111].

³ Using CCS v6.0 for a 25 KBases library on 2x64 cores at 2.4 GHz, adapted from [114].

overcomes the applicability limitations of UNCALLED to large genomes by optimizing the raw signal mapping pipeline. This optimization leads to a 4.4x improvement in detection performance. SquiggleFilter [126] matches reads against a target genome using hardware-accelerated dynamic time warping algorithm [120,129]. SquiggleFilter needs fewer signal measurements to identify irrelevant DNA segments compared to UNCALLED, allowing for earlier ejection from the pores. It is computationally cheaper and particularly suited for sequencing in the field, where the computational resources are limited. The computational step of SquiggleFilter can be as fast as 0.027 ms per DNA segment, depending on the size of the to-be-compared-with reference genome. Analysis in raw signal space can also provide information that would otherwise be lost during basecalling, such as chemical modifications of nucleotides, or estimating the length of the poly(A) tail of mRNA [123].

5.3. PacBio

For PacBio sequencing, 30 h of continuous fluorescent light pulses are recorded as a movie. The movie can be directly interpreted into bases, resulting in multiple subreads (stored in BAM format), where each subread corresponds to a single pass over the circular DNA fragment. The current PacBio basecalling workflow is CCS [130], which includes aligning the subreads to each other using computationally expensive sequencing alignment tools, such as KSW2 [131] and Edlib [132], and other polishing steps. We observe that there are currently neither hardware acceleration nor alternative more efficient (more customized than KSW2 and Edlib) algorithms for improving the runtime of PacBio basecalling.

6. Quality control

The goal of the quality control (QC) step (Fig. 1.3) is to examine the quality of some regions in the read sequence or the entire read sequence and trim them if they are of low quality or not needed anymore (as in adapters for sequencing). Some of the causes for low quality regions are library preparation (e.g., fragmenting the DNA into very short fragments) and sequencing (e.g., low base quality during sequencing) [96,133]. The QC step ensures high quality of the reads and hence high quality downstream analysis.

The QC step evaluates the quality of the entire read sequence by (1) examining the intrinsic quality of the read sequence, that is the average quality score generated by the sequencing machine before or after basecalling, (2) assessing the length of the read, and (3) evaluating the number of ambiguous bases (N) in a read. The QC step also evaluates the quality of individual bases for (1) masking out an untrustworthy region in a read if the average quality score of the region's bases is low and (2) trimming beginning or/and trailing regions of a read if it is. For example, Illumina sequences have degraded quality towards the ends of the reads, while adapter sequences are added to both head and tail of the DNA fragment for sequencing. Looking at quality distribution over base positions can help decide the trimming sites. There are also other quality control steps that can be applied during or after read mapping, such as those provided by Picard [134], and can be hardware accelerated as in [135]. Some other quality control steps can be carried out directly after sequencing and before basecalling, such as trimming barcode sequences used for labeling different samples to be sequenced within the same sequencing run [136,137].

There exists a number of software tools for performing QC. FastQC [138] is the most popular tool for controlling the quality of Illumina sequencing data. FastQC takes a FASTQ file as its input and performs ten different quality analyses. A given FASTQ file may

pass or fail each analysis and a FASTQ file is usually accepted when it passes all quality analyses. There are also a large number of QC software tools for long read (both nanopore and HiFi), such as LongQC [139], which uses expensive read mapping, minimap2 [131], for low coverage detection. RabbitQC [140] exploits modern multicore CPUs to parallelize the QC computations and provides an order of magnitude faster performance for the three prominent types of sequencing technologies. We observe that there are currently a large number of software tools for QC, but we still lack efficient hardware accelerators for improving the QC step.

7. Read mapping

The goal of read mapping is to locate possible subsequences of the reference genome sequence that are similar to the read sequence while allowing at most E edits, where E is the *edit distance threshold*. Tolerating a number of differences is essential for correctly finding possible locations of each read due to sequencing errors and genetic variations. Mapping billions of reads to the reference genome is *still* computationally expensive [27,35,36,53,141]. Therefore, most existing read mapping algorithms exploit two key heuristic steps, *indexing* and *filtering*, to reduce the search space for each read sequence in the reference genome. Read mapping includes four computational steps (Fig. 1.4), indexing, seeding, pre-alignment filtering, and sequence alignment. First, a read mapper starts with building a large index database using subsequences (called *seeds*) extracted from a reference genome to enable quick and efficient querying of the reference genome. Second, the mapper uses the prepared index database to determine one or more possible regions of the reference genome that are likely to be similar to each read sequence by matching subsequences extracted from each read with the subsequences stored in the index database.

Third, the read mapper uses filtering heuristics to quickly examine the similarity for every read sequence and one potential matching segment in the reference genome identified during seeding. As only a few short subsequences are matched between each read sequence and each reference genome segment, there can be a large number of differences between the two sequences. Hence filtering heuristics aim to eliminate most of the dissimilar sequence pairs by performing minimal computations. Fourth, the mapper performs sequence alignment to check whether or not the remaining sequence pairs that pass the filter are actually similar. Due to potential differences, the similarity between a read and a reference sequence segment must be identified using an approximate string matching (ASM) algorithm. The ASM typically uses a computationally-expensive dynamic programming (DP) algorithm to *optimally* (1) examines all possible *prefixes* of two sequences and tracks the prefixes that provide the highest possible *alignment score* (known as *optimal alignment*), (2) identify the type of each difference (i.e., insertion, deletion, or substitution), and (3) locate each difference in one of the two given sequences. Such alignment information is typically output by read mapping into a sequence alignment/map (SAM, and its compressed representation, BAM) file [142]. The alignment score is a quantitative representation of the quality of aligning each base of one sequence to a base from the other sequence. It is calculated as the sum of the scores of all differences and matches along the alignment implied by a user-defined scoring function. DP-based approaches usually have quadratic time and space complexity (i.e., m^2) for a sequence length of m , but they avoid re-examining the same prefixes many times by storing the examination results in a DP table. The use of DP-based approaches is unavoidable when optimality of the alignment results is desired [143].

We evaluate the performance of three state-of-the-art read mappers, BWA-MEM2, minimap2, and pbmm2, for three different types of sequencing data, short reads, ultra-long reads, and accurate long reads, respectively (Table 3). We make three key observations. 1) The indexing time, indexing peak memory, and index size provided by the three read mappers all differs significantly though the three mappers index the same reference genome. BWA-MEM2 provides the highest indexing time, the highest indexing peak memory, and the highest index size. 2) The mapping throughput is also rapidly different between the three read mappers as they perform different algorithms. We define the mapping throughput as the number of input bases processed in 1 second. 3) The ratio of the number of primary alignments to the total number of records in the output SAM file provided by BWA-MEM2, minimap2, and pbmm2 is 98.8%, 46.3%, and 92.1%, respectively. Given the execution time spent on read mapping, all four steps of read mapping have been targeted for acceleration.

7.1. Accelerating indexing and seeding

Indexing and seeding fundamentally use the same algorithm to extract the subsequences from the reference genome or read sequences. The only difference is that the indexing step stores the seeds extracted from the reference genome in an indexing database, while the seeding step uses the extracted seeds to query the indexing database. The indexing step populates a lookup data structure that is indexed by the contents of a seed (e.g., its hash value), and identifies all locations where a seed exists in the reference genome (Fig. 1 .4.A and Fig. 1 .4.B). Indexing needs to be done only once for a reference genome, thus it is not on the critical path for most bioinformatics applications. Seeding is performed for every read sequence and thus it contributes to the execution time of read mapping. However, the number of extracted seeds in both steps (indexing and seeding), the length of each seed, and the frequency of each seed can significantly impact the overall memory footprint, performance, and accuracy of read mapping

[131,144,145]. For example, querying very short seeds leads to a large number of mapping locations that need to be checked for a sequence alignment, which makes later steps more computationally costly. In contrast, querying very long seeds may prevent identifying some mapping locations. This is because the querying process usually requires the entire seed to exactly appear in the indexing database, and longer seeds have a higher probability of containing mismatches. This can lead to missing some mapping locations and causing a low accuracy (defined in this context as *sensitivity*, the ability of a read mapper to find the location of a read sequence that already exists in the reference genome). The properties of the data affect the tradeoffs between these choices, for example long reads tend to have a higher error rate, thus shorter seed lengths are appropriate for good sensitivity. There are three major directions for improving the indexing and seeding steps: (1) better seed sampling techniques, (2) better indexing data structures, and (3) accelerating the task and minimizing its data movement through specialized hardware.

7.1.1. Sampling seeds

The goal of sampling the seeds is to reduce redundant information that can be inferred from extracted seeds. For example, choosing all possible overlapping subsequences of length k , called k -mers, as seeds causes each base to appear in k seeds, causing unnecessarily high memory footprint and inefficient querying due to large number of seed hits. Thus, state-of-the-art read mapping algorithms (e.g., minimap2 [131]) typically aim to reduce the number of seeds that are extracted for the index structure by sampling all possible k -mers into a smaller set of k -mers. A common strategy to choose such a smaller set is to impose an ordering (e.g. lexicographically or by hash value) on every group of w overlapping k -mers and choosing only the k -mer with the smallest order as a seed, known as the minimizer k -mers [146]. Minimizer-based approach guarantees finding at least one seed in a group of k -mers, known as *windowing guarantee*, ensuring low information loss depending on the values of k and w .

Table 3
Evaluation analysis of three state-of-the-art read mappers, BWA-MEM2, minimap2, and pbmm2, for three different types of sequencing data, short reads, ultra-long reads, and accurate long reads, respectively.

	Short reads	Ultra-long reads	Accurate long reads
Read mapping tool	BWA-MEM2	minimap2	pbmm2
Version	2.2.1	2.24-r1122	1.7.0
Reference genome	Human genome (HG38), GCA_000001405.15, FASTA size 3.2 GB		
Indexing time (CPU)	2002 sec	163 sec	144 sec
Indexing peak memory	72.3 GB	11.4 GB	14.4 GB
Indexing size*	17 GB	7.3 GB	5.7 GB
Read set (accession number)	https://www.ebi.ac.uk/ena/browser/view/ERR194147	https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG003_NA24149_father/UCSC_Ultralong_OxfordNanopore_Promethion/GM24149_1.fastq.gz	https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG003_NA24149_father/PacBio_CCS_15kb_20kb_chemistry2/reads/PBmixSequel729_1_A01_PBTH_30hours_19kbV2PD_70pM_HumanHG003.fastq.gz
Number of reads	1,430,362,384	6,724,033	1,289,591
Number of bases	144,466,600,784	82,196,263,791	24,260,611,730
Read mapping time	868.9 hours ¹	48.6 h	17.5 h
Mapping peak memory	131.2 GB ¹	9.6 GB	20 GB
Number of mapped reads ²	2,842,576,947 ¹	3,854,572	1,286,256
Mapping throughput (input bases/mapping time)	92,369 bases/sec	469,801 bases/sec	385,090 bases/sec
Number of output mappings (SAM)	2,875,143,231	8,322,218	1,396,899
File size of output mappings (SAM)	222.6 GB	190.2 GB	54.4 GB

¹ For paired-end read mapping.

² After excluding secondary, supplementary, and unmapped alignments using SAMtools and SAM flag 2308.

Additionally, the frequency (i.e., the size of the location list) of each minimizer seed can be restricted up to a certain threshold to reduce the workload for querying and filtering the seed hits [144,147,148]. Similar to minimizers, the syncmer approach [149] is a more recent type of sampling method with a different selection criteria than minimizers that has shown to provide more uniform distribution of seeds to achieve better sensitivity. The syncmer approach chooses a seed as a minimizer whose substring located at a fixed location achieves the smallest order compared to that of substrings of other seeds. This strategy ensures a certain gap between any two consecutive minimizers, which enables read mappers to report mapping locations for reads that are unmapped using minimizers-based read mappers [150].

To increase the sensitivity of read mappers, other approaches can be applied, such as spaced [151] and strobemer [152] seeds. Spaced seeds [151] exclude some bases from each seed following a predetermined pattern, where the resulting seeds are composed of multiple shorter substrings [153]. Spaced seeds can achieve high sensitivity when finding seed matches by allowing the excluded bases to mismatch (substitute) with their corresponding bases. S-conLSH [154] is a recent approach that applies the locality-sensitive hashing idea and uses multiple patterns on the same sequence to enable excluding different sets of characters belonging to a sequence. A recent approach, known as strobemers [152], improves on spaced seeds by varying the sizes of the spaces dynamically based on selection criterias. These strategies for joining/linking seeds are orthogonal to minimizers and syncmers, and recent work shows such approaches can be combined for additional sensitivity improvements [155].

There are also other works that use the minimizer sampling strategy without providing a windowing guarantee, such as MinHash [156]. MinHash finds a single minimizer k-mer from an entire sequence (i.e., w equals the sequence length). To find many minimizer k-mers from the same sequence, the idea is to use many hash functions and find the minimizer k-mer from each hash function. The goal is to find a minimizer k-mer at n -many regions of a sequence using n -many different hash functions, providing a sampled set of k-mers. Although the MinHash approach is effective when comparing sequences of similar lengths, it uses many redundant minimizers when comparing sequences of varying length to ensure high accuracy, which comes with a high cost of performance and memory overhead [157].

7.1.2. Improving data structures for seed lookups

After choosing the appropriate method for extracting seeds, the goal is to store or query them using the index. The straightforward data structure to find seed matches is a hash table that stores the hash value of each seed as a key, and location lists of each seed as values. Hash tables have been used since 1988 in read mapping [18] and are still used even by the state-of-the-art read mappers as they show good performance in practice [131]. Choosing a hash function is an important design choice for hash tables. It is desired to use hash functions with low collision rates so that different seeds are not assigned to the same hash value. It is also desired to increase the collision rate for highly similar seeds to improve the overall sensitivity. A recent approach, BLEND [145], aims to generate hash values such that highly similar seeds can have the same hash value while dissimilar seeds are still assigned to different hash values with low collision rates. Such an approach can find approximate (i.e., fuzzy) matches of seeds directly using their hash values, which can be applied to other seeding approaches that enable finding inexact matching, such as spaced seeds and strobemers.

There are also other data structures that can be efficiently used with the aim of reduced memory footprint and improved querying time. One example of such data structures is FM-index

(implemented by Langarita et al. [158]), which provides a compressed representation of the full-text index, while allowing for querying the index without the need for decompression. This approach has two main advantages. 1) We can query seeds of arbitrary lengths, which helps to reduce the number of queried seeds. 2) It typically has less (by 3.8x) memory footprint compared to that of the indexing step of minimap2 [18]. However, there is no significant difference in read mapping runtime due to the use of either indexing data structure [18]. One major bottleneck of FM-indexes is that locating the exact matches by querying the FM-index is significantly slower than that of classical indexes [158,159]. The FM-index can be accelerated by at least 2x using SIMD-capable CPUs [160]. BWA-MEM2 [159] proposes an uncompressed version of the FM-index that is at least 10x larger than the compressed FM-index to speed up the querying step by 2x. The seeding step of BWA-MEM2 can be further accelerated by 2x by using *enumerated radix trees* on recent CPUs to reduce the number of memory accesses and improve the access patterns [161]. Hash-based minimizer lookup can be replaced with learned indexes [162]. The learned indexes use machine learning models to predict the locations of the queried minimizers. The expected benefit of such a machine learning-based index is the reduced size of the index as it does not store the locations of the seeds. However, it is shown that a learned-index based read mapper has the same memory footprint as the hash-table based read mapper [163].

7.1.3. Reducing data movement during indexing

Indexing and seeding remain memory-intensive tasks [53], and hence do not fit modern processor centric systems well. An alternative approach is PIM, where processing happens either inside the memory chip itself, or close to it [34]. This approach can improve both energy efficiency, by moving data a shorter distance, as well as throughput, by providing more total memory bandwidth [34]. MEDAL [164] proposes integrating small ASIC accelerators for seeding close to off-the-shelf DRAM chips on standard LRDIMM memory modules. GenStore [53] proposes a seeding and filtering accelerator inside SSDs, providing comparable advantages to PIM approaches, but with the key difference that the index and reads do not have to be moved outside of the storage device for seeding. The higher internal bandwidth of SSDs provides increased throughput, and the reduced data movement improves energy efficiency. RADAR [165] implements the search for exact matches in an index database by storing the database in 3D Resistive Random Access Memory (ReRAM) based Content Addressable memory (ReCAM). The database can be directly queried without offloading it, leading to a small amount of data movement and highly energy efficient operation.

7.2. Accelerating pre-alignment filtering

After finding one or more potential mapping locations of the read in the reference genome, the read mapper checks the similarity between each read and each segment extracted at these mapping locations in the reference genome. These segments can be *similar* or *dissimilar* to the read, though they share common seeds. To avoid examining dissimilar sequences using computationally-expensive sequence alignment algorithms, read mappers typically use filtering heuristics that are called *pre-alignment filters* (Fig. 1.4.C). The key idea of pre-alignment filtering is to quickly estimate the number of edits between two given sequences and use this estimation to decide whether or not the computationally-expensive DP-based alignment calculation is needed – if not, a significant amount of time is saved by avoiding DP-based alignment. If two genomic sequences differ by more than the edit distance threshold, then the two sequences are identified as dissimilar sequences and hence DP calculation is not needed.

Edit distance is defined as the minimum number of single character changes needed to convert a sequence into the other sequence [166]. In practice, only genomic sequence pairs with an edit distance less than or equal to a user-defined threshold (i.e., E) provide useful data for most genomic studies [18,51,52,141,167,168]. Pre-alignment filters use one of four major approaches to quickly filter out the dissimilar sequence pairs: (1) the pigeonhole principle, (2) base counting, (3) q -gram filtering, or (4) sparse DP. Long read mappers typically use q -gram filtering or sparse DP, as their performance scales linearly with read length and independently of the edit distance.

7.2.1. Pigeonhole principle

The pigeonhole principle states that if E items are put into $E + 1$ boxes, then one or more boxes would be empty. This principle can be applied to detect dissimilar sequences and discard them from the candidate sequence pairs used for ASM. If two sequences differ by E edits, then they should share at least a single subsequence (free of edits) among any set of $E + 1$ non-overlapping subsequences [141], where E is the edit distance threshold. Pigeonhole-based pre-alignment filtering can accelerate read mappers even without specialized hardware. For example, the Adjacency Filter [148] accelerates sequence alignment by up to $19 \times$. The accuracy and speed of pre-alignment filtering with the pigeonhole principle have been rapidly improved over the last seven years. Shifted Hamming Distance (SHD) [168] uses SIMD-capable CPUs to provide high filtering speed, but supports a sequence length up to only 128 base pairs due to the SIMD register widths. GateKeeper [167] utilizes the large amounts of parallelism offered by FPGA architectures to accelerate SHD and overcome such sequence length limitations. MAGNET [169] provides a comprehensive analysis of all sources of filtering inaccuracy of GateKeeper and SHD. Shouji [141] leverages this analysis to improve the accuracy of pre-alignment filtering by up to two orders of magnitude compared to both GateKeeper and SHD, using a new algorithm and a new FPGA architecture.

SneakySnake [51] achieves up to four orders of magnitude higher filtering accuracy compared to GateKeeper and SHD by mapping the pre-alignment filtering problem to the single net routing (SNR) problem in VLSI chip layout. SNR finds the shortest routing path that interconnects two terminals on the boundaries of a VLSI chip layout in the presence of obstacles. SneakySnake is the only pre-alignment filter that efficiently works on CPUs, GPUs, and FPGAs. To further reduce data movements, SneakySnake is redesigned to exploit the near-memory computation capability on modern FPGA boards equipped with high-bandwidth memory (HBM) [54]. Near-memory pre-alignment filtering improves performance and energy efficiency by 27.4x and 133x, respectively, over SneakySnake running on a 16-core (64 hardware threads) IBM POWER9 CPU [54]. GenCache [170] proposes to perform highly-parallel pre-alignment filtering inside the CPU cache to reduce data movement and improve energy efficiency, with about 20 % cache area overhead. GenCache shows that using different existing pre-alignment filters together (a similar approach to [171]), each of which operates only for a given edit distance threshold (e.g., using SHD only when is between 1 and 5), provides a 2.5x speedup over GenCache with a single pre-alignment filter. Several pigeonhole principle based pre-alignment filters are evaluated for wide-range FPGA platforms [172].

7.2.2. Base counting

The base counting filter compares the numbers of bases (A, C, G, T) in the read with the corresponding base counts in the reference segment. The sum of absolute differences of the base counts provides an upper bound on the edit distance of the read and reference segment. If one sequence has, for example, three more Ts than

another sequence, then their alignment has at most three edits. If half of the sum of absolute differences between the four base counts is greater than E , then the two sequences are dissimilar and the reference segment is discarded. The base counting filter is used in mrsFAST-Ultra [173] and GASSST [171]. Such a simple filtering approach rejects a significant fraction of dissimilar sequences (e.g., 49.8 %–80.4 % of sequences, as shown in GASSST [171]) and thus avoids a large fraction of expensive verification computations required by sequence alignment algorithms. A PIM implementation of base counting can improve filtering time by 100x compared to its CPU implementation [174].

7.2.3. q -gram filtering approach

The q -gram filtering approach considers all of the sequence's possible overlapping substrings of length q (known as q -grams). Given a sequence of length m , there are $m - q + 1$ overlapping q -grams that are obtained by sliding a window of length q over the sequence. A single difference in one of the sequences can affect at most q overlapping q -grams. Thus, differences can affect no more than $q \cdot E$ q -grams, where E is the edit distance threshold. The *minimum number* of shared q -grams between two similar sequences is therefore $(m - q + 1) - (q \cdot E)$. This filtering approach requires very simple operations (e.g., sums and comparisons), which makes it attractive for hardware acceleration, such as in GRIM-Filter [52]. GRIM-Filter exploits the high memory bandwidth and computation capability in the logic layer of 3D-stacked memory to accelerate q -gram filtering in the DRAM chip itself, using a new representation of reference genome that is friendly to in-memory processing. q -gram filtering is generally robust in handling only a small number of edits, as the presence of edits in any q -gram is significantly underestimated (e.g., counted as a single edit) [175]. The data reuse in GRIM-Filter can be exploited for improving both performance and energy efficiency of filtering [176].

7.2.4. Sparse dynamic programming

Sparse DP algorithms exploit the exact matches (seeds) shared between a read and a reference segment to reduce execution time. These algorithms exclude the corresponding locations of these seeds from estimating the number of edits between the two sequences, as they were already detected as exact matches during indexing. Sparse DP filtering techniques link the overlapping seeds together to build longer chains and use the total length of the calculated chains as a metric for filtering the sequence pairs. This approach is also known as *chaining*, and is used in minimap2 [131] and rHAT [177]. GPU and FPGA accelerators [178] can achieve 7x and 28x acceleration, respectively, compared to the sequential implementation (executed with 14 CPU threads) of the chaining algorithm used in minimap2. mm2-fast [163] accelerates minimap2's chaining step by up to 3.1x with SIMD instructions. mm2-ax [179] accelerates mm2-fast's chaining step by up to 12.6x using a GPU. Modular Aligner [180] introduces an alternative to chaining based on two seed filtering techniques, achieving better performance than minimap2 in terms of both accuracy and runtime.

7.3. Accelerating sequence alignment

After filtering out most of the mapping locations that lead to dissimilar sequence pairs, read mapping calculates the sequence alignment information for every read and reference segment extracted at each mapping location (Fig. 1.4.D). Sequence alignment calculation is typically accelerated using one of two approaches: (1) accelerating optimal affine gap scoring DP-based algorithms using hardware accelerators, and (2) developing heuristics that sacrifice the optimality of the alignment score solution in

order to reduce alignment time. Affine gap scores are typically calculated using the Smith-Waterman-Gotoh algorithm [181], allowing for linear integer scores for matches/substitutions, and affine integer scores for gaps. Affine gap scores are more general than linear or unit (edit distance) costs, but are more costly to compute by a constant factor. Despite more than three decades of attempts to accelerate sequence alignment, the fastest known edit distance algorithm [182] has a nearly quadratic running time, $O(m^2/\log_2 m)$ for a sequence of length m , which is proven to be a tight bound, assuming the strong exponential time hypothesis holds [143]. A common approach to reducing the algorithmic work without sacrificing optimality is to define an *edit distance threshold*, limiting the maximum number of allowed single-character edits in the alignment. In this case, only a subset of the entries of the DP table is computed, called *diagonal vectors*, as first proposed in Ukkonen's banded algorithm [183]. The number of diagonal vectors required for computing the DP matrix is $2E + 1$, where E is the edit distance threshold. This reduces the runtime complexity to $O(m^*E)$. This approach is effective for short reads, where the typical sequencing error rates are low, thus a low edit distance threshold can be chosen. Unfortunately, as long reads have high sequencing error rates (up to 20 % of the read length), the edit distance threshold for long reads has to be high, which results in calculating more entries in the DP matrix compared to that of short reads. The use of heuristics (i.e., the second approach) helps to reduce the number of calculated entries in the DP matrix and hence allows both the execution time and memory footprint to grow only linearly or less with read length (as opposed to quadratically with classical DP). Next, we describe the two approaches in detail.

7.3.1. Accurate alignment accelerators

From a hardware perspective, sequence alignment acceleration has five directions: (1) using SIMD-capable CPUs, (2) using multicore CPUs and GPUs, (3) using FPGAs, (4) using ASICs, and (5) using processing-in-memory architectures. Parasail [184] and KSW2 (used in minimap2 [131]) exploit both Ukkonen's banded algorithm and SIMD-capable CPUs to compute banded alignment for a sequence pair with a configurable scoring function. SIMD instructions offer significant parallelism to the matrix computation by executing the same vector operation on multiple operands at once. mm2-fast [163] accelerates KSW2 by up to 2.2x by matching its SIMD capability to recent CPU architectures. KSW2 is nearly as fast as Parasail when KSW2 does not use heuristics (explained in Section 7.3.2). The wavefront algorithm (WFA) [185] reformulates the classic Smith-Waterman-Gotoh recursion such that the runtime is reduced to $O(m^*s)$ for a sequence pair of length m and affine gap cost of s without fixing the value of s ahead of time. It is SIMD-friendly and shows significant speedups for sequence pairs that have high similarity. The memory footprint and runtime complexity of WFA can be improved to $O(s^2)$ [186]. However, this improved runtime is not practical due to a large constant factor. The memory footprint of the WFA algorithm can be improved to $O(s)$ at the expense of an increase in runtime complexity to $O(m^*s)$ [187]. LEAP [188] formulates what can be considered a more general version of WFA, which is applicable to any convex penalty scores.

The multicore architecture of CPUs and GPUs provides the ability to compute alignments of many independent sequence pairs concurrently. GASAL2 [189] exploits the multicore architecture of both CPUs and GPUs for highly-parallel computation of sequence alignment with a user-defined scoring function. Unlike other GPU-accelerated tools, GASAL2 transfers the bases to the GPU, without encoding them into binary format, and hides the data transfer time by overlapping GPU and CPU execution. GASAL2 is up to 20x faster than Parasail (when executed with 56 CPU threads). BWA-MEM2 [159] accelerates the banded sequence alignment of its predecessor (BWA-MEM [190]) by up to 11.6x,

by leveraging multicore and SIMD parallelism. A GPU implementation [191] of the WFA algorithm improves the original CPU implementation by 1.5–7.7x using long reads.

Other designs, such as FPGASW [192], exploit the very large number of hardware execution units in FPGAs to form a linear systolic array [193]. Each execution unit in the systolic array is responsible for computing the value of a single entry of the DP matrix. The systolic array computes a single vector of the matrix at a time. The data dependency between the entries restricts the systolic array to computing the vectors sequentially (e.g., top-to-bottom, left-to-right, or in an anti-diagonal manner). FPGASW has a similar execution time as its GPU implementation, but is 4x more power efficient. SeedEx [194] designs an FPGA accelerator similar to FPGASW, but improves hardware utilization by speculatively computing fewer than $2E + 1$ diagonal bands, and then applying optimality checking heuristics to guarantee correct results. An FPGA accelerator [195] can accelerate the WFA algorithm by up to 8.8x and improve its energy efficiency by 9.7x for only short reads.

Specialized hardware accelerators (i.e., ASIC designs) provide application-specific, power- and area-efficient solutions to accelerate sequence alignment. For example, GenAx [196] is composed of SillaX, a sequence alignment accelerator, and a second accelerator for finding seeds. SillaX supports both a configurable scoring function and traceback operations. SillaX is more efficient for short reads than for long reads, as it consists of an automata processor whose performance scales quadratically with the edit distance. GenAx is 31.7x faster than the predecessor of BWA-MEM2 (i.e., BWA-MEM [190]) for short reads. Recent PIM architectures such as RAPID [197] exploit the ability to perform computation inside or near the memory chip to enable efficient sequence alignment. RAPID modifies the DP-based alignment algorithm to make it friendly to in-memory parallel computation by calculating two DP matrices (similar to Smith-Waterman-Gotoh [181]): one for calculating substitutions and exact matches and another for calculating insertions and deletions. RAPID claims that this approach efficiently enables higher levels of parallelism compared to traditional DP algorithms. The main two benefits of RAPID and such PIM-based architectures are higher performance and higher energy efficiency [33,34], as they alleviate the need to transfer data between the main memory and the CPU cores through slow and energy hungry buses, while providing high degree of parallelism with the help of PIM. RAPID is on average 11.8x faster and 212.7x more power efficient than 384-GPU cluster implementation of sequence alignment, known as CUDAlign [198]. A recent PIM architecture of WFA algorithm implemented in real hardware provides up to 4.87x higher throughput than the 56-thread CPU implementation using short reads [199].

7.3.2. Alignment accelerators with limited functionality

The second direction is to *limit* the functionality of the alignment algorithm or *sacrifice* the optimality of the alignment solution in order to reduce execution time. The use of restrictive functionality and heuristics limits the possible applications of the algorithms that utilize this direction. Examples of limiting functionality include limiting the scoring function (e.g. allowing only linear gap or unit scores), and calculating only the alignment score without performing the backtracking step [200]. There are several existing algorithms and corresponding hardware accelerators that limit scoring function flexibility. An example of limiting the scoring function is Myers' bit-vector algorithm [201], where the scoring function is limited to edit distance [166]. In this case, all types of edits are penalized equally when calculating the total alignment score. Restrictive scoring functions enable computation with smaller bit-widths, such that either smaller registers can be used, or multiple DP entries fit into a single SIMD register. This

reduces the total execution time of the alignment algorithm by operating on multiple DP entries in parallel in a SIMD fashion. In the case of Myers's bit-vector algorithm a single 64-bit register can hold the values of 64 entries of the DP matrix. BitPAL [202] expands on the idea by limiting the scoring function to linear gap scores and achieves speedups through bit-parallel execution. ASAP [203] accelerates edit distance calculation by up to 63.3x using FPGAs compared to its CPU implementation. The use of a fixed scoring function as in Edlib [132], which is the state-of-the-art implementation of Myers' bit-vector algorithm, helps to outperform Parasail (which uses a flexible scoring function) by 12–1000x. One downside of a limited scoring function is that it may lead to the selection of a suboptimal sequence alignment, relative to an affine gap scoring function as in the Smith-Waterman-Gotoh algorithm. There are also a large number of edit distance approximation algorithms that provide a reduction in time complexity (e.g., $m^{1.647}$ instead of m^2), but they suffer from providing overestimated edit distance [204–207].

There are other algorithms and hardware architectures that provide low alignment time by trading off accuracy. Darwin [36] builds a customized hardware architecture to speed up the alignment process, by dividing the DP matrix into overlapping submatrices and greedily processing each submatrix using systolic arrays. Darwin provides three orders of magnitude speedup compared to Edlib [132]. Greedily processing each submatrix reduces the number of calculated entries of the full DP matrix and hence reduces the memory footprint and algorithmic workload, but it leads to suboptimal alignment calculation [171]. Darwin claims that choosing a large submatrix size ($\geq 320 \times 320$) and ensuring sufficient overlap (≥ 128 entries) between adjacent submatrices may provide optimal alignment calculation for some datasets. GenASM [35] is a framework that uses bit-vector-based ASM and a similar strategy as Darwin to accelerate multiple steps of the genome analysis pipeline, and is designed to be implemented inside 3D-stacked memory. Through a combination of hardware–software co-design to unlock parallelism, and processing-in-memory to reduce data movement, GenASM achieves 111x/116x speedup over state-of-the-art software read mappers while reducing power consumption by 33x/37x.

There are other proposals that limit the number of calculated entries of the DP matrix based on one of two approaches: (1) using sparse DP or (2) using a greedy approach to maintain a high alignment score. Both approaches suffer from producing possibly suboptimal alignments [208,209]. The first approach uses the same sparse DP algorithm used for pre-alignment filtering but as an alignment step, as done in the exonerate tool [208]. This includes applying DP-based alignment algorithms only between every-two non-overlapping chains to quickly estimate the total number of edits. The second approach is employed in X-drop [209], which (1) avoids calculating entries (and their neighbors) whose alignment scores are more than below the highest score seen so far (where is a user-specified parameter), and (2) stops early when a high alignment score is not possible. The X-drop algorithm is guaranteed to find the optimal alignment between relatively-similar sequences for *only some* scoring functions [209]. A similar algorithm (known as Z-drop) makes KSW2 at least 2.6x faster than Parasail. A recent GPU implementation [210] of the X-drop algorithm is 3.1–120.4x faster than KSW2. A related approach is *adaptive banding* [211] (and its improved algorithm Block aligner [212]), where the band of calculated diagonals is shifted up or down depending on the highest score in the last calculated anti-diagonal.

8. Variant calling

The goal of variant calling (Fig. 1.5) is to find the differences between an individual or a group of individuals (i.e., population)

compared to a reference genome of a species. Calling the variants is an essential step in genome analysis because the attributes of an individual or a population (e.g., phenotypes or diseases) are determined from these variations. To determine these variants, there are several steps that need to be performed as outlined by tools such as GATK's best practices [213] and DeepVariant's workflow [214]. Variant calling usually iterates over the read mapping information to identify the variants such as SNPs, short indels, and SVs. Although there are several algorithms to find SNPs and short indels, the main idea of most of these tools is to find the locations in a genome where the reference genome and the sequencing reads have different bases. To find such regions, several mapped reads should consistently provide the same edit operation at the same location in a reference genome to call the variant with a high quality.

Calling the variants with high quality is essential for performing accurate downstream analysis (e.g., validating a genetic disease) [215]. There are several parameters that contribute to calling high quality variants such as high sequencing depth of coverage, highly accurate sequencing reads (i.e., Illumina and PacBio HiFi), long reads, accurate and deterministic read mappers. The sequencing depth of coverage refers to the average number of reads mapped to each location in the reference genome. This helps variant callers to better distinguish the genetic mutations from errors (both sequencing errors and read mapping artifacts [31]). It is also known that variant calling tools may also be nondeterministic such that running the same tool multiple times may result in different results [31]. Thus, it is essential for the community to provide the best practices to achieve high accuracy due to many parameters involved in high quality variant calling.

There are several efforts in the field to provide best practices when performing genomic analysis that includes variant calling. One of the efforts is to provide benchmarking studies to evaluate the accuracy of the variant calling output [216]. Such comparisons are usually done by comparing the output from a variant caller with a ground truth dataset (e.g., GiAB [217]). Another effort is to suggest the best pipeline of tools to achieve the best accuracy for variant calling [215]. The computational steps for variant calling differ from one variant caller to another as each variant caller focuses on detecting one or a few types of structural (larger than 50 bp) and large variations [218,219]. However, we can generally categorize the computational steps for variant calling into three key steps: 1) processing read mapping data, 2) variant classification, and 3) generating variant calls.

8.1. Processing read mapping data

The first step prepares read mapping output data (e.g., read bases, base quality, edit information, strand information) for variant calling. The output of read mapping (i.e., SAM file) has some irrelevant information that needs to be cleaned. This includes identifying and removing read duplicates (reads originated from the same subsequence of genome sequence) that can be a result of library preparation using PCR, as they do not lead to any useful information for variant calling. Most variant callers, including the best practices of DeepVariant and GATK [213,220], require converting the input SAM file into its compressed version, BAM file, and sorting the reported alignments in the BAM file by their coordinates (mapping locations) to facilitate processing overlapping alignments only once for a fast variant calling.

Some variant callers, such as GATK [221,222], rely on base quality scores that are reported by sequencing machines. Such variant callers require an additional step called *base quality score recalibration* (BQSR) for recalibrating the base quality scores to account for various sources of potential systematic (non-random) technical errors introduced during sequencing. In DeepVariant, processing read mapping data, called *make examples*, converts read mapping

information (read bases, base quality, edit information, strand information) into images. In DeepVariant, BQSR is not needed, sorting BAM file is required before performing make examples, and marking read duplicates is optional.

8.2. Variant classification

The second step of typical variant calling is to detect the actual genomic variations in the processed read mapping data and classify the variations into SNPs, short indels, and SVs. This step is called HaplotypeCaller in GATK and it calls the variants using three stages. First, it finds the alternate alleles from the BAM/SAM file. Second, it creates a local graph assembly that shows all alternate and reference alleles including both homozygous and heterozygous alternate loci. Third, it uses PairHMM with the graph to classify the probability of each allele at each locus to call variants. Probabilities determine whether a certain alternate loci is homozygous or heterozygous.

DeepVariant considers the task of identifying genetic variants as an image classification problem. DeepVariant directly applies a deep neural network based approach, convolutional neural network (CNN), to classify images that are extracted in make examples step to perform the detection and classification. Each image is represented as a multi-channel tensor, where each channel represents one information extracted from read mapping data. Though DeepVariant is not designed for calling structural variations [223], recent efforts show that deep neural network based approaches can detect structural variations [224,225].

8.3. Generating variant calls

The last step filters out the low quality variations and interprets the classification output as variant calls stored in VCF format.

We evaluate the performance of DeepVariant, a state-of-the-art variant caller for the three types of sequencing data (Table 4). DeepVariant uses three phases, make examples, call variants, post-process variants, that follow the three steps we describe above [226]. Based on Table 4, we make three key observations. 1) The first step of DeepVariant consumes about 50 % of the execution time of the second step, variant classification. 2) Using nearly 10x more bases of short read mapping data compared to that of accurate long reads leads to nearly the same number of variant calls (PASS) and half of the number of called variants (RefCALL). 3) Variant calling using ultra-long reads is computationally very expensive, which can be mainly because of the errors in the raw sequencing data, as we discuss in Section 3.4.

Given the high computational cost of variant calling, there are efforts focusing on identifying the computational bottlenecks in the best practices for variant calling and accelerating these bottlenecks to achieve high performance in variant calling [227,228]. It is observed that the first step of variant calling, processing read

mapping data, includes unnecessary several passes over the read mapping output data to perform different processing algorithms, which incurs significant IO overhead, excess memory accesses, and performance overhead [229]. elPrep [229] takes advantage of this observation and provides a multithreading processing using only a single pass over the read mapping output data to perform key processing steps such as sorting, duplicate marking, BQSR, and variant calling. It is claimed that elPrep provides 8-16x speedup compared to GATK (version 4) [230]. A similar approach is followed in [231] where it overlaps the execution of the first step with the second step of DeepVariant and enables parallel/distributed processing to accelerate DeepVariant by 30x using 8 GPUs. The first step of variant calling, processing read mapping data, can be separately accelerated by 2x using modern FPGAs as in [135]. Compared to the number of proposed software and hardware accelerators for read mapping, there are still only a limited number of hardware accelerators for variant calling. We believe that there is still a huge need for accelerating the state-of-the-art variant calling tools, such as DeepVariant.

9. Discussion and future opportunities

Despite more than three decades of attempts, bridging the performance gap between sequencing machines and computational analysis is still challenging. We summarize six main challenges below.

First, we need to accelerate the entire genome analysis process rather than its individual steps. Accelerating only a single step of genome analysis limits the overall achieved speedup according to Amdahl's Law. However, some of the computational steps included in genome analysis pipeline are also included in other genomics pipelines. For example, improving read mapping performance positively impacts almost all genomic analyses that use sequencing data [13,27,35,36,53]. The contribution of read mapping to the entire analysis pipeline varies depending on the application. For example, read mapping takes up to 1) 45 % of the execution time when discovering sequence variants in cancer genomics studies [232], and 2) 60 % of the execution time when profiling the taxonomy of a multi-species (i.e., metagenomic) sample [13]. Illumina and NVIDIA started following a more holistic approach, and they claim to accelerate genome analysis by more than 48x, mainly by using specialization and hardware/software co-design. Illumina has built an FPGA-based platform, called DRAGEN [233], that accelerates all steps of genome analysis, including read mapping and variant calling. DRAGEN reduces the overall analysis time from 32 CPU hours to only 37 min [234]. NVIDIA has built Parabricks, a software suite accelerated using the company's latest GPUs. Parabricks [235] can analyze whole human genomes at 30x coverage in about 45 min.

Second, we need to reduce the high amount of data movement that takes place during genome analysis. Moving data (1) between compute units and main memory, (2) between multiple hardware

Table 4

Evaluation analysis of variant calling, using DeepVariant tool, using read mapping results of three different types of sequencing data, short reads, ultra-long reads, and accurate long reads.

	Short reads	Ultra-long reads	Accurate long reads
Variant calling tool	DeepVariant	PEPPER + DeepVariant	DeepVariant
Version	1.3.0	0.8	1.3.0
Total number of bases in input SAM file	250,103,434,512	56,958,985,752	23,944,354,059
Phase 1 (make examples) CPU Time (sec)	250,359	1,136,356	230,066
Phase 2 (call variants) CPU Time (sec)	473,962	3,480,419	549,272
Phase 3 (post-process variants) CPU Time (sec)	2,193	2,765	6,201
Total Run Time (sec)	746,514	4,619,540	785,539
Number of Called Variants (PASS)	4,644,980	6,054,168	4,589,024
Number of Called Variants (RefCall)	1,313,292	6,722,265	2,603,968
Variant Calling Throughput (Number of called variants / sec)	7.9 variants/sec	2.77 variants/sec	9.2 variants/sec

accelerators, and (3) between the sequencing machine and the computer performing the analysis incurs high costs in terms of execution time and energy. These costs are a significant barrier to enabling efficient analysis that can keep up with sequencing technologies, and some recent works try to tackle this problem [33,34,52]. DRAGEN reduces data movement between the sequencing machine and the computer performing analysis by adding specialized hardware support inside the sequencing machine for data compression. However, this still requires movement of compressed data. GenStore [53] mitigates data movement from the storage devices to the rest of the system (processors and main memory) by processing more than 80% of the input read set inside the storage device.

Third, we need to build more specialized hardware accelerators that are mainly developed for genomics. Computer programs are currently widely used for analyzing genomic data, which limits their scaling capability and efficiency to handle population-level analyses. We are witnessing a paradigm shift to near-data computing with more specialized hardware accelerators for other key applications such as self-driving cars [236], and artificial intelligence with the largest chip ever [237]. This already fuelled huge interest in genomics especially from large companies, such as NVIDIA, which introduces GPU H100 boards that are equipped with HBM3 and hardware support for building and calculating DP matrix for sequence alignment. UPMEM also shows significant benefits for using their PIM-capable memory devices for genome analysis [238]. We envision that performing genome analysis inside the sequencing machine itself using emerging technologies (e.g., PIM-enabled systems) can significantly improve efficiency by eliminating sequencer-to-computer movement, and embedding a single specialized chip for genome analysis within a portable sequencing device can potentially be a key enabler for new applications of genome sequencing (e.g., rapid surveillance of diseases such as Ebola [7] and COVID-19 [6,239], near-patient testing, bringing precision medicine to remote locations). Unfortunately, efforts in this direction remain very limited.

Fourth, an emerging problem with using a single reference genome for an entire species is the reference genome bias. The use of a single reference genome can bias the mapping process and downstream analysis towards the DNA composition and variations present in the reference organism due to population-specific genetic variations, individual's genetic variations, and sequencing errors [240,241]. An emerging technique to overcome reference bias is the use of graph-based representations of a species' genome, known as genome graphs [242]. A genome graph represents the reference genome and known genetic variations in the population as a graph-based data structure. Genome graphs are growing in popularity for genome analysis, which requires modifying existing tools or introducing new tools and hardware accelerators for supporting genome graphs instead of linear representations of reference genomes. Hardware acceleration is demonstrated to greatly benefit sequence mapping to genome graphs. SeGraM is the first hardware acceleration framework for sequence-to-graph mapping and alignment, where it provides an order of magnitude faster and more energy efficient performance compared to software sequence-to-graph mapping tools [243]. A new direction to alleviate the computation overhead of using different reference genomes is to update the existing results of one step of the genome analysis pipeline for the new reference genome without re-running the step's algorithm again. The efforts in this direction are still limited to only read mapping [62,63,244].

Fifth, we need to develop flexible hardware architectures that do not conservatively limit the range of supported parameter values at design time. Commonly-used read mappers (e.g., minimap2) have different input parameters, each of which has a wide range of input values. For example, the edit distance threshold is typically user defined and can be very high (15–20% of the read length) for recent

long reads. A configurable scoring function is another example, as it determines the number of bits needed to store each entry of the DP matrix (e.g., DRAGEN imposes a restriction on the maximum frequency of seed occurrence). Due to rapid changes in sequencing technologies (e.g., high sequencing error rate and longer read lengths) [97,99], these design restrictions can quickly make specialized hardware obsolete. Thus, read mappers need to adapt their algorithms and their hardware architectures to be modular and scalable so that they can be implemented for any sequence length and edit distance threshold based on the sequencing technology.

Sixth, we need to adapt existing genomic data formats for hardware accelerators or develop more efficient file formats. Most sequencing data is stored in the FASTQ/FASTA format, where each base takes a single byte (8 bits) of memory. This encoding is inefficient, as only 2 bits (3 bits when the ambiguous base, N, is included) are needed to encode each DNA base. The sequencing machine converts sequenced bases into FASTQ/FASTA format, and hardware accelerators convert the file contents into unique (for each accelerator) compact binary representations for efficient processing. This process that requires multiple format conversions wastes time. For example, only 43% of the sequence alignment time in BWA-MEM2 [159] is spent on calculating the DP matrix, while 33% of the sequence alignment time is spent on pre-processing the input sequences for loading into SIMD registers, as provided in [159]. To address this inefficiency, we need to widely adopt efficient hardware-friendly formats, such as UCSC's 2bit format (<https://genome.ucsc.edu/goldenPath/help/twoBit>), to maximize the benefits of hardware accelerators and reduce resource utilization. We are not aware of any recent read mapper that uses such formats. Basecalling can also benefit from data formats that support parallel file accesses, such as the recent file formats called SLOW5 and BLOW5 [245]. Other computational steps can benefit from efficient file formats, such as k-mer analyses with the new format called KFF [246].

Looking into the late future, even if accurately sequencing the entire genome as a single string might be possible, we believe that most of the tools and hardware accelerators involved in the intelligent genome analysis pipeline will continue to remain a crucial component in analyzing and comparing the sequencing data. For example, we still need to quickly and efficiently compare complete genomes together for inferring variations and identifying metagenomic taxonomy profiles. The acceleration efforts we highlight in this work represent state-of-the-art efforts to reduce current bottlenecks in the genome analysis pipeline. We hope that these efforts and the challenges we discuss provide a foundation for future work in making genome analysis faster, more accurate, privacy-preserving, more energy-efficient, and cost-effective; *simply more intelligent*.

CRediT authorship contribution statement

Mohammed Alser: Conceptualization, Methodology, Validation, Investigation, Writing – original draft, Project administration. **Joel Lindegger:** Writing – original draft, Validation. **Can Firtina:** Writing – original draft, Validation, Investigation. **Nour Almadhoun:** Validation, Visualization. **Haiyu Mao:** Writing – original draft. **Gagandeep Singh:** Writing – original draft, Investigation. **Juan Gomez-Luna:** Writing – review & editing. **Onur Mutlu:** Writing – review & editing, Funding acquisition, Conceptualization.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the SAFARI group members for feedback and the stimulating intellectual environment. We acknowledge the generous gifts and support provided by our industrial partners: Google, Huawei, Intel, Microsoft, VMware, and the Semiconductor Research Corporation. M.A. dedicates this paper to the memory of his father, who passed away on 9th March 2022.

References

- [1] Ginsburg GS, Phillips KA. Precision medicine: from science to value. *Health Aff* 2018;37:694–701.
- [2] Farnaes L, Hildreth A, Sweeney NM, Clark MM, Chowdhury S, Nahas S, et al. Rapid whole-genome sequencing decreases infant morbidity and cost of hospitalization. *NPJ Genom Med* 2018;3:10.
- [3] Clark MM, Hildreth A, Batalov S, Ding Y, Chowdhury S, Watkins K, et al. Diagnosis of genetic diseases in seriously ill children by rapid whole-genome sequencing and automated phenotyping and interpretation. *Sci Transl Med* 2019;11. <https://doi.org/10.1126/scitranslmed.aat6177>.
- [4] Sweeney NM, Nahas SA, Chowdhury S, Batalov S, Clark M, Caylor S, et al. Rapid whole genome sequencing impacts care and resource utilization in infants with congenital heart disease. *NPJ Genom Med* 2021;6:29.
- [5] Ginsburg GS, Willard HF. Genomic and personalized medicine: foundations and applications. *Transl Res* 2009;154:277–87.
- [6] Bloom JS, Sathe L, Munugala C, Jones EM, Gasperini M, Lubock NB, et al. Massively scaled-up testing for SARS-CoV-2 RNA via next-generation sequencing of pooled and barcoded nasal and saliva samples. *Nat Biomed Eng* 2021;5:657–65.
- [7] Quick J, Loman NJ, Duraffour S, Simpson JT, Severi E, Cowley L, et al. Real-time, portable genome sequencing for Ebola surveillance. *Nature* 2016;530:228–32.
- [8] Yelagandula R, Bykov A, Vogt A, Heinen R, Özkan E, Strobl MM, et al. Multiplexed detection of SARS-CoV-2 and other respiratory infections in high throughput by SARSeq. *Nat Commun* 2021;12:3132.
- [9] Le VTM, Diep BA. Selected insights from application of whole-genome sequencing for outbreak investigations. *Curr Opin Crit Care* 2013;19:432–9.
- [10] Nikolayevskiy V, Kranzer K, Niemann S, Drobniowski F. Whole genome sequencing of Mycobacterium tuberculosis for detection of recent transmission and tracing outbreaks: A systematic review. *Tuberculosis* 2016;98:77–85.
- [11] Danko D, Bezdán D, Afshin EE, Ahsanuddin S, Bhattacharya C, Butler DJ, et al. International MetaSUB Consortium, A global metagenomic map of urban microbiomes and antimicrobial resistance. *Cell* 2021;184:3376–3393.e17.
- [12] Meyer F, Fritz A, Deng ZL, Koslicki D, Gurevich A. Critical Assessment of Metagenome Interpretation—the second round of challenges. *BioRxiv* 2021. <https://doi.org/10.1101/2021.07.12.451567.abstract>.
- [13] LaPierre N, Alser M, Eskin E, Koslicki D, Mangul S. Metalign: efficient alignment-based metagenomic profiling via containment min hash. *Genome Biol* 2020;21:242.
- [14] LaPierre N, Mangul S, Alser M, Mandric I, Wu NC, Koslicki D, et al. Microbial Community Profiling method for detecting viral and fungal organisms in metagenomic samples. *bioRxiv* 2018;. <https://doi.org/10.1101/243188>.
- [15] Meyer F, Fritz A, Deng Z-L, Koslicki D, Lesker TR, Gurevich A, et al. Critical Assessment of Metagenome Interpretation: the second round of challenges. *Nat Methods* 2022. <https://doi.org/10.1038/s41592-022-01431-4>.
- [16] Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC, Baldwin J, et al. International Human Genome Sequencing Consortium, Initial sequencing and analysis of the human genome. *Nature* 2001;409:860–921.
- [17] Reuter JA, Spacek DV, Snyder MP. High-throughput sequencing technologies. *Mol Cell* 2015;58:586–97.
- [18] Alser M, Rotman J, Deshpande D, Taraszka K, Shi H, Baykal PI, et al. Technology dictates algorithms: recent developments in read alignment. *Genome Biol* 2021;22:249.
- [19] Mangul S, Martin LS, Hill BL, Lam A-K-M, Distler MG, Zelikovsky A, et al. Systematic benchmarking of omics computational tools. *Nat Commun* 2019;10:1393.
- [20] Misra BB, Langefeld CD, Olivier M, Cox LA. Integrated omics: tools, advances, and future approaches. *J Mol Endocrinol* 2018. <https://doi.org/10.1530/JME-18-0055>.
- [21] Markowitz F. All biology is computational biology. *PLoS Biol* 2017;15:e2002050.
- [22] Sanger F. The free amino groups of insulin; 1945.
- [23] Shendure J, Balasubramanian S, Church GM, Gilbert W, Rogers J, Schloss JA, et al. DNA sequencing at 40: past, present and future. *Nature* 2017;550:345–53.
- [24] Nielsen R, Paul JS, Albrechtsen A, Song YS. Genotype and SNP calling from next-generation sequencing data. *Nat Rev Genet* 2011;12:443–51.
- [25] Ho SS, Urban AE, Mills RE. Structural variation in the sequencing era. *Nat Rev Genet* 2020;21:171–89.
- [26] Jacquemont S, Reymond A, Zufferey F, Harewood L, Walters RG, Kutalik Z, et al. Mirror extreme BMI phenotypes associated with gene dosage at the chromosome 16p11.2 locus. *Nature* 2011;478:97–102.
- [27] Alser M, Bingöl Z, Cali DS, Kim J, Ghose S, Alkan C, et al. Accelerating genome analysis: a primer on an ongoing journey. *IEEE Micro* 2020;40:65–75. <https://doi.org/10.1109/mm.2020.3013728>.
- [28] Friedman JM, Bombard Y, Cornel MC, Fernandez CV, Junker AK, Plon SE, et al. Paediatric Task Team of the Global Alliance for Genomics and Health Regulatory and Ethics Work Stream, Genome-wide sequencing in acutely ill infants: genomic medicine's critical application? *Genet Med* 2019;21:498–504.
- [29] Marshall CR, Chowdhury S, Taft RJ, Lebo MS, Buchan JG, Harrison SM, et al. Medical Genome Initiative, Best practices for the analytical validation of clinical whole-genome sequencing intended for the diagnosis of germline disease. *NPJ Genom Med* 2020;5:47.
- [30] Baruzzo G, Hayer KE, Kim EJ, Di Camillo B, FitzGerald GA, Grant GR. Simulation-based comprehensive benchmarking of RNA-seq aligners. *Nat Methods* 2017;14:135–9.
- [31] Firtina C, Alkan C. On genomic repeats and reproducibility. *Bioinformatics* 2016;32:2243–7.
- [32] Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, et al. Big data: astronomical or genomic? *PLoS Biol* 2015;13:e1002195.
- [33] Mutlu O, Ghose S, Gómez-Luna J, Ausavarungnirun R. Processing data where it makes sense: Enabling in-memory computation. *Microprocess Microsyst* 2019;67:28–41.
- [34] S. Ghose, A. Boroumand, J.S. Kim, J. Gómez-Luna, O. Mutlu, Processing-in-memory: A workload-driven perspective, *IBM J. Res. Dev.* 63 (2019) 3:1–3:19.
- [35] Cali DS, Kalsi GS, Bingöl Z, Firtina C, Subramanian L, Kim JS, et al. A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis. In: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). p. 951–66.
- [36] Y. Turakhia, G. Bejerano, W.J. Dally, Darwin, Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. (2018). 10.1145/3173162.3173193.
- [37] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun, A Modern Primer on Processing in Memory, *arXiv [cs.AR]*. (2020). <http://arxiv.org/abs/2012.03112>.
- [38] Boroumand A, Ghose S, Kim Y, Ausavarungnirun R, Shiu E, Thakur R, et al. Google workloads for consumer devices: mitigating data movement bottlenecks. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: Association for Computing Machinery; 2018. p. 316–31.
- [39] Boroumand A, Ghose S, Akin B, Narayanaswami R, Oliveira GF, Ma X, et al. Google neural network models for edge devices: analyzing and mitigating machine learning inference bottlenecks. In: 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). p. 159–72.
- [40] Horowitz M. 1.1 computing's energy problem (and what we can do about it), in: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE, 2014; pp. 10–14.
- [41] Oliveira GF, Gómez-Luna J, Orosa L, Ghose S, Vijaykumar N, Fernandez I, Sadosadati M, Mutlu O., DAMOV: A new methodology and benchmark suite for evaluating data movement bottlenecks, *IEEE Access*. 9 (undefined 2021) 134457–134502.
- [42] Mutlu O, Subramanian L. Research problems and opportunities in memory systems. *Supercomp Front Innov* 2014;1:19–55.
- [43] Mutlu O. Memory scaling: A systems architecture perspective, in: 2013 5th IEEE International Memory Workshop, 2013; pp. 21–25.
- [44] Langmead B, Nellore A. Cloud computing for genomic data analysis and collaboration. *Nat Rev Genet* 2018;19:325.
- [45] Almadhoun N, Ayday E, Ulusoy Ö. Differential privacy under dependent tuples—the case of genomic privacy. *Bioinformatics* 2019;36:1696–703.
- [46] Almadhoun N, Ayday E, Ulusoy Ö. Inference attacks against differentially private query results from genomic datasets including dependent tuples. *Bioinformatics* 2020;36:i136–45.
- [47] Alser M, Almadhoun N, Nouri A, Alkan C, Ayday E. Can you really anonymize the donors of genomic data in today's digital world? Cham: Springer International Publishing; 2016. p. 237–44.
- [48] Alser NA, Ulusoy O, Ayday E, Mutlu O. GenShare: Sharing accurate differentially-private statistics for genomic datasets with dependent tuples, *arXiv [q-bio.GN]*. (2021). <http://arxiv.org/abs/2112.15109>.
- [49] Alser NA, Kale G, Mutlu O, Tastan O, Ayday E. Near-Optimal Privacy-Utility Tradeoff in Genomic Studies Using Selective SNP Hiding, *arXiv [cs.CR]*. (2021). <http://arxiv.org/abs/2106.05211>.
- [50] Ghiasi NM, Park J, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D.S. Cali, C. Firtina, H. Mao, N.A. Alser, R. Ausavarungnirun, N. Vijaykumar, M. Alser, O. Mutlu, GenStore: A high-performance and energy-efficient in-storage computing system for genome sequence analysis, *arXiv [cs.AR]*. (2022). <http://arxiv.org/abs/2202.10400>.
- [51] Alser M, Shahroodi T, Gómez-Luna J, Alkan C, Mutlu O. SneakySnake: A fast and accurate universal genome pre-alignment filter for CPUs, GPUs, and FPGAs. *Bioinformatics* 2020. <https://doi.org/10.1093/bioinformatics/btaa1015>.
- [52] Kim JS, Senol Cali D, Xin H, Lee D, Ghose S, Alser M, et al. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC Genomics* 2018;19:89.

- [53] Mansouri Ghiasi N, Park J, Mustafa H, Kim J, Olgun A, Gollwitzer A, et al. GenStore: a high-performance in-storage processing system for genome sequence analysis. In: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: Association for Computing Machinery; 2022. p. 635–54.
- [54] Singh G, Alser M, Cali DS, Diamantopoulos D, Gómez-Luna J, Corporaal H, et al. FPGA-based near-memory acceleration of modern data-intensive applications. *IEEE Micro* 2021;41:39–48.
- [55] O. Mutlu, Intelligent Architectures for Intelligent Machines, in: 2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), 2020; pp. 1–4.
- [56] M. Alser, Z. Bingöl, D.S. Cali, J. Kim, S. Ghose, C. Alkan, O. Mutlu, Accelerating Genome Analysis: A Primer on an Ongoing Journey, arXiv [cs.AR]. (2020). <http://arxiv.org/abs/2008.00961>.
- [57] Sanger F, Nicklen S, Coulson AR. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A* 1977;74:5463–7.
- [58] The Nobel Prize in Chemistry 1958, NobelPrize.org. (n.d.). <https://www.nobelprize.org/prizes/chemistry/1958/sanger/lecture/> (accessed March 2, 2022).
- [59] Maxam AM, Gilbert W. A new method for sequencing DNA. *Proc Natl Acad Sci U S A* 1977;74:560–4.
- [60] White House press release, (n.d.). https://web.ornl.gov/sci/techresources/Human_Genome/project/clinton1.shtml (accessed March 2, 2022).
- [61] Introduction to Patches, (n.d.). <https://www.ncbi.nlm.nih.gov/grc/help/patches/> (accessed March 2, 2022).
- [62] Kim JS, Firtina C, Cali DS, M. Alser, N. Hajinazar, C. Alkan, O. Mutlu, AirLift: A Fast and Comprehensive Technique for Translating Alignments between Reference Genomes, arXiv Preprint arXiv:1912.08735. (2019). https://www.researchgate.net/profile/Damla-Senol-Cali/publication/338036201_AirLift_A_Fast_and_Comprehensive_Technique_for_Translating_Alignments_between_Reference_Genomes/links/5f7382e692851c14bc9ff96e/AirLift-A-Fast-and-Comprehensive-Technique-for-Translating-Alignments-between-Reference-Genomes.pdf.
- [63] Mun T, Chen N-C, Langmead B. LevioSAM: Fast lift-over of variant-aware reference alignments. *Bioinformatics* 2021. <https://doi.org/10.1093/bioinformatics/btab396>.
- [64] Nurk S, Koren S, Rhie A, M. Rautiainen, A.V. Bzikadze, The complete sequence of a human genome, bioRxiv. (2021). <https://www.biorxiv.org/content/10.1101/2021.05.26.445798v1.abstract>.
- [65] Syed F, Grunenwald H, Caruccio N. Next-generation sequencing library preparation: simultaneous fragmentation and tagging using in vitro transposition. *Nat Methods* 2009;6:i–ii. <https://doi.org/10.1038/nmeth.f.272>.
- [66] van Dijk EL, Jaszczyszyn Y, Thermes C. Library preparation methods for next-generation sequencing: tone down the bias. *Exp Cell Res* 2014;322:12–20.
- [67] Kelley DR, Schatz MC, Salzberg SL. Quake: quality-aware detection and correction of sequencing errors. *Genome Biol* 2010;11:R116.
- [68] Erlich HA, Gelfand D, Sninsky JJ. Recent advances in the polymerase chain reaction. *Science* 1991;252:1643–51.
- [69] Alser M, Waymost S, Ayyala R, B. Lawlor, R.J. Abdill, N. Rajkumar, N. LaPierre, et al., Packaging, containerization, and virtualization of computational omics methods: Advances, challenges, and opportunities, arXiv [q-bio.GN]. (2022). <http://arxiv.org/abs/2203.16261>.
- [70] Home - SRA - NCBI, (n.d.). <https://www.ncbi.nlm.nih.gov/sra> (accessed March 27, 2022).
- [71] Overview: Main : Sequence read archive : NCBI/NLM/NIH, (n.d.). <https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi> (accessed March 3, 2022).
- [72] EMBL-EBI, ENA browser, (n.d.). <https://www.ebi.ac.uk/ena> (accessed March 27, 2022).
- [73] RefSeq: NCBI Reference Sequence Database, (n.d.). <https://www.ncbi.nlm.nih.gov/refseq> (accessed March 27, 2022).
- [74] Nasko DJ, Koren S, Phillippy AM, Treangen TJ. RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biol* 2018;19:165.
- [75] Escalona M, Rocha S, Posada D. A comparison of tools for the simulation of genomic next-generation sequencing data. *Nat Rev Genet* 2016;17:459–69.
- [76] Ono Y, Asai K, Hamada M. PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics* 2021;37:589–95.
- [77] Yang C, Chu J, Warren RL, Birol I. NanoSim: nanopore sequence read simulator based on statistical characterization. *GigaScience* 2017;6. <https://doi.org/10.1093/gigascience/gix010>.
- [78] Holtgrewe M. Mason: a read simulator for second generation sequencing data, (2010). <https://refubium.fu-berlin.de/handle/fub188/18686>.
- [79] Portik DM, Titus Brown C, Tessa Pierce-Ward N. Evaluation of taxonomic profiling methods for long-read shotgun metagenomic sequencing datasets, bioRxiv. (2022). 2022.01.31.478527. 10.1101/2022.01.31.478527.
- [80] Huang W, Li L, Myers JR, Marth GT. ART: a next-generation sequencing read simulator. *Bioinformatics* 2012;28:593–4.
- [81] Schmeing S, Robinson MD. ReSeq simulates realistic Illumina high-throughput sequencing data. *Genome Biol* 2021;22:67.
- [82] Ono Y, Asai K, Hamada M. PBSIM: PacBio reads simulator—toward accurate genome assembly. *Bioinformatics* 2012;29:119–21.
- [83] Castro-Wallace SL, Chiu CY, John KK, S.E. Stahl, K.H. Rubins, A.B.R. McIntyre, J. P. Dworkin, M.L. Lupisella, D.J. Smith, D.J. Botkin, T.A. Stephenson, S. Juul, D.J. Turner, F. Izquierdo, S. Federman, D. Stryke, S. Somasekar, N. Alexander, G. Yu, C.E. Mason, A.S. Burton, Nanopore DNA Sequencing and Genome Assembly on the International Space Station, (n.d.). 10.1101/077651.
- [84] van Dijk EL, Auger H, Jaszczyszyn Y, Thermes C. Ten years of next-generation sequencing technology. *Trends Genet* 2014;30:418–26.
- [85] Quail MA, Kozarewa I, Smith F, Scally A, Stephens PJ, Durbin R, et al. A large genome center's improvements to the Illumina sequencing system. *Nat Methods* 2008;5:1005–10.
- [86] Singular Genomics, Singular Genomics. (2020). <https://singulargenomics.com> (accessed March 4, 2022).
- [87] Glenn TC. Field guide to next-generation DNA sequencers. *Mol Ecol Resour* 2011;11:759–69.
- [88] NGS vs. Sanger sequencing, (n.d.). <https://emea.illumina.com/science/technology/next-generation-sequencing/ngs-vs-sanger-sequencing.html> (accessed March 4, 2022).
- [89] Mardis ER. DNA sequencing technologies: 2006–2016. *Nat Protoc* 2017;12:213–8.
- [90] Medžiūnė J, Kapustina Ž, Žeimytė S, Jakubovska J, Sindikevičienė R, Čikotienė I, et al. Advanced preparation of fragment libraries enabled by oligonucleotide-modified 2',3'-dideoxynucleotides. *Commun Chem* 2022;5:1–8.
- [91] 2-channel SBS technology, (n.d.). <https://emea.illumina.com/science/technology/next-generation-sequencing/sequencing-technology/2-channel-sbs.html> (accessed March 5, 2022).
- [92] Run time estimates for each sequencing step on Illumina sequencing platforms, (n.d.). <https://emea.support.illumina.com/bulletins/2017/02/run-time-estimates-for-each-sequencing-step-on-illumina-sequenci.html> (accessed March 6, 2022).
- [93] Company history, Oxford Nanopore Technologies. (2021). <https://nanoporetech.com/about-us/history> (accessed March 7, 2022).
- [94] Wang Y, Zhao Y, Bollas A, Wang Y, Au KF. Nanopore sequencing technology, bioinformatics and applications. *Nat Biotechnol* 2021;39:1348–65.
- [95] Huang Y-T, Liu P-Y, Shih P-W. Homopolish: a method for the removal of systematic errors in nanopore sequencing by homologous polishing. *Genome Biol* 2021;22:95.
- [96] Amarasinghe SL, Su S, Dong X, Zappia L, Ritchie ME, Gouil Q. Opportunities and challenges in long-read sequencing data analysis. *Genome Biol* 2020;21:30.
- [97] Firtina C, Kim JS, Alser M, Cali DS, Ercument Cicek A, Alkan C, et al. Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm. *Bioinformatics* 2020;36:3669–79. <https://doi.org/10.1093/bioinformatics/btaa179>.
- [98] Logsdon GA, Vollger MR, Eichler EE. Long-read human genome sequencing and its applications. *Nat Rev Genet* 2020;21:597–614.
- [99] Senol Cali D, Kim JS, Ghose S, Alkan C, Mutlu O. Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Brief Bioinform* 2019;20:1542–59.
- [100] Suzuki Y. Advent of a new sequencing era: long-read and on-site sequencing. *J Hum Genet* 2020;65:1.
- [101] Hon T, Mars K, Young G, Tsai Y-C, Karalius JW, Landolin JM, et al. Highly accurate long-read HiFi sequencing data for five complex genomes. *Sci Data* 2020;7:399.
- [102] Wenger AM, Peluso P, Rowell WJ, Chang P-C, Hall RJ, Concepcion GT, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nat Biotechnol* 2019;37:1155–62.
- [103] Chaisson MJP, Wilson RK, Eichler EE. Genetic variation and the de novo assembly of human genomes. *Nat Rev Genet* 2015;16:627–40.
- [104] Jain M, Koren S, Miga KH, Quick J, Rand AC, Sasani TA, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat Biotechnol* 2018;36:338–45.
- [105] Gong L, Wong C-H, Idol J, Ngan CY, Wei C-L. Ultra-long read sequencing for whole genomic DNA analysis. *J Vis Exp* 2019. <https://doi.org/10.3791/58954>.
- [106] Deschamps S, Zhang Y, Llaca V, Ye L, Sanyal A, King M, et al. A chromosome-scale assembly of the sorghum genome using nanopore sequencing and optical mapping. *Nat Commun* 2018;9:4844.
- [107] Koren S, Walenz BP, Berlin K, Miller JR, Bergman NH, Phillippy AM. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res* 2017;27:722–36.
- [108] Chen Y, Nie F, Xie S-Q, Zheng Y-F, Dai Q, Bray T, et al. Efficient assembly of nanopore reads via highly accurate and intact error correction. *Nat Commun* 2021;12. <https://doi.org/10.1038/s41467-020-20236-7>.
- [109] Gehrig JL, Portik DM, Driscoll MD, Jackson E, Chakraborty S, Gratalo D, et al. Finding the right fit: evaluation of short-read and long-read sequencing approaches to maximize the utility of clinical microbiome data. *Microb Genom* 2022;8. <https://doi.org/10.1099/mgen.0.000794>.
- [110] High performance long read assay enables contiguous data up to 10Kb on existing illumina platforms, (n.d.). <https://www.illumina.com/science/genomics-research/articles/infinity-high-performance-long-read-assay.html> (accessed April 5, 2022).
- [111] Benton M. Guppy GPU benchmarking (nanopore basecalling), (n.d.). https://esr-nz.github.io/gpu_basecalling_testing/gpu_benchmarking.html (accessed March 3, 2022).
- [112] Cacho A, Smirnova E, Huzurbazar S, Cui X. A Comparison of base-calling algorithms for illumina sequencing technology. *Brief Bioinform* 2016;17:786–95.

- [113] Lindner MS, Strauch B, Schulze JM, Tausch S, Dabrowski PW, Nitsche A, et al. HiLive – real-time mapping of illumina reads while sequencing. *Bioinformatics* 2016;btw659. <https://doi.org/10.1093/bioinformatics/btw659>.
- [114] Performance, CCS Docs. (n.d.). <https://ccs.how/faq/performance> (accessed April 5, 2022).
- [115] Rang FJ, Kloosterman WP, de Ridder J. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. *Genome Biol* 2018;19:90.
- [116] Wick RR, Judd LM, Holt KE. Performance of neural network basecalling tools for Oxford Nanopore sequencing. *Genome Biol* 2019;20:129.
- [117] Lou Q, Janga SC, Jiang L. Helix: Algorithm/Architecture Co-design for Accelerating Nanopore Genome Base-calling. In: Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques. New York, NY, USA: Association for Computing Machinery; 2020. p. 293–304.
- [118] Ferreira JD, Falcao G, Gómez-Luna J, M. Alser, L. Orosa, M. Sadrosadati, J.S. Kim, G.F. Oliveira, T. Shahroodi, A. Nori, O. Mutlu, PLUTO: Enabling massively parallel computation in DRAM via lookup tables, arXiv [cs.AR]. (2021). <http://arxiv.org/abs/2104.07699>.
- [119] Hajmazar N, Oliveira GF, Gregorio S, Ferreira JD, Ghiasi NM, Patel M, et al. SIMDGRAM: a framework for bit-serial SIMD processing using DRAM. In: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: Association for Computing Machinery; 2021. p. 329–45.
- [120] Fernandez I, Quisilant R, Gutiérrez E, Plata O, Giannoula C, Alser M, et al. A near-data processing accelerator for time series analysis. In: 2020 IEEE 38th International Conference on Computer Design (ICCD). p. 120–9.
- [121] Schuiki F, Schaffner M, Gürkaynak FK, Benini L. A scalable near-memory architecture for training deep neural networks on large in-memory datasets. *IEEE Trans Comput* 2019;68:484–97.
- [122] Xu Z, Mai Y, Liu D, He W, Lin X, Xu C, et al. Fast-bonito: A faster deep learning based basecaller for nanopore sequencing. *Artificial Intelligence Life Sci* 2021;1:100011.
- [123] Wan YK, Hendra C, Pratanwanich PN, Göke J. Beyond sequencing: machine learning algorithms extract biology hidden in Nanopore signal data. *Trends Genet* 2022;38:246–57.
- [124] Gamaarachchi H, Lam CW, Jayatilaka G, Samarakoon H, Simpson JT, Smith MA, et al. GPU accelerated adaptive banded event alignment for rapid comparative nanopore signal analysis. *BMC Bioinf* 2020;21:343.
- [125] Loose M, Malla S, Stout M. Real-time selective sequencing using nanopore technology. *Nat Methods* 2016;13:751–4.
- [126] Dunn T, Sadasivan H, Wadden J, Goliya K, Chen K-Y, Blaauw D, et al. An Accelerator for Portable Virus Detection. In: MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. New York, NY, USA: Association for Computing Machinery; 2021. p. 535–49.
- [127] Kovaka S, Fan Y, Ni B, Timp W, Schatz MC. Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. *Nat Biotechnol* 2021;39:431–41.
- [128] Zhang H, Li H, Jain C, Cheng H, Au KF, Li H, et al. Real-time mapping of nanopore raw signals. *Bioinformatics* 2021;37:i477–83.
- [129] Using Dynamic Time Warping to Find Patterns in Time Series. (n.d.). <https://www.aaai.org/Library/Workshops/1994/ws94-03-031.php> (accessed April 5, 2022).
- [130] How does CCS work, CCS Docs. (n.d.). <https://ccs.how/how-does-ccs-work.html> (accessed March 25, 2022).
- [131] Li H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 2018;34:3094–100.
- [132] Šošić M, Šikić M. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics* 2017;33:1394–5.
- [133] Trivedi UH, CĀzard T, Bridgett S, Montazam A, Nichols J, Blaxter M, Gharbi K. Quality control of next-generation sequencing data without a reference. *Front Genet* 2014;5. <https://doi.org/10.3389/fgene.2014.00111>.
- [134] Picard, (n.d.). <https://broadinstitute.github.io/picard> (accessed March 27, 2022).
- [135] Ham TJ, Bruns-Smith D, Sweeney B, Lee Y, Seo SH, Gyeong Song U, et al. Genesis: A Hardware Acceleration Framework for Genomic Data Analysis. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). <https://doi.org/10.1109/isca45697.2020.00031>.
- [136] Hebert PDN, Gregory TR. The promise of DNA barcoding for taxonomy. *Syst Biol* 2005;54:852–9.
- [137] Baccaro A, Steck A-L, Marx A. Barcoded nucleotides. *Angew Chem Int Ed Engl* 2012;51:254–7.
- [138] Andrews S. Others, FastQC: a quality control tool for high throughput sequence data, (2010).
- [139] Fukasawa Y, Ermini L, Wang H, Carty K, Cheung M-S. LongQC, a quality control tool for third generation sequencing long. *Read Data* 2020;G3 (10):1193–6.
- [140] Yin Z, Zhang H, Liu M, Zhang W, Song H, Lan H, et al. RabbitQC: high-speed scalable quality control for sequencing data. *Bioinformatics* 2021;37: 573–4.
- [141] Alser M, Hassan H, Kumar A, Mutlu O, Alkan C. Shouji: a fast and efficient pre-alignment filter for sequence alignment. *Bioinformatics* 2019;35:4255–63.
- [142] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. 1000 genome project data processing subgroup, the sequence alignment/map format and SAMtools. *Bioinformatics* 2009;25:2078–9.
- [143] Backurs A, Indyk P. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false), in: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, NY, USA, 2015; pp. 51–58.
- [144] Xin H, Nahar S, Zhu R, Emmons J, Pekhimenko G, Kingsford C, et al. Optimal seed solver: optimizing seed selection in read mapping. *Bioinformatics* 2016;32:1632–42.
- [145] Firtina C, Park J, Kim JS, M. Alser, D.S. Cali, T. Shahroodi, N.M. Ghiasi, G. Singh, K. Kanellopoulos, C. Alkan, O. Mutlu, BLEND: A Fast, Memory-Efficient, and Accurate Mechanism to Find Fuzzy Seed Matches, arXiv [q-bio.GN]. (2021). <http://arxiv.org/abs/2112.08687>.
- [146] Schleimer S, Wilkerson DS. A. Aiken, Winnowing: local algorithms for document fingerprinting, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2003; pp. 76–85.
- [147] Jain C, Rhie A, Zhang H, Chu C, Walenz BP, Koren S, et al. Weighted minimizer sampling improves long read mapping. *Bioinformatics* 2020;36:i111–8.
- [148] Xin H, Lee D, Hormozdiari F, Yedkar S, Mutlu O, Alkan C. Accelerating read mapping with FastHASH. *BMC Genomics* 2013;14(Suppl 1):S13.
- [149] Edgar R. Synckmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ* 2021;9:e10805.
- [150] Pellow D, Dutta A, Shamir R. Using synckmers improves long-read mapping, bioRxiv. (2022) 2022.01.10.475696. 10.1101/2022.01.10.475696.
- [151] Ma B, Tromp J, Li M. PatternHunter: faster and more sensitive homology search. *Bioinformatics* 2002;18:440–5.
- [152] Sahlin K. Effective sequence similarity detection with strobemers. *Genome Res* 2021;31:2080–94.
- [153] Giroto S, Comin M, Pizzi C. Efficient computation of spaced seed hashing with block indexing. *BMC Bioinf* 2018;19:441.
- [154] Chakraborty A, Morgenstern B, Bandyopadhyay S. S-conLSH: alignment-free gapped mapping of noisy long reads. *BMC Bioinf* 2021;22:64.
- [155] Sahlin K. Flexible seed size enables ultra-fast and accurate read alignment, (n. d.). 10.1101/2021.06.18.449070.
- [156] Berlin K, Koren S, Chin C-S, Drake JP, Landolin JM, Phillippy AM. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat Biotechnol* 2015;33:623–30.
- [157] Li H. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* 2016;32:2103–10.
- [158] Langarita R, Armejach A, Setoain J, P.E.I. Marin, J. Alastruey-Benedé, M.M. Planas, Compressed sparse FM-index: Fast sequence alignment using large k-steps, IEEE/ACM Trans. Comput. Biol. Bioinform. (2020). <https://ieeexplore.ieee.org/abstract/document/9109660/>.
- [159] Vasimuddin M, Misra S, Li H, Aluru S. Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems, in: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019: pp. 314–324.
- [160] Anderson T, Wheeler TJ. An optimized FM-index library for nucleotide and amino acid search. *Algorithms Mol Biol* 2021;16:25.
- [161] Subramanian A, Wadden J, Goliya K, Ozog N, Wu X, Narayanasamy S, et al. Accelerated seeding for genome sequence alignment with enumerated radix trees. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). p. 388–401.
- [162] Ho D, Ding J, Misra S, N. Tatbul, V. Nathan, Vasimuddin, T. Kraska, LISA: Towards Learned DNA Sequence Search, arXiv [cs.DB]. (2019). <http://arxiv.org/abs/1910.04728>.
- [163] Kalikar S, Jain C, Vasimuddin SM. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nat. Comput. Sci.* 2022;2:78–83.
- [164] Huangfu W, Li X, Li S, X. Hu, P. Gu, Y. Xie, MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Association for Computing Machinery, New York, NY, USA, 2019; pp. 587–599.
- [165] Huangfu W, Li S, Hu X, Y. Xie, RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture, in: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018; pp. 1–6.
- [166] Levenshtein VI, et al., Binary codes capable of correcting deletions, insertions, and reversals, in: Soviet Physics Doklady, Soviet Union, 1966; pp. 707–710.
- [167] Alser M, Hassan H, Xin H, Ergin O, Mutlu O, Alkan C. GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping. *Bioinformatics* 2017;33:3355–63.
- [168] Xin H, Greth J, Emmons J, Pekhimenko G, Kingsford C, Alkan C, et al. Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics* 2015;31:1553–60. <https://doi.org/10.1093/bioinformatics/btu856>.
- [169] Alser M, Mutlu O, Alkan C. MAGNET: Understanding and Improving the Accuracy of Genome Pre-Alignment Filtering, arXiv [q-bio.GN]. (2017). <http://arxiv.org/abs/1707.01631>.
- [170] Nag A, Ramachandra CN, Balasubramonian R, Stutsman R, Giacomini E, Kambalabramanyam H, et al. Leveraging in-cache operators for efficient sequence alignment. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. New York, NY, USA: Association for Computing Machinery; 2019. p. 334–46.
- [171] Rizk G, Lavenier D. GASSST: global alignment short sequence search tool. *Bioinformatics* 2010;26:2534–40.
- [172] D. Castells-Rufas, S. Marco-Sola, J.C. Moure, Q. Aguado, A. Espinosa, FPGA Acceleration of Pre-Alignment Filters for Short Read Mapping With HLS, IEEE Access. 10 (undefined 2022) 22079–22100.

- [173] Hach F, Sarrafi I, Hormozdiari F, Alkan C, Eichler EE, Sahinalp SC. mrsFAST-Ultra: a compact, SNP-aware mapper for high performance sequencing applications. *Nucleic Acids Res* 2014;42:W494–500.
- [174] Khalifa M, Ben-Hur R, Ronen R, Leitersdorf O, Yavits L, Kvatinisky S. FiltPIM: in-memory filter for DNA sequencing. In: 2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS). p. 1–4.
- [175] Weese D, Holtgrewe M, Reinert K. RazerS 3: faster, fully sensitive read mapping. *Bioinformatics* 2012;28:2592–9.
- [176] Hameed F, Khan AA, Castrillon J. ALPHA: A Novel Algorithm–Hardware Co-design for Accelerating DNA Seed Location Filtering. *IEEE Transactions on Emerging Topics in Computing*. (undefined 2021) 1–1.
- [177] Liu B, Guan D, Teng M, Wang Y. rHAT: fast alignment of noisy long reads with regional hashing. *Bioinformatics* 2016;32:1625–31.
- [178] Guo L, Lau J, Ruan Z, Wei P, Cong J. Hardware acceleration of long read pairwise overlapping in genome sequencing: a race between FPGA and GPU. In: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). p. 127–35.
- [179] Sadasivan H, Maric M, Dawson E, Iyer V, Israeli J, Narayanasamy S. Accelerating Minimap2 for accurate long read alignment on GPUs. *bioRxiv*. (2022) 2022.03.09.483575. 10.1101/2022.03.09.483575.
- [180] Schmidt M, Heese K, Kutzner A. Accurate high throughput alignment via line sweep-based seed processing. *Nat Commun* 2019;10:1939.
- [181] Gotoh O. An improved algorithm for matching biological sequences. *J Mol Biol* 1982;162:705–8.
- [182] Masek WJ, Paterson MS. A faster algorithm computing string edit distances. *J Comput System Sci* 1980;20:18–31.
- [183] Ukkonen E. Algorithms for approximate string matching. *Inf Control* 1985;64:100–18.
- [184] Daily J. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinf* 2016;17:81.
- [185] Marco-Sola S, Moure JC, Moreto M, Espinosa A. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics* 2021;37:456–63.
- [186] Eizenga JM, Paten B. Improving the time and space complexity of the WFA algorithm and generalizing its scoring. *bioRxiv*. (2022) 2022.01.12.476087. 10.1101/2022.01.12.476087.
- [187] Marco-Sola S, Eizenga JM, Guarracino A, B. Paten, E. Garrison, M. Moreto, Optimal gap-affine alignment in O(s) space. *bioRxiv*. (2022) 2022.04.14.488380. 10.1101/2022.04.14.488380.
- [188] Xin H, Kim J, Nahar S, Kingsford C, Alkan C, Mutlu O, et al. A Generalization of the Landau-Vishkin Algorithm with Custom Gap Penalties. *bioRxiv* 2017;. <https://doi.org/10.1101/133157>
- [189] Ahmed N, Lévy J, Ren S, Mushtaq H, Bertels K, Al-Ars Z. GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data. *BMC Bioinf* 2019;20:520.
- [190] Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv [q-bio.GN]*. (2013). <http://arxiv.org/abs/1303.3997>.
- [191] Aguado-Puig Q, Marco-Sola S, J.C. Moure, C. Matzoros, D. Castells-Rufas, A. Espinosa, M. Moreto, WFA-GPU: Gap-affine pairwise alignment using GPUs. *bioRxiv*. (2022) 2022.04.18.488374. 10.1101/2022.04.18.488374.
- [192] Fei X, Dan Z, Lina L, Xin M, Chunlei Z. FPGASW: Accelerating large-scale smith–Waterman sequence alignment application with backtracking on FPGA linear systolic array. *Interdiscip Sci* 2018;10:176–88.
- [193] Kung. Why systolic architectures? *Computer* 1982;15:37–46.
- [194] Fujiki D, Wu S, Ozog N, Goliya K, Blaauw D, Narayanasamy S, et al. SeedEx: a genome sequencing accelerator for optimal alignments in subminimal space. In: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). p. 937–50.
- [195] Haghi A, Marco-Sola S, Alvarez L, Diamantopoulos D, Hagleitner C, Moreto M. An FPGA accelerator of the wavefront algorithm for genomics pairwise alignment. In: 2021 31st International Conference on Field-Programmable Logic and Applications (FPL). p. 151–9.
- [196] Fujiki D, Subramanian A, Zhang T, Zeng Y, Das R, Blaauw D, et al. A genome sequencing accelerator. In: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). p. 69–82.
- [197] Gupta S, Imani M, Khaleghi B, V. Kumar, T. Rosing, RAPID: A ReRAM Processing in-Memory Architecture for DNA Sequence Alignment, in: 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2019: pp. 1–6.
- [198] Sandes EF de O, de Oliveira Sandes EF, Miranda G, Martorell X, Ayguade E, Teodoro G, Melo ACM. CUDAAlign 4.0: incremental speculative traceback for exact chromosome-wide alignment in GPU clusters. *IEEE Trans Parallel Distrib Syst* 2016;27:2838–50. <https://doi.org/10.1109/tpds.2016.2515597>.
- [199] Diab S, Nassereldine A, Alser M, J.G. Luna, O. Mutlu, I. El Hajj, High-throughput Pairwise Alignment with the Wavefront Algorithm using Processing-in-Memory. *arXiv [cs.AR]*. (2022). <http://arxiv.org/abs/2204.02085>.
- [200] Chen P, Wang C, Li X, Zhou X. Accelerating the next generation long read mapping with the FPGA-based system. *IEEE/ACM Trans Comput Biol Bioinform* 2014;11:840–52.
- [201] Myers G. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J ACM* 1999;46:395–415.
- [202] Loving J, Hernandez Y, Benson G. BitPAI: a bit-parallel, general integer-sequence alignment algorithm. *Bioinformatics* 2014;30:3166–73.
- [203] Banerjee SS, El-Hadedy M, Lim JB, Kalbarczyk ZT, Chen D, Lumetta SS, et al. ASAP: accelerated short-read alignment on programmable hardware. *IEEE Trans Comput* 2019;68:331–46.
- [204] Charikar M, Geri O, M.P. Kim, W. Kuzmaul, On estimating edit distance: alignment, dimension reduction, and embeddings, *arXiv [cs.DS]*. (2018). <http://arxiv.org/abs/1804.09907>.
- [205] Batu T, Ergun F, C. Sahinalp, Oblivious string embeddings and edit distance approximations, *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm - SODA '06*. (2006). 10.1145/1109557.1109644.
- [206] Andoni A, Onak K. Approximating edit distance in near-linear time. *SIAM J Comput* 2012;41:1635–48. <https://doi.org/10.1137/090767182>.
- [207] Chakraborty D, Das D, Goldenberg E, Koucky M, Saks M. Approximating edit distance within constant factor in truly sub-quadratic time. In: 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS). <https://doi.org/10.1109/focs.2018.00096>.
- [208] Slater GSC, Birney E. Automated generation of heuristics for biological sequence comparison. *BMC Bioinf* 2005;6:31.
- [209] Zhang Z, Schwartz S, Wagner L, Miller W. A greedy algorithm for aligning DNA sequences. *J Comput Biol* 2000;7:203–14.
- [210] Zeni A, Guidi G, Ellis M, Ding N, Santambrogio MD, Hofmeyr S, et al. High-performance GPU-based X-drop long-read alignment. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). <https://doi.org/10.1109/ipdps47924.2020.00055>.
- [211] Suzuki H, Kasahara M, Acceleration of Nucleotide Semi-Global Alignment with Adaptive Banded Dynamic Programming, (n.d.). 10.1101/130633.
- [212] Liu D, Steinegger M. Block aligner: fast and flexible pairwise sequence alignment with SIMD-accelerated adaptive blocks. *bioRxiv*. (2021) 2021.11.08.467651. 10.1101/2021.11.08.467651.
- [213] Best Practices Workflows – GATK, (n.d.). <https://gatk.broadinstitute.org/hc/en-us/sections/360007226651-Best-Practices-Workflows> (accessed March 25, 2022).
- [214] Poplin R, Chang P-C, Alexander D, Schwartz S, Colthurst T, Ku A, et al. A universal SNP and small-indel variant caller using deep neural networks. *Nat Biotechnol* 2018;36:983–7.
- [215] Koboldt DC. Best practices for variant calling in clinical sequencing. *Genome Med* 2020;12:91.
- [216] Zook JM, Hansen NF, Olson ND, Chapman L, Mullikin JC, Xiao C, et al. A robust benchmark for detection of germline large deletions and insertions. *Nat Biotechnol* 2020;38:1347–55.
- [217] Genome in a bottle, NIST. (n.d.). <https://www.nist.gov/programs-projects/genome-bottle> (accessed March 25, 2022).
- [218] Liu Y, Zhang M, Sun J, Chang W, Sun M, Zhang S, et al. Comparison of multiple algorithms to reliably detect structural variants in pears. *BMC Genomics* 2020;21:61.
- [219] Sarwal V, Niehus S, Ayyala R, Kim M, Sarkar A, Chang S, et al. A comprehensive benchmarking of WGS-based deletion structural variant callers. *Brief Bioinform* 2022;23. <https://doi.org/10.1093/bib/bbac221>.
- [220] trio-merge-case-study.md at r1.4 · google/deepvariant, Github, n.d. <https://github.com/google/deepvariant> (accessed August 3, 2022).
- [221] Poplin R, Ruano-Rubio V, DePristo MA, Fennell TJ, Carneiro MO, Van der Auwera GA, et al. Scaling accurate genetic variant discovery to tens of thousands of samples. *bioRxiv* 2018;. <https://doi.org/10.1101/201178>.
- [222] Van der Auwera GA, O'Connor BD. Genomics in the Cloud: Using Docker, GATK, and WDL in Terra, "O'Reilly Media, Inc." 2020.
- [223] Liu Z, Roberts R, Mercer TR, Xu J, Sedlazeck FJ, Tong W. Towards accurate and reliable resolution of structural variants for clinical diagnosis. *Genome Biol* 2022;23:68.
- [224] Cai L, Wu Y, Gao J. DeepSV: accurate calling of genomic deletions from high-throughput sequencing data using deep convolutional neural network. *BMC Bioinf* 2019;20:665.
- [225] Park H, Chun S-M, Shim J, Oh J-H, Cho EJ, Hwang HS, et al. Detection of chromosome structural variation by targeted next-generation sequencing and a deep learning application. *Sci Rep* 2019;9:3644.
- [226] deepvariant-details.md at r1.3 · google/deepvariant, Github, n.d. <https://github.com/google/deepvariant> (accessed April 5, 2022).
- [227] Sampietro D, Crippa C, Di Tucci L, Del Sozzo E, Santambrogio MD. FPGA-based PairHMM forward algorithm for DNA variant calling. In: 2019 IEEE 29th International Conference on Application-Specific Systems, Architectures and Processors (ASAP). <https://doi.org/10.1109/asap.2018.8445119>.
- [228] Freed D, Aldana R, Weber JA, J.S. Edwards, The Sentieion Genomics Tools – A fast and accurate solution to variant calling from next-generation sequence data. *bioRxiv*. (2017) 115717. 10.1101/115717.
- [229] Herzeel C, Costanza P, Decap D, Fostier J, Reumers J. elPrep: high-performance preparation of sequence alignment/map files for variant calling. *PLoS ONE* 2015;10:e0132868.
- [230] Herzeel C, Costanza P, Decap D, Fostier J, Wuyts R, Verachtert W. Multithreaded variant calling in elPrep 5. *PLoS ONE* 2021;16:e0244471.
- [231] Yang C-H, Zeng J-W, C.-Y. Liu, S.-H. Hung, Accelerating Variant Calling with Parallelized DeepVariant, in: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, Association for Computing Machinery, New York, NY, USA, 2020: pp. 13–18.
- [232] Luo R, Wong Y-L, Law W-C, Lee L-K, Cheung J, Liu C-M, et al. BALSAs: integrated secondary analysis for whole-genome and whole-exome sequencing, accelerated by GPU. *PeerJ* 2014;2:e421.
- [233] Illumina DRAGEN Bio-IT Platform, (n.d.). <https://www.illumina.com/products/by-type/informatics-products/dragen-bio-it-platform.html> (accessed March 26, 2022).

- [234] Goyal A, Kwon HJ, Lee K, Garg R, Yun SY, Kim YH, et al. Ultra-fast next generation human genome sequencing data processing using DRAGENTM bio-IT processor for precision medicine. *Open J. Genetics* 2017;7:9–19.
- [235] NVIDIA Genome Sequencing Analysis, NVIDIA. (n.d.). <https://developer.nvidia.com/clara-parabricks> (accessed March 26, 2022).
- [236] Talpes E, Sarma DD, Venkataramanan G, Bannon P, McGee B, Floering B, et al. Compute solution for Tesla's Full Self-Driving Computer. *IEEE Micro* 2020;40:25–35.
- [237] Lauterbach G. The path to successful wafer-scale integration: the cerebras story. *IEEE Micro* 2021;41:52–7.
- [238] Lavenier D, Cimadomo R, Jodin R, Variant Calling Parallelization on Processor-in-Memory Architecture, in: 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2020: pp. 204–207.
- [239] Alser M, Kim JS, Alser NA, Tell SW, Mutlu O. COVIDHunter: An Accurate, Flexible, and Environment-Aware Open-Source COVID-19 Outbreak Simulation Model, arXiv [q-bio.PE]. (2021). <http://arxiv.org/abs/2102.03667>.
- [240] Sherman RM, Forman J, Antonescu V, Puiu D, Daya M, Rafaels N, et al. Assembly of a pan-genome from deep sequencing of 910 humans of African descent. *Nat Genet* 2019;51:30–5.
- [241] Ballouz S, Dobin A, Gillis JA. Is it time to change the reference genome? *Genome Biol* 2019;20:159.
- [242] Paten B, Novak AM, Eizenga JM, Garrison E. Genome graphs and the evolution of genome inference. *Genome Res* 2017;27:665–76.
- [243] Cali DS, Kanellopoulos K, Lindegger J, Bingöl Z, Kalsi GS, Z. Zuo, C. Firtina, M.B. Cavlak, J. Kim, N.M. Ghiasi, G. Singh, J. Gómez-Luna, N.A. Alser, M. Alser, S. Subramoney, C. Alkan, S. Ghose, O. Mutlu, SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping, arXiv [cs.AR]. (2022). <http://arxiv.org/abs/2205.05883>.
- [244] Kim JS, Firtina C, Cavlak MB, D.S. Cali, C. Alkan, O. Mutlu, FastRemap: A Tool for Quickly Remapping Reads between Genome Assemblies, arXiv [q-bio.GN]. (2022). <http://arxiv.org/abs/2201.06255>.
- [245] Gamaarachchi H, Samarakoon H, Jenner SP, Ferguson JM, Amos TG, Hammond JM, et al. Fast nanopore sequencing data analysis with SLOW5. *Nat Biotechnol* 2022;40:1026–9.
- [246] Dufresne Y, Lemane T, Marijon P, Peterlongo P, Rahman A, Kokot M, et al. The K-mer File Format: a standardized and compact disk representation of sets of k-mers. *Bioinformatics* 2022. <https://doi.org/10.1093/bioinformatics/btac528>.