CAROLINA TENG

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine

Corrected version

São Paulo
2022

CAROLINA TENG

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine

Corrected version

Dissertation presented to the Polytechnic School of University of São Paulo for obtainment of the Title of Master of Science.

São Paulo
2022

CAROLINA TENG

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine

Corrected version

Dissertation presented to the Polytechnic School of University of São Paulo for obtainment of the Title of Master of Science.

Area of Concentration:

Microelectronics (Graduate Program in Electrical Engineering)

Advisor:

Fernando Josepetti Fonseca

São Paulo
2022

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 31 de Agosto de 2022

Assinatura do autor: Carolina Teng

Assinatura do orientador:

Catalogação-na-publicação

,

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine / Volte e preencha o campo Autor -- versão corr. -- São Paulo, 2022.
132 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.CIRCUITOS FPGA 2.COMPUTAÇÃO EM NUVEM 3.BIOINFORMÁTICA 4.GENÔMICA 5.ALGORITMOS I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

ACKNOWLEDGMENTS

This author is extremely grateful first and foremost to Professor Wang Jiang Chau, for accompanying, teaching, and advising her throughout the entirety of her Master's degree. The amount of learning acquired in this trajectory is of immeasurable value, from technical subjects to introduction to what it is to have scientific thinking. Thanks to Professor Wang for the friendship, tutoring, and great dedication, without which the quality and detail present in this project and in this text would not have been obtained. The shared excitement for solving bugs, relief for achieving good results, and exhaustion from writing until late will never be forgotten and will be greatly missed.

Thanks to the author's colleague Renan Weege Achjian, who shares the same interests in Bioinformatics and FPGA projects, who have helped and followed this author with many issues in the development stages, and have also cheered for all the accomplishments on the way. It has been the best companionship and teamwork one could ask for, and hopefully many more projects and events will be shared in the future.

Thanks to Mauricio Perez for maintaining the Wiener server from Department of Electronic Systems Engineering, and for helping this author in many technical issues. Thanks to Professor Carlos Menck for providing access to the Seal server from *Instituto de Ciências Biomédicas*. Thanks to the qualification board members Ricardo Pires and Fernando Josepetti Fonseca for reviewing the qualification text and suggesting improvements and corrections.

Finally, thanks to the author's family and friends for all the support.

ABSTRACT

TENG, C. **Accelerating the alignment phase of Minimap2 genome assembly algorithm using GACT-X in a commercial Cloud FPGA machine**. 2022. Dissertation (Masters) - Polytechnic School, University of São Paulo, São Paulo, 2022.

Genetic sequencing can provide crucial information in medicine and in biology studies. The technologies developed in the field are advancing rapidly and the current third-generation of genome sequencers have significant improvements over the second-generation. In parallel to that, sequencing throughput has been increasing at an exponential rate, which, coupled with price reduction, has resulted in a leap of generation of genomic data to be processed. Transistor technology is reaching its fundamental limits, and Moore's Law is becoming obsolete, so other alternatives are required to efficiently process such an amount of data.

Long-reads from the third generation of sequencers are shown to be an emerging type of genetic data, with average lengths of thousands of nucleotides each. State-of-the-Art algorithm Minimap2 is able to assemble these reads into the genome that was sampled, but it is a computationally-intensive process: for the human genome size with sufficient coverage, running times can reach up to dozens of CPU hours. Hardware acceleration has been proposed as an effort to make Minimap2 more efficient, but up to the present moment, only one of its main bottlenecks, the chaining step, has been successfully accelerated on FPGA. No efficient solution has been proposed for the aligning step, implemented as the ksw function. GACT-X is a Cloud FPGA design that performs a banded SWG alignment with fixed memory, suitable for any size of input.

GACT-X with tiles of size 4,000 can be 2x faster than ksw when aligning long sequences. Replacing the alignment function ksw in Minimap2 with GACT-X on a Cloud hybrid system can provide up to 1.41x acceleration on the entire execution to the software counterpart, with comparable accuracy for data that have high similarity to the reference genome. This dissertation presents all the relevant background information, the development stages and methods, the results achieved on three different datasets, and the proposed future work on this acceleration project.

Keywords: Cloud Computing, Minimap2, Field Programmable Gate Arrays, Smith-Waterman-Gotoh, Co-processors, Acceleration, Genomics.

RESUMO

TENG, C. ***Accelerating the alignment phase of Minimap2 genome assembly algorithm using GACT-X in a commercial Cloud FPGA machine***. 2022. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2022.

O sequenciamento genético pode forneccer informações cruciais em medicina e em estudos de biologia. As tecnologias desenvolvidas na área estão avançando rapidamente e a atual terceira-geração de sequenciadores de genoma possuem melhorias significantes sobre a segunda-geração. Paralelamente a isso, a taxa de sequenciamento vem aumentando exponencialmente, o que, aliado à redução de preços, resultou em um salto de geração de dados genômicos a serem processados. A tecnologia de transistores está atingindo seus limites fundamentais, e a Lei de Moore está se tornando obsoleta, então outras alternativas são necessárias para processar tal quantidade de dados.

Long-reads da terceira geração de sequenciadores são um tipo emergente de dados genéticos, com comprimentos médios de milhares de nucleotídeos cada. O algoritmo do Estado-da-Arte Minimap2 é capaz de montar essas *reads* de volta ao genoma que foi amostrado, mas é um processo computacionalmente intensivo: para o tamanho do genoma humano com cobertura suficiente, os tempos de execução podem chegar a dezenas de horas de CPU. Aceleraçao em hardware foi proposta como uma aplicação para tornar o Minimap2 mais eficiente, mas até o presente momento, apenas um de seus principais gargalos, a etapa de *chaining*, foi acelerada com sucesso em FPGA. Nenhuma solução eficiente foi proposta para a etapa de alinhamento, implementada como a função ksw. O GACT-X é um design de FPGA em nuvem que executa o alinhamento de SWG em banda, com consumo de memória fixo, adequado para qualquer tamanho de entrada.

O GACT-X com *tiles* de tamanho 4.000 pode ser 2x mais rápido que o ksw ao alinhar sequências longas. Substituir a função de alinhamento ksw no Minimap2 pelo GACT-X em um sistema híbrido na nuvem pode proporcionar aceleração de até 1,41x sobre toda a execução do software, com precisão comparável para dados que têm alta similaridade com o genoma de referência. Esta dissertação apresenta todas as informações básicas relevantes, as etapas e os métodos desenvolvimento, os resultados alcançados em três conjuntos de dados diferentes e o trabalhos futuros propostos para este projeto de aceleração.

Palavras-Chave – Computação na Nuvem, Minimap2, Arranjo de Porta Programável em Campo, Smith-Waterman-Gotoh, Co-processadores, Aceleração, Genômica.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AFI	Amazon FPGA Image
AMI	Amazon Machine Image
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
AWS	Amazon Web Services
AXI	Advanced eXtendible Interface
BAM	Binary (Sequence) Alignment Map
BRAM	Block RAM
BSW	Banded SW
BWA	Burrows-Wheeler Aligner
CIGAR	Compact Idiosyncratic Gapped Alignment Report
CLB	Configurable Logic Block
CL	Custom Logic
CLR	Continuous Long Read
CMOS	Complementary Metal-Oxide-Semiconductor
CNV	Copy Number Variation
CPU	Central Process Unit
CUDA	Compute Unified Device Architecture
DDR	Dual Data Rate
DFE	Data-Flow Engine
DMA	Direct Memory Access
DNA	DeoxyriboNucleic Acid
DP	Dynamic Programming
DSA	Domain-Specific Architecture
ENA	European Nucleotide Archive
FCCM	Field-programmable Custom Computing Machines
FF	Flip-Flop
FIFO	First In, First Out
FPL	Field Programmable Logic
GATK	Genome Analysis ToolKit
GPU	Graphics Processing Unit
GRCh38	Genome Reference Consortium human build 38
HDL	Hardware Description Language
HGP	Human Genome Project
HLS	High Level Synthesis
hsa	homo sapiens organism code
IOB	Input/Output Block
I/O	Input/Output
ISA	Instruction Set Architecture
KNL	KNights Landing
LUT	Look-Up Table
MEM	Maximal Exact Match

MM	Memory Mapped
MPSoC	Multi-Processor System-on-Chip
MUX	MUltipleXer
NCBI	National Center for Biotechnology Information
NUMA	Non-Uniform Memory Access
NW	Needleman-Wunsch
ONT	Oxford Nanopore Technologies
OpenCL	Open Computing Language
OS	Operating System
PacBio	Pacific Biosciences
PCIe	Peripheral Component Interconnect express
PCR	Polymerase Chain Reaction
PE	Processing Element
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computer
RNA	RiboNucleic Acid
RTL	Register Transfer Level
SAM	Sequence Alignment Map
SD	Secure Digital
SD	Standard Deviation
SDK	Software Development Kit
SIMD	Single Instruction, Multiple Data
SK	Suzuki-Kasahara
SLR	Super Logic Region
SMRT	Single Molecule Real-Time
SNV	Single Nucleotide Variant
SRAM	Static RAM
SSE	Streaming SIMD Extensions
SV	Structural Variant
SW	Smith-Waterman
sw	software
SWG	Smith-Waterman-Gotoh
USD	United States Dollar
VCF	Variant Call Format
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WC	Watson-Crick
WGA	Whole Genome Alignment
XML	eXtensible Markup Language

CONTENTS

1 INTRODUCTION

Genetic sequencing can provide information in medicine for a wide variety of uses. In preventative medicine, some genetic variations are shown to raise the risk for certain diseases, such as breast cancer, heart diseases, and type II diabetes, for which therapies or preventative strategies are available. Genetic sequencing can also be used to identify current or future genetic diseases, such as muscular dystrophy and Waardenburg syndrome. This information can help with life planning and earlier treatment of symptoms. Another use of genetic sequencing is to help asses whether a person is a carrier of variants that might cause disease in their children, but does not affect themselves, such as cystic fibrosis, fragile X syndrome, and sickle cell anemia (KUSHNICK, 1992).

Collecting and analyzing genetic data is also crucial in biology studies, such as in biodiversity (HENG; HENG, 2021), evolution (ORTEU; JIGGINS, 2020) and metabolic pathways (GEORGAKOPOULOS-SOARES et al., 2020). It has even become a commodity of public interest, providing people insights on their ancestry and personal phenotypes (GENERA, 2022).

Current sequencing technology is not capable of reading straightaway a complete human genome strand, so each DNA (deoxyribonucleic acid) molecule in the sample needs to be cleaved into many much smaller sequences, called fragments. The sequenced fragments are then called reads. Each read can also be understood as the equivalent sequence translated into nucleotide bases represented by characters (A, T, C, G) for computer processing. Depending on the technology used in the sequencing process, there can be two types of reads: short-reads, with lengths of a few hundreds of bases (ILLUMINA, 2022), and long-reads, with many thousands of bases on average (PACBIO, 2022)(OXFORD. . . , 2022).

Currently, the so-called second-generation of sequencing technology produces short-reads with high throughput, dominates the market (ADEWALE, 2020) and is expected to stay prevalent for the next years. However, only the emerging third-generation of sequencers that produce long-reads is able to identify long alterations in the DNA (MAN-

TERE; KERSTEN; HOISCHEN, 2019). The slow transition to long-reads is explained mainly by their higher sequencing cost compared to short-reads. Nowadays hybrid long- and short-reads solutions for genome assembly are implemented in needed situations (ANTIPOV et al., 2015).

Several algorithms and programs have been developed to assemble reads back into the original complete DNA sequence (LI, 2018)(BURROWS-WHEELER..., 2010)(LANGMEAD; SALZBERG, 2012). Some of the assembling algorithms map the reads to a genome reference, which is used as a guide (GRCH38, 2013). First, approximate mapping positions are identified for the read in the reference. Then, the reads are aligned to the reference's mapping region to pinpoint the variations in the genome that has been sequenced.

Some classic algorithms, such as the Smith-Waterman-Gotoh (SWG) algorithm (SMITH; WATERMAN, 1981)(GOTOH, 1990), use dynamic-programming to align two character strings. It was published in 1981-1990 and is still used to this day in read assembly programs, but with many heuristics and transformations added on, such as calculating only a portion of the matrix where the optimal alignment is more probable to be (FUJIKI et al., 2020)(LIAO et al., 2018).

Read assembly is currently considered a major bottleneck in the entire genome analysis pipeline (ALSER et al., 2020), that includes laboratory sampling, sequencing, processing and annotating. Sequencing technologies have been increasing their output capacity in terms of number of reads at an exponential rate (REUTER; SPACEK; SNYDER, 2015), whereas computational power has been slowly reaching the limits of transistor technology (HENNESSY; PATTERSON, 2019). Read assembly is also the bottleneck of the genome processing pipeline (GENOME..., 2022), which corresponds to the technical activities involved in genetic sequencing.

Although the cost of sequencing DNA is a more significant impediment for the technology's diffusion in clinical settings, the speed of acquiring and interpreting genomic information is crucial in certain applications. For instance, prenatal testing is performed on women during pregnancy to assess whether the fetus could be born with a genetic condition or birth defect, which can be helpful to determine the management of the pregnancy and delivery (SAMURA, 2020; GADSBøLL et al., 2020). Pathogen genomics can be used in diagnosing infections, investigating outbreaks and describing transmission patterns, and has been highly present in the latest COVID-19 pandemic (LIU et al., 2021; THIEL et al., 2003).

In order to improve the computational performance in software-based algorithms, one alternative is designing specialized hardware for the assembly task; particularly with application specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). ASICs are more costly due to customized manufacturing; only a high-volume production would justify its adoption. FPGAs, on the other hand, are configurable devices that can surpass software performance without requiring a high manufacturing capital.

Several FPGA acceleration articles can be found in the read assembly acceleration literature, and some have achieved improvements in processing time (KOLIOGEORGI et al., 2019)(FUJIKI et al., 2020)(GUO et al., 2019). Given the variety of new algorithms and tools that are developed in the field every year, and the steady evolution of sequencing technology that now produces considerably longer reads, it is crucial for the hardware research to constantly adapt to the new changes.

1.1 Motivation

Many programs, such Bowtie2 (LANGMEAD; SALZBERG, 2012) have been developed to efficiently assemble short-reads at the second generation technology. However, their strategies are often not suitable to process long-reads, having their performance hampered. Hardware acceleration that has been proposed for alignment in these algorithms on a few hundred nucleotides (KOLIOGEORGI et al., 2019) (FUJIKI et al., 2020) can't support sequences of many thousands of nucleotides. This is because the SWG algorithm with banding has two scaling aspects: the number of stored backtracking pointers grows with linear proportion to the inputs' lengths, and the cumulative alignment scores used in the wave-front expansion increase with the number of matching nucleotides.

BWA-MEM (BURROWS-WHEELER..., 2010) and Minimap2 (LI, 2018) are examples of programs/algorithms proposed to fill the software gap, the latter being 50 times faster than the former one, besides having better mapping accuracy than most other programs; therefore, Minimap2 is one of the current State-of-the-Art algorithms for assembling long-reads. Minimap2 has a chaining algorithm that takes advantage of the higher load of information carried by long-reads to find approximate mapping positions with better performance and accuracy. It also uses a transformed version of the SWG algorithm, proposed by Suzuki and Kasahara (SUZUKI; KASAHARA, 2018), for the alignment (or extending) step, that limits data size in Streaming SIMD (Single Instruction, Multiple Data) Extensions (SSE) vector instructions to optimize parallelization of computation of cells.

Still, mapping one human sample of reads to a reference using Minimap2 is very time consuming, taking dozens of Central Processing Unit (CPU) hours on an Intel Xeon processor, which further increases the already expensive genome testing budget. This is because the human genome is really long (approximately 3.2 GB of data) and read information need to be 8-17 times larger in order to get enough map coverage for reliable clinical standard (i.e. to statistically cover sequencing errors and mapping heuristics, and to identify heterozygous states) (AMARASINGHE et al., 2020). Some laboratories even need to invest in powerful and power hungry compute clusters just to meet the computational demands.

Previous studies have determined that the chaining and extending steps of Minimap2 are its run-time bottlenecks, taking together around 70% of the total execution time. Focused on this, authors of such works have already successfully accelerated Minimap2's chaining step on FPGA and Graphic Processing Unit (GPU) (GUO et al., 2019). Others

have also implemented the extending step on CPU, GPU and KNL (Knights Landing) (FENG et al., 2019), with an acceleration of 3.9x times with 128 CUDA (Compute Unified Device Architecture) streams with 512 threads each in the GPU case; but achieved lower performance per thread in the hardware designs compared to software. To the author's knowledge, no work in the literature has successfully accelerated Minimap2's extending step on an FPGA.

Designing an application as hardware is more complex than programming it as software. In addition, the design of an FPGA accelerator takes a longer time, and is usually more expensive than programming a GPU accelerator. So it is important to assess which application can achieve a better performance. According to the authors of (GUO et al., 2019), FPGA may achieve a better performance in several applications, as in the case of Minimap2, for the following reasons:

- When the algorithm includes a great amount of checking and filtering options, many control and branch conditions are required, almost at the same rate as the integer and float arithmetics. On FPGAs, these control logic will add to the pipeline without having a significant impact on throughput;
- Tasks with irregular-width integer type might be more suitable for FPGAs;
- GPUs need to unpack the data structure, adding instructions to some applications, whereas FPGAs' inputs are kept in BRAM (Block Random-Access Memory) and used immediately with combinatorial logic;
- GPUs' performance relies on parallelism, but read assembling, although parallelizable along reads, has an extensive processing pipeline for each read.

The extending step of Minimap2 is based on a banded SWG algorithm optimized for wave-front parallelization of cells with SSE vector instructions, that consumes time and memory in linear proportion to the input's lengths. The chaining algorithm of Minimap2 breaks the sequence pairs into many sub-sequences that are sent separately to the extending function. With this, the extending module is prepared to deal with input data that has an average length of a few hundred bases, but that sometimes can also be very long. This variation of sizes requires a new hardware design that can compensate for two issues: first, short sequences won't be worth sending to co-processors separately, because even if they can be processed faster in the FPGA, the time spent transferring the data alone will be higher than the processing time taken by the software (TENG et al., 2021);

second, long sequences can extrapolate the hardware's resources that calculate banded SWG matrices.

Darwin (TURAKHIA; BEJERANO; DALLY, 2018) was published at about the same time as Minimap2 and is a software-hardware hybrid system that also assembles long-reads, achieving up to 183.8x speedup over Edlib (ŠOŠIC; ŠIKIC, 2017). Its initial filtering stage runs on software, presenting a high RAM (Random-Access Memory) consumption (30-64GB of RAM for the human genome), which makes Darwin efficient only for sequencing cases with references of limited size. Their extending step GACT, which also calculates the SWG matrix, has been accelerated on an FPGA and was adapted to work with inputs of any length, since they limit BRAM consumption in hardware by dividing the matrix into tiles of fixed size.

The same authors of Darwin also published the software-hardware hybrid whole genome aligner Darwin-WGA that came with an improved extending architecture GACT-X (TURAKHIA et al., 2019). It is not a read assembler, but a single long stream aligner. It would be possible to adapt Darwin-WGA to run as a read assembler by transferring the GACT-X module into Darwin's software support (since both of them use the D-SOFT filtering algorithm) after taking care of the RAM consumption issue. GACT-X is 2x faster than GACT and was developed on the Amazon Web Services (AWS) Cloud platform (AMAZON..., 2022c), so it can be easily replicated for other uses. The re-purposing of Darwin-WGA's GACT-X module, designed on an FPGA, could potentially accelerate Minimap2's extending bottleneck with a Cloud hybrid architecture.

In recent years, Cloud FPGAs (AMAZON..., 2022a) (HUAWEI..., 2022) (VMACCEL, 2022) have become a compelling alternative to reduce the initial cost of hardware implementation, by charging a smaller value per time of usage. Cloud FPGAs work the same way as Cloud computers: a big cluster of processors is physically stored and managed by a company that offers their clients scalable storage and computing capacity, charging in proportion to their usage demand. Some companies like Amazon (AMAZON..., 2022c) offer other advantages to developers. For instance, their FPGA comes with a Shell that wraps any kernel to fit the FPGA's I/O (Input/Output) resources, and projects developed on their platform can be encrypted and advertised on their marketplace, with all intellectual property secured. Therefore, an acceleration scheme can be tested with reduced implementation overhead; only after the implementation has shown to be successful, the designer may decide to move to a customized board implementation, or to rely on the Amazon platform for business or research.

Although Cloud FPGA providers announce large accelerations for applications when compared to software implementations (for instance, Amazon claims that AWS FPGA Instances can accelerate compute-bound applications up to 100x), the real acceleration is limited by virtualized PCIe (Peripheral Component Interconnect Express) transfer's throughput and latency, which is dependent on the driver's technology and whether virtual machines are adding extra virtual interrupts. Previous work (WANG et al., 2020) has measured the performance gap between Cloud FPGAs and physical implementation PCIe transfers and indicated that it is the limiting bottleneck for communication-intensive workloads, which in their examples did not perform over 2x faster than the software counterparts. Therefore, the adoption of Cloud FPGAs for Minimap2 must be analyzed and carefully considered.

1.2 Objectives

This Master's dissertation presents the development steps of a software-hardware hybrid system that accelerates the State-of-the-Art program Minimap2's aligning step by using the Cloud FPGA module GACT-X from Darwin-WGA in substitution to the Suzuki-Kasahara's algorithm. The specific objectives are:

- **Data acquisition.** The selection of a set of human genome data suitable to the experiments proposed in this dissertation is needed. Human sequencing data can be downloaded from many different open source repositories, such as the United State's National Center for Biotechnology Information (NCBI) (NCBI, 2022) and the European Nucleotide Archive (ENA) (EUROPEAN..., 2022); reads data can also be generated with simulating tools such as PBSIM (ONO; ASAI; HAMADA, 2013). The consensus human genome reference is updated from time to time and its versions are submitted at NCBI (GRCH38, 2013).

- **Evaluation of Minimap2.** First, Minimap2's processing time bottlenecks need to be determined for long-read datasets that have different read length distributions. Minimap2's performance running on AWS' CPU machines need to be measured and used as a reference to assess the acceleration. Minimap2's mapping and aligning accuracies need to be collected for the same purpose. Finally, some internal data aspects need to be measured, to identify potential issues when integrating with the FPGA design.

- **Adapting GACT-X and Evaluating the Implementation.** The original GACT-X module published on GitHub must be adapted with changes in the host for alignment of different pairs of sequences produced by the chaining step of Minimap2's software. The data transfer must be analyzed in order to verify the best sequence format to be adopted. GACT-X's Verilog files may need to be altered to exploit the best performance for the implementation.

- **Integration.** Minimap2's compiler should be merged with GACT-X's compiler, and the accelerator's host needs to be included in Minimap2's code for an integrated design. The integration has to be compatible with the multi-threading capacity of Minimap2 with proper synchronization between multiple kernels and multiple cores in the host. With this, it is possible to measure the total acceleration.

- **Measuring acceleration.** Execution time measurements must be made for the GACT-X module and for the hybrid software-hardware accelerated system as well,

in order to analyze the optimization options and difficulties. The total acceleration and the acceleration by read length are to be measured for analysis.

- **Measuring accuracy.** Comparing the alignment accuracy is important to assure the proposed accelerated solution is acceptable. The alignment accuracy is impacted by the heuristics applied to accelerate the calculation of the SWG scores. GACT-X and Minimap2 use different banding algorithms and both also break the pairs of sequences into smaller sub-sequences with different methods, chaining for Minimap2 and tile for GACT-X.

1.3 Text Organization

This dissertation is divided into 7 main chapters. The introduction to the context of this work has been presented in this chapter. Chapter 2 probes the human genome, genome sequencing, and FPGA concepts and fields. Chapter 3 lightly introduces works that are related to the one proposed, presenting their main strategies and results. Chapter 4 is more theoretical and explains in detail and examples the main algorithms used in this project. Chapter 5 shows the first steps of collecting and generating data and quantitatively studying the State-of-the-Art program Minimap2. Chapter 6 describes the steps of adaptation and integration performed on the hybrid Cloud design GACT-X to be used to accelerate Minimap2, and the final accuracy and speed measurements are presented in the end. Chapter 7 summarizes all the results obtained in each step and proposes some future work options.

2 BACKGROUND

Bioinformatics is inherently a multidisciplinary field, that uses algorithms and digital processing to help solve biological problems. This chapter gives a broad introduction to each relevant topic involved, with concepts from both fields. It starts with an overview of the human genome and its relevance in medicine. Then sequencing technologies are briefly introduced, allowing readers to understand the data that is being dealt with, the differences between long-reads and short-reads, the GATK best practices' steps for the genome processing pipeline, and the current reference-guided read assembly *seed-and-extend* strategy. FPGAs are described in the last subsection; this can help understand how they differ from general processors and ASICs, and how they are able to accelerate software counterparts. At last the AWS Cloud FPGA Instance used in this project is referenced and detailed, followed the description of the OpenCL (Open Computing Language) toolset.

2.1 Overview of the Human Genome

DNA is a polymeric nucleic acid macro molecule, composed of three types of unities: a sugar of five carbons (desoxirribose), one base containing nitrogen, and a phosphate group. The bases can be of type Adenine (A), Guanine (G), Thymine (T) and Cytosine (C). Any of them links to the desoxirribose by the nitrogen atom, and to a phosphate group, forming a corresponding nucleotide (Figure 1). Poly-nucleotide chains form a double helix structure, in which one ribbon is the Watson-Crick (WC) complement of the other (Figure 2). For the purpose of genome sequencing, the actual molecule configuration is not relevant and the nucleotides are abstracted to the characters A, C, T, and G.

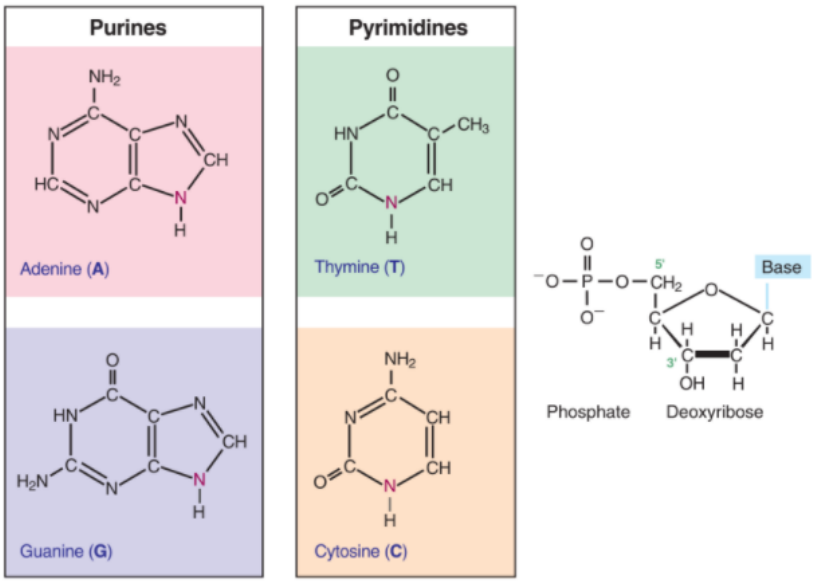

Figure 1: Lewis structures of the four types of DNA nucleotides (KUSHNICK, 1992)

On the left, DNA bases can be Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). Each base links to the deoxyribose, which is connected to a phosphate group (on the right), by the nitrogen in magenta, to form the corresponding nucleotides.

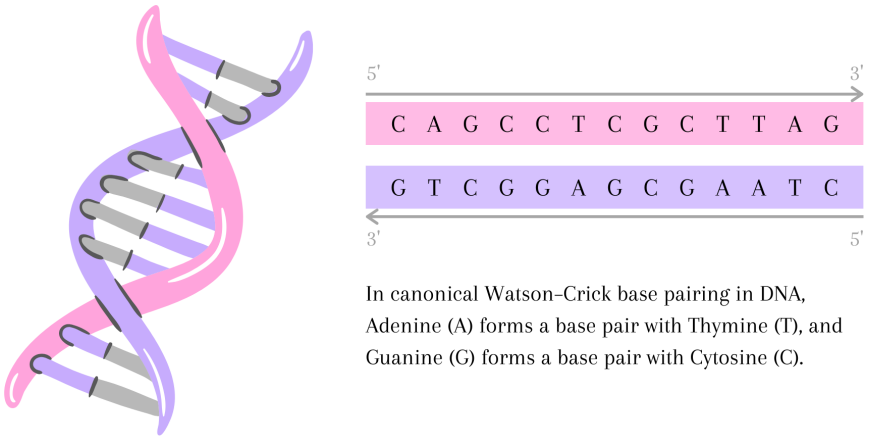

Figure 2: Canonical Watson-Crick base pairing in DNA

The strands are conventionally read in the direction 5' to 3', which is determined by the phosphodiester 5'-3' bonds between desoxirribose adjacents. One strand is the inverted Watson-Crick complement of the other. This complementability enables efficient and correct repair of DNA damage.

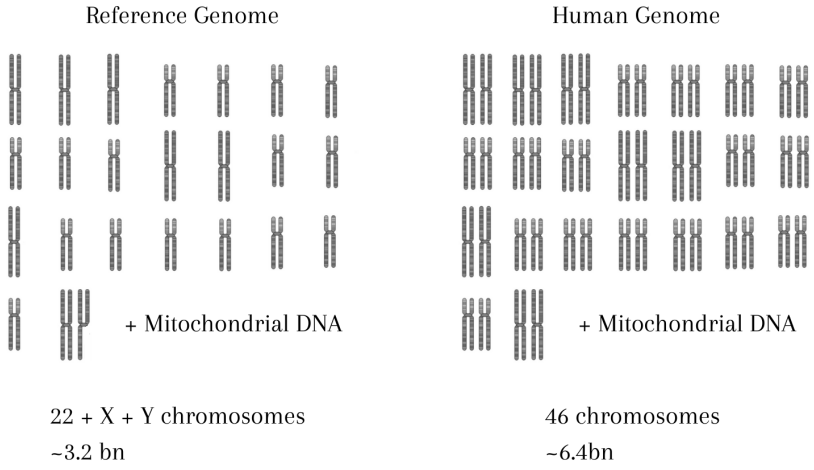

Figure 3: Human genome reference composition vs. somatic cell composition

On the left, the human genome reference has the 22 types of autosome human chromosomes, the 2 types of sex chromosomes, and mitochondrial DNA. On the right, human somatic cells' autosome chromosomes have two copies of each gene and both of them can express and generate a product, except in special cases of allelic imbalance.

Every somatic cell (diploid cell) of the body carries it's copy of the human genome, which has around 6.2 billion nucleotides. Given that humans are diploid, and receive 23 chromosomes from each parent, the established reference to the human genome (more in Subsection 2.1.1) has around 3.2 billion nucleotides containing all 24 types of chromosomes (Figure 3). The first human genome reference was obtained by an international collaborative research effort called The Human Genome Project (HGP) (HUMAN..., 2003), which started in 1990 and completed in 2003. The reference is updated as better sequencing technologies emerge and more research is done. The latest version, Genome Reference Consortium Human Build 38 (GRCh38), was published by NCBI in 2013. The references are presented in one of the DNA strands with direction 5' to 3'.

There are around 30,000 genes (functional units of genetic information) in the human genome. They can be responsible for the production of proteins or functional RNAs (ribonucleic acids). Only less than 1.5% of the human genome codifies proteins and it is believed that around 5% of the genome influences or determines gene expression patterns during development or in different tissues. It is still heavily debated whether the remaining portion of the genome could also provide relevant signals for the genome's functions (PALAZZO; GREGORY, 2014). These sub-sequences can contribute in an interconnected manner to the phenotypes of an organism. Genes appear in the DNA sequence in different densities per chromosome, as shown in Figure 4.

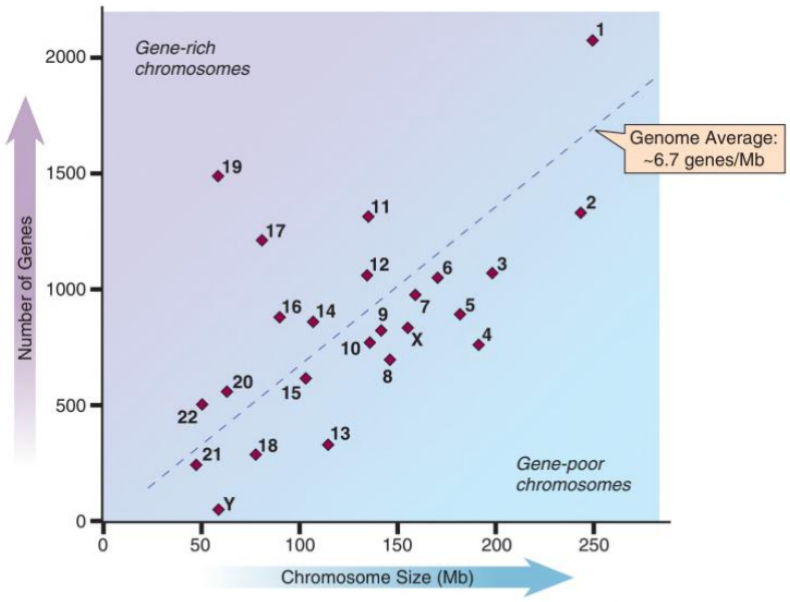

Figure 4: Human gene concentration graph for each chromosome (KUSHNICK, 1992)

The genome has on average 6.7 encoder genes for every mega-base; chromosomes under this line are called gene-poor chromosomes and above this line are called gene-rich chromosomes.

Analysis of the genome sequence shows that about half of it consists of unique DNA copies, which means that they only appear once (or a few times) throughout the whole genome. Most of the unique DNA copies are short sequences (a couple of kilo-bases or less) and are scattered in the genome. The other half of the DNA sequence consists of repetitive DNA that shows up hundreds of thousands of times, identically or with small changes, throughout the genome. They contribute to maintaining the chromosome's structures (e.g. centromere and telomer), are an important source of variation between different individuals, and can be responsible for up to one every 500 genetic diseases.

Given the number of individuals in our species, it is expected that every base-pair in the human genome varies in someone somewhere in the world. There's about 0.5% of genome sequence variation between any two individuals chosen at random. The majority of these differences consists of insertions or deletions of short stretches, number or copies of repetitive elements, or inversion of order of sequences in a certain position.

Genetic diseases can be categorized into three types. Chromosomal disorders, caused by excess or lack of localized genes, affect around 7 out of 1,000 humans born alive and are responsible for around half of all spontaneous abortions that happen in the first three months of pregnancy. Monogenic diseases are caused by mutations in individual genes and follow the autosomal recessive, autosomal dominant, or linked to X patterns. It is a rare

Figure 5: Frequency of genetic disorder types in the population and common examples

disorder, but causes serious disturbances in 1 out of 300 humans born alive and prevails in 1 out of 50 people throughout life. Multi-factorial diseases with complex heritage result from the combination of several gene variations and environmental factors. It is a prevalent genetic disorder and affects 5% of the pediatric population and more than 60% of the population in general (Figure 5).

2.1.1 The Human Genome Reference

Genome references are created as synthetic hybrids (an archetype of the most common variants) aimed to represent a common standard in a species. The purpose is to make it easier to identify which variations are commonly observed in a population and which variations are more unique. Humans are diploid organisms, which means that they carry two copies of each autosome chromosome, one copy originating from each parent. Genome references, however, are haploid, representing each type of chromosome only once. This requires additional steps for identifying heterozygous states (situations where a person carries different variants in the same position on both chromosomes).

The primary assembly of a genome reference contains the assembled chromosomes, the unlocalized sequences (known to belong to a specific chromosome but with unknown order or orientation) and unplaced sequences (with unknown chromosome). The latest human genome GRCh38 came out in 2013 (GRCH38, 2013). In the ".fna" format file, the primary assembly has 193 sub-sequences (24 chromosomes + 127 unplaced + 42 unlocalized), and was separated from the rest of the file for use in this dissertation.

2.2 Genome Sequencing and Genome Analysis

The Human Genome Project (HUMAN..., 2003) prompted the development of cheaper DNA sequencing technologies, with the purpose of obtaining reads: pieces of ordered genetic coding collected from a human sample material. It was a 20 year conjoint research effort of many scientists around the world that cost around 3 billion USD (United States Dollar) in total. At the end of the project, the cost to sequence a human genome was reduced to 100 million USD, using traditional Sanger sequencing.

Sanger sequencing (SANGER; NICKLEN; COULSON, 1977) follows the steps: chain-termination PCR (polymerase chain reaction) is applied to a DNA fragment, generating multiple copies of it that are randomly terminated at different lengths, and with the last nucleotide emitting a fluorescent label corresponding to its base; the copies are subjected to gel electrophoresis: by applying an electric current, since DNA is negatively charged, it moves to one direction with speed determined by its size; the final arrangement of the gel matrix can be translated into the DNA fragment's sequence. The original manual Sanger sequencing method performed four PCR reactions, one for each base type, resulting in fours columns, as shown in Figure 6. Automated Sanger sequencing performs only one PCR amplification for all base types.

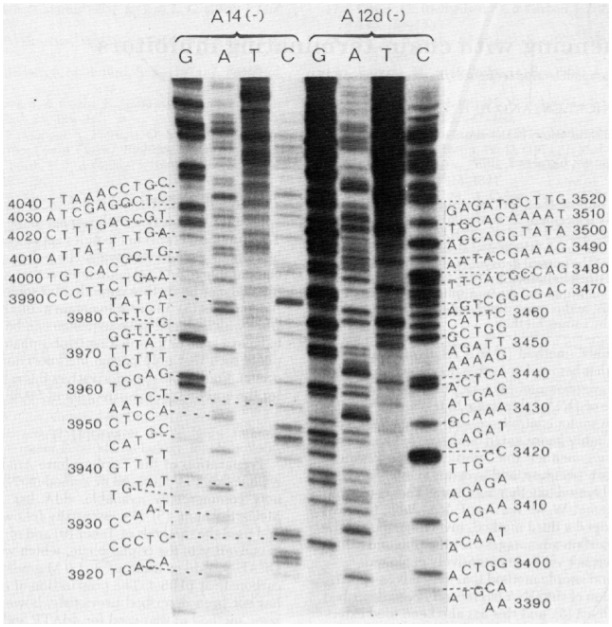

Figure 6: Sanger sequencing gel electrophoresis (SANGER; NICKLEN; COULSON, 1977)

The sequence is written from left to right and upwards, beside each corresponding band.

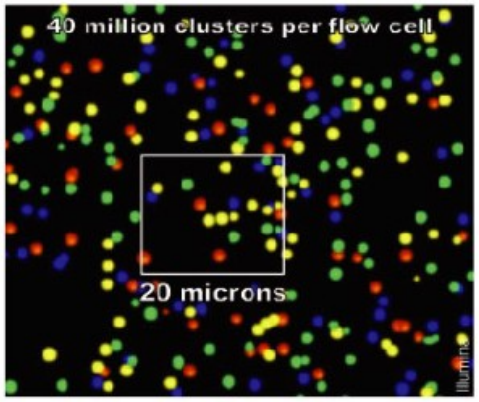

Figure 7: Illumina sequencer DNA clusters captured by a microscope (CHI, 2008)

Each color corresponds to a base type being read from a cluster at a given moment.

To really understand functions of the human genome, thousands of genomes need to be sequenced. This was only made possible with the introduction of the second-generation sequencers. Illumina sequencing (ILLUMINA, 2022) currently takes around 90% of the sequencing market (ADEWALE, 2020). It uses flow-cells: microscope slides with channels inside. The sequencing process follows the steps: the sample's library with the reads is expanded using normal PCR amplification; adapter sequences are added to both ends of the reads, these adapter sequences correspond to primer and capture sequences; the DNA is denatured and captured by complement sequences that are fixed in the flow-cell; DNA polymerase fills the complementary strand on top of the fixed capture sequence and the original sequence is washed off; the captured sequences are copied many times until a cluster is formed with many copies of the same sequence; fluorescent terminators are added to the flow cell in cycles: when incorporated to the sequence, they emit a specific frequency for the base type; a powerful microscope takes pictures of the clusters and sequences the reads one base at a cycle (Figure 7).

In Illumina sequencing, the read lengths are limited because not every strand in a cluster can be captured in each cycle, so as sequencing progresses, more and more strands lag behind, affecting the fluorescent color of the cluster and reducing accuracy. This technology achieves high throughput due to the ability to read millions of clusters of sequences and many different samples simultaneously.

The third-generation of sequencers introduced long-reads, many orders of magnitude longer than in the second-generation (more details in Subsection 2.2.1). One of them is from Oxford Nanopore Technology (ONT) (OXFORD. . . , 2022). They use 18 nm nanopores embedded in lipid membranes to sequence DNA and RNA. The single stranded molecules are forced to thread individually through the pores and each base changes the

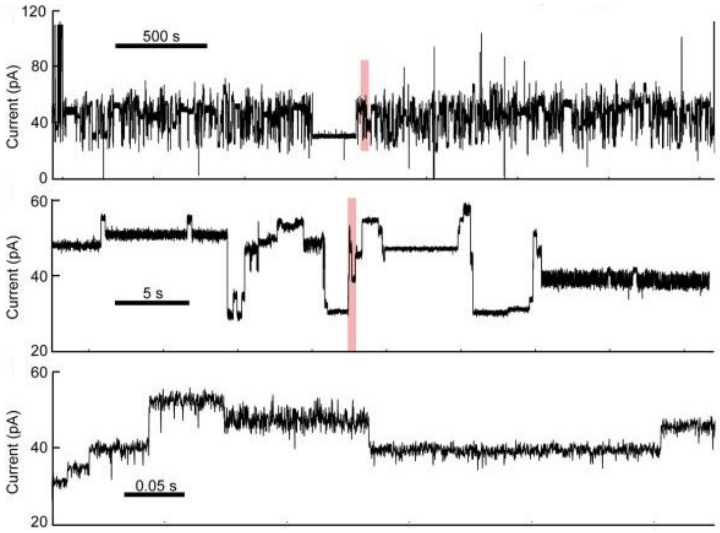

Figure 8: ONT sequencer Ion current (LASZLO et al., 2014)

Each consecutive graph is a zoom-in of the red section in the previous graph.

electric current that is being measured (Figure 8). However, a pore contains around 6 bases at a time, affecting the measurement and increasing the error rate, but this is alleviated by error correction methods ($< 5\%$) (AMARASINGHE et al., 2020). The reads produced can be really long (up to hundreds of Mega-bases). The sequencer is really portable and can be directly plugged into a personal computer.

Pacific Biosciences (PacBio) (PACBIO, 2022) sequencers are also from the third-generation, with long-reads. They use nanometer wells (Figure 9) with DNA polymerase at the bottom. A camera underneath captures videos that register a fluorescence spike when a new base is incorporated. The fluorescent component is able to leave after a time, dropping the intensity of light. The error rates are similar to ONT, but are random, which makes it possible to generate consensus sequences. For that, the same strand is connected end-to-end as a circle and is re-sequenced many times to remove sequencing errors. With this, the achieved accuracy can be higher than the ones achieved with Illumina technology (AMARASINGHE et al., 2020).

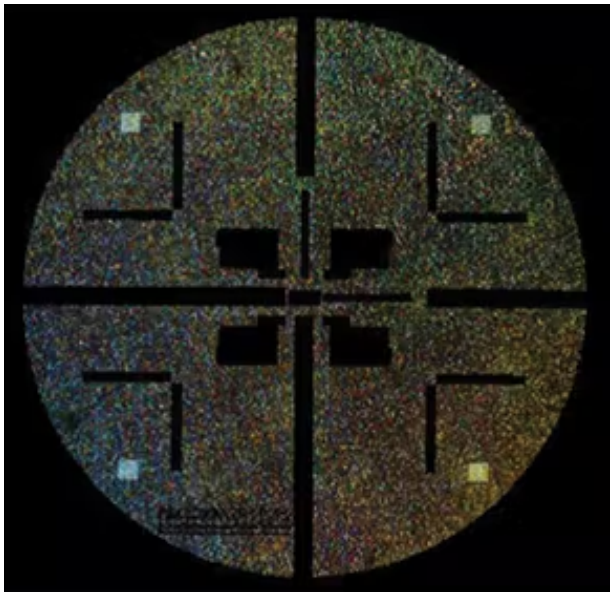

Figure 9: PacBio sequencer nanometer wells (PFEUFER; SCHULZE, 2015)

2.2.1 Short-Reads vs. Long-Reads

The second-generation of sequencing technology leveraged the clinical use of genetic testing by steeply reducing the cost to sequence a human genome or exome (fraction of the genome that codifies for genes). Read sequences from this generation commonly have only a few hundreds of bases, and so, are called short-reads. Even nowadays, short-reads are still known for being some of the most cost-effective, accurate, and popular types of genetic data, even after the introduction of the third-generation of sequencing technology (ADEWALE, 2020).

The third-generation came along with long-reads, that have average length of kilobases, which solved many sequences in the human genome reference that short-reads were not able to cover. Firstly, short-reads are unable to, or are really ineffective at, detecting structural variants (SVs), which are insertions, deletions, duplications, inversions, or translocations in the genome that affect more than 50 nucleotide bases. Each human genome has $>$ 20,000 SVs and additional thousands of indels (insertion–deletion mutations with $<$ 50 base-pairs or bp), and most of them have remained undetected until long-reads were introduced. This poses a serious problem since SVs account for the greatest number of divergent bases across human genomes.

Secondly, short-read sequencing technologies often require the DNA to be fragmented and subjected to PCR amplification, which introduces GC-content coverage bias (high GC content can affect the efficiency of PCR due to the tendency of these templates to

Table 2: Comparison between short-read and long-read data

	Short-Reads	**Long-Reads**
	Second Generation	*Third Generation*
Technologies	Illumina, BGI, Thermo Fisher	PacBio, ONT
Length	up to 600 bases	up to hundreds of kilo-bases
Error rate	$< 0.1\%$	$< 5\%$
Cost (human genome)	USD 942	USD 1500
Assembly algorithms	BWA-MEM, Bowtie2, SNAP, Minimap2	Minimap2, blasr-mc, BWA-MEM

fold into complex secondary structures) and hampers the detection of base modifications, such as methylation (when a methyl group is added to a base, changing the DNA's activity without changing the sequence) (AMARASINGHE et al., 2020). Long-reads have also shown to be essential at detecting copy number variations (CNVs) (regions with multiple copies of short sequences), at phasing alleles (assigning gene variants to paternal or maternal chromosomes), and at differentiating pseudo-genes (sequences that resemble functional genes but can't produce functional proteins) (MANTERE; KERSTEN; HOISCHEN, 2019).

Long-reads were known for having a large sequencing error rate compared to short-reads, but a recent article published in 2020 (AMARASINGHE et al., 2020) showed that error correction strategies (such as the one mentioned for the PacBio technology in Section 2.2) reduced it considerably, closing the gap to short-reads. Short-reads now have an error rate $< 0.1\%$ and long-reads $< 5\%$. One of the factors that are slowing the adoption of long-read sequencers is the higher cost to sample a human genome (USD 942 using short-reads and USD 1500 using long-reads in 2020 (ADEWALE, 2020)) (see comparison between short- and long-reads in Table 2). Some researchers argue that it also takes time for the scientific community to become adapted to the new technologies from the third-generation (ADEWALE, 2020).

2.2.2 The GATK Pipeline

The next step after collecting genomic reads is processing the data to acquire useful information, such as the specific variants in the sequenced individual's genome. An extensive pipeline of tools is used in this step, which is currently considered the bottleneck of the complete genome analysis pipeline (ALSER et al., 2020). The most conventionalized processing pipelines come from the Genome Analysis Toolkit (GATK) (GENOME..., 2022), directed by the GATK Best Practices for different ends, with the workflows designed mainly for Illumina short-reads. This subsection will skip through the processing tools in GATK4 and also comment on changes that would be applied to process long-reads

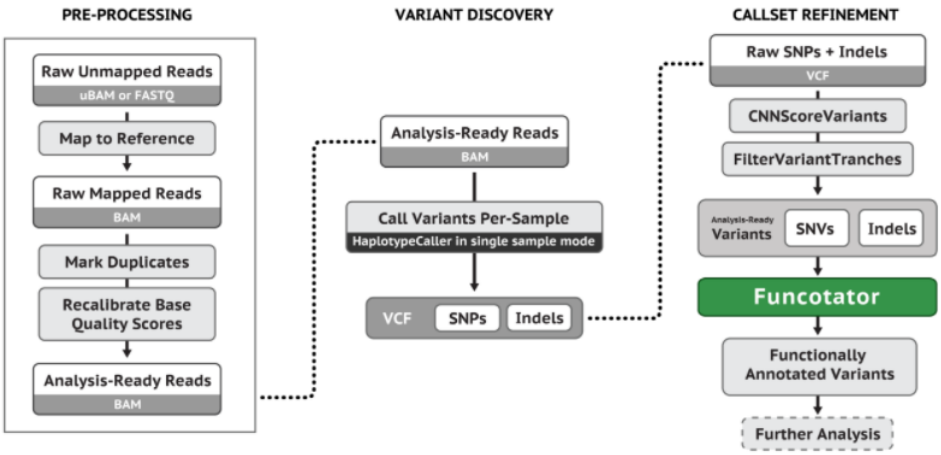

Figure 10: GATK4's pipeline for variant calling (GENOME. . . , 2022)

```
@S1_1
AATCCCTGGCTTCAGCCGGGGTTTGGACACACGTTTGGAGGGAGGGACAGCGAATCAAGAAGGGTTATGAAAGGATATGGACTAAGTGATTGAGAA
+S1_1
.++*../(,.-(--$###%,+'##'*-##$$*#&-,"('##'*,%"&(+%-*&%+#$%-..'&,+-$")(*("+)"-,.+,(#&*&*-.+.//,..
```

Figure 11: Example of a read in ".fastq" format

The read sequence and read quality strings were pruned for clarity, showing only the first line of characters.

from the third-generation. The processing pipeline is divided into three main sections: pre-processing, variant discovery, and post-processing.

The description is based on the generic germline short variant per-sample calling pipeline, referred to as Germline Single-Sample Data in GATK4 (Figure 10). Here, germline refers to alignment to a conventionalized genome reference, and single-sample data refers to the analysis made on an individual genome.

Pre-processing

The pipeline's input is composed of raw unmapped reads produced by the sequencers, that usually come in format ".fastq". Each read in the ".fastq" file has 4 lines: the first and third lines are headers, the second line is the nucleotide sequence, and the fourth line is the quality of each base of that sequence. The quality score is related to the probability of the corresponding base call being incorrect; the lower the probability, the higher the quality score. The scores are encrypted with ASCII (American Standard Code for Information Interchange) codes. For the example, in Figure 11, "@S1_1" and "+S1_1" are headers, and the quality codes for the four initial nucleotides AATC, are .++* respectively, which means that their P_error values are 0.05012, 0.10000, 0.10000, and 0.12589.

The reads are mapped to a reference genome to produce a SAM (Sequence Alignment

Map) file, or its binary version BAM, with mapping information of each read. The purpose of mapping reads to a reference genome is to "rebuild" the original DNA sample, from which the molecules had to be cleaved into much shorter sub-sequences. In the processing pipeline, read assembly is the most computationally intensive step. For Illumina short-reads, GATK recommends the BWA-MEM (BURROWS-WHEELER..., 2010) assembler tool. For long-reads, Minimap2 (LI, 2018) has better accuracy and speed.

The output file after the mapping step is of extension ".bam" or ".sam". The first lines (194 for the primary assembly reference) are headers. After them, for each read, the records display read name, read sequence, read quality, alignment information, custom tags, mapped chromosome, start coordinate, alignment quality, and the Compact Idiosyncratic Gapped Alignment Report (CIGAR) string. The alignment flag (Figure 12), the mapped chromosome, and the start coordinate are the mapping results for the read. Figure 13 shows an example of part of a ".sam" file. The first 5 lines are still part of the header of the file. Then the first read S1_1 is mapped with the forward direction (flag 0) to the position 164,654,461 of the RefSeq NC_000001.11, with mapping quality 60. The CIGAR string starts with 5M1D2M1I (explained later), the read sequence starts with AATCCCTGG, and the sequencing base quality starts with .++*../(,.

The CIGAR strings are alignment structures between the reads and the region to which they were mapped on the reference. The CIGAR string contains pairs — number operator — that represent the alignment path between reference and read. The number indicate the count of sequential occurrences of the operator. The operators can be: "M" for match or mismatch; "I" for insertion to the read; "D" for deletion from the read; and "S" for soft-clipping, which results from semi-global or local alignments that lose the edges of the sequences. Less often used operators are "N", used to differentiate introns

Bit		Description
1	0x1	template having multiple segments in sequencing
2	0x2	each segment properly aligned according to the aligner
4	0x4	segment unmapped
8	0x8	next segment in the template unmapped
16	0x10	SEQ being reverse complemented
32	0x20	SEQ of the next segment in the template being reverse complemented
64	0x40	the first segment in the template
128	0x80	the last segment in the template
256	0x100	secondary alignment
512	0x200	not passing filters, such as platform/vendor quality controls
1024	0x400	PCR or optical duplicate
2048	0x800	supplementary alignment

Figure 12: SAM file alignment flags (SEQUENCE..., 2021)

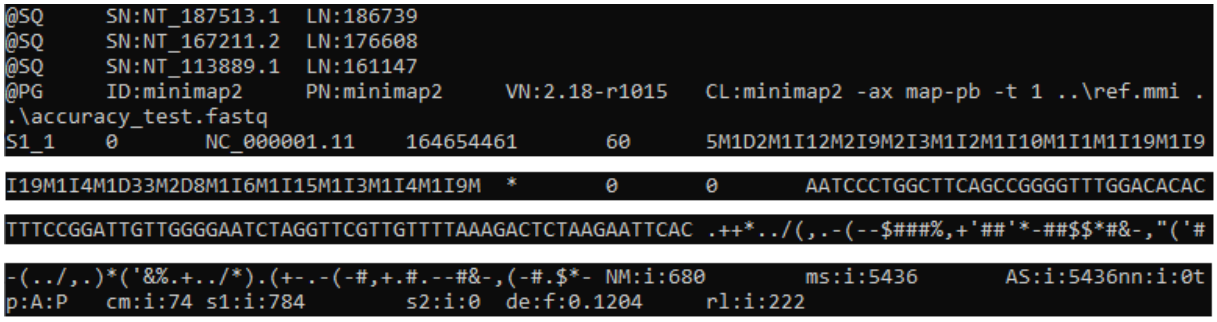

Figure 13: Example of last headers and first alignment report in a SAM file

This is a screenshot of part of a SAM file after the first header lines until, including, the first mapping report. Part of the CIGAR, read sequence, and read quality strings were pruned for clarity.

from deletions; and "H", used for hard-clipping, which works similarly to soft-clipping, but the read string also loses the clipped edges. Section 4.1 presents the genome sequence alignment process and some CIGAR examples.

The last step of pre-processing, which is more specifically for Illumina reads, marks duplicates that may emerge from the replication step during library preparation. This is done by verifying if two or more reads have the same orientation, mapping position and length. After marking duplicates, the base quality scores are re-calibrated using the BQSR tool (BASE..., 2020), which detects the systematic errors created by the sequencer using a machine learning model, to compensate for the tendency of sequencers to overestimate quality scores. PacBio and ONT long-read sequencing technologies do not use PCR amplification, so this step is not required.

Section 2.2.3 will dive more into the pre-processing step of the genome analysis pipeline, as it is the area of focus of this work.

Variant Discovery

This step identifies genomic variations in the sequenced individual using the tool HaplotypeCaller (HAPLOTYPECALLER, 2022). It reassembles the reads in regions presenting variation signs using the *de novo* process, which glues sequences suffix-to-prefix instead of aligning them with a reference; and then calls the variants of each base position. HaplotypeCaller has a very high sensitivity, meaning that it detects many variants that can sometimes be irrelevant. The output file format is VCF (Variant Call Format), which contains variants in each line and columns with chromosome, position, identifier, reference sequence, list of alternative alleles, quality score, a filtration flag, description of the variation, and other information. PacBio suggests using pbsv (PBSV, 2022) to call structural variants joined with DeepVariant (POPLIN et al., 2018) to call

small variants.

Post-Processing

The germline analysis pipeline is focused on identifying variants from the individual that are not common for the species. The undesired variants can be filtered out with the CNNScoreVariants (CNNSCOREVARIANTS, 2020) and the FilterVariantTranches (FILTERVARIANTTRANCHES, 2019) tools. CNNScoreVariants is a trained convolutional neural network model. With relevant variants identified, the funcotator (FUNCtional annOTATOR) (FUNCOTATOR, 2022) links them to their respective functions, using a set of data sources provided by the user.

2.2.3 Reference Guided Read Assembly

There are several ways to re-assemble reads back into the complete genome strand. Two of the most common methods are the *de novo* assembly and the reference-guided assembly. The *de novo* assembly method matches suffix to prefix of different reads, expanding the sequence in a scaffolding way into many long sequences called *contigs*. Genome references themselves, including the human genome reference, and short genomes are assembled with this method. The second method is mapping and aligning the reads to a pre-established reference for the species. This method demands considerably less computation and is more commonly used for assembling long genomes that have a published reference to follow. The reference-guided assembly method is extensively used in clinical sampling of human genomes and exomes.

Reference-guided assembly relies on the similarity between genomes of the same species (about 99.5% for humans). It can generate reference biases; for example, variants that are predominant in an ethnicity might not be represented in the reference. The post-processing step can be tailored to these situations and filter out variations that are actually expected for an individual's ancestry. The reference-guided method also relies on statistical inference. Clinical applications require at least 8 times coverage for each base for an acceptable accuracy (AMARASINGHE et al., 2020). This should cover random sequencing and alignment errors and help identify heterozygous states, where a different variant is inherited from each of the biological parents.

Figure 14 illustrates a reference-guided assembly. The reference sequence is on the top line in red, while the read samples are in gray in following lines. The coverage is represented by each line and each column is linked to the nucleotide of a specific position

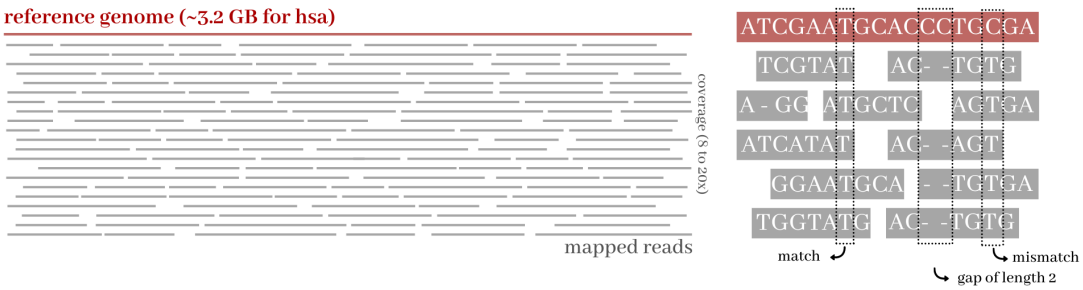

Figure 14: Illustrated reference-guided assembly and statistical inference for base-call

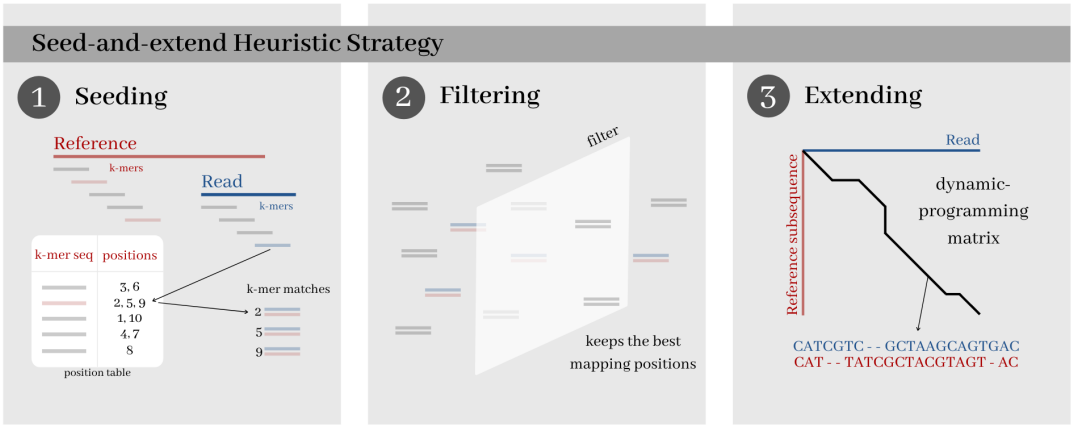

Figure 15: The seed-and-extend read assembly strategy

In the seeding stage, the algorithm finds exact or non-exact matches of very short sequences between the read and the reference. Indexing strategies are applied to the reference to accelerate the seeding stage. Because the seeding stage can result in a great number of false positives, often an intermediate filtering stage is required to reduce the number of selected items. On the extending stage, the Smith-Waterman-Gotoh algorithm (described in Section 4.1.3) is used to align the entire read sequence to the reference region.

in the reference. Variants are determined by dissimilarities between the sequenced sample and the reference genome. In the zoomed figure on the right, under the nucleotides "CC" in the reference, there are two columns missing, with dashes in the reads. That would be identified as a deletion in the sequenced genome. There is also a column where the reference genome has the nucleotide "C" whereas the reads have the nucleotide "T", which would be identified as a homozygous SNV (single nucleotide variant). The variant would be classified as homozygous because all the reads in the coverage have this SNV; if it was heterozygous, around half of them would have the SNV.

There are many methods to map and align reads to a reference. The most successful method has been the *seed-and-extend* approach (Figure 15). It was pioneered by BLAST (ALTSCHUL et al., 1990) and it is used by almost all read assembly tools currently.

Some examples of seeding strategies are: searching for identical matches of fixed length scanning through fixed intervals, used in Bowtie2 (LANGMEAD; SALZBERG, 2012) and Minimap2 (LI, 2018) (more in Section 4.2.2); finding maximal exact matches (MEMs), the longest exact matches between read and reference, used in BWA-MEM (BURROWS-WHEELER..., 2010); non-identical seeds can also be used by mapping algorithms.

Indexing the reference is essential for finding the seeds in short time. Each seeding method can be better suited for different indexation methods. The FM-Index (FERRAGINA; MANZINI, 2000) is a memory efficient indexation method used in BWA-MEM and Bowtie2 that sorts all the rotations of the sequence, and indexes each type of nucleotide in the first and last columns. The hash table indexing method is used in Minimap2; it converts seed sequences into an index for a table containing all positions for that seed in the reference (more in Section 4.2.1).

The seeding process can end up pointing to many regions to align in the reference. Some techniques can be applied to filter out the false positive seeds. Filtering for short-reads is harder because they carry little extra information that would not result already in a complete extension of the read. For example, the work described in Section 3.2 is at the same time a filtering step, that calculates the alignment between read and mapped region in the reference using a very narrow band; and an extending step, because most of the results of these alignments correspond to the optimal alignment. For long-reads, filtering can be applied by extending a small region near the seed and evaluating the score obtained before deciding to extend the entire read (TURAKHIA; BEJERANO; DALLY, 2018), or by grouping the parallel seeds from the same read to find its best mapping position in the reference as in Minimap2 (LI, 2018) (more in Section 4.2.3).

Finally, with the one or few remaining mapping positions, the algorithm needs to align the read to the mapped region in the reference to identify potential variants. For this alignment, a matrix is adopted, with the reference and query (read) sequences positioned in the matrix' axes. The alignments that are performed can be global (when both sequences are fully aligned to each other), which is preferred when they are similar and have similar lengths; local (only a portion of the sequences is aligned), used to find regions of similarity between two sequences; and semi-global (alignment is extended from a fixed point between the two sequences and stops when the similarity ends), commonly used in the *seed-and-extend* method.

The Smith-Waterman-Gotoh (SWG) algorithm has quadratic time and memory complexities in relation to the input's lengths. Because of that, the assembly tools usually do

not calculate the entire SWG matrices for this step. Since the alignment path is expected to stay around the anti-diagonal of the matrices, different banding techniques can be used to limit computation to cells around the anti-diagonals, reducing the complexities to a linear relation; they can adopt fixed bands or dynamic bands. Fixed bands limit the calculation to cells in a fixed area of the matrix (FUJIKI et al., 2020). Dynamic bands try to follow the deviation of the alignment path (LI, 2018). Other algorithms compute overlapping quadrants of the matrix, named tiles, one at a time, limiting the memory usage to a constant value, independent of the input's lengths (TURAKHIA et al., 2019).

Chapter 4.2 describes the specific algorithms used in the seeding, filtering and extending steps of Minimap2. Chapter 4.3 describes a hardware implementation of the alignment process that is taken in the extending stage. These two are combined in this work to provide an acceleration for Minimap2.

2.3 FPGA Acceleration

Currently, almost every 32-bit and 64-bit processor uses reduced instruction set computer (RISC) instructions to communicate software and hardware. It is a type of instruction set architecture (ISA), where instructions are typically as simple as microinstructions, being executed directly by the hardware. In RISC, recently executed instructions are stored in a fast memory called cache, since they are more likely to be reused. This architecture adds extensive flexibility to general purpose processors, but also increases considerably the number of clock cycles required to perform the same task as an ASIC.

The computers' most basic components are the transistors. Their purpose is to reliably and accurately control electric currents, used to switch or maintain voltage representing zeroes or ones in binary logic. Specific combinations of transistors in a circuit can create logic gates for conjunction, disjunction, and negation. Combined logic gates can perform all sorts of more complex calculations. Transistors are fabricated with a semiconductor like silicon, treated with elements that give it an electron emitting or electron absorbing characteristic. Arranging these types of semiconductors in layers gives transistors the ability to control the current propagating between its terminals based on the voltage applied to its gate.

In 1965, Gordon Moore first predicted that the transistor density on processors would double every year, and in 1975, he revised it to be every two years. His prediction was so precise that it became known as Moore's Law. From around 2000, Moore's Law began to slow down, and by 2018 there was a 15-fold gap between the predictions and the technology at that time. This gap is only going to increase as CMOS (complementary metal-oxide-semiconductor) technology approaches fundamental limits, and the costs to deal with this skyrocket (Figure 16).

In parallel to that, Robert Dennard stated that as transistor density increased, power consumption per transistor would decrease, allowing power consumption per mm^2 to stay nearly constant. With power density increasing, "Dennard scaling" also began to slow in 2007 and became almost nonexistent by 2012 (Figure 17).

The slowdown of Moore's Law and the end of Dennard scaling induced architects to exploit better data parallelism. They developed a branch prediction strategy to keep modern processor cores' pipelines full. However, the average misprediction rate of 19% (on Intel Core i7 in this example) made the energy efficiency of modern processors even worse, as additional energy is needed to restore the initial state.

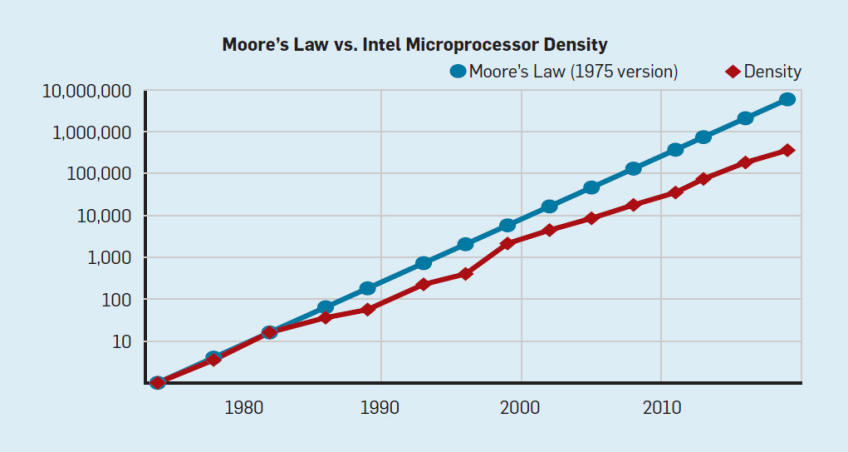

Figure 16: Moore's Law vs. Density (HENNESSY; PATTERSON, 2019)

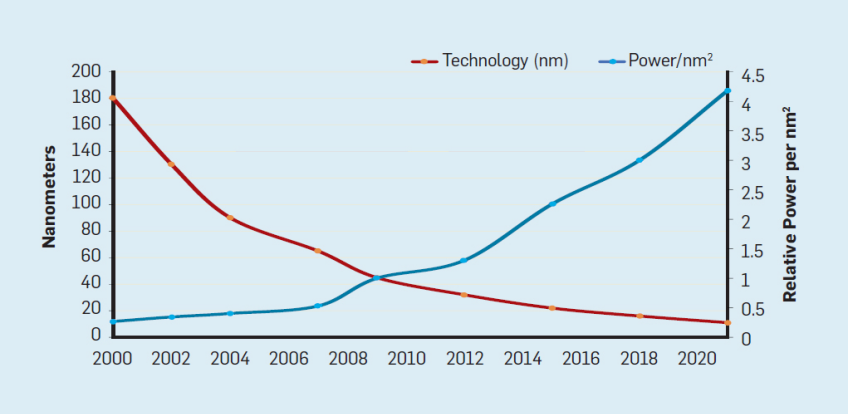

Figure 17: Transistors' length vs. power/nm^2 (HENNESSY; PATTERSON, 2019)

The multi-core (more than a single CPU) approach allowed the programmer to set the best use of thread parallelism for each implementation, but did not solve the energy consumption problem, since every active core consumes power whether or not it is contributing to boost the performance. Also, according to Amdahl's Law, the speedup from parallel computing is limited by the portion of the program that is sequential.

Quantum effects on extremely small transistors lead to high thermal dissipation and CMOS technology is approaching its fundamental limits (i.e. reaching lengths equivalent to a few atoms), but there's still a growing need for computational power. Genomics is an example of it: the Illumina NovaSeq 6000 system can sequence about 48 human whole genomes at 30x genome coverage in about two days. However, analyzing (performing assembling and variant calling to) the sequencing data of a single human genome requires over 32 CPU hours on an Intel Xeon processor, 23 of which are spent on read assembly (GOYAL et al., 2017).

Domain-specific architectures (DSAs) are one of the best solutions to this. They are architectures tailored to a specific problem domain, which in this case would be genome assembly. Some examples of DSAs are GPUs and neural network processors. DSAs can exploit a more efficient type of parallelism for the domain, can make more effective strategies for memory access, can use less precision when adequate, and are energy efficient. FPGAs are programmable hardware used on DSA development that constantly appear on genome alignment acceleration research and were argued to be the most efficient option for the genomics domain (GUO et al., 2019).

ASICs are able to deliver a higher performance when compared to CPUs or DSAs (GPUs and FPGAs), because their operating frequency can be adjusted and optimized to the longest path of the circuit, becoming over 3 times higher than the operating frequency of FPGAs (KAPLAN; YAVITS; GINOSAR, 2019); and because the components can be distributed on the chip in the most efficient manner, under same technology. However, the fabrication cost for ASICs is high, requiring a high volume production to become viable. Since the genomics field is evolving extremely fast, and new algorithms are published every year, it is not prudent to adopt a fixed solution.

2.3.1 Field Programmable Gate Array

Before explaining FPGAs, some basic electronic components need to be introduced. Lookup Tables (LUTs) are tables that produce an output based on the input values. They act like logic gate circuits and, in great numbers, can produce any combinational logic function and even perform very complex computations. LUTs can be looked at as being very small RAMs that are loaded with data when configuring the FPGA. The input would represent a storage address and the output would be the stored data. Lookup tables reduce significantly computation time as they can produce the same results of a circuit that has many logic gates. They are also components that attribute a re-configurable aspect for FPGAs.

Flip-flops (FFs) are logic circuits that work as 1 bit memories. Clock pulses can keep or alter their output value, depending on the value set on the input. A series of flip-flops can constitute a register that also has similar characteristics to memories. A latch is a circuit that registers data if the input has ever been at a high value, keeping output high after this occurs, and only going back to zero if reset is activated. Multiplexers (MUXes) are able to switch the output to one of its multiple inputs through control signals.

FPGAs are, in a simplified way, ICs that contain configurable blocks of logic and

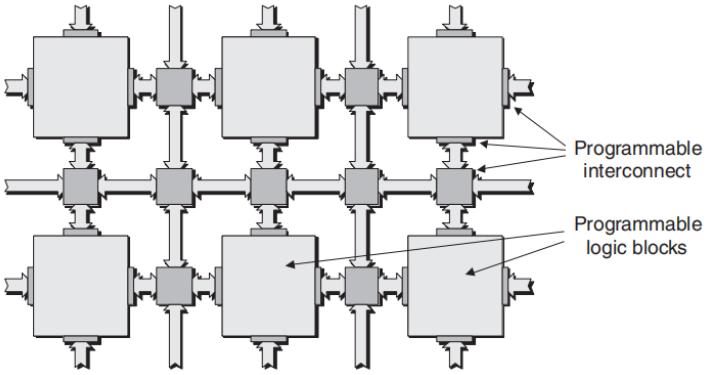

Figure 18: Simplified illustration of an FPGA architecture (MAXFIELD, 2004)

Switch Matrices are the keys arranged in rows and columns that interconnect CLBs and connect them to the inputs and outputs of the FPGA.

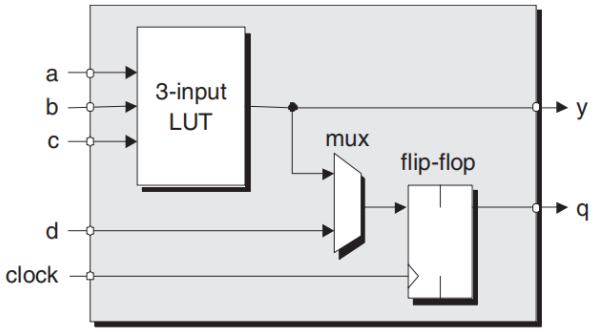

Figure 19: Key elements of a configurable logic block in an FPGA (MAXFIELD, 2004)

configurable interconnects between them (Figure 18). In the illustration in Figure 19, each configurable block is formed by three key elements: a 3-input lookup table, a register that could act as a flip-flop or a latch, and a multiplexer. A 3-input function can be loaded in the LUT, the multiplexer could accept the output from the LUT or another input from the logic block, and the register could be configured to act as a FF or a latch.

Another important aspect of FPGAs is how the logic is loaded into the board. Most FPGAs use, besides LUTs for configurable logic, SRAM (static RAM) configuration cells to hold control signals for interconnections, which is a fast re-configurable method, with a downside of requiring configuration in every system power-up. Another issue with this method is the hindrance to protect the intellectual property, since the configuration file is commonly stored in an external non-volatile memory. Bit-stream encryption can be applied with additional circuitry to avoid this, or Cloud implementations can block access from the end user to private components.

Several modern FPGAs available in the market include in their fabric embedded Block

RAMs (BRAMs) and high-speed Input/Output Blocks (IOB) that make the interface between internal blocks of the FPGA and its external pins. Some FPGAs come with embedded processor cores. Considering FPGAs adopt volatile SRAM for configuration, booting with an external micro-controller is required to load the memory value into the FPGA every time it is initialized. It is also possible to reconfigure the FPGA while it is running a project with a complex system.

The FPGA project developer generates the circuit description at behavior level, at RTL (register transfer level) with a hardware description language (HDL) or with a block diagram graphic description. Most common HDLs are Verilog and VHDL (VHSIC (Very High Speed Integrated Circuit) HDL). Some tools like VIVADO, from Xillinx, can translate high level programming languages such as C and C++ into HDL, process known as high-level synthesis.

On one hand, it is considerably easier and faster to design FPGA projects than ASIC projects, since no complex backend design steps (e.g. layout generation and layout optimization) is needed, and the reprogrammability aspect allows the design to promptly adapt to the market changes. On the other hand, FPGA projects are limited by on-chip resources and mostly don't reach the same level of ASIC performance. Manufacturing costs per chip are lower for ASICs in large production runs, but are prohibitively high for small reproductions. FPGAs also have high manufacturing costs, but they are shared by the high number of users or purchasers.

Designing an FPGA accelerator takes a longer time, and is usually more expensive than programming a GPU accelerator, although it may deliver, in general, faster processing. GPUs are considered still efficient, when the related application present strong data parallelism. According to the authors of (GUO et al., 2019), FPGAs may achieve better performance than GPUs in several applications, as in the one considered in this dissertation.

2.3.2 Cloud FPGAs (the AWS F1 Instance)

FPGA designs can be implemented in a physical board or, nowadays, even in the Cloud. Genomic tools, specially read assembly tools, deal with great amounts of data, requiring processors to have higher volatile and non-volatile memory capacities. FPGA boards with high resource count can be very costly and inaccessible for development projects. Even if the board is accessible, developers must deal with the hardware-software interface configurations and with system maintenance for new platform support tools.

Cloud computing services can reduce the initial cost by charging a smaller price for usage time. Designs that are implemented in popular Cloud servers, such as Amazon Web Services (AMAZON..., 2022c) and Google Cloud (GOOGLE..., 2022); and that use more unique processors, such as FPGAs and GPUs; have been increasingly used in the research environment, because they can be more easily replicated and more people can avail the results. Considering the reasons above, this project was developed for, and was implemented on, a Cloud machine containing many CPUs and an FPGA.

Amazon Web Services (AWS) is a Cloud platform that offers technological services through the web. AWS provides on-demand delivery of IT resources over the internet with pay-as-you-go pricing. On-demand Instances are more suitable for short-term, irregular workloads that cannot be interrupted, which is typical for development and research projects. A wide variety of Instance types are available, with different number of CPUs, RAM and storage capacities, and network and data transfer bandwidths (AMAZON..., 2022b).

Some Instance types offer accelerated computing with hardware integration. The F1 Instances are the only AWS Instances that contain FPGAs. There are f1.2xlarge, f1.4xlarge and f1.16xlarge Instances, in increasing performance potential, with their respective resources listed in Table 3. For the purposes of this dissertation, the f1.2xlarge will suffice.

The f1.2xlarge Instance has a server with eight CPU Cores and one Xilinx UltraScale+ VU9P FPGA in a separate board. Since this Instance is the one used in this dissertation, from this point on, it will be referred simply as AWS F1, unless otherwise explicitly

Table 3: Resources available on AWS F1 Instances

Instance	FPGAs	vCPU	Mem (GiB)	SSD (GB)	Network Performance (Gbps)
f1.2xlarge	1	8	122	470	10
f1.4xlarge	2	16	244	940	10
f1.16xlarge	8	64	976	4 x 940	25

stated. The server is equipped with 4 DDR4 (Double Data Rate) channels, providing a bandwidth of 4x16 GiB/s, each accessing a 72-bit wide ECC-protected memory (48 Gb/s bandwidth). The interface with the host is done with dedicated PCIe Gen3x16 connection (30 Gb/s bandwidth).

The advantages of AWS F1 Instances with respect to FPGAs are that they already come integrated to a CPU host server with fast interface and drivers installed; AWS FPGAs are pre-loaded with a Shell that wraps the kernels (HDL synthesized hardware in FPGA that run algorithms and are called via OpenCL functions) to fit the FPGA's I/O range, presenting a standard AXI (Advanced eXtensible Interface) to the kernels. The VU9P FPGAs are very powerful, and have more SRAM capacity than FPGAs currently available at the author's research institution.

The project's development environment is SDAccel (SDACCEL..., 2019) (default changed to Vitis in 2020 (VITIS..., 2022)). On the host CPU, the custom application (written in C/C++) interacts with the FPGA by using the OpenCL API (Application Programming Interface). OpenCL Runtime manages and services the requests sent to the FPGA. The drivers handle the PCIe transfers between the host and devices. The FPGA comes preloaded with the necessary logic to DMA (Direct Memory Access) the data in local DDR memory. The custom kernels read this data, process it, and write the results back to DDR, using standard AXI-4 (Figure 20).

The steps for hardware development in AWS are:

- Create SDAccel/Vitis kernels from C/C++, OpenCL, or RTL models; the resulting containers (".xo" files) contain kernel XML (eXtensible Markup Language) metadata, RTL files, and Vivado IP project;

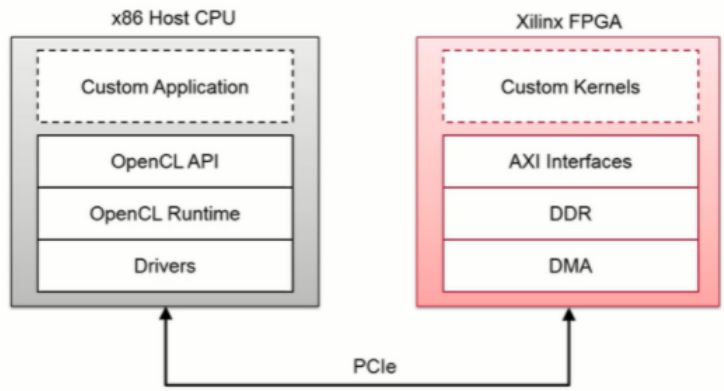

Figure 20: CPU-FPGA interconnection in AWS F1 Instances (DEVELOPING..., 2022)

- Compile the platform (the SDAccel/Vitis compiler links the kernels and other required hardware components), instantiate the kernels and the F1 Shell, generate DDR interfaces and interconnect logic, make all the necessary connections, run synthesis and place&route, resulting in a ".xclbin" binary file;
- Create the encrypted Amazon FPGA Image (AFI), which is stored by an AWS back-end service.

VU9P is a stacked silicon device, meaning that a silicon interposer connects 2 or more FPGA dies (Super Logic Regions). SLR is a slice of a device containing a subset of its resources. VU9P has 3 SLRs, and each has access to one or more DDR interfaces. The Shell includes the PCIe link and hardware necessary to transfer data between the host and kernels in the CL (Custom Logic) region (Figure 21). The Shell's clock runs at a fixed 250 MHz. The CL supports clock frequency of up to 500 MHz.

The kernels access DDRs via AXI4 MM (Memory Mapped) that have 512 bit busses clocked at 250 MHz. Each DDR is accessible by a maximum of 16 AXI4 busses, so an F1 FPGA can have at most 64 kernels. The DDR global memory has an inherent latency overhead when accessed, both for host via PCIe, and for kernel via DDR memory controllers. One cycle of data transfer takes in total 4 DDR transfers.

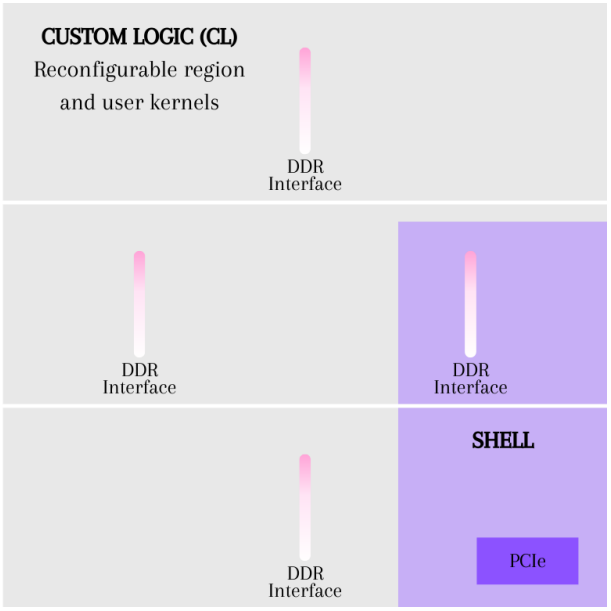

Figure 21: The AWS F1 Instance's VU9P FPGA with 3 SLRs, 4 DDRs, and AWS Shell

2.3.3 The OpenCL Tool-Set

OpenCL is a tool-set that augments C and C++ languages with data types, data structures, and functions for building parallel programs to run on high-performance processors (OPENCL..., 2011). It is a standardized interface that allows developers to program different devices without having to learn multiple languages.

In 2008, the company Apple was selling very popular consumer electronic products (e.g. iPhone, iPad, iPod, and MAC). They were built using devices from third party companies, that benefited from this with the rise in market share and developer interest. Apple faced the need for a common interface that suited any device, so together with its vendors, Apple formed the OpenCL Working Group, one of many in the Khronos Group (a consortium of companies for graphics advancement).

OpenCL targets platforms as the one presented in Figure 20, which describes a hardware implementation in FPGA, but the device instances could be instead, for example, a set of GPUs running concurrent code. A custom application in C/C++ runs in the host, along with the acceleration platform's execution commands and its device's functional code. In the case of GPUs, this code corresponds to C/C++ blocks, while for FPGAs it consists of HDL compiled binaries. OpenCL is a tool-set with a large number of specific commands; the implementation's host in this research practices the following steps with their corresponding commands/functions:

- **Create a platform structure:** with the software development kit (SDK) previously installed on the machine, the vendor is informed (e.g. Xillinx, Nvidia, etc.) in this step; the cl_platform_id data type and the clGetPlatformIDs and clGetPlatformInfo functions are used;
- **Create a device structure:** linked devices from the specified vendor are identified, and one or more are chosen for the application; the cl_device_id data type and the clGetDeviceIDs and clGetDeviceInfo functions are used;
- **Create a context:** devices that work together are grouped into a context, that will have a command queue designated to it; the cl_context data type and the clCreateContext function are used;
- **Create a command queue:** various types of commands can be en-queued to the device with this data structure, from transferring buffer objects to executing kernels; the queue also synchronizes commands by using event flags, and by deter-

mining whether the commands are ordered by FIFO (first in, first out) or not; the cl_command_queue data type and the clCreateCommandQueue function are used;

- **Create a program:** programs are containers of kernels; both of them store executable code, but kernel represents a single function whereas program can represent more than one function; at this step, the binary file with instructions to build logic in the FPGA is read and loaded in the memory; the cl_program data type and the clCreateProgramWithBinary, clBuildProgram, and clGetProgramBuildInfo functions are used;

- **Create kernels:** kernels can be associated to a specific memory bank, such as one of the DDRs from the AWS FPGA (see Section 2.3.2); the cl_kernel and cl_mem_ext_ptr_t data types and the clCreateKernel function are used;

- **Create buffers:** one or more buffers (regions of a memory used to temporarily store data) are reserved in the device for it to read or write input and output data; this step requires maximum data size and memory bank information; the cl_mem data type and the clCreateBuffer and clEnqueueMapBuffer functions are used;

- **Transfer data:** data is written into or read from the buffers by the host; this is a command order sent to the command queue; the clEnqueueWriteBuffer and clEnqueueMapBuffer functions are used;

- **Set the kernel arguments:** any I/O configured kernel argument is set at this stage; the clSetKernelArg function is used;

- **Execute the kernels:** this is a kernel execution command order sent to the command queue; the clEnqueueTask function is used.

3 RELATED WORK

Thousands of relevant works related to genomic read assembling can be found in the literature. This literature review is focused on papers that use hardware designs (e.g. GPU, FPGA, KNL, and ASIC) to achieve some improvement with respect to their software counterparts. The first two works accelerate short-read assembly algorithms and the other works are related to long-read assembly algorithms.

3.1 SWG on FPGA for Short-Reads (KOLIOGEORGI et al., 2019)

This work consists of accelerating the short-read assembly program Bowtie2's alignment step with FPGA implementation. In Bowtie2, the SWG step takes 60% of the execution time, being 56% for matrix-fill and 4% for traceback. A communication overhead on the hardware-software co-designed architectures was identified and addressed.

The wave-front parallelism is explored by an array of PEs (Processing Elements), with the same length as the read sequence (average 270 bp for their dataset), that fills the matrices in a skewed pattern in $n + m - 1$ steps (Figure 22). The matrices are stored on on-chip memory for use by the traceback step, which is also implemented in hardware.

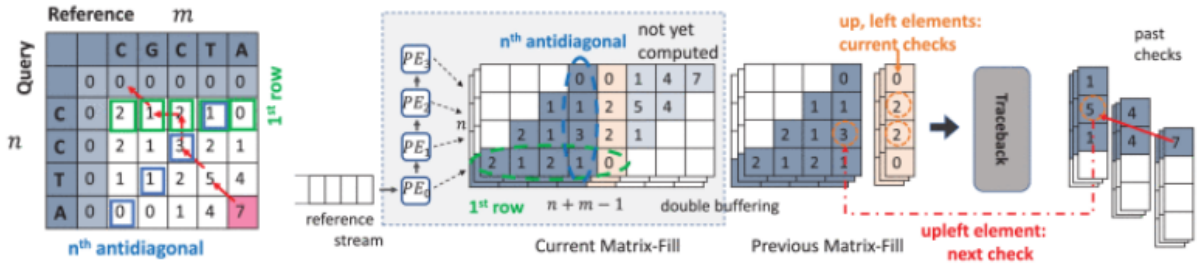

Figure 22: Related work - data-flow for SWG (KOLIOGEORGI et al., 2019)

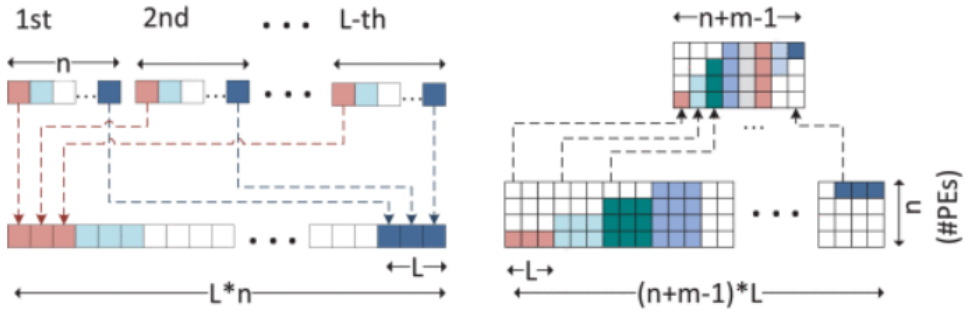

Figure 23: Related work - interleaving read sequences (KOLIOGEORGI et al., 2019)

Interleaving of L read sequences causes a skewed memory pattern for storing score matrices in BRAM.

In their architecture, they noticed that the computation of a cell value requires a chain of multiplexers, which introduces a latency L between consecutive diagonals. To alleviate this, L reads are interleaved in a round robin manner (Figure 23). Double buffering was also implemented to avoid halting matrix-fill operation while streaming data to traceback.

The design was implemented on Xilinx VU9P Ultrascale FPGAs running at 200 MHz and MAX5C DFE (data-flow engine). Compared to Bowtie2 with SIMD optimization running in an Intel Xeon E5-2658A processor at 2.2 GHz, the accelerator achieved 18x speed-up and the integrated system had 35% performance gain. This design is only suited for aligning short-reads, since the number of PEs is directly linked to the length of the query.

3.2 SeedEx: BSW on FPGA for Short-Reads (FUJIKI et al., 2020)

SeedEx is a co-processor implemented on an FPGA that performs seeding expansion with narrow banded SW (BSW) as a filtering algorithm. First, they noticed that in the BWA-MEM algorithm, more than 98% of the seed expansions required a band $w \leq 10$ for short-reads. So they developed a narrow banded accelerator with a three step optimality check to find and rerun the non-optimal expansions on the host.

The first step calculates the theoretical highest score (upper-bound score) and compares it with the score obtained within the band. Thresholds $S1$ and $S2$ are calculated for the smaller region and the bigger region outside the band respectively (difference is due to length asymmetry between query and reference sub-sequence) (Figure 24). The highest score would be the seed's score subtracted by a gap with the band's length and added to a match in the remaining sequence. If the score is under $S1$, it is automatically

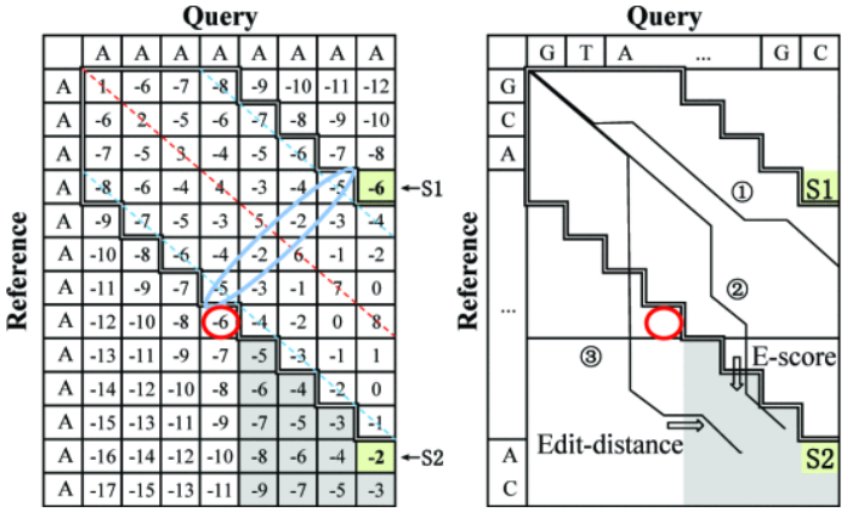

Figure 24: Related work - SeedEx' three step optimality check (FUJIKI et al., 2020)

Thresholds $S1$ and $S2$ from 2 regions outside the band where optimal alignment could occur. On the left is the matrix of thresholds. On the right are the three possible optimal alignment paths with scores higher than $S1$.

sent to rerun on the host. If it is higher than $S2$, optimality is guaranteed. If it is between $S1$ and $S2$, further checking is required.

It is proven by contradiction that if the optimal score is outside the band, then it must be in the shaded region in Figure 24. There are two ways of reaching this region, from above or from the side. The second step performs an E-score check by dislocating the border cells out of the band and performing matches for all subsequent alignments. The maximum value resulting from these interactions is the threshold of acceptance of the band's alignment.

Finally, in the third step they perform the Edit-distance check where they perform a seed expansion with the lower gap penalty and edit-distance scoring on the shaded region using $S1$. The best resulting score is an optimistic threshold for alignments coming into the gray area. If the alignment score in the band passes both E-score and Edit-distance thresholds, then it is optimal.

The design was implemented on AWS F1 instance (f1.2xlarge). They chose a band size of 41, which resulted in a thresholding passing rate of 71.76% and a rerun rate of 1.81%.

12 BSW cores generated the narrow-band and E-check scores. 4 Edit machines performed the Edit-distance check's expansion. With perfect prefetching and appropriate buddering, the memory access time was completely hidden. They achieved 43.9 M seed extensions/s and 1.5 M reads/s (the latest was coupled with seeding acceleration), a 1.3x

speed-up over BWA-MEM and BWA-MEM2.

SeedEx could be repurposed as an accelerator of a filtering step for long-read assembly (it would be equivalent to the BSW modules in Darwin-WGA, see 3.8), but can't be used for long-read alignment for two reasons: the traceback memory requirement would be prohibitive, and long-reads would hardly be limited to narrow bands.

3.3 RASSA: ASIC for Finding Mapping Positions (KAPLAN; YAVITS; GINOSAR, 2019)

RASSA consists of an accelerator for the filtering or pre-processing step of genome assembly. A circuit of resistive memories called *memristors* stores the reference genome and, at the same time, compares it in parallel with chunks (100-200 bp) of the read sequence to measure the Hamming distance between them (Figure 25). A threshold of around 50% determines the mapping position(s) of the read (Figure 26).

RASSA achieved 16-77x speedup over Minimap2 with higher sensitivity. While Minimap2 only mapped about 20% of all reads from their dataset, RASSA always mapped more than 72% of the reads with false positives between 6.9% and 39.2%. However, RASSA's tested circuit only supported small genomes (< 31.5 Mbps), and there was no evaluation for human genome size performance.

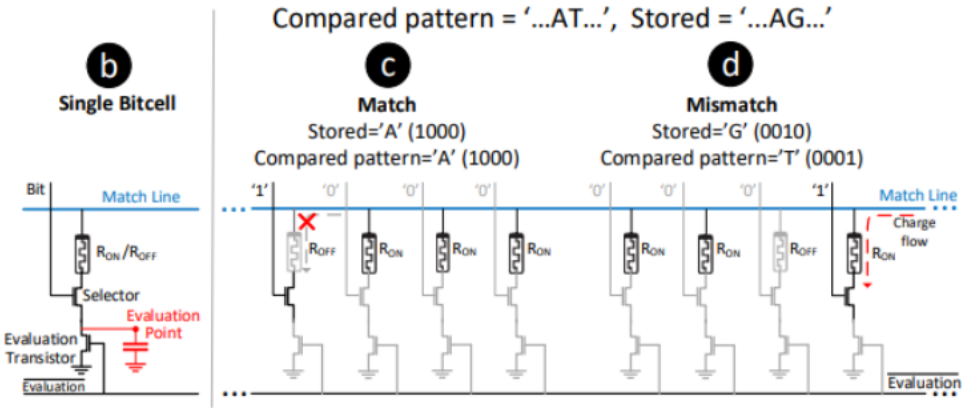

Figure 25: Related work - circuit of memristors (KAPLAN; YAVITS; GINOSAR, 2019)

b) Single RASSA bitcell. **c)** 'A' base from reference stored in-memory compared to 'A' base from read resulting in no charge loss. **d)** mismatch between 'G' and 'T', resulting in match line voltage reduction.

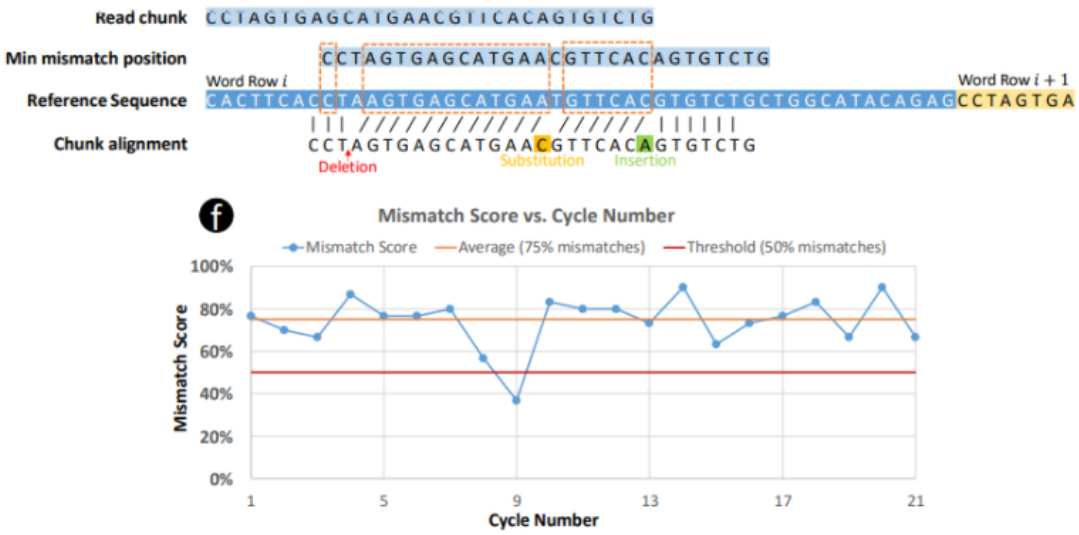

Figure 26: Related work - mapping threshold (KAPLAN; YAVITS; GINOSAR, 2019)

Read is scanned through the reference and when it hits a potential mapping position, the mismatch rate drops under a 50% threshold.

3.4 Minimap2's Chaining Step on FPGA and GPU (GUO et al., 2019)

This project aimed to accelerate the chaining step of Minimap2 in 3 processors: CPU, GPU and FPGA, and compared the results. For the CPU, they binded a thread to a designed core, allocated data of tasks in a neighbor NUMA (non-uniform memory access) node, and implemented SIMD so that the weights between eight pairs of anchors can be computed concurrently. With this, they achieved almost linear speedup to the number of cores (13.9x with 14 cores).

For hardware implementation, they changed the order of the operation sequence because of a loop-carried dependency. Instead of comparing the current anchor with N previous anchors to find a maximum, they compare it with N later anchors and update the temporary value of each of them (Figure 27). They also dispatch a data batch so that, at any clock cycle, every PE can fetch input data. This solves the problem of different input sizes from different tasks (Figure 28).

The architecture was implemented on an AWS F1 Instance, running at 250 MHz with 8 PEs. PCIe bandwidth was the limiting factor. For long-reads, the FPGA accelerator is 277x faster than a single-thread software, 28x faster than 14-core optimized software and 4x faster than their GPU implementation. Since the chaining step is one of the bottlenecks for Minimap2's runtime on long-reads, with this acceleration the bottleneck

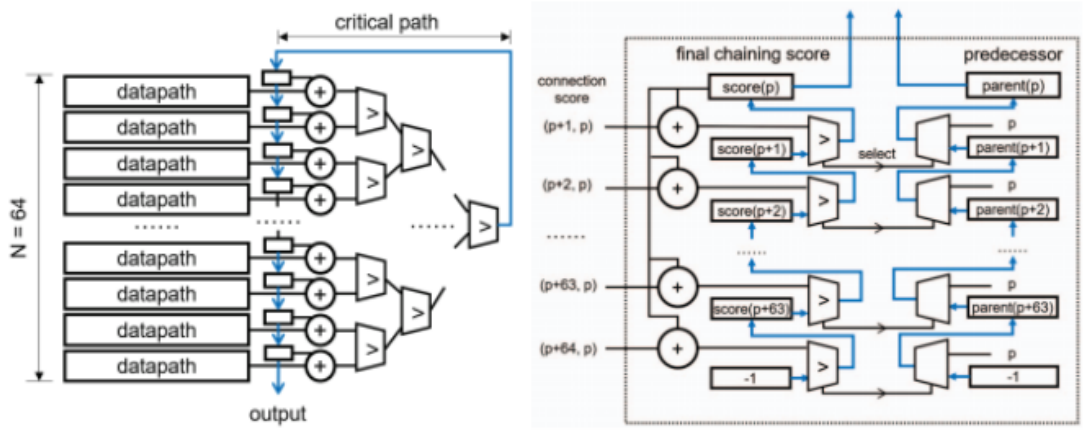

Figure 27: Related work - reordered operation sequence (GUO et al., 2019)

On the left, the original algorithm with a long critical path. On the right, the reordered operation sequence reduces the critical path.

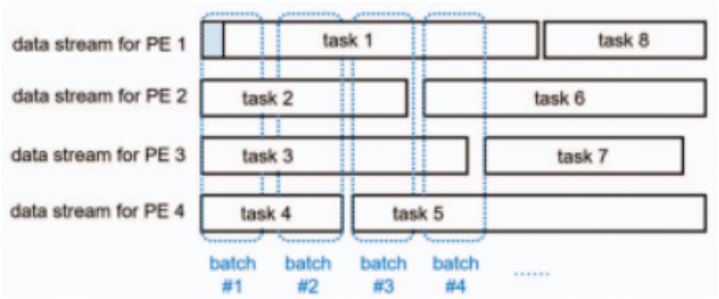

Figure 28: Related work - fine-grained task dispatching scheme (GUO et al., 2019)

is passed to the extending step. This project has not been fully integrated into Minimap2, but the separate module is available on GitHub (MINIMAP2-ACCELERATION, 2021).

3.5 Minimap2's Extending Step on CPU, GPU and KNL (FENG et al., 2019)

This article implemented acceleration techniques on Minimap2's base-level alignment step for long-reads on CPU, GPU and KNL. For PacBio simulated dataset on the human genome, this step consumes 65.42% of the time. They noticed that the Suzuki-Kasahara transformation uses linear arrays to store the matrices, reducing memory usage, but introducing intra-loop data dependency. They proposed a new memory layout with another coordinate transformation that does not alter the memory requirement (Figure 29). The vector load procedure was reduced to a single load instruction (Figure 30).

On their setup, the CPU was running with 20 cores, GPU with 5120 and KNL with

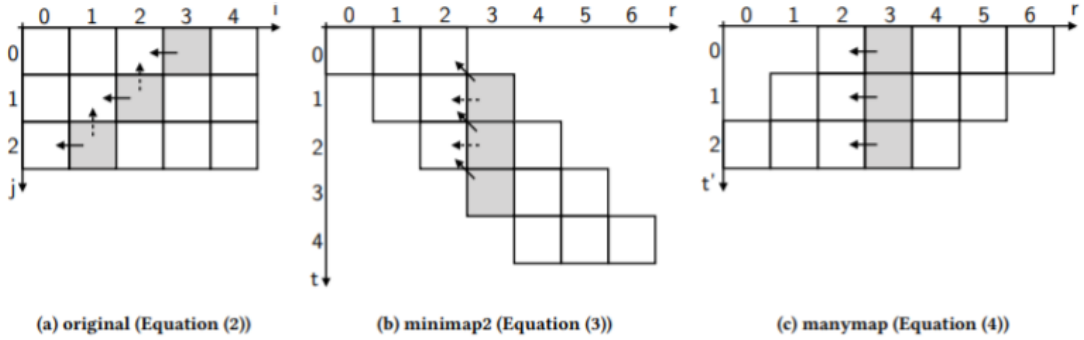

Figure 29: Related work - optimized memory layout for SWG (FENG et al., 2019)

```
xt  = _mm_load_si128(&X[t]);  // xt  = x[r-1][t..t+15]
x1  = _mm_srli_si128(xt, 15);
xt  = _mm_slli_si128(xt, 1);
xt  = _mm_or_si128(xt, tmp);  // xt  = x[r-1][t-1..t+14]
tmp = x1;                     // tmp = x[r-1][t+15]
```

(a) minimap2

```
xt = _mm_loadu_si128(&X[t - r + qlen]);
```

(b) manymap

Figure 30: Related work - vector load reduced to a single instruction (FENG et al., 2019)

64. For matrix fill and traceback, CPU was 1.3 to 4.5 times faster than Minimap2, GPU was 3.2 times faster and KNL was 3.9 times faster. They concluded that, although KNL and GPU both outperformed CPU, a high-end server CPU was still the most efficient platform due to the others' low single thread performance and occupancy issue.

3.6 Darwin: A Hybrid Design for Long-Read Assembly (TURAKHIA; BEJERANO; DALLY, 2018)

Darwin presented the deployment of a complete genome assembly algorithm that is optimized for hardware acceleration. It is divided in two parts: Diagonal-band Seed Overlapping based Filtration Technique (D-SOFT) for the seeding and filtering steps, and Genome Alignment using Constant memory Traceback (GACT) for a second filtering stage and the sequence alignment step. With D-SOFT implemented on CPU and GACT implemented on an Arria 10 FPGA, clocked at 150 MHz, Darwin achieves up to 183.8x speedup over GraphMap (ŠOŠIC et al., 2016) with a single thread on a dual socket Intel Xeon E5-2658 processor (2.2 GHz).

The reference is first indexed with a seed position table, where seed hits are stored

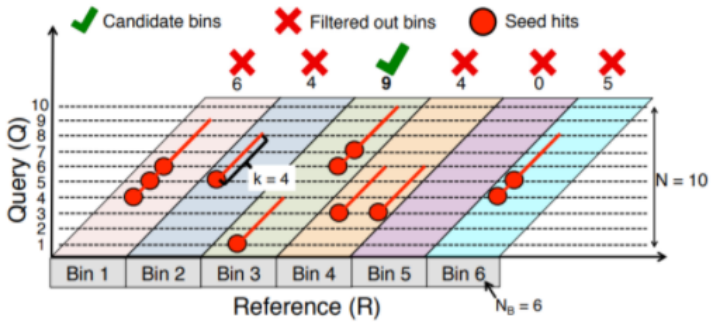

Figure 31: Related work - D-SOFT mapping (TURAKHIA; BEJERANO; DALLY, 2018)

In the example, $k = 4$, $BinSize = 10$, and $h = 8$.

sequentially, which enables faster and fewer memory accesses. High frequency seeds that occur more than $32 \times |R|/4^k$ times, where $|R|$ is the size of the reference, are discarded. Then, in D-SOFT, the reference is divided into bins of size 128. Seeds of size $k = 15$ are collected from a read with $stride = 1$ between start and end. When matched to the position table, the hits are computed so that each bin has a count of unique bases that account for seed hits that may overlap. When the count for a bin first exceeds a threshold $h = 13$, the last hit in the bin is added to the candidate positions list.

Figure 31 shows an example with a reference with 6 bins of 10 nucleotides each. Each of the query's 4-mers, with stride (or dislocation) of 1 is searched in the position table. The first k-mer finds a match in Bin 3, illustrated by a red circle and a line spanning the k-mer's length. When a match is found, the Bin updates its count, without considering overlapping bases that have already been counted in previous k-mers. After iterating through all the query's 4-mers, the last hit from the Bins with counts that exceeded the threshold of 8 are sent as seeds for the next stage. In the example, only Bin 3 passed the filtration process.

GACT performs a second filtration stage and the extending step for the seeds that passed. It computes sub-squares of the SWG matrix, called tiles, with fixed size $T = 320$ and overlap border $O = 128$, creating sequentially a band around the anti-diagonal of the matrix. The first tile starts from the expansion point; it is the only one that starts the traceback from the highest score and it is bigger than the other tiles ($T = 384$) because it works as a second filtering stage. The expansion will only proceed if the first tile's score surpasses $h_{tile} = 90$. The traceback in each tile proceeds until the overlap boundary is reached. The other tiles are clipped in the position where the previous tile stopped backtracking and perform their traceback from the bottom-right cell. Figure 32 presents an left-extension example from the seeding point in green, with $T = 4$ and $O = 1$.

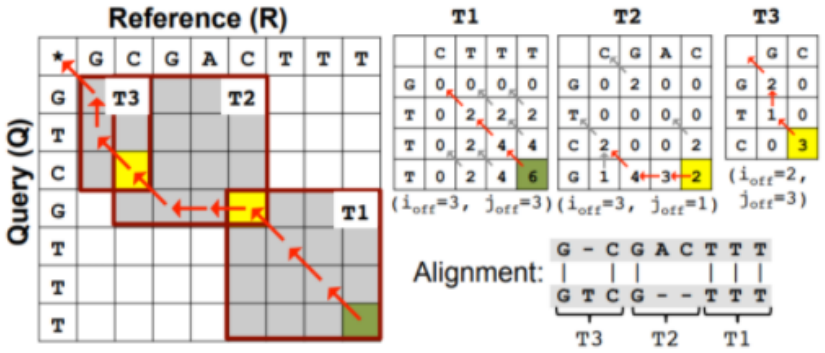

Figure 32: Related work - GACT extension (TURAKHIA; BEJERANO; DALLY, 2018)

In the example, $T = 4$ and $O = 1$.

The memory requirement is fixed at $O(T^2)$, ideal for hardware implementation with limited resources, and the authors affirm that GACT gives optimal results empirically. The hardware design is based on a systolic array. An array of $N_{PE} = 32$ processing elements (PEs) computes blocks of N_{PE} rows from the tile, exploiting the wave-front parallelism of the matrix. In each block, each PE holds its corresponding nucleotide from the query sequence and the reference sequence is streamed through the array. In each clock cycle, each PE computes the three scores and the traceback pointer. Figure 33 shows an example with $N_{PE} = 4$ and $T = 9$.

They synthesized 4 GACT arrays with traceback support (limited by on-chip memory — each PE required 2 KB SRAM, which equals 64 KB SRAM per array), and 36 GACT arrays that do not perform traceback and only compute the first tile's score. The peak throughput was 1.3 M tiles/s. The reported 183.8x speed-up over GraphMap was in relation to single-threaded software, but the hardware design takes multiple kernels. A

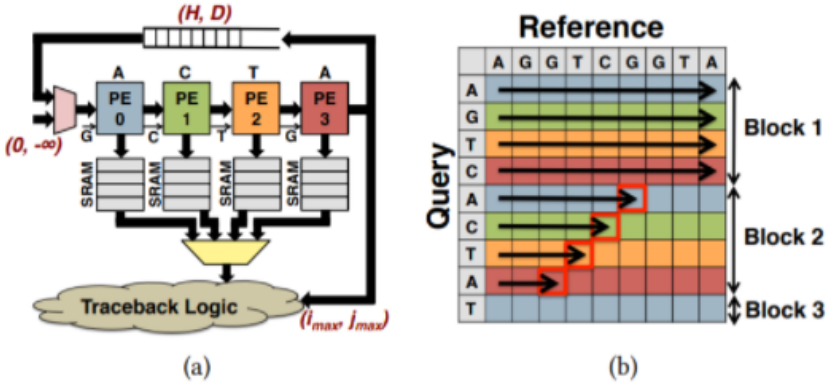

Figure 33: Related work - GACT architecture (TURAKHIA; BEJERANO; DALLY, 2018)

(a) Systolic architecture of GACT with $N_{PE} = 4$. **(b)** Matrix fill scheme for $T = 9$.

one to one comparison should have had a reduced speed-up.

Darwin was published about the same time as Minimap2, and the latter achieved significantly better accuracy and also better performance in comparison to GraphMap. There has not been any performance evaluation of Darwin in relation to Minimap2. D-SOFT was developed to achieve best performance in hardware implementation and, although the authors have simulated a fully accelerated Darwin ASIC, which would provide a great speed-up, in the current implemented FPGA design, D-SOFT runs on software and consumes prohibitive amount of RAM for long references, such as the human genome.

3.7 Improved GACT Algorithm Using BSW (LIAO et al., 2018)

This article presents an improved version of Darwin's GACT accelerator. The authors replaced the tiles with bands without affecting accuracy empirically and achieved a 2.5x speed-up in hardware (Figure 34). They also developed a dynamic programming algorithm for band overlapping. The algorithm consists of searching for regions with high match rates that are more likely to be in the optimal alignment, and overlapping on these instances. Traceback was performed on the hardware.

They used a systolic array of PEs, with length equal to the band, $L = 512$ and $B = N_{PE} = 128$. Score cells were treated as 12-bit integers. Compared to GACT, they reduced the total required clock cycles to 40%. They implemented the accelerator on an Intel 40nm Stratix IV FPGA, EP4SGX230KF40C2, running at 125 MHz, with RIFFA data communication through PCIe interface. No open-source code has been provided by the authors for eventual third part improvements. Later, the authors of Darwin also published an improved architecture that bands GACT and achieves higher speed-up, which will be presented in the next Section 3.8.

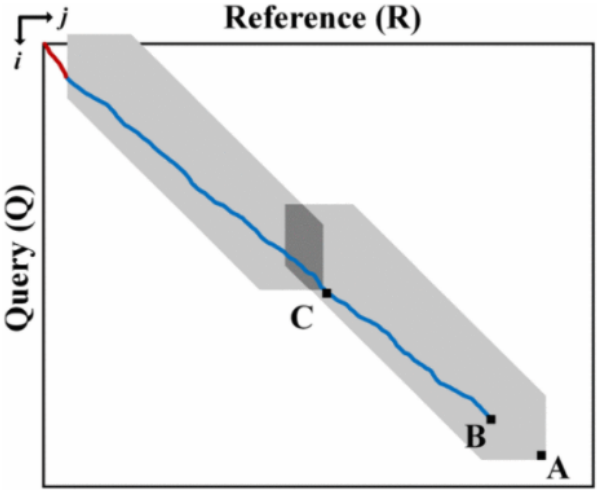

Figure 34: Related work - GACT adapted with bands (LIAO et al., 2018)

3.8 Darwin-WGA: A Hybrid Design for Whole Genome Alignment (TURAKHIA et al., 2019)

The authors of Darwin developed a whole genome alignment (WGA) hybrid design by adapting the D-SOFT and the GACT algorithms used in Darwin. WGA is a method that aligns two complete genomes. It is used for identification and prediction of functional elements (genes and regulatory sequences), for deducing the evolutionary relationship between species, and for ancestral genome reconstruction. Darwin-WGA uses an adapted D-SOFT algorithm for seeding, BSW for gaped filtering, and the improved GACT-X design for aligning. The main differences in Darwin-WGA compared to Darwin are: the query is a long genome sequence, instead of many shorter reads; aligned sequences have a significantly lower similarity for originating from different species; longer indels are expected in the result.

The adaptations aimed at increasing sensitivity, so that true positives could pass on through the filtering stage. D-SOFT uses spaced seed patterns instead of perfect matches to find seeds. A BSW filtering stage, implemented on an FPGA, is added to the design, which also increases the algorithm's sensitivity by allowing gaps in the filtration. Finally, longer gaps require wider tiles to show up in the final alignments, but GACT has a quadratic consumption of on-chip SRAM in relation to tile size. To linearize this ratio, the X-drop algorithm was added to create a dynamic band around the alignment path in the tiles, naming the new algorithm GACT-X. GACT-X is further detailed in Chapter 4.3.

In their architecture, the software runs D-SOFT, configures the arrays, controls the execution of tiles, and reconstructs the alignments from traceback pointers. 50 BSW arrays and 2 GACT-X arrays are implemented on an AWS FPGA, operating at 150 MHz. GACT-X achieved 4.6 k tiles/s (14.6 MB/s) throughput, which is over 4x speed-up over GACT, using 2x less memory. Although Darwin-WGA is not a read assembler, the GACT-X module in it computes the SWG alignment for arbitrarily long sequences, which can be repurposed to accelerate read assembling algorithms.

3.9 Minimap2's Extending Step on FPGA (TENG et al., 2021)

This is a work related to the author's under-graduation project, accomplished with her colleagues Renan Weege Achjian and Caio da Costa Braga, which consisted on accelerating the extending step of Minimap2, using a Xilinx MPSoC Zynq Ultra96v1 board, containing an FPGA and an Arm processor. First, the extending step was identified as Minimap2's main bottleneck when mapping short-reads (58.6% of the total execution time).

One of ksw_extd2_sse41's main loops, commented as "gap left-alignment" in Minimap2's original code, was synthesized into VHDL, using the VIVADO High Level Synthesis (HLS) tool. The SSE instructions were changed to regular C and pragma commands were added to optimize the logic description. From the RTL files, an IP block was generated and the block design was completed in VIVADO to generate the bit-stream.

The host was loaded into the Arm processor, using PYNQ OS (operating system) and Jupyter interface. The intermediate data was stored on an SD (Secure Digital) card inserted in the board. This design only consumed 3% of FFs and 11% of LUTs. The required clock cycles were reduced 155x compared to software. However, the acceleration scheme did not provide performance increase to the ksw function, due to a data transfer overhead that took 99.9% of the time taken by the whole design.

4 ASSEMBLY AND ALIGNMENT ALGORITHMS

This chapter describes the key algorithms that form the basis of the acceleration project of this work: SWG, Minimap2, and GACT-X. The first and last algorithms perform sequence alignment; and the second performs read-to-reference assembly, which also includes a sequence alignment stage. SWG is a classic biological sequence alignment algorithm used in almost any read assembly program, usually with added heuristics to improve performance and reduce memory consumption. The SWG's predecessor is the algorithm Needleman-Wunsch. Minimap2 is the State-of-the-Art algorithm for assembling long-reads to a genome reference. GACT-X has the same function as Minimap2's extending step, and is an FPGA design suited for alignment of long sequences. GACT-X is implemented on an AWS Instance and is going to be used in this project to accelerate Minimap2's extending step.

4.1 Biological Sequence Alignment

All the following methods are based on matrices. The examples given are simple for didactic reasons, but real short and long-reads can measure up to 600 bp and hundreds of kbp respectively.

4.1.1 The Needleman-Wunsch Algorithm

Saul Needleman and Christian Wunsch first published a method to find the best alignments between two sequences using a matrix based dynamic programming algorithm in 1970 (NEEDLEMAN; WUNSCH, 1970). Their method allowed comparisons involving matches, when the letters are identical in the same position of the alignment; mismatches, when the letters are different in the same position; and gaps, when either one of the sequences have an insertion or a deletion of a nucleotide or sub-sequence. In other words, it is a metric that represents the minimum number of "mutational events" required to

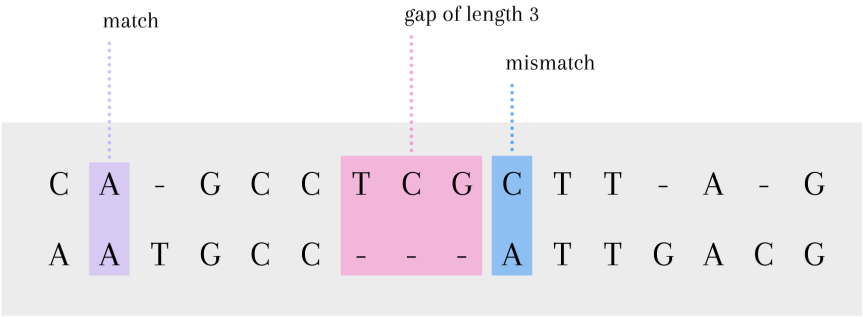

Figure 35: Example of an alignment between two DNA sequences

Alignment between sequences “CAGCCTCGCTTAG” and “AATGCCATTGACG”. Each position of the alignment has to correspond to a match, a mismatch, or a gap. Gaps can also be called indels, insertions or deletions, depending on the situation.

convert one sequence into another. Take the example of an alignment in Figure 35.

Given a matrix S, the axes of the Needleman-Wunsch matrix correspond, respectively, to the query and the reference being aligned, $A = a_1a_2...a_n$ and $B = b_1b_2...b_m$. Matches and mismatches are compensated with function $s(a_i, b_j)$ which has a positive value when it is a match and a negative value when it is a mismatch. Gaps are penalized with a constant w. In dynamic-programming fashion, each cell contains the best score up to that position and the best alignment interaction(s) between every pair of bases that resulted in this score. The equation 4.1 shows how each cell of matrix S is computed.

$$S_{ij} = max\{S_{i-1,j-1} + s(a_i, b_j), S_{i-k,j} - w, S_{i,j-1} - w\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \quad (4.1)$$

As an example, take the alignment between $A = ATGACTCTCAGAC$ (reference) and $B = ATCTCGAGTGAGC$ (query) in Figure 36. A match score is 1, a mismatch penalty is 0.4 and a gap penalty is 0.4. First column and row are filled as gaps, since it’s the only possible interaction between an empty sequence and a non-empty one. The filled Needleman-Wunsch (NW) matrix is shown in the figure. The arrows indicate the origin(s) of the maximum score for each cell and the backtracking generates the best alignment path(s) as shown in purple.

The backtrack pointers indicate that there is a match/mismatch when the path is diagonal, an insertion when the path is vertical, and a deletion when the path is horizontal (when the reference is in the x axis). The computation in $S_{1,1}$ results in 1 due to the match of “A”s, which scores +1 from the upper diagonal cell, resulting in a value larger than the ones coming from above or from the left. After all cells are calculated, a traceback phase,

S		A	T	C	T	C	G	A	G	T	G	A	G	C
	0	-0.4	-0.8	-1.2	-1.6	-2	-2.4	-2.8	-3.2	-3.6	-4	-4.4	-4.8	-5.2
A	-0.4	1	0.6	0.2	-0.2	-0.6	-1	-1.4	-1.8	-2.2	-2.6	-3	-3.4	-3.8
T	-0.8	0.6	2	1.6	1.2	0.8	0.4	0	-0.4	-0.8	-1.2	-1.6	-2	-2.4
G	-1.2	-2	1.6	1.6	1.2	0.8	1.8	1.4	1	0.6	0.2	-0.2	-0.6	-1
A	-1.6	-2	1.2	1.2	1.2	0.8	1.4	2.8	2.4	2	1.6	1.2	0.8	0.4
C	-2	-0.6	0.8	2.2	1.8	2.2	1.8	2.4	2.4	2	1.6	1.2	0.8	1.8
T	-2.4	-1	0.4	1.8	3.2	2.8	2.4	2	2	3.4	3	2.6	2.2	1.8
C	-2.8	-1.4	0	1.4	2.8	4.2	3.8	3.4	3	3	3	2.6	2.2	3.2
T	-3.2	-1.8	-0.4	1	2.4	3.8	3.8	3.4	3	4	3.6	3.2	2.8	2.8
C	-3.6	-2.2	-0.8	0.6	2	3.4	3.4	3.4	3	3.6	3.6	3.2	2.8	3.8
A	-4	-2.6	-1.2	2	1.6	3	3	4.4	4	3.6	3.2	4.6	4.2	3.8
G	-4.4	-3	-1.6	-2	1.2	2.6	4	4	5.4	5	4.6	4.2	5.6	5.2
A	-4.8	3.4	-2	-0.6	0.8	2.2	3.6	5	5	5	4.6	5.6	5.2	5.2
C	-5.2	-3.8	-2.4	-1	0.4	1.8	3.2	4.6	4.6	4.6	4.6	5.2	5.2	6.2

Figure 36: Example of a Needleman-Wunsch global alignment matrix

Needleman-Wunsch global alignment matrix between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$, $MismatchScore = 0.4$ and $GapScore = 0.4$.

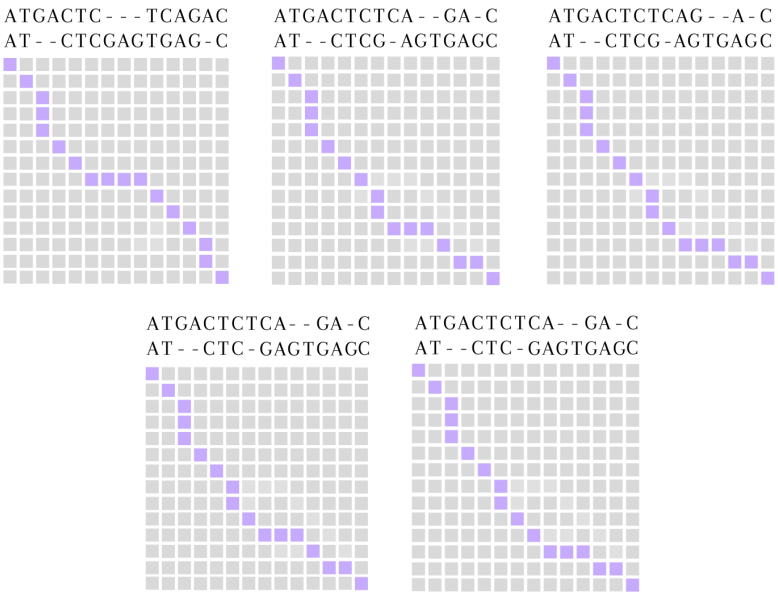

Figure 37: Needleman-Wunsch optimal alignment paths found in the example

Best alignments produced by NW global alignment between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$, $MismatchScore = 0.4$ and $GapScore = 0.4$.

starting from the last cell up until it reaches the origin, marks the possible interactions and finds all the best paths for the alignment. There were in total 5 alignments that obtained the $MaximumScore = 6.2$, as shown in Figure 37. The caption shows the alignment

and the graph shows the feasible path in the matrix that represents the alignment. The CIGAR string for the first alignment would be "2M2D3M3I4M1D1M".

4.1.2 The Smith-Waterman Algorithm

Later in 1981, Temple Smith and Michael Waterman had noticed that the frequency of occurrence of short insertions or deletions in nature was length-dependent, so they added an affine function to represent the gap score (SMITH; WATERMAN, 1981). They have also adapted the method to compute local alignment, which is useful in applications, such as searching for similarities between two sequences that have different sizes or are from different phylogenies.

Now gaps are penalized by the affine function $W_k = a + bk$, where the constant a represents the gap opening score, the multiplier b represents the gap extension score, and the variable k represents the length of the gap. Each cell then has to take into consideration gaps coming from all previous cells in the same line and column. The new equation below shows how each cell of matrix S is computed.

$$S_{ij} = max\{S_{i-1,j-1} + s(a_i, b_j), max_{k \leq 1}\{S_{i-k,j} - W_k\}, max_{l \leq 1}\{S_{i,j-1} - W_l\}\} \\ , l \leq i \leq n \text{ and } l \leq j \leq m \tag{4.2}$$

Using the same example, but aligning two sequences with an affine gap cost: A =ATGACTCTCAGAC (reference), B = ATCTCGAGTGAGC (query), match score is 1, mismatch penalty is 0.4, gap opening score is 1 and gap extension score is 0.3. Figure 38 shows the final Smith-Waterman matrix. $S_{1,2}$, for example, evaluates to -0.3 because the maximum value comes from $S_{1,1}$ scoring $1 - 1.3 = -0.3$, while, from $S_{1,0}$ the score is $-1.3 - 1.3 - 0.3 = -2.9$, from the upper diagonal the score is $-1.3 - 1.0 = -2.3$, and from the top the score is $-1.6 - 1.3 = -2.9$.

This example produced one best alignment with $Score = 3.8$, shown below. The CIGAR string for this alignment would be "2M2D3M3I4M1D1M".

A	T	G	A	C	T	C	-	-	-	T	C	A	G	A	C
A	T	-	-	C	T	C	G	A	G	T	G	A	G	-	C

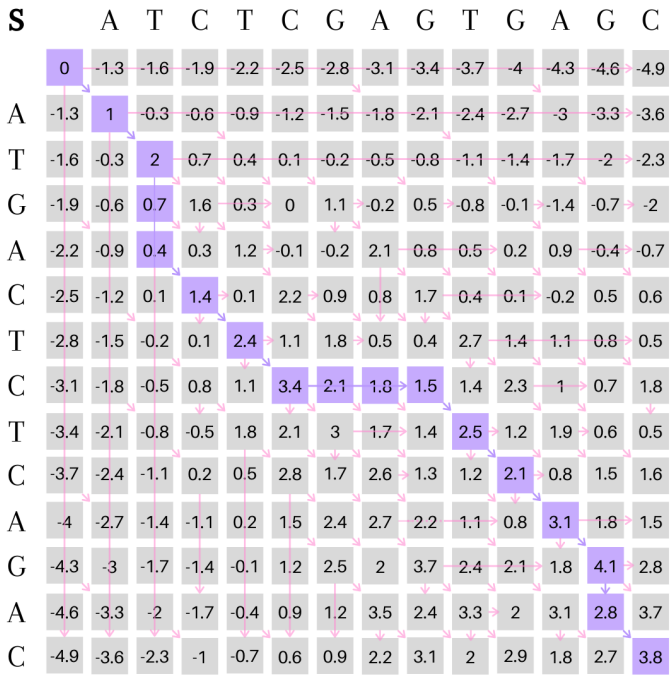

Figure 38: Example of a Smith-Waterman global alignment matrix

Smith-Waterman global alignment matrix between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$, $MismatchScore = 0.4$, $GapOpeningScore = 1$ and $GapExtensionScore = 0.3$.

To turn this process into a local alignment between two sequences, a new arrow pointing to the origin without penalty or reward has to be created (illustrated as a circle in Figure 39). Therefore all the values in the matrix are going to be positive or zero, since the score in the origin is zero. The backtracking should also start from the positions in the table with the maximum score (the purple scores in the matrix), instead of starting from the last cell, and end if it encounters an origin arrow. This will remove the requirement for the alignment to stretch from the beginning to the end of each sequence.

The algorithm found two solutions for best local alignments with $Score = 4.7$, for which, below, the strings are shown:

T	C	T	C	-	A	G		T	C	T	C	A	G	A
T	C	T	C	G	A	G		T	C	T	C	-	G	A

There is also the semi-global alignment, that is frequently used in read mapping algorithms that apply the seed-and-extend strategy. The alignment starts at the origin, that would be the seeding region, and extends until the max score in the matrix, so no origin arrow is used.

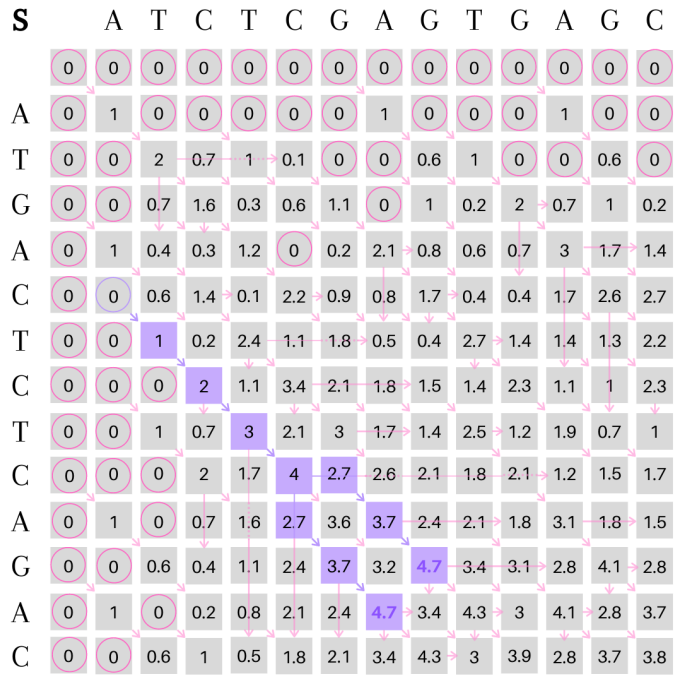

Figure 39: Example of a Smith-Waterman local alignment matrix

Smith-Waterman local alignment matrix between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$,$MismatchScore = 0.4$, $GapOpeningScore = 1$ and $GapExtensionScore = 0.3$.

Although the SW Algorithm generated alignments that were more consistent biologically, it increased considerably the time complexity compared to the NW solution, from $O(MN)$ to $O(MN(M+N))$, where M and N are the lengths of the sequences being aligned. Also, the affine gap method frequently excluded very long gaps that could be generated from a single mutational event, such as crossing-over or transposition of a movable element. Logarithmic, quadratic, or other concave functions can be implemented in the SW algorithm to alleviate this issue, but will further increase the complexity of it.

4.1.3 The Smith-Waterman-Gotoh Algorithm

Osamu Gotoh came up with an elegant solution to the problems mentioned in the previous section and published his algorithm in 1990 (GOTOH, 1990). By adding matrices for each gap type, he could perform more levels of dynamic programming, eliminating the need to check the gap score coming from each previous cell in every iteration. His method could also extend to a piece-wise linear gap-weighting function that could approximate concave curves without increasing the complexity extensively.

For an example with one affine gap function, two extra matrices are added, one for

the horizontal gap score H and another for the vertical gap score V. The cells of the 3 matrices are filled simultaneously, starting from H and V. Cells in H and V only need to compute the best score between opening a new gap from the main matrix S, or expanding the gap from its neighbor. S needs to get the maximum score between a diagonal path from its neighbor, or the scores computed in matrices H and V in the same cell position. The summary of the algorithm is shown in the equations below.

$$\begin{aligned} H_{ij} &= max\{S_{i-1,j} - a, H_{i-1,j} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\ V_{ij} &= max\{S_{i,j-1} - a, V_{i,j-1} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\ S_{ij} &= max\{S_{i-1,j-1} + s(a_i, b_j), H_{ij}, V_{ij}\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \end{aligned} \tag{4.3}$$

Adding two more matrices will increase the cumulative memory requirement to store the traceback pointers by 3 times, but will reduce the time complexity in relation to the SW algorithm from cubic back to quadratic. This is because, with SWG, each cell will only have to consider scores from previous neighboring cells (considering "neighbor" cells from other matrices), instead of all previous cells in the same row/column as in the SW algorithm.

Taking the same example and aligning two sequences with an affine gap cost: A =ATGACTCTCAGAG (reference), B = ATCTCGAGTGAGC (query), match score is 1, mismatch penalty is 0.4, gap opening score is 1 and gap extension score is 0.3. Figure 40 shows the final Smith-Waterman-Gotoh matrices. The first matrix V computes the vertical gaps, the second matrix H computes the horizontal gaps, and the third matrix is the main matrix S. As in the case of $S_{1,2}$ seen in Section 4.1.2 from Figure 38, it also evaluates to -0.3, since it is the maximum among the values obtained from the upper diagonal cell $S_{0,1}$, $-1.3 - 1 = -2.3$, from $V_{1,2}$, -2.9, and $H_{1,2}$, -0.3. On their turn, $V_{1,2}$ is obtained from the max from $S_{0,2}$, $-1.6 - 1.3 = -2.9$, and from $V_{0,2}$, $-\infty - 0.3 = -\infty$. Similarly, $H_{1,2}$ is obtained from $S_{1,1}$, $1 - 1.3 = -0.3$, and from $H_{1,1}$, $-2.6 - 0.3 = -2.9$.

Both the SW and the SWG Algorithms will generate the same matrix S and compute the same alignments, but each with a different strategy. With the SWG Algorithm, any number of affine functions L can be added to approximate a desired curve. The time complexity is now $O(MN(L + C))$, where C is a constant.

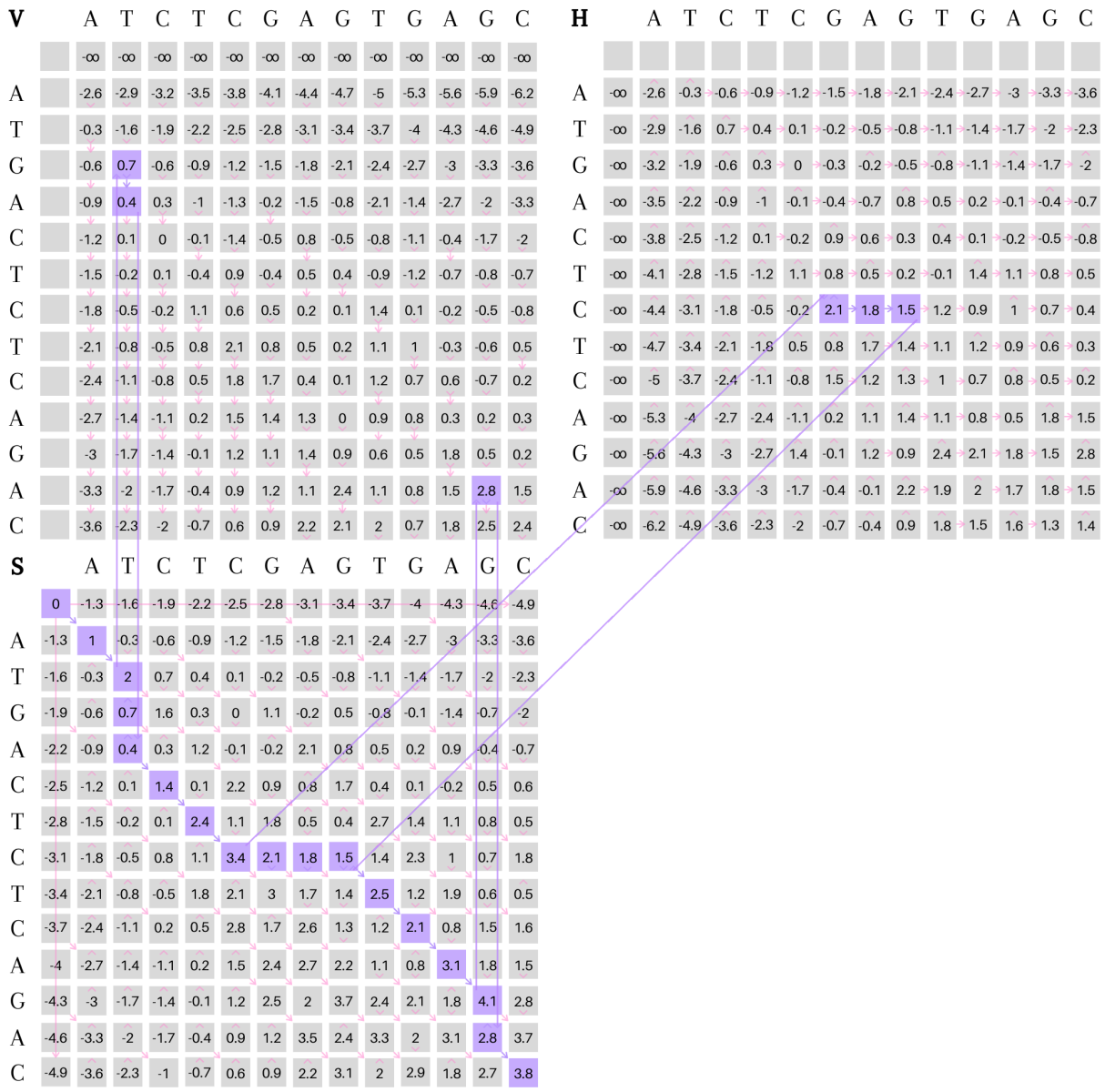

Figure 40: Example with the Smith-Waterman-Gotoh global alignment matrices

Smith-Waterman-Gotoh global alignment matrices between sequences ATGACTCTCAGAC (query) and ATCTCGAGTGAGC (reference), using $MatchScore = 1$, $MismatchScore = 0.4$, $GapOpeningScore = 1$ and $GapExtensionScore = 0.3$.

4.2 Minimap2: Assembly Algorithm for Long-Reads

Minimap2 (LI, 2018) is a State-of-the-Art algorithm formulated by Heng Li and published in 2018 in Bioinformatics. This section will present Minimap2's heuristic strategies to assemble genomes from short- and long-reads, achieving better accuracy and time performance for long-reads than other previous works.

Minimap2 was developed to tackle the problem of assembling long-reads produced by SMRT (Single Molecule Real-Time) sequencing technology and ONT. These technologies can produce reads longer than 10 kbp, at the cost of higher error rate ($\sim 15\%$), but error correction methods have reduced this rate to $< 5\%$. In general, it is not feasible for such data to be processed by mainstream short-read assemblers, mainly due to memory and higher error rate impediments.

The new strategies adopted in Minimap2 allowed it to be over 30 times faster than traditional long-read assemblers, with higher accuracy: Figure 41 (a) shows that, for a given mapping quality threshold (x axis), Minimap2 achieves the highest percentage of mapped reads (y axis).

The algorithms that had been developed specifically to assemble long-reads, such as BLASR (CHAISSON; TESLER, 2012) and BWA-MEM, would often perform five times slower than the ones developed for short-reads, when processing the same amount of data. Minimap2 running on short-reads performs 3 to 4 times faster than mainstream short-read assemblers, such as Bowtie2, and other long-read assemblers, such as BWA-MEM, but it is 1.3 times slower than SNAP. However, it is more accurate than Bowtie2 and SNAP and less accurate than BWA-MEM. The accuracy can be seen in Figure 41 (b), where, Minimap2 shows a second higher percentage of mapped reads, given a mapping quality threshold. The author in (LI, 2018) affirmed that Minimap2 showed to be worse than BWA-MEM in accuracy because the last one tries to locally align a read in a small region close to its mate.

Although Minimap2 uses the typical index and seed-and-extend procedure (see the method in Chapter 2.2.3), each of these steps has had a new improvement that was proven to be more effective. The next topics will present each one of these new adaptations.

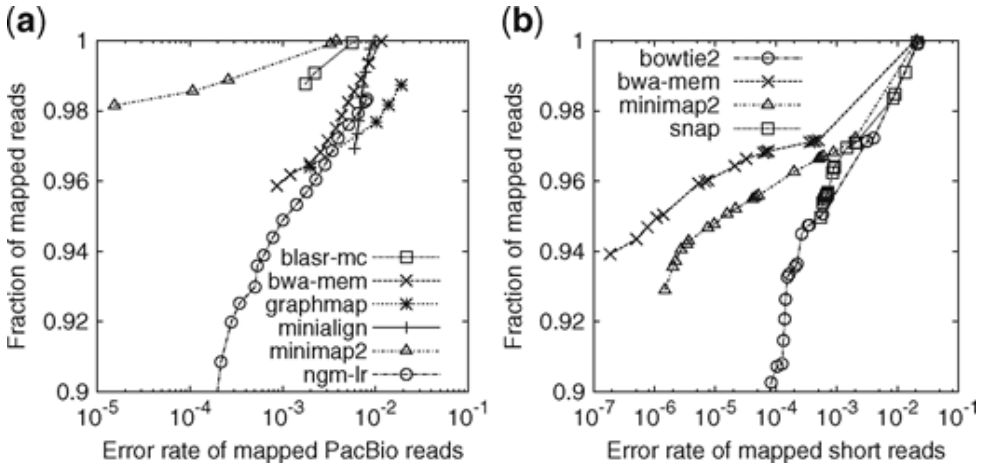

Figure 41: Comparison of mapping performances between read assemblers (LI, 2018)

A read was considered correctly mapped when it overlapped the true interval by at least 10%. **(a)** Long-read assembly evaluation. 33088 $\geq$ 1000 bp reads were simulated. **(b)** Short-read assembly evaluation. 10 million pairs of 150 bp reads were simulated.

4.2.1 Hash Table Indexing

Mainstream aligners often use a full-text index, such as suffix array or FM-index (FERRAGINA; MANZINI, 2000), to index reference sequences. This allows higher seed uniqueness which will reduce the number of unsuccessful extensions. Minimap2 uses a table for indexing fixed length code. The fixed length allows more efficient computation: query seeds with multiple hits can be skipped without affecting the final accuracy, compensating for the advantage that seed uniqueness offers. The author ended up concluding that a hash table is the ideal data structure for mapping long noisy sequences.

On Minimap2, a k-mer is a code sequence of length k. The k-mers are collected from the reference during indexing stage, and later on from the reads during mapping stage, through the same function. A window of size k scans through the target sequence with w steps. The hash function from eq. 4.4 is applied to all these k-mers and their respective inverted Watson-Crick complements (see Figure 2). The sub-sequence or the complement that has the smallest hash is taken as a minimizer and is added to the set of minimizers M (Figure 42). Minimap2 uses $k = 15$ and $w = 5$, but these can be changed by the user.

$$\varphi(s) = \varphi(a_1) \times 4^{k-1} + \varphi(a_2) \times 4^{k-2} + \cdots + \varphi(a_k) \tag{4.4}$$

If $\varphi(A) = 0$, $\varphi(C) = 1$, $\varphi(G) = 2$ and $\varphi(T) = 3$ are adopted, the hash function will always map a k-mer to a distinct 2k-bit integer. Only poly-A will always get zero.

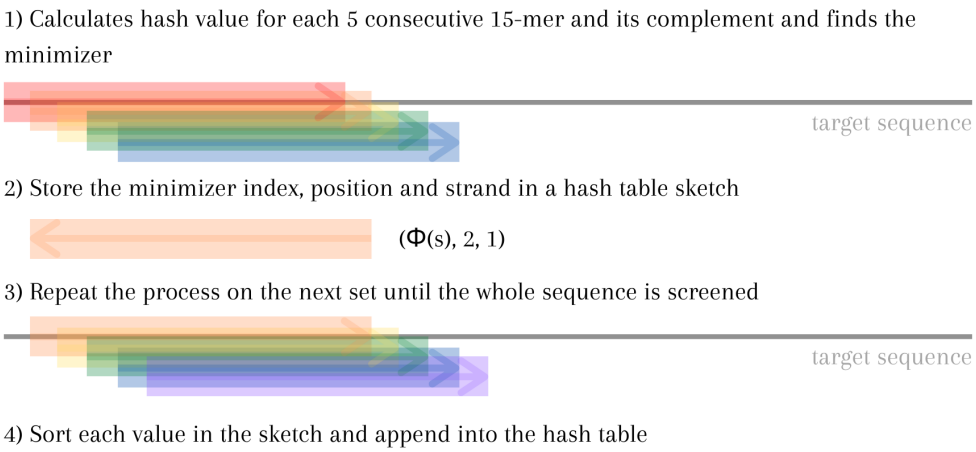

Figure 42: Minimap2's algorithm for collecting minimizers and indexing

In this example, the minimizer is the reverse complement of the orange k-mer. The minimizer index, calculated with function $\varphi(s)$; the choice of minimizer in the set (2); and the reverse complement flag "1" are stored.

Because of this, Minimap2 uses $\varphi' = h \cdot \varphi$, where h is an invertible integer hash function on $[0, 4^k)$. In short, a hash table is a dictionary where the key is the minimizer hash and the value is a set of target sequence index, position of minimizer, and sequence. Each value list is sorted before being added to the hash table to reduce heap allocations and cache misses.

4.2.2 Perfect Match Seeding

For each query sequence (referring to read sequences), Minimap2 takes query minimizers as seeds and finds exact matches, called anchors, in the reference, using the previously indexed hash table (Figure 43). From the figure, it can be noticed that a seed in the read may be anchored in different parts of reference. This step is often used in other assemblers but in Minimap2, the seeds have fixed length and are matched to the reference through the hash table. The seed is skipped when it has too many occurrences in the reference.

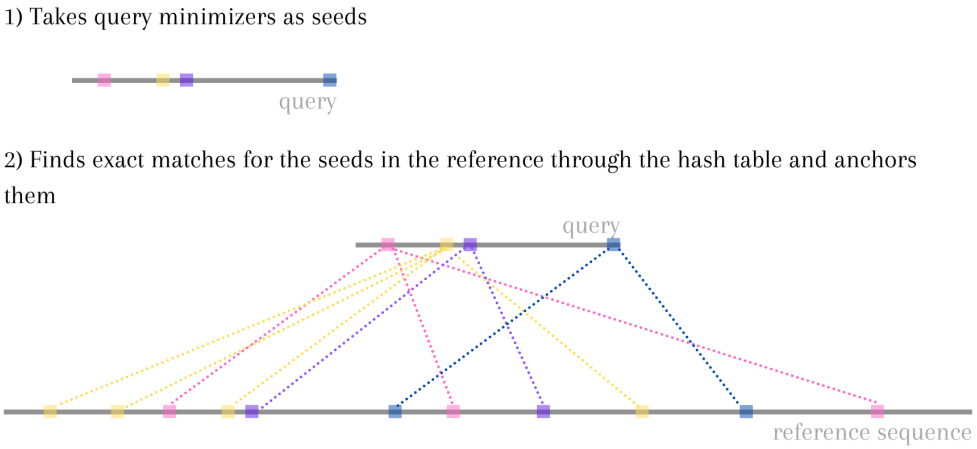

Figure 43: Minimap2's seeding and anchoring steps

4.2.3 Chaining for a Filtering Stage

All sets of collinear anchors are grouped into chains (Figure 44). An anchor is a 3-tuple (x, y, w) that indicates that interval $[x - w + 1, x]$ on the reference matches interval $[y - w + 1, y]$ on the query. The chaining score up to the $i - th$ anchor in the sorted chain is given by the function 4.5. The function can be computed with dynamic programming.

$$f(i) = max\{\{f(j) + \alpha(j, i) - \beta(j, i)\}, w_i\} \tag{4.5}$$

In function 4.5, $\alpha(j, i) = min\{min\{y_i - y_j, x_i - x_j\}, w_i\}$ is the number of matching bases between the two anchors. $\beta(j, i) = \gamma_c((y_i - y_j) - (x_i - x_j))$ is the gap cost and equals to ∞ when $y_j \geq y_i$ or when the distance between two anchors is higher than G. A

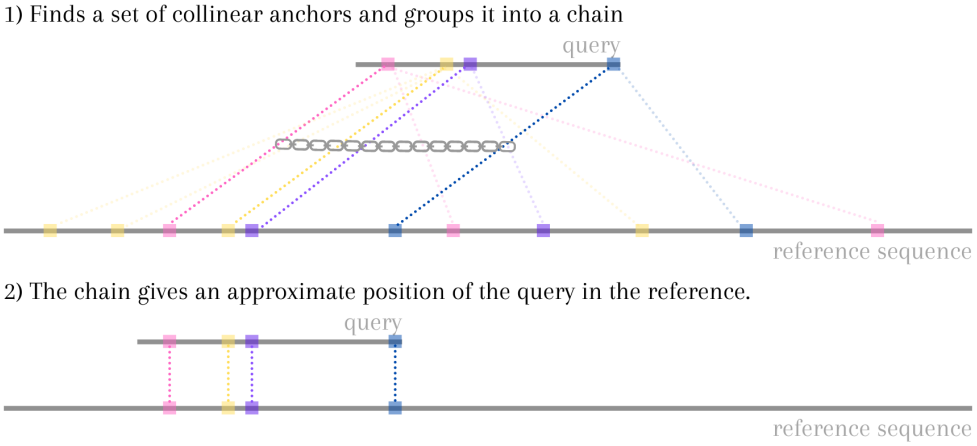

Figure 44: Minimap2's chaining step

gap of length l for the average seed length $\bar{w}$ costs:

$$\begin{aligned} \gamma_c(l) &= 0.01 \cdot \bar{w} \cdot |l| + 0.5 \cdot log_2|l| \ (l \neq 0) \\ \gamma_c(l) &= 0 \ (l = 0) \end{aligned} \tag{4.6}$$

For N anchors, chaining has complexity $O(N^2)$, but is able to take a generic gap cost function, is simple to implement, and is not associated with a large constant like in other algorithms. Minimap2 applies an heuristic strategy to accelerate this step. It limits the number of iterations to $h = 50$, since it is unlikely that an anchor i chained with j has better score when chained against j's predecessors. This reduces the complexity to $O(hN)$ and, according to Minimap2's author, it can almost always find the optimal chain, and even when it fails, the optimal chain is often close.

As mentioned before, the chaining process is a dynamic-programming algorithm, so during chaining, there is a process of backtracking that appends to a chain new anchors in order that provide the best score. When backtracking, anchors are marked as 'used', so that no anchor is used in more than one chain. Sometimes chains can have significant or complete overlaps due to repeats in the reference. To identify the primary chains, all chains are sorted according to their scores, and an empty list Q is created. For each query, if the chain overlaps with a chain in Q by 50% or more, the chain is marked as secondary; otherwise, it is added to Q so, in the end, Q contains all the primary chains.

Although many chains can be detected, usually each read should only be mapped to one place in the reference. The best primary chain is picked for this purpose to be aligned in the next stage. At the end, each read with its associated chain will be mapped to the reference, with each chain composed of several anchors. Sub-sequences between adjacent anchors are called in this dissertation as anchor-separated sub-sequences, and are individually aligned in the Smith-Waterman-Gotoh algorithm, as to be seen in the next section. The sub-sequence in the chain that extends to the right, from the leftest anchor to the end of the sequences is called anchor-extended sub-sequence, and will be used at Chapter 6.1.2 in the GACT-X's implementation.

4.2.4 SSE Optimized Alignment

After getting the primary chains, Minimap2 uses the dynamic-programming-based algorithm Smith-Waterman-Gotoh, with the Suzuki-Kasahara formulation (SUZUKI; KASAHARA, 2018), to perform global alignment of anchor-separated sub-sequences in a chain, and semi-global alignment for the head and tail sequence pairs. In this way, the final result is a local alignment that maps the read on the reference and aligns the read sequence to a reference sub-sequence.

Minimap2 implements its alignment with a 2-piece affine gap cost, as shown in eq. 4.7. In this way, long insertions and deletions, that often occur in complex genomes such as the human genome, can be recovered.

$$\gamma_a = min\{q + |l| \cdot e, \tilde{\mathrm{q}} + |l| \cdot \tilde{\mathrm{e}}\} \tag{4.7}$$

The Suzuki-Kasahara Formulation

Hajime Suzuki and Masahiro Kasahara published in 2018 a reformulation of the Smith-Waterman-Gotoh Algorithm (SUZUKI; KASAHARA, 2018), allowing it to be the fastest SSE SWG implementation, with a 2.1 fold higher performance than that of the fastest implementation for the semi-global alignment of long-reads. They noticed that, as the read length increases, the values of the cells in the dynamic-programming matrix increase. Traditionally, SSE implementations could achieve 16-way parallelization for short sequences, but only 4-way parallelization for long sequences, when the peak alignment score reached 32,767, which was turning the alignment step into a significant bottleneck. This happens because SSE vector instructions are vectors of fixed length with data that have limited size; if the data surpasses the limit value, more cells are needed to represent it, reducing the parallelization capacity of the vector instructions.

Difference Recurrence Relation

Instead of calculating and storing all the values in a matrix, they proposed a "difference recurrence relation", where the differences between the neighboring cells are stored, not the integral value of each cell. In this way, they could guarantee that the values would have a limited range and would be able to be stored as an 8-bit integer, enabling 16-way SSE vectorization.

First consider the following small change to the SWG formulation:

$$
\begin{aligned}
S_{ij} &= max\{S_{i-1,j-1} + s(a_i, b_j), H_{i-1,j} - b, V_{i,j-1} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\
H_{ij} &= max\{S_{ij} - a, H_{i-1,j} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\
V_{ij} &= max\{S_{ij} - a, V_{i,j-1} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m
\end{aligned} \tag{4.8}
$$

With this, four difference matrices are created, as shown in eq. 4.9 and Figure 45.

$$
\begin{aligned}
\Delta H_{ij} &= S_{ij} - S_{i-1,j}, (i \geq 1) \\
\Delta V_{ij} &= S_{ij} - S_{i,j-1}, (j \geq 1) \\
\Delta H'_{ij} &= H_{ij} - S_{ij} \\
\Delta V'_{ij} &= V_{ij} - S_{ij}
\end{aligned} \tag{4.9}
$$

A new intermediate variable I is introduced to represent the diagonal difference $S_{ij} - S_{i-1,j-1}$. The new recurrences then become:

$$
\begin{aligned}
I_{ij} &= max\{s(a_{i-1}, b_{j-1}), \Delta H'_{i-1,j} + \Delta V_{i-1,j} - b, \Delta V'_{i,j-1} + \Delta H_{i,j-1} - b\} \\
\Delta H_{ij} &= A_{ij} - \Delta V_{i-1,j} \\
\Delta V_{ij} &= A_{ij} - \Delta H_{i,j-1} \\
\Delta H'_{ij} &= max\{-a, \Delta H'_{i-1,j} - \Delta H_{ij} - b\} \\
\Delta V'_{ij} &= max\{-a, \Delta V'_{i,j-1} - \Delta V_{ij} - b\}
\end{aligned} \tag{4.10}
$$

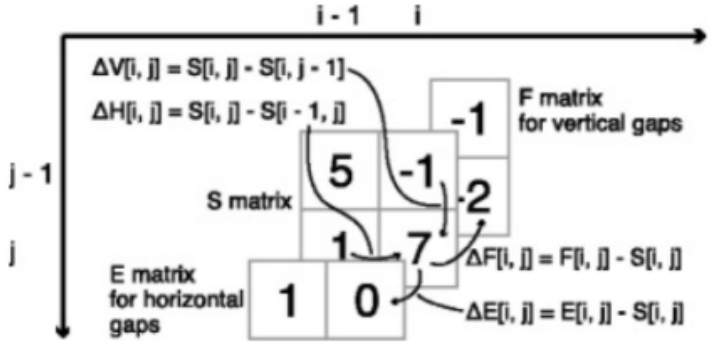

Figure 45: The Suzuki-Kasahara transformation (SUZUKI; KASAHARA, 2018)

In the first transformation of the Suzuki-Kasahara algorithm, the new values ΔH, ΔV, $\Delta H'$ and $\Delta V'$ are stored in four new matrices instead of the original values S, H and V.

All the values in the cells are now limited by:

$$
\begin{aligned}
-a - b \leq \Delta H \leq M + a + b \\
-a - b \leq \Delta V \leq M + a + b \\
-a \leq \Delta H' \leq 0 \\
-a \leq \Delta V' \leq 0
\end{aligned}
$$

, where M is the maximum value of $s(x, y)$, or the match score in nucleotide alignment.

The row-column coordinate is transformed into the diagonal-antidiagonal coordinate by letting $r \leftarrow i + j$ and $t \leftarrow i$. In this way, cells with the same antidiagonal index r are independent of each other, allowing them to fully vectorize the computation of all cells on the same diagonal. Using smaller integer types also reduces the memory requirements. Now all the matrices go through one more transformation:

$$
\begin{aligned}
A_{G_{ij}} &= A_{ij} + 2a + 2b \\
\Delta H_{G_{ij}} &= \Delta H_{ij} + a + b \\
\Delta V_{G_{ij}} &= \Delta V_{ij} + a + b \\
\Delta H'_{G_{ij}} &= \Delta H'_{ij} + \Delta V_{ij} + 2a + b \\
\Delta V'_{G_{ij}} &= \Delta V'_{ij} + \Delta H_{ij} + 2a + b \\
s_G(x, y) &= s(x, y) + 2a + 2b
\end{aligned}
\tag{4.11}
$$

The new recurrences then become:

$$
\begin{aligned}
I_{G_{ij}} &= max\{s_G(a_{i-1}, b_{j-1}), \Delta H'_{G_{i-1,j}}, \Delta V'_{G_{i,j-1}}\} \\
\Delta H_{G_{ij}} &= A_{G_{ij}} - \Delta V_{G_{i-1,j}} \\
\Delta V_{G_{ij}} &= A_{G_{ij}} - \Delta H_{G_{i,j-1}} \\
\Delta H'_{G_{ij}} &= max\{A_{G_{ij}}, \Delta H'_{G_{i-1,j}} + a\} - \Delta H_{G_{i,j-1}} \\
\Delta V'_{G_{ij}} &= max\{A_{G_{ij}}, \Delta V'_{G_{i,j-1}} + a\} - \Delta V_{G_{i-1,j}}
\end{aligned}
\tag{4.12}
$$

Finally, the values can be stored as an array of unsigned integers:

$$
0 \leq \Delta H_G, \Delta V_G, \Delta H'_G, \Delta V'_G \leq M + 2a + 2b
$$

The critical path, or the longest operation dependency chain, is reduced to 4 from 8 in the first difference recurrences transformation, and from 5 in the non-difference semi-global alignment algorithm.

Adaptive Banded Dynamic-Programming Algorithm

In the SK (Suzuki-Kasahara) algorithm, an adaptive banded dynamic-programming strategy is used, so that only part of the cells, specifically those around the antidiagonal, is calculated, since it is the region where the alignment between two somewhat similar sequences is expected to be. A forefront vector of constant width iteratively moves right and down and forms a band, trying to move away from the cells with lower scores. The band is divided into blocks containing vectors calculated in 32 successive updates each (Figure 46).

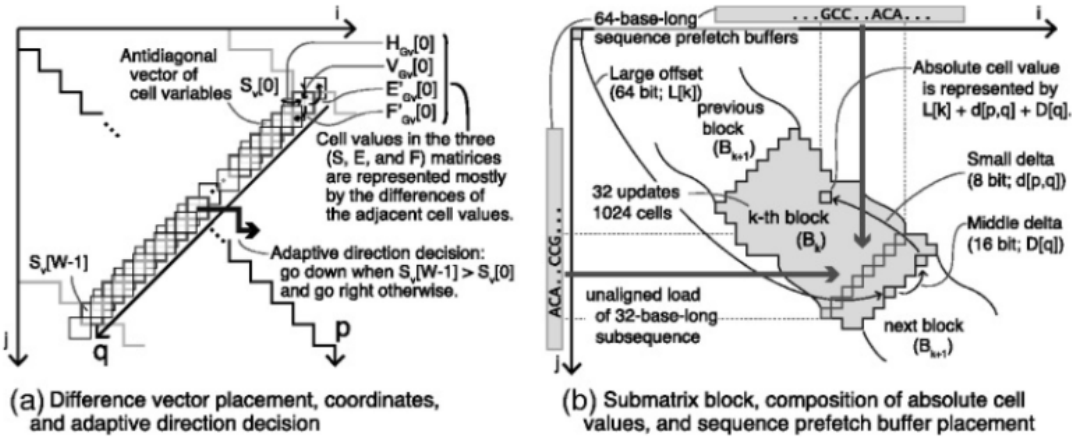

Figure 46: Suzuki-Kasahara's banding strategy (SUZUKI; KASAHARA, 2018)

(a) In the new coordinates, $p = i + j$ corresponds to the position of the vectors, and q is a local position within a vector. The advancing direction (right or down) is determined by the smallest difference between values of cells $S_v[0]$ and $S_v[W-1]$. **(b)** Data structure: each block has 32 vectors and is indexed as k. $L(k)$ is an offset integer that helps calculating the absolute value of a cell in the block.

The Z-drop heuristic

To avoid computing global alignment of unrelated sub-sequences, in query and reference, between two adjacent anchors (e.g. when there is a short inversion), Minimap2 computes an accumulative alignment score and breaks the alignment when the score drops too fast in the anti-diagonal direction. The condition follows eq. 4.13.

$$S(i', j') - S(i, j) > Z + e \cdot |(i - i') - (j - j')| \tag{4.13}$$

, where e is the gap extension cost and Z is an arbitrary threshold. The Z-drop strategy was first used in BWA-MEM and is similar to the X-drop algorithm implemented in BLAST (ALTSCHUL et al., 1990), but allows the presence of a single long gap.

When the alignment is broken, Minimap2 performs local alignment in the same region but with one sub-sequence reverse complemented. This may help identify short inversions that were missed during the chaining process.

4.3 GACT-X: an FPGA Accelerated SWG Implementation with Fixed Memory Usage (TURAKHIA et al., 2019)

As seen in Section 3.8, GACT-X is a hardware aligner, implemented by its developers on an AWS Cloud FPGA, that was replicated in this work to accelerate Minimap2. GACT-X computes the SWG matrix with limited traceback memory for any input lengths. The memory usage is limited because the computation is performed in tiles of fixed size (instead of on a long band as in other alignment algorithms); one tile overlaps a previous one by a certain number of bases to expand the alignment, forming a scaffold band on the SWG matrix. GACT-X differs from GACT from Darwin (Section 3.6) by computing cells within an X-drop band (to be clarified next) in each tile, further reducing on-chip SRAM usage, and allowing tiles to be considerably wider; and by using a different overlapping strategy.

GACT-X's expansion starts from the anchoring point and stops when the last tile's max score is negative or zero, or when the alignment has reached the border of query or target sequences. The overlap system is similar to GACT's, but if the max score cell lies outside the overlap boundary, the next tile's origin will be placed on that cell. In Figure 47, tile T1 performs left-extension from the anchor, and tiles T2 and T3 perform right-extension. T3's origin is placed on the intersection of the alignment path from T2 and its overlap border, so the alignment tail produced by T2 in green is ignored. The blue area in the zoomed image represents the X-drop band in T2.

X-drop is the concept of interrupting further computation of cells or further expansion

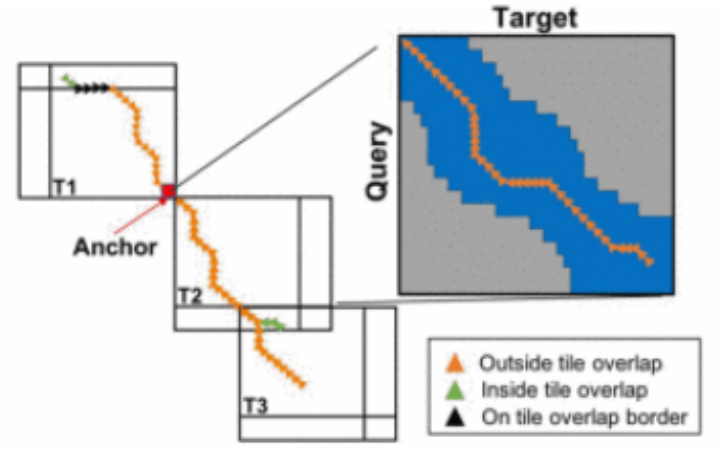

Figure 47: GACT-X tile overlap algorithm (TURAKHIA et al., 2019)

of alignment if the new score drops below a threshold of X (Y in GACT-X) in comparison to the maximum score registered. It is a broad definition because the term X-drop is used in different algorithms for different purposes. Minimap2 uses X-drop to interrupt global-alignment extension of sequences between anchors if they are too dissimilar (see Subsection 4.2.4). GACT-X uses X-drop to create a dynamic band around the alignment path inside of tiles.

In each tile, the matrix is filled row-wise while tracking the maximum score V_{max} of the tile. Stripes with $N_{PE} = 32$ rows are calculated with wave-front parallelism. The computation in a stripe is terminated when all the scores in a column fall below $(V_{max} - Y)$, Y being a given threshold. The next stripe also begins computation from the first column with all cells exceeding $(V_{max} - Y)$, generating a dynamic band around the alignment (Figure 48).

The processor implementation follows approximately the same architecture as in Darwin. The host transfers the target and the query sequences to the DRAM. N_{PE} query characters are loaded to the PEs, and target elements are streamed in a systolic fashion. A fixed memory of 1 BRAM bank per PE is allocated to store the 4-bit traceback pointers sequentially — 2 bits for the main matrix and 2 bits for gap matrices. Start and stop positions of each stripe are stored in separate BRAMs (Figure 49).

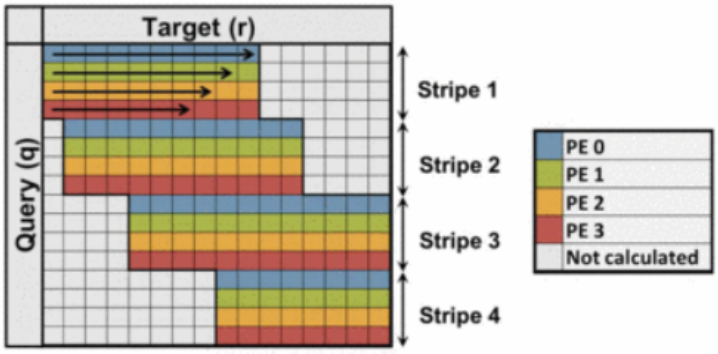

Figure 48: GATC-X tile with X-drop banding (TURAKHIA et al., 2019)

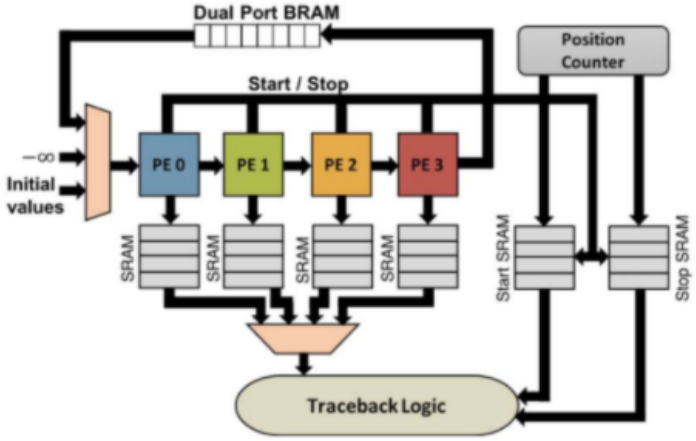

Figure 49: GACT-X systolic array with 4 PEs (TURAKHIA et al., 2019)

5 FIRST STEPS FOR PREPARATION

This chapter describes a methodical analysis of Minimap2, carried out to help develop the acceleration strategy. The objective is to obtain and apply the algorithm on different datasets (long-read sequences) and assess the algorithm's characteristics assembling human genomes. First, it shows how simulated and real human input data were collected. Then, it presents the execution time evaluation for the different datasets. Some graphic results for mapping quality assessment and alignment characteristics are presented.

5.1 Simulating Data with PBSIM

PBSIM, a model-based simulator (ONO; ASAI; HAMADA, 2013), collects stretches of a reference sequence, mimicking the read length distribution; and adds variants and sequencing errors. The GRCh38 PacBio continuous long reads (CLRs) were generated with the PBSIM tool, based on the GRCh38 reference, with error profile sampled from file 'm131017_060208_42213_*.1.*' downloaded at (HUMAN..., 2014). This group of reads will be referred as Simulated PacBio reads. Simulated reads are useful for evaluating the mapping accuracy, due to lack of truth in real data. Since the original reference is known, the simulated reads can be mapped in Minimap2 to check its mapping accuracy. Minimap2's author simulated PacBio data with this same process to compare their mapping accuracy with other tools (LI, 2018).

PBSIM simulates differences introduced to reads by analyzing alignments performed on real PacBio reads with respect to reference sequences using LAST (FRITH; HAMADA; HORTON, 2010) ($match = 1$, $mismatch = -2$, $gap_{opening} = -1$ and $gap_{extension} = -1$). Length distribution is log-normal; average accuracy over the read length has normal distribution with parameters given by the user; single molecule errors are stochastic (random); coverage depth is nearly uniform. Since PacBio nucleotide insertions have higher probability of being the same as their following neighbors, half of inserted nucleotides are the same as their following nucleotides and the other half is random.

PBSIM outputs simulated reads in FASTQ format, and read origins in MAF format, for each of the 193 sequences in the GRCh38 reference's Primary Assembly. The MAF files can be used to evaluate the mapping accuracy, since it describes where the reads came from. The algorithm was executed and the 193 ".fastq" and 193 ".maf" files were merged into one of each, to facilitate the file manipulation, resulting in 121GB of CLRs. Figure 50 shows PBSIM's log report; the default coverage depth is 20; the read length thresholds were 100 to 25,000; the mean length, deviation, mean accuracy, and accuracy deviation were sampled from the given file. Some simulated reads had lengths outside of the threshold determined, and were filtered out, resulting in the distribution in Table 4.

Over seven million reads were generated in total, which corresponds to a 20x coverage depth. The read lengths mean, standard deviation (SD), minimum, and maximum are respectively 8310, 106.26, 100, and 24,988. The reads' similarity rate to the reference strand is on average 0.85, with 0.00017 SD. Substitution, insertion, and deletion rates are 0.015, 0.090, and 0.046 respectively. Note that, on the sample given, there were considerably more insertions than deletions.

```
:::: Simulation parameters :::

Simulated by fastq sampling.

prefix : sd
data-type : CLR
depth : 20.000000
length-mean : (sampling FASTQ)
length-sd : (sampling FASTQ)
length-min : 100
length-max : 25000
accuracy-mean : (sampling FASTQ)
accuracy-sd : (sampling FASTQ)
accuracy-min : 0.750000
accuracy-max : 1.000000
difference-ratio : 10:60:30

:::: FASTQ stats ::::

file name : m131017_060208_42213_c100579642550000001823095604021496_s1_p0.1.subreads.fastq

:: all reads ::
read num. : 23365
read total length : 195771725
read min length : 35
read max length : 33422

:: filtered reads ::
read num. : 23001
read total length : 191169143
read min length : 104
read max length : 24988
read length mean (SD) : 8311.340507 (5448.702016)
read accuracy mean (SD) : 0.846338 (0.026921)
```

Figure 50: PBSIM simulation parameters and statistics from the ".fastq" sample used

Table 4: Statistics of CLRs generated with PBSIM in sampling mode

number of reads	coverage depth	read length				read accuracy		rates		
		mean	SD	min	max	mean	SD	substitution	insertion	deletion
7,460,510	20	8310	106.26	100	24,988	0.85	0.00017	0.015	0.090	0.046

5.2 Collecting Real Reads from Genomic Databases

Real datasets, obtained from real human samples, can give insights on real runtime and accuracy behaviors of Minimap2. The datasets were chosen by size, which is expected to be over the minimum coverage for clinical applications ($\geq 8\times$), and by current and known sequencing technologies. The first datasets encountered are from ethnicities that lack representation in the most renowned databases, therefore it is expected that they might diverge more from the genome reference used. This divergence is better for evaluating assembly algorithms because they expose them to more situational adversities. Other factor that should be considered when choosing the datasets is potential issues in the library preparation, for example, contamination of foreign DNA, but this can only be said after the pre-processing step of the genome analysis pipeline.

The first dataset was collected from real human ONT PromethION reads at the European Nucleotide Archive (RUN..., 2018), from a project that sequenced and produced the Yoruba reference genome NA19240, using 5 PromethION flowcells (COSTER et al., 2019). Some researchers are working on producing regional genome references to reduce the bias from the GRCh38 reference. However, the best practices still recommend using the later, while these projects mature with more sequencing data. The reads have on average 16,900 bases according to ENA's base and read counts (28,528,692,209 and 1,688,000).

The second dataset was collected from real PacBio Sequel II reads at NCBI (SRX9063500..., 2019). The reads were collected from a Sri Lankan individual. The reads have an average length of 13,329 bases according to NCBI's base and read counts (52.2G and 3,916,231).

A script that collects the read length histograms of the simulated and the two real datasets was written in Python, and the measurements are displayed in Figure 51. The average read length alone sometimes can't provide the whole information. For example, the real PacBio dataset and the real ONT dataset have close mean lengths (13,329 and 16,900), but their length distribution is drastically different. It is important to observe that the last length interval to the right corresponds to the accumulated frequency of all reads larger than 30,000. In the case of real ONT dataset, the last item does not correspond to a peak, but to a flat low frequency distribution.

The length distribution is going to be relevant when assessing the performance, as seen in Section 5.3.

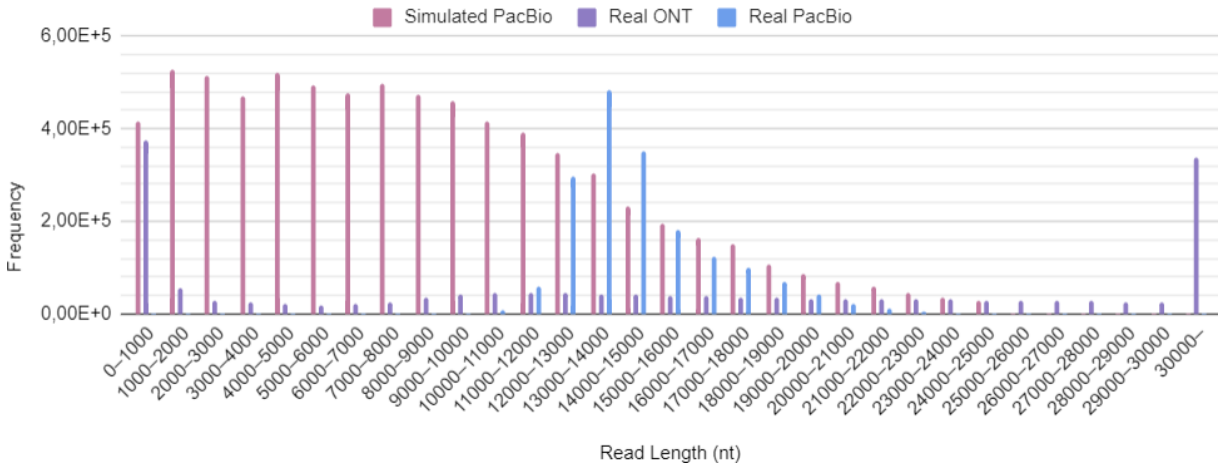

Figure 51: Simulated PacBio, real ONT and real PacBio read length histograms

5.3 Profiling Minimap2's Execution Time

It was already presented previously that assembly algorithms are computationally demanding and constitute the bottleneck of the genome analysis pipeline. Before accelerating an algorithm, some measurements on the software's throughput and its runtime bottlenecks should be taken.

Profiling Minimap2 with the three read sets presented in previous Sections 5.1 and 5.2 were performed on a Dell PowerEdge R910 server[1], running Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-201-generic x86_64), with further details in Figure 52 (information was obtained with the instruction "lscpu" on the command prompt). All three read sets were mapped to the Primary Assembly of the GRCh38 reference. To save time, the minimizer ".mmi" reference index was previously saved and used to substitute the reference ".fasta" file.

The datasets were mapped using 40 threads to collect Minimap2's throughput and internal data information presented in Section 5.4. Minimap2 was executed to output the alignment in ".sam" format and with optimization for ONT or PacBio reads. The gprof Linux tool (GNU..., 1998) was used to profile the execution time, and each execution was done with one thread. Since only one thread was being used for profiling, only the first 200,000 reads of each dataset were selected for use, otherwise the execution would take too long. The results are presented in Table 5.

[1]Professor Carlos Menck from *Instituto de Ciências Biomédicas* at University of São Paulo granted access to the Bioinformatics Seal server, with 4 Intel® Xeon® CPU E7-4870, 80 cores and 504GB of memory.

```
cteng@seal:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   44 bits physical, 48 bits virtual
CPU(s):                          80
On-line CPU(s) list:             0-79
Thread(s) per core:              2
Core(s) per socket:              10
Socket(s):                       4
NUMA node(s):                    4
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           47
Model name:                      Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:                        2
CPU MHz:                         2394.129
BogoMIPS:                        4788.25
Virtualization:                  VT-x
L1d cache:                       1.3 MiB
L1i cache:                       1.3 MiB
L2 cache:                        10 MiB
L3 cache:                        120 MiB
NUMA node0 CPU(s):               0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,76
NUMA node1 CPU(s):               1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61,65,69,73,77
NUMA node2 CPU(s):               2,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,66,70,74,78
NUMA node3 CPU(s):               3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63,67,71,75,79
```

Figure 52: ICB Seal server's CPU information

Table 5: Minimap2's throughput and bottlenecks running on three datasets

	Average Length (nt)	Real Time (hours) *	CPU Time (hours) *	Dataset Size (Gbases)	Throughput (kbases/s)	Chaining Time **	Extending Time **
Simulated PacBio Reads	8,300	2:46	42:41	61.99	403.46	8%	49%
Real PacBio Reads	13,300	4:55	88:21	52.20	164.11	50%	25%
Real ONT Reads	16,900	2:14	34:27	28.53	230.00	27%	42%

* 40 threads, complete dataset
** 1 thread, first 200,000 reads

The algorithm's speed and profile could have been influenced by input data characteristics, such as genome reference length, read length distribution, and similarity between reference and reads. Throughput was calculated based on the datasets' sizes and the CPU time. The simulated PacBio reads showed the highest throughput, but also the lowest average read length. Real PacBio reads had the lowest throughput and highest chaining time. More analyses presented in Section 5.4 can also help explain these differences.

The Minimap2's functions ksw_extd2_sse41 (referred to as ksw in this dissertation) and mm_chain_dp, responsible for the base-level alignment and chaining steps respectively, took together between 57% and 75% of the execution time. Other functions in order took 10% or less of the execution time. It is possible to conclude that the aligning and chaining steps are the bottlenecks of Minimap2's performance when assembling long-reads. Previous works reinforce this conclusion (GUO et al., 2019)(FENG et al., 2019) or add that these are also the same bottlenecks of Minimap2 on short-read execution (TENG et al., 2021).

5.4 Analyzing Minimap2's Intermediate Processing Data

5.4.1 Indel Sizes, Alignment Deviation, and Mapping Quality

Another important preparation step before accelerating Minimap2 is assessing the mapping accuracy and the characteristics of the alignments produced by it. Mapping quality can be measured by the proportion of reads that have been mapped to the reference and, particularly for the simulated data, mapping quality can also be assessed by the proportion of reads that were mapped near the original position. Other information pieces, such as indel sizes and deviation from anti-diagonal, help assess how the banding algorithm in Minimap2 behaves with respect to different long-read profiles. For that, two Python scripts were written to obtain statistical data from the outputs of simulated and real data.

The first set of data refers to histograms of indel sizes as shown in Figure 53. The indel sizes are directly collected from the numbers preceding the "I" or "D" alignment flags in the CIGAR strings reported in the SAM files. For all three datasets, the graphs showed that most of the indels are of a single nucleotide, and that the frequency decreases steeply, the longer the indels.

A second set of data refers to histograms of minimum fixed half band required to find the same alignment for the read as found in the Suzuki-Kasahara algorithm calls

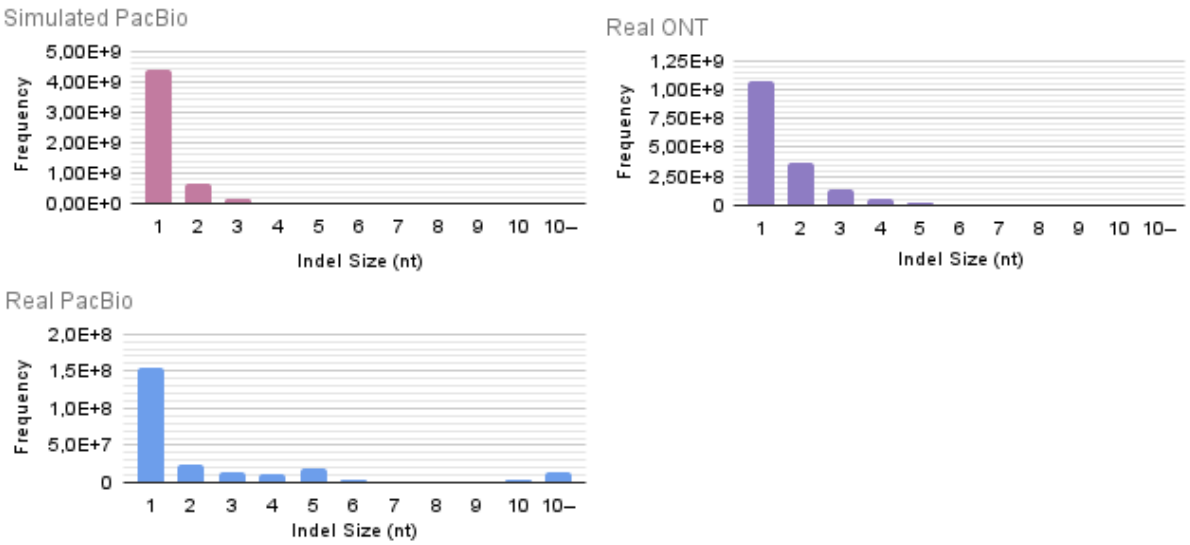

Figure 53: Indel size histograms from CIGAR strings reported on the SAM files

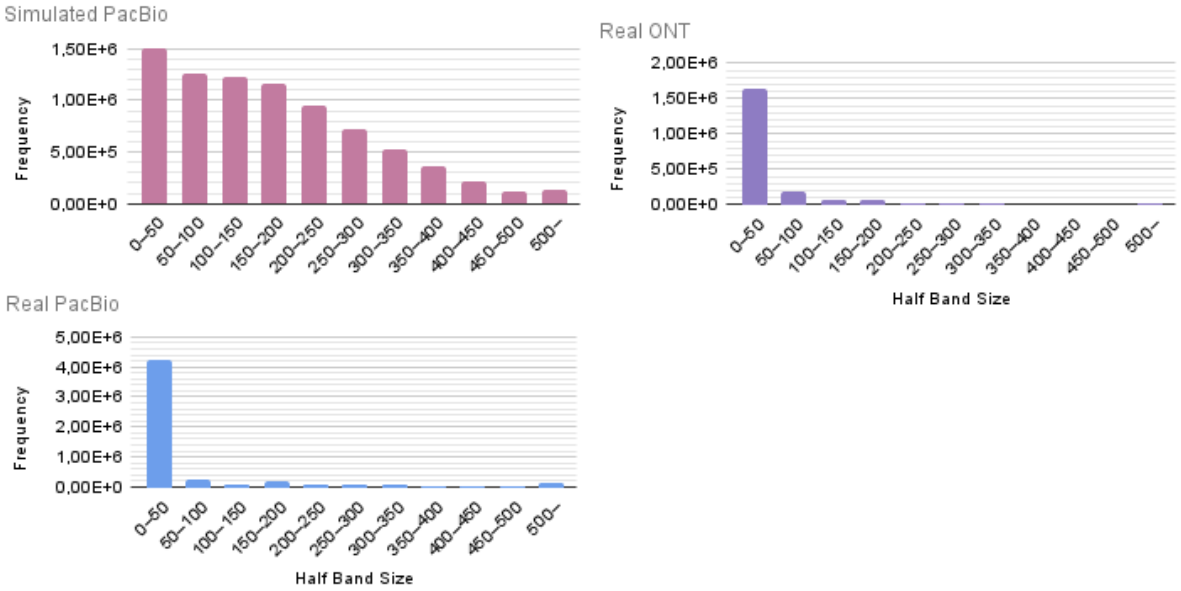

Figure 54: Histograms of deviation from the anti-diagonal from CIGAR strings

coupled with the chaining partitioning, and it is shown in Figure 54. The minimum half-band represents how much the resulting alignment has deviated from the anti-diagonal; it is calculated by adding cumulative insertions and deletions in the CIGAR string while canceling each other. It is important to note that, for Minimap2, this deviation for reads does not represent the deviation from the anti-diagonal in the SK matrices, to be presented in the next section, since the reads are irregularly divided by anchors.

For the simulated data, the deviation was significant up to a number of 500 from the anti-diagonal; this could have resulted from the higher insertion rate, compared to the deletion rate, as shown in Table 4. On the other hand, a great portion of the alignments stayed in a range of 50 from the anti-diagonal for real ONT and real PacBio cases, meaning a tighter band can be adopted for better performance.

One second script collects histograms for mapping overlap and histograms for SAM alignment flag reports. The mapping overlap can only be obtained from simulated reads, due to lack of truth in real data. It measures how much Minimap2's alignment overlaps with the original position of the read, calculated in percentage of the read's length. With Minimap2 running on the simulated PacBio reads, 99.45% of the reads were mapped to the correct RefSeq. From these, over 99.6% of the reads were mapped with an overlap greater than 90% to the expected mapping region.

The histograms on SAM flags, shown in Figure 55, were obtained directly from the SAM file, and the main report refers to the items "mapped" and "4" (see Figure 12 at

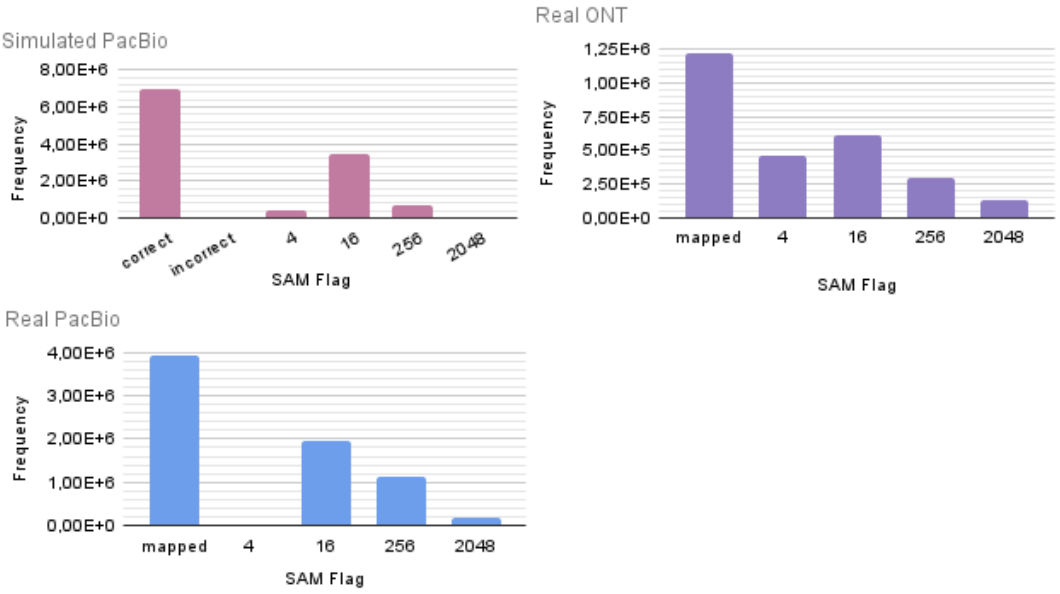

Figure 55: Map report histograms from the SAM flags

page 42), which corresponds to reads that could not be mapped to the reference with Minimap2. For the simulated data, the "mapped" column is separated into "correct" and "incorrect", since it possible to evaluate if the read was mapped to the correct reference sub-sequence, as discussed in Section 2.1.1. For simulated PacBio, real ONT and real PacBio reads, 6%, 27.5% and 0.04% of reads were not mapped, respectively.

With these analyses, it is possible to infer that Minimap2 has a good mapping accuracy and can map an acceptable portion of the real data. Further investigation is required to explain the high rate of unmapped reads in the real ONT dataset. The indel size histograms and the deviation from anti-diagonal histograms can be used to tune the accelerator's banding strategy in a way that does not harm the alignment accuracy.

5.4.2 Uniformity and Distribution of Anchors

Minimap2's original code was also edited to collect some other important internal data information, able to attest Minimap2's capability of generating uniform processing data, particularly regarding adjacent anchors. Initially, as shown in Figure 56, query and target length histograms were obtained, corresponding to the lengths of anchor-separated sub-sequences, in read and reference streams, respectively. The first three histograms refer to the sub-sequences in reads, while the other three refer to the sub-sequences in the reference.

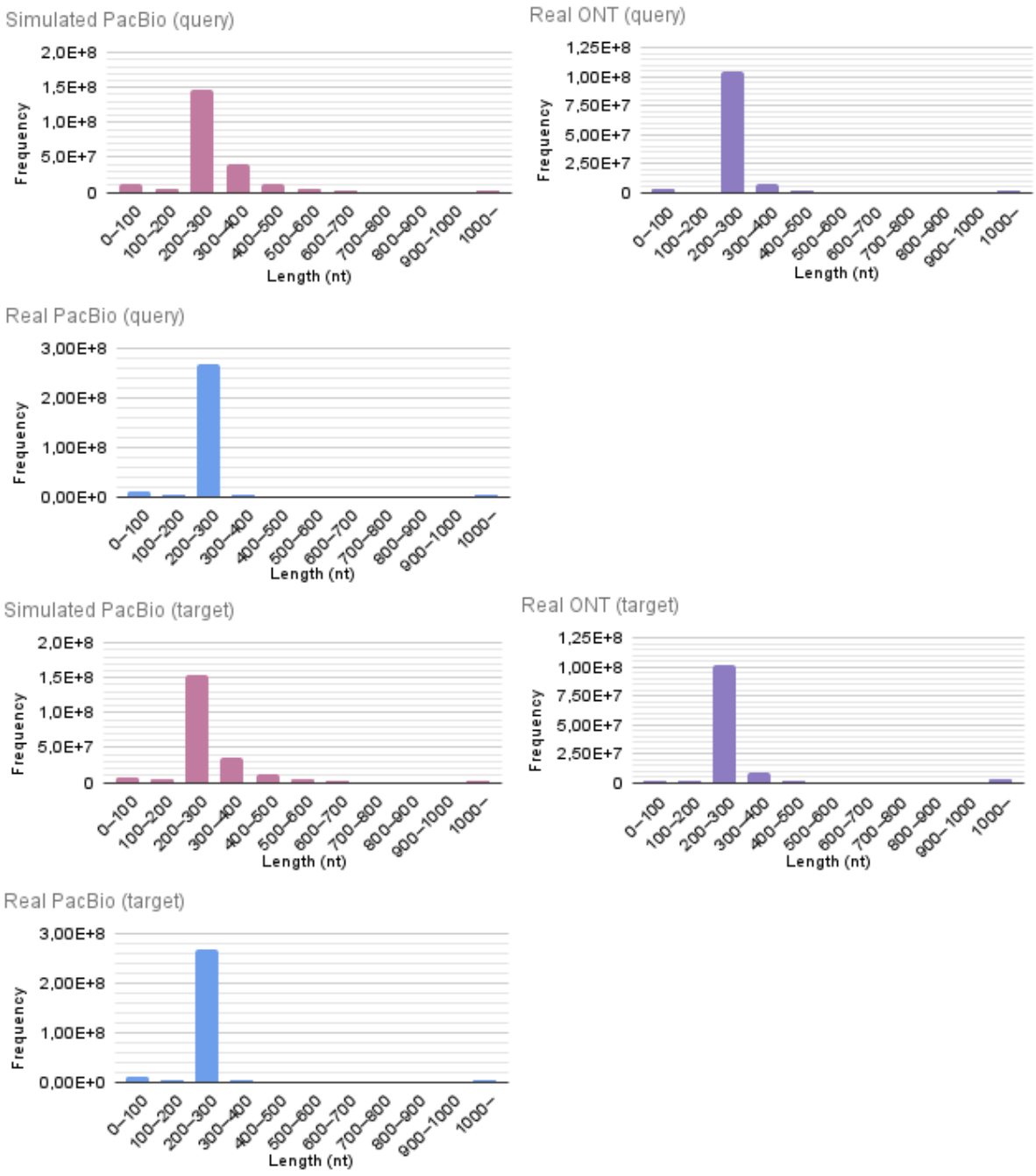

Figure 56: Anchor-separated query and target length histograms

Minimap2's intermediate data analysis for all the three datasets showed that the anchoring and chaining processes divided most of the reads into sub-sequences quite uniformly, with sizes between 200 and 300 nucleotides, as one can see in the peaks of the graphs. However, some sub-sequences were a little larger, and a few remained with lengths over 1000 nucleotides, meaning the need of larger SK matrices. As expected, the histograms for query and target for each dataset were very similar, since the chaining algorithm searches for more parallel anchors, meaning that the distances between consecutive

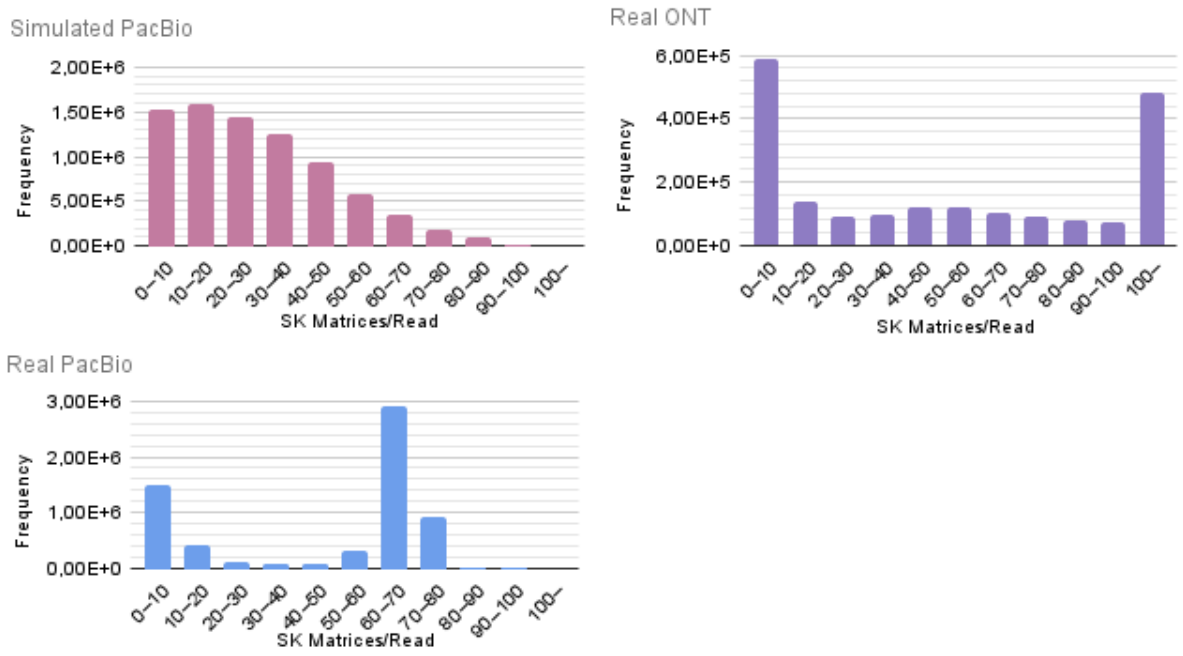

Figure 57: Histograms of number of SK matrices per read

anchors in the read and in the reference are similar.

Another form of verifying the uniformity is through the histogram with the number of SK matrices per read, i.e. the number of ksw calls to anchor-separated sub-sequences in each read, which is shown in Figure 57. The distribution seems to be different for the three datasets, but they can be better understood side-by-side with the read length histograms shown in Figure 51. The graphs for the simulated data and the real ONT data follow the curve of their read length histograms, indicating that the seed distribution was homogeneous across reads. If the sizes of anchor-separated sub-sequences are similar for most chains in a dataset, it is expected that the larger the read, the larger the number of sub-sequences. For the simulated reads, the SK Matrices/Read decreases steadily since the frequency also decreases when the read size diminishes. For the real ONT dataset, the frequency of small sized reads is high, then dropping and staying constant for varied larger sized reads; the same occurs for the SK Matrices/Read numbers. Only the real PacBio dataset did not show exactly this behavior because small SK Matrices/Read occurred frequently although there was no frequency of short reads in the dataset, indicating that some longer sub-sequences could appear.

Finally, SK deviation from anti-diagonal histogram is shown in Figure 58. The deviation works the same way as the previous deviation from anti-diagonal, shown in Figure 54, but it is for alignments between adjacent anchors. For simulated reads and real ONT reads, the deviation from the anti-diagonal in each SK matrix was more compressed to

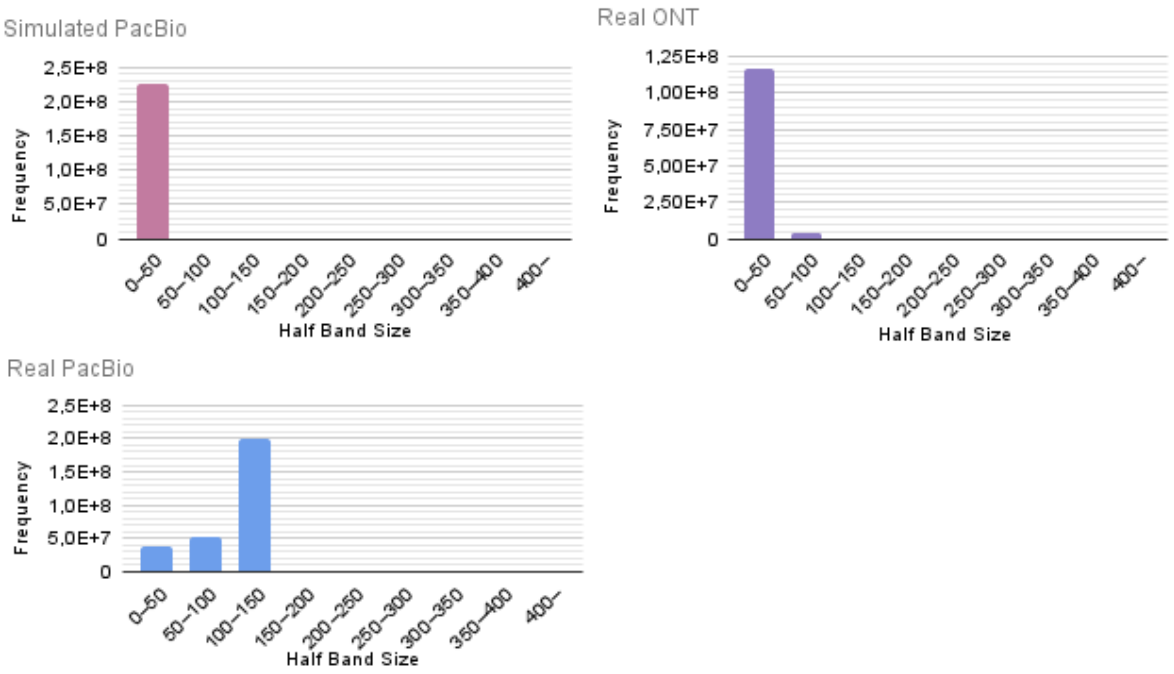

Figure 58: Histograms of deviation from the anti-diagonal in SK matrices

under 50, but for real PacBio reads, most of the matrices had a deviation of 100 to 150. This indicates that applying a fixed narrow band, such as the one described in Section 3.2, could be inefficient for some datasets in anchor-separated sub-sequences.

6 MINIMAP2 WITH AWS FPGA ACCELERATED GACT-X: ADAPTATION, INTEGRATION AND RESULTS

This chapter describes the process of adapting, installing, and integrating the GACT-X module in the Cloud AWS FPGA Instance to the software implementation of Minimap2. The module has as inputs the sequences generated in Minimap2 after the seeding and chaining steps, that are separately aligned to the reference in the ksw function, and as outputs the alignment results. The first section describes specific adaptations done to GACT-X's host and Verilog codes to improve its performance and turn it compatible with Minimap2. Then it shows an integration option like in the original Minimap2 that has anchor-separated sub-sequences, and results on performance are presented. It also presents a new adaptation to the integration method, aimed to overcome problems observed in the results in the first option; in this new version, anchor-extended sub-sequences are used from the chaining step; results in performance and accuracy are presented. A second section presents the integration process between Minimap2 and GACT-X, compatible with the multi-kernel and multi-threading capabilities, using OpenCL synchronization methods; total acceleration measurements are collected for all combinations of number of threads and number of kernels possible in the F1 Instance.

6.1 Adapting GACT-X for Better Performance

Darwin-WGA was published on GitHub (DARWIN-WGA, 2019). The entire algorithm, as well as the separated modules BSW and GACT-X, can be mirrored and implemented on AWS. Only the GACT-X module was used for this project, as mentioned in Section 4.3.

First, an AWS Instance was created with the FPGA Developer AMI (Amazon Machine Image), version 1.4.1, in region US West (Oregon). Currently, AWS in Brazil does not provide any Amazon FPGA instances. As mentioned before, the chosen instance type was

f1.2xlarge, the smallest one that has an FPGA, enough for our analysis purposes. This AWS instance is structured in its host-device environment as seen in Section 2.3.2. From Windows, Puttygen was used to access the terminal and WinSCP was used to manage files. tmux (TMUX, 2022), a terminal multiplexer for controlling terminals even if detached from a screen, was installed to run long jobs remotely. The aws-fpga library was swapped to an older version, since GACT-X was developed on the SDAccel environment and, from 2020 on, the default environment changed to Vitis. A bucket was created to keep the Amazon FPGA Images. The VIVADO tool version was set up to version 2017.4.

Part of software in OpenCL for GACT-X at the host had to be adapted for this implementation. Since Darwin-WGA is a whole genome aligner, originally it was developed to send the two entire genomes to be aligned to the DDR, looping the pairs of positions to be extended from them. For read mapping, as performed in Minimap2, every pair of reference sub-sequence and read is streamed to the DDR to get the extending results. The process of creating and managing tiles was not implemented in the original GACT-X host, and had to be added as well.

Figure 59 shows the fluxogram for transferring data to the FPGA with the tile algorithm. The inputs correspond to many pairs of read and reference sub-sequences, which were written in a separate file after the seeding and chaining steps in Minimap2 (left of the figure). In the example, the loop cycles 5 times, since there were 5 tiles in total. The sequence of steps for the software-hardware interaction follows:

- Each pair is transferred to the FPGA into the ref_seq and query_seq buffers (see item **1)** in the figure);
- A loop sets the tile's starting index with variables ref_offset and query_offset, and the tile's size with variables ref_len and query_len (see item **2)** in the figure);
- The FPGA kernel or kernels calculate the alignment for the tile and write in h_tile_output the alignment score, max score index with variables ref_pos and query_pos, and the number of traceback pointers in the band; the traceback pointers are written in h_tb_output.

The alignment parameters were changed to have about the same proportions as the default in Minimap2 (Table 6). The match, mismatch, and gap scores are 2, -4, -4, -2 for Minimap2, so they were set as 10, -20, -30, -10 for the adapted GACT-X. The difference in the gap opening score is due to different interpretations on whether an extension occurs when opening a gap, but the resulting score is the same. GACT-X

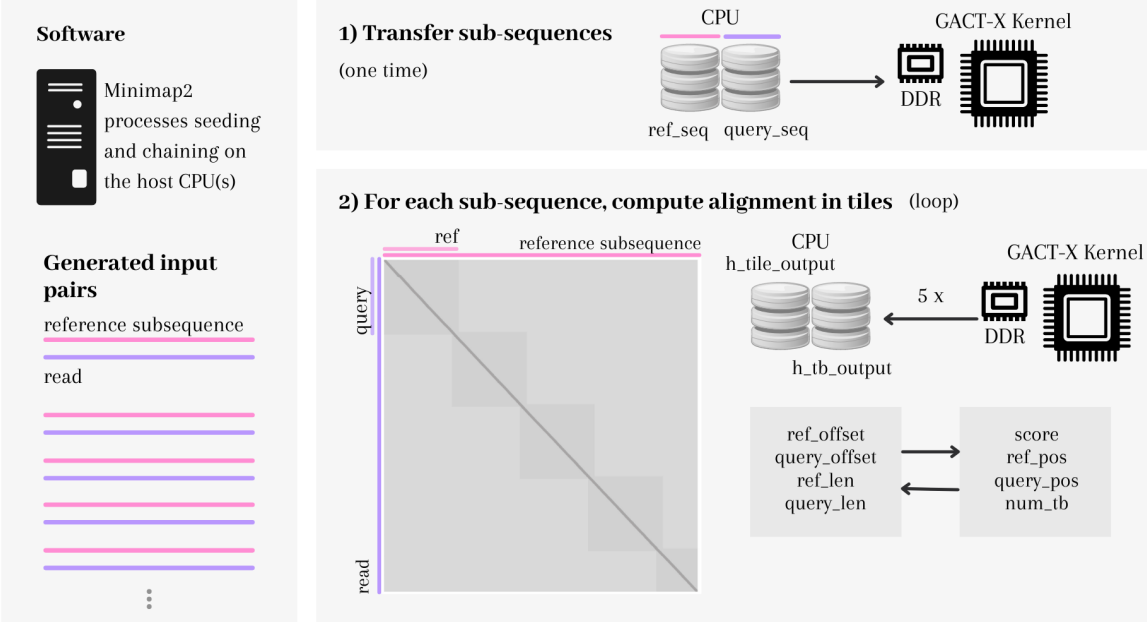

Figure 59: Example of GACT-X's host-FPGA data transfer fluxogram

Table 6: Minimap2 and GACT-X's alignment parameters

		match	mismatch	1st function		2nd function		Y-drop
				gap open	gap extend	gap open	gap extend	
Minimap2		2	-4	-4	-2	-24	-1	-
GACT-X	**original**	91 to 100	-31 to -125	-430	-31	-	-	9530
	adapted	10	-20	-30	-10	-	-	943

originally uses substitution matrices for match and mismatch scores, where different pairs of nucleotides result in different scores. GACT-X only supports one gap function; only the Minimap2's affine gap function with higher slope and lower translocation was used in GACT-X because it picks the more frequent short indels. The Y-drop value was adjusted according to GACT-X's new gap function, allowing the presence of a same size of gap in the band as originally allowed.

6.1.1 GACT-X with Anchor-Separated Sub-Sequences from Minimap2

The initial strategy to accelerate the extending step was to send, individually, every anchor-separated sub-sequence that lay between two anchors to be aligned in the FPGA, using global alignment. This is the strategy used originally by Minimap2, that at the same time reduces sequence lengths and avails the matching information produced by the chaining process.

The sequence of steps in the host for the software-hardware interaction described in OpenCL (see commands in Section 2.3.3) is:

- The input files are uploaded to the AWS Instance;
- The host creates input and output buffers in the DDR with sufficient space for any input length in the dataset; for that, the clCreateBuffer command is used;
- The pairs of anchor-separated sub-sequences are collected from the input files, and sent to the buffers in the FPGA through the clEnqueueWriteBuffer command;
- The arguments to the compute kernel (e.g. tile beginning and end positions, alignment scores, and buffers) are set through the clSetKernelArg command;
- The kernel is executed through the clEnqueueTask command;
- The output buffers are read back in through the clEnqueueMapBuffer command and the results are processed to recover the alignment from the direction pointers (traceback);
- The alignments for each pair of sequences are written into an output file.

The simulated PacBio dataset was adopted for this testing phase, and only its first 100,000 reads were used, due to the high execution time. The real ONT and PacBio datasets are not used in this development stage, but the final design's performance was measured on them and will be presented later. The datasets contain some millions of input reads each, but it is reasonable to consider that the sequencing technologies produce uniformly distributed read sizes, therefore, 100,000 samples can be considered representative of the whole dataset.

The query and target anchor-separated sub-sequences that are aligned in Minimap2, generated after the chaining step, were collected. The input length histograms have been presented in Fig. 56 for all three datasets, showing the anchor-separated sub-sequence sizes concentrated between 200 and 400 bases.

This Minimap2-GACT-X implementation showed performance problems, as they can be seen in Figure 60, which presents average execution times in using Minimap2's ksw function and a GACT-X kernel for the alignment step. The results for the simulated PacBio dataset showed that GACT-X performed considerably worse than ksw from Minimap2 in this design, that mostly aligns short sequences (although originated from long-reads). The performances for ksw and GACT-X are similar only for sequences larger than

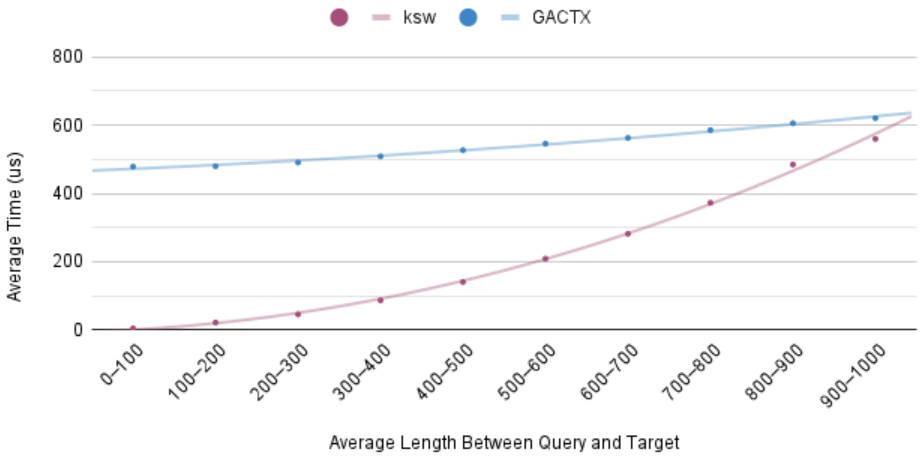

Figure 60: Average processing times per length for anchor-separated sub-sequences

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

1,000 bases. This is because GACT-X's processing time starts with a shifted constant that corresponds to the data transfer time from host to FPGA and back (see item **1)** in Figure 59), for every anchor-separated sub-sequence, creating a heavy transfer latency that undermines the gains obtained with a faster alignment performed in the kernel. For the real ONT an PacBio datasets, the transfer latency is expected to be similar, since the average lengths of sub-sequences, shown in the histograms of Figure 56, are highly concentrated in the 200-300 bases range.

6.1.2 GACT-X with Anchor-Extended Sub-sequences from Minimap2

For a more efficient solution, instead of aligning each anchor-separated sub-sequence, the strategy changed to considering, in each read, the anchor-extended sub-sequences, i.e., the portion from the first anchor to the end of the sequence and its corresponding reference sub-sequence, expanding the alignment in a semi-global fashion. The software-hardware interaction follows the fluxogram of Figure 59, however the item **1)** is altered to a reduced number of transfers of longer sub-sequences.

The experiments were also performed on the first 100,000 reads from the simulated PacBio dataset. In order to confirm the effects of the new transfer, Figure 61 was generated. It shows that, for the simulated PacBio dataset, the new transferred data have lengths to be aligned increased in one to two orders of magnitude. As expected, the histogram follows the shape of the one presented in Fig. 51 for the reads, therefore, it is expected that the real ONT and PacBio datasets also present increased lengths in their

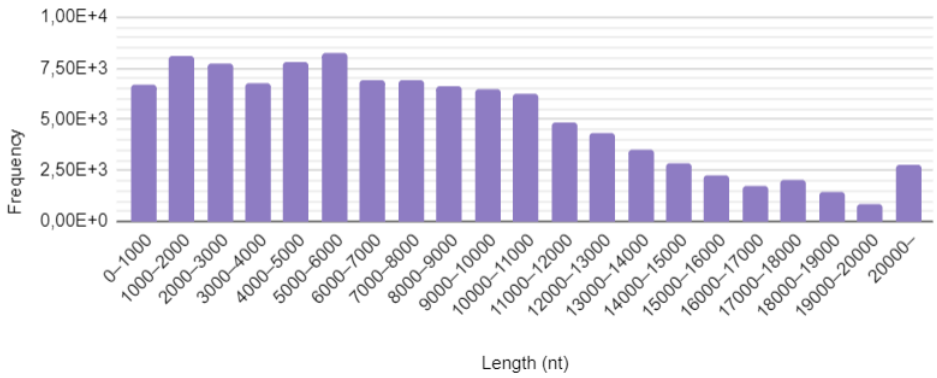

Figure 61: Anchor-extended query-target average length histogram

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

transferred sub-sequences.

By increasing the lengths of the transferred sub-sequences, the tile processing had to be activated, and the heuristics described in Section 4.3 had to be implemented due to limitations in hardware resources. In the original Verilog files, used to build the binary code of the kernels, the tile size is set up to be at most $2,048$, determined by wires in internal modules of size $log_2(2,048) = 11$. If the host forces an alignment between longer sequences, either the result is going to turn out incorrect, or the execution will potentially freeze, requiring forced interrupt, cleaning and reloading the AFI into the FPGA.

Each tile requires a new data transfer cycle, as at item **2)** in Figure 59, so it was expected that, as tile size increases, fewer tiles would be required in each expansion, and better the overall performance would become. In order to test the behavior of the hardware-software implementation, its performance was measured under different tile sizes. The max tile size value in the Verilog files was changed to $8,192$ and a new AFI was generated.

The performances for the new strategy were measured with varying tile sizes and are shown in Figures 62 and 63. The first one presents the execution times with respect to the many ksw calls in software along with the ones of GACT-X, for different sub-sequence sizes. The second one refers to the total execution time for the ksw function in Minimap2 and for GACT-X.

Regarding Figure 62, as expected, GACT-X's performance improved as tile sizes increased from 1,000 to peak at 4,000, considering the same size of the input data. However, after that, the performance started worsening, and it is suspected that it is due to BRAM saturation, since direction pointer storage is linearly cumulative and increases with tile

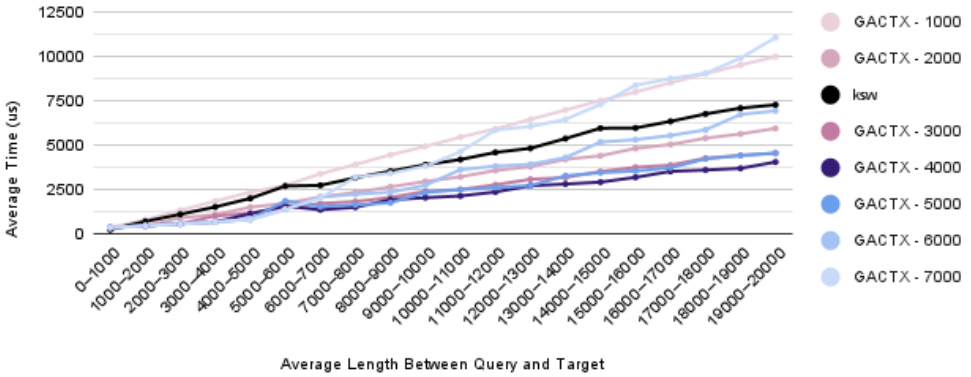

Figure 62: Average processing times per length for anchor-extended sub-sequences

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

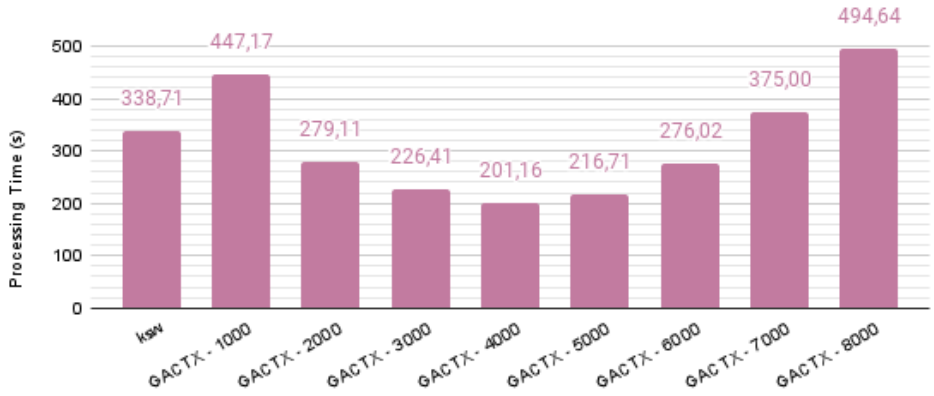

Figure 63: Total processing times for ksw and GACT-X

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

size. Still, it was already possible to observe an acceleration compared to ksw's performance. As for the real ONT and PacBio datasets, it is expected also a steady increase of performance for larger tiles, since the transfer time will be similarly reduced; due to the BRAM limitation, tile of size 4,000 should be the one with highest performance improvement too.

Figure 63 shows that GACT-X with tile size of 4,000 achieved the largest speed-up of 1.68x compared to Minimap2's ksw software execution time. Other changes can further improve this number, and are discussed later in this section and in Section 7.

GACT-X's accuracy was also evaluated against Minimap2's ksw function in a range of tile sizes. Figure 64 shows two sets of results: a) from the alignments, it was possible to compare the alignment scores obtained from the Suzuki-Kasahara algorithm, coupled with chaining divisions, with the alignment scores obtained from the GACT-X algorithm;

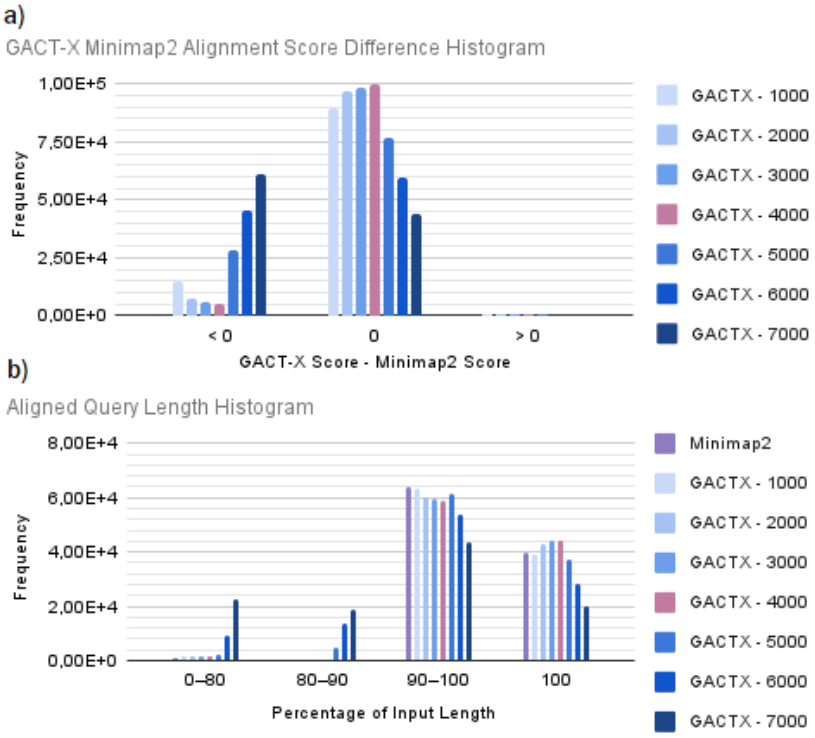

Figure 64: Histograms of GACT-X and Minimap2's score and query percentage differences

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

b) the percentage of aligned inputs of query sequences.

The first histogram shows the proportion of alignments where GACT-X had a worse (< 0), equal (0), and better (> 0) score in relation to ksw from Minimap2, for varying tile sizes. 95.42% of alignments found with GACT-X with tile $4,000$ had the exact same score as the ones produced in Minimap2, even with chaining information hidden from GACT-X. It means that GACT-X with tile $4,000$ can provide a accuracy at the same level of the ksw algorithm. The cases with tiles of 5,000 bases or more show lower accuracy and also are the cases where the performance is low, as seen in Figures 62 and 63. Again, it is believed that BRAM saturation causes some data dis-alignment, leading to wrong results. The small percentage of alignments' lower scores may be explained by these factors: the tile heuristic adds uncertainty in the borders of each tile; GACT-X does not support a second gap function, so frequently loses longer gaps that should be expected to appear.

The second histogram corresponds to the percentage of query sub-sequences that were used in each alignment in Minimap2's ksw (with global alignment with anchor-separated sub-sequences and semi-global alignment after the last anchor) and in each alignment for GACT-X with different tile sizes (considering a semi-global alignment from the first anchor

to the end of the sequences). When an input sequence is aligned, it may happen that the max score, which is where the backtrack starts in semi-global alignment (see Section 4.1.2), does not occur in the last (right-down) cell, and only part of the sequence is used. Ideally, the query sequence should be aligned entirely to reduce waste. By the histogram of Figure 64, 60.97% of Minimap2's tracked alignments consumed between 90% and 100% of the corresponding sequence, and 37.62% consumed the entire query sequence. For GACT-X with tile $4,000$, the percentages were 55.21% and 43.04% respectively, indicating a better usage. The accuracy for tiles from $5,000$ on decreased steeply, which could also be linked to BRAM saturation.

Given that the tests with tile size of 4,000 showed the best performances in both accuracy and speed for the simulated data, the same measurements were made with the two real datasets with this same configuration, and are presented in Tables 7 and 8.

Table 7 shows that GACT-X's accuracy was very similar to Minimap2's on the simulated PacBio data, considerably worse on the real ONT data (41.21% of the alignments produced had lower scores), and dissimilar for the real PacBio data, presenting high rate of both worse (14.90%) and better (18.28%) alignments. Table 8 reinforces GACT-X's similar accuracy performances for the PacBio datasets, with similar ratio of aligned query lengths, but worse for the real ONT dataset, with lower ratio of aligned query lengths. With these results and the alignment reports presented in Figure 55, it is possible to induce that the real ONT dataset has a significant disparity to the genome reference used, which could be affecting the mapping and alignment accuracies of both of the algorithms.

The partial results obtained to this point have provided a strong indication that it is possible to accelerate Minimap2's extending step, even with SSE optimization, by using a software plus hardware hybrid architecture. GACT-X's tile heuristic allows alignment be-

Table 7: Minimap2 and GACT-X's alignment score differences for three datasets

GACT-X Minimap2 Score Difference	< 0	**0**	> 0
simulated PacBio	3.90%	95.42%	0.68%
real ONT	41.21%	47.79%	11.00%
real PacBio	14.90%	66.83%	18.28%

Table 8: Minimap2 and GACT-X's aligned percentage of query sequences for three datasets

	Percentage Aligned	**0-80**	**80-90**	**90-100**	**100**
simulated PacBio	**Minimap2's ksw**	1.06%	0.22%	60.97%	37.62%
	GACT-X 4,000	1.52%	0.23%	55.21%	43.04%
real ONT	**Minimap2's ksw**	21.56%	2.59%	73.57%	2.29%
	GACT-X 4,000	36.24%	3.64%	58.47%	1.64%
real PacBio	**Minimap2's ksw**	18.42%	0.76%	13.15%	67.67%
	GACT-X 4,000	12.01%	2.71%	19.59%	65.69%

tween arbitrarily long sequences, which was an issue that long-read mapping faced on limited FPGA resources. GACT-X's Y-drop heuristic mimics Minimap2's fixed bandwidth heuristic, both decreasing memory and time complexities of the SWG algorithm from quadratic to linear. GACT-X's accuracy remained satisfactory compared to Minimap2's (considering alignment scores) and GACT-X was able to align a similar percentage of the read sequences on the PacBio datasets. GACT-X's accuracy is aggravated in comparison to Minimap2 on the real ONT dataset that seems to have lower similarity to the genome reference.

Satisfactory acceleration and accuracy has occurred as far as the tile size is not larger than 4,000 bases, limitation imposed by the allocated BRAM size for one kernel. The use of larger tiles may be desirable since it could further improve the performance, likely maintaining the accuracy. With the FPGA device in the AWS F1 Instance, it is possible to double the tile size, since just less than half of the BRAM available is consumed by one GACT-X kernel. This ratio is explored in the next section by implementing 2 parallel GACT-X modules in the same device. Bigger tiles can be accomplished with the use of more expensive FPGA platforms with larger BRAM resources.

6.2 Integrating GACT-X into Minimap2 with Multi-kernel and Multi-threading

With the results obtained in Section 6.1, GACT-X was integrated to Minimap2 on the host, and a series of experiments were made for the software-hardware hybrid implementation. The tile size was set to 4,000 following the results in Section 6.1.2. The hybrid system accommodates the multi-threading and multi-kernel capabilities of the system.

In order to set the host, a new C++ file "gactx.cpp" was created and added to Minimap2's source to accommodate the hardware's host lines. It contains the functions to configure and load the binary file, initiate the acceleration context, align with GACT-X, and clean the system. Detailed description of the OpenCL code implementation can be found in Appendix A.

Minimap2's original implementation allows multi-threading. Each thread performs the sequential processing of seeding, chaining, and aligning one read at a time (process shown in Figures 43, 44 and 46); threads can be distributed to cores in the host server for concurrent processing. OpenCL supports multi-kernels in the same FPGA, as many as the resources can fit.

Generally speaking, any thread could be assigned to any kernel in order to optimize concurrent processing. However, in Minimap2, only a limited form of concurrency in using the kernels related to GACT-X is possible, because of the sequential aspect mentioned in the previous paragraph:

- The number of threads need to correspond to at least the number of kernels, otherwise the excess kernels will be idle;
- A kernel cannot carry the execution of different threads interchangeably.

The restrictions above require that, whenever there are more threads than kernels running, and the corresponding sequences generated by the chaining steps are ready for alignment processing, a waiting list must be used in order to access an available/free kernel. On the other hand, in situations of a single thread run (single core), whatever the number of kernels, only one may be used.

The AWS f1.2xlarge Instance allows the implementation with up to 64 kernels and 8 cores, as explained in Section 2.3.2. This project was tested with up to 2 GACT-X kernels and 8 Minimap2 cores, due to the number of BRAMs available in the FPGA device, that

could only support 2 GACT-X modules with tiles of size 4,000. This test size is enough to give a good insight on the problem, as it will be seen later in this section, and there will be discussion on how to use it with a larger number of modules.

The analyses were made with the 100,000 first reads from the simulated PacBio, and the real ONT and PacBio datasets. Two types of run were executed for time measurements. In both of them, the measurements were made with all combinations of 1 to 8 software (sw) threads and 1 to 2 kernels.

The first run is for the comparison between the hybrid Minimap2 with GACT-X implementation, and Minimap2 in software with ksw-based alignment. This run was to measure the total execution time, displayed by Minimap2 in the command prompt at the end of the execution; regarding the GACT-X kernel, command queue was used to stream commands to the FPGA and, right away, execution proceeded to execute the next line without waiting for the command to finish, except in synchronization points. The results are presented in Table 9.

In the table, all three datasets, as seen in column 1, are processed in the three implementations for Minimap2 in software, Minimap2 with GACT-X implemented in 1 kernel, and Minimap2 with GACT-X implemented in 2 kernels, as shown in column 2. Results are shown for increasing number of threads, which were tried up to 8, as listed in columns 4 to 11. The total execution times in seconds were measured for each test instance. Two types of acceleration were computed: a) the acceleration obtained for each implementation with respect to the software, for the same number of threads (set for each column); b) the integrated system's thread acceleration, which is the acceleration for increasing the num-

Table 9: Execution times and acceleration in the Minimap2-GACT-X integrated system

	Number of threads (sw)		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
simulated PacBio	**software**	**total execution (s)**	759.157	390.198	266.926	206.255	197.863	186.727	177.508	169.927
	1 kernel	**total execution (s)**	539.56	299.76	233.88	208.93	203.74	253.84	409.03	453.26
		acceleration	1.41	1.30	1.14	0.99	0.97	0.74	0.43	0.37
		thread acceleration	1.00	1.80	2.31	2.58	2.65	2.13	1.32	1.19
	2 kernels	**total execution (s)**	538.996	289.557	211.837	198.700	203.366	228.516	273.066	313.150
		acceleration	1.41	1.35	1.26	1.04	0.97	0.82	0.65	0.54
		thread acceleration	1.00	1.86	2.54	2.71	2.65	2.36	1.97	1.72
real ONT	**software**	**total execution (s)**	3,105.50	1,559.59	1,056.77	806.37	774.30	719.29	684.98	646.35
	1 kernel	**total execution (s)**	2,243.69	1,168.48	837.14	712.91	653.68	660.44	937.68	1,258.13
		acceleration	1.38	1.33	1.26	1.13	1.18	1.09	0.73	0.51
		thread acceleration	1.00	1.92	2.68	3.15	3.43	3.40	2.39	1.78
	2 kernels	**total execution (s)**	2,197.265	1,126.070	794.728	638.109	586.025	576.393	672.990	858.846
		acceleration	1.41	1.38	1.33	1.26	1.32	1.25	1.02	0.75
		thread acceleration	1.00	1.95	2.76	3.44	3.75	3.81	3.26	2.56
real PacBio	**software**	**total execution (s)**	4,255.21	2,132.71	1,431.38	1,083.29	1,037.82	965.57	904.33	858.31
	1 kernel	**total execution (s)**	3,570.25	1,813.39	1,243.76	1,003.40	919.39	889.68	1,077.63	1,575.97
		acceleration	1.19	1.18	1.15	1.08	1.13	1.09	0.84	0.54
		thread acceleration	1.00	1.97	2.87	3.56	3.88	4.01	3.31	2.27
	2 kernels	**total execution (s)**	3,470.799	1,765.341	1,239.144	994.346	885.107	831.761	855.454	1,117.282
		acceleration	1.23	1.21	1.16	1.09	1.17	1.16	1.06	0.77
		thread acceleration	1.00	1.97	2.80	3.49	3.92	4.17	4.06	3.11

ber of threads, with respect to the single threaded execution, for the software-hardware integrated implementation.

The execution times of the software-only implementation for all datasets have shown a steady decrease for increasing number of threads, as it is expected (it can be seen in line 2, for example, for the simulated PacBio dataset); the relation is not linear due to the increase in management complexity for larger number of threads. The execution times for the integrated software-hardware systems also decreased in the initial addition of threads, as expected, but stabilized and increased again for about 5 or 6 threads or more, whatever the number of kernels, reflecting in reduced thread acceleration (see, for example, lines 3 and 6, for simulated PacBio dataset, and can be observed clearly through the figures on thread acceleration in lines 5 and 8).

Another issue is that data with 7 and 8 threads had their execution times exploded, being inconsistent with the change rate for the other measured times, and making them unreliable for analysis (therefore, they will not be considered in later discussions); considering there are 8 available cores for processing, probably two of them are used for system management and conflicts with the GACT-X's host lines.

Although the integrated software-hardware systems show decreased time execution with increased number of threads, that occurs in a lower rate than for the full software implementation. This can be seen through the acceleration numbers in lines 3, 6, 10, 13, 17 and 20. These results show that, for the integrated implementations built in this work, the higher number of threads do not help much. In fact, a more detailed study on the bottlenecks must be provided, what is done in next paragraphs.

The table shows that the use of 2 kernels improved the execution time if compared to the single kernel case, as expected, but this trend changes after 5 threads. Besides that, the reduced time with 3 or more threads was limited, with the maximum close to 10%. Also, as commented before, for 1 or 2 threads, the availability of 2 kernels does not help. It is expected that, with more available kernels, the workload from a larger number of threads can be alleviated, with better acceleration, but that would require increased hardware resources and costs.

The highest achieved acceleration therefore was of 1.41x with 1 thread for simulated PacBio with 1 and 2 kernels, and real ONT with 2 kernels, and second highest was 1.35x for simulated PacBio with 1 thread and 1 kernel, and real ONT with 2 threads and 2 kernels. It is worth to mention that this study with GACT-X's acceleration only refers to the alignment phase in Minimap2; the chaining step may also be hardware accelerated

(GUO et al., 2019), improving the total performance of Minimap2.

Although the results in Table 9 showed some expected trends, several expectations were not accomplished:

- The acceleration decreased as threads outnumbered kernels, performing worse than the software counterpart when the gap is too big;
- Acceleration with 2 kernels was only slightly better than with one kernel, when a speed-up of 2x might be expected.

The suspicion for the above behavior is the performance limitation due to competition of the PCIe transfer channel. In order to verify it, a second run of experiments were made; time measurements were taken with different tools for each situation, in order to deal with the actions in separate. Every kernel processing, data transfer and waiting in line instance has had the time measured and accumulated, respectively. These instances in each thread were measured independent of any other of Minimap2's execution that could occur concurrently. Therefore, the total accumulated time cannot be directly correlated to the total execution time, shown in Table 9, except for the case of a single thread, which is totally sequential.

The kernel processing times were measured using the function "clGetEventProfiling-Info" and START and END event flags. The data transfer times, which includes the times to transfer sequences and tile information, as can be seen by the items 1 and 2, respectively, in Figure 59, are measured similarly. The time spent by the threads waiting in line for a kernel was measured using "clock_t". Every writing, reading, and kernel event had to be completed for the execution of the code to proceed to the next line by setting the "blocking_write" argument as CL_TRUE, or by using "clWaitForEvents", so that the measurements could be taken.

The results are displayed in Table 10. For each of the three datasets, shown in column 1, the two integrated implementations, with 1 and 2 kernels, were considered, as displayed in column 2. For each implementation, the case of kernel processing, data transfer and in line times are shown in column 3, referring to different thread numbers, from 1 to 8, as indicated in columns 4 to 11.

Considering the kernel processing time reflects the computation of the total amount of the inputs, which is constant throughout the different kernel implementations, the results have shown to be as expected: for all datasets, it remained quite constant comparing 1 kernel with 2 kernels cases. Also, considering any particular implementation for a dataset, the

Table 10: Processing, transferring, and waiting times in the Minimap2-GACT-X system

	Number of threads (sw)		1	2	3	4	5	6	7	8
simulated PacBio	1 kernel time (s)	processing	94.14	94.62	95.16	96.16	96.71	98.11	132.26	150.40
		data transfer	26.90	28.81	32.07	33.24	34.13	37.59	152.30	150.51
		time in line	0.21	83.11	318.66	845.77	1,789.56	3,377.48	13,990.11	21,404.43
	2 kernels time (s)	processing	94.20	94.99	95.43	95.89	96.49	110.83	129.61	141.75
		data transfer	27.00	34.77	37.87	37.25	47.99	146.70	165.48	166.65
		time in line	0.15	0.24	44.49	186.04	619.47	3,813.98	7,592.47	12,094.00
real ONT	1 kernel time (s)	processing	322.29	323.02	324.33	326.42	327.82	331.18	388.75	476.91
		data transfer	12.20	48.54	51.66	53.72	59.88	75.27	361.88	503.51
		time in line	0.15	148.91	544.15	1,310.52	2,548.16	4,926.27	24,567.76	50,482.29
	2 kernels time (s)	processing	322.66	324.16	324.98	326.04	327.69	346.86	376.96	420.51
		data transfer	42.80	53.15	61.25	60.11	86.25	227.04	347.46	476.46
		time in line	0.24	0.27	54.48	242.43	802.66	4,210.88	11,777.99	25,100.46
real PacBio	1 kernel time (s)	processing	255.12	255.52	257.62	259.56	261.43	264.64	324.07	465.84
		data transfer	53.28	57.38	61.37	65.27	74.19	91.04	383.26	723.37
		time in line	0.20	63.13	219.84	499.16	962.39	1,909.10	16,175.72	53,765.80
	2 kernels time (s)	processing	255.79	257.62	258.97	259.90	261.93	284.62	333.82	388.27
		data transfer	53.64	64.29	70.02	70.59	91.89	275.96	452.71	683.43
		time in line	0.10	0.41	15.78	68.70	228.42	2,063.87	8,240.65	23,054.56

kernel processing times remained relatively constant among different number of threads. With 6 or more threads, an increasing deviation occurs; since the OpenCL's clEnqueue-Task command is a macro, probably the management of large numbers of threads start to affect the measurements of its time span.

For all datasets, the data transfer time increased with the number of threads (for a fixed implementation), and with the number of kernels (for a fixed number of threads); since the measurement is made on OpenCL commands, which involves transfer channel liberation, probably a channel access latency component was included.

With respect to the time in line, the consistency for all datasets can be observed in the table, by spotting the single thread case, which has a sequential nature; independent of the implementation (1 or 2 kernels), there will always be a kernel available and the observed waiting time is close to zero. In the case of 2 threads, for a 2 kernels implementation, the same occurs; however, for a 1 kernel implementation, a second thread has to wait for the kernel to be liberated from the computation of a first thread, therefore, the waiting time in line is larger, for example, 83.11 seconds for the simulated PacBio dataset.

Another observation regarding the time in line is that, for all datasets and implementations, it increases with the number of threads. That indicates that the kernels are getting more occupied and less available to take new jobs and clear the waiting queue. The management complexity for the threads, what affects the data dispatching time, is specially critic for 7 or 8 threads, probably due to the superposition with the operating system execution. This time in line effect has a strong impact on the total execution time, leading to the figures observed in Table 9.

The increase in time in line has indicated that 1 or 2 kernels were not always available for the threads, becoming the bottleneck of the integrated system. Besides that, not all availability of a second kernel is occupied, indicating a dispute in the PCIe interface. Still, the waiting time for the 2 kernels implementation, for all datasets, showed to be, for the varied number of threads, significantly shorter than the corresponding ones in the 1 kernel implementation. For example, comparison can be made between lines 4 and 7, 10 and 13, 16 and 19. This was expected since for 2 kernels implementation, chances of kernel availability is increased; this suggests that a larger number of kernels (therefore, more hardware resources) can bring better results as the number of threads also increases.

What the results tell about optimization:

- As expected, according to (WANG et al., 2020), PCIe turns out to be a bottleneck if the transfer rate is high. The traffic was reduced in this research by manipulating the data in the host, basically using anchor-extended sub-sequences instead of anchor-separated ones;
- Another improvement option would be addressing twice the number of BRAMs to a kernel, allowing the tile size to be twice as large, which would further reduce the transfer rate; this would allow fitting only one kernel in the device, but would also eliminate the PCIe channel conflict, and could potentially achieve a better performance than 2 parallel kernels in a device;
- The number of kernels should correspond to the number of software threads for the integrated system's best performance; if multi-threading is wished, a corresponding increment on hardware resources must accompany it;
- Further improvement, including hardware changes, is adding a traceback logic to the kernel (changing the Verilog implementation), which would reduce the data transfer sizes.

7 CONCLUSION AND FUTURE WORK

This project started with the identification of genome sequencing data as tending to migrate to longer reads (seen in the third-generation of sequencers), as they present several crucial advantages over short-reads from the previous generations. The computing stage has been identified as being the genome analysis pipeline's bottleneck, mainly because of the advances of highly parallelized sequencing of data which provides a great amount of genomic data to be processed, and because of the slow-down of computing power advances. One option for resolving this bottleneck is the development of DSAs, that use advantages of alternative architectures to target specific domains.

Minimap2 was identified as the State-of-the-Art algorithm for assembly of long-reads, and some works were already made on several types of processors, such as GPUs, FPGAs, and KNL to accelerated one of its main bottlenecks, the chaining or the aligning stages. No work that has successfully accelerated Minimap2's aligning step on an FPGA has been identified on the literature, which culminates on the proposition of this project.

GACT-X, an FPGA Cloud design for the SWG algorithm, was considered to be a good option for accelerating Minimap2's aligning step running on long-reads because of its limited memory consumption characteristic, which solves banded SWG's inherent linear complexity to the inputs' lengths.

Through the development of this work, the following conclusions could be drawn:

- There are genome data and tools available in public repositories to be used. Two sets of real human genome long-reads with high coverage from PacBio and ONT sequencing technologies were obtained from the NCBI and ENA databases. Another set of reads was generated with the PBSIM tool, used to simulate PacBio long-reads. These datasets were considered to be a reasonable sample for the experiments done in this research. The datasets presented different read length distributions. Simulated reads concentrated on shorter to medium lengths with average of 8,300 nucleotides; real PacBio reads concentrated on medium lengths with average of

13,300; and real ONT reads had a homogeneous distribution of lengths up to 100,000 nucleotides, with average of 16,900;

- An analysis of the Minimap2 algorithm running the datasets was important to understand its dynamics for later changes in the alignment or mapping step. Minimap2's accuracy and speed performances varied for the three datasets. The highest throughput of 403.46 kbases/s was for the simulated reads, that also had the lowest average length. The lowest throughput of 164.11 kbases/s was for the real PacBio reads, where the profiling indicated that it had the chaining step as a more significant bottleneck (50%) than the extending step (25%), whereas the other data had an inverted proportion. The highest frequencies of indels that showed in the final alignments were of a single nucleotide, although simulated reads caused an abnormal deviation from the alignment matrix's anti-diagonal, justified by the higher ratio of insertions. PacBio data presented high mapping rates, whereas 27.5% of ONT reads were unmapped;

- Minimap2's chaining process divided the read-reference pairs of sequences into many short (200-300 nt) sub-sequences. A direct substitution of the ksw alignment function in Minimap2 by a hybrid hardware implementation, aligning anchor-separated sub-sequences, resulted in a low performance due to data transfer latency. An alternative sequence execution by the aligner was developed and tested. Aligning longer anchor-extended sub-sequences resolved this (low performance) issue;

- GACT-X's alignment parameters were adapted to produce the most similar results to Minimap2, changing the Y-drop threshold to keep the maximum gap size in the band as in the original values. Internal wires defined in the Verilog code were increased to accommodate bigger tiles, and with tests on the simulated data, it was found that tiles of size 4,000 provided the best speed and accuracy performances, and such a configuration would be adopted in the Minimap2 GACT-X integrated solution. Experiments showed that, for part of the simulated data, GACT-X was 1.68x faster than ksw. With simulated data, 95.42% of the alignments had the same score as in Minimap2; real PacBio data presented a more dissimilar accuracy, with high rate of higher and lower scores; real ONT data had a considerable worse alignment (41.21%). Minimap2 and GACT-X aligned a similar proportion of the reads for the PacBio datasets, but interrupted early the alignment of many ONT reads;

- A Minimap2 GACT-X integrated system was developed to support multi-threading

and multi-kernel. Up to 8 threads and 2 kernels were implemented; the number of kernels was limited by the device's BRAM resources. The highest acceleration of 1.41x was observed on simulated and real ONT runs on 1 thread, and of 1.23x on real PacBio data (this lower rate can be linked to the lower occupation of the extending step observed in profiling); the general Minimap2 performance can be improved by hardware accelerating the chaining step;

- Detailed measurements for the kernel processing, data transfer and thread waiting execution times were performed in order to understand the limitations on the Minimap2-GACT-X integrated system. Acceleration decreased with more threads and could be explained by two factors: the kernels were not always available to process incoming data, increasing the time of threads waiting in line; multi-kernel generated conflict of data transfer in the PCIe channel. The first factor indicates that a larger amount of kernels would be needed for a larger number of threads; the second factor could be eased by implementing multi-devices (FPGAs) with one kernel in each having a dedicated PCIe channel.

Considering the results obtained in Chapter 6 and the analysis made on them, some future developments could be carried on to improve these results:

- A multi-FPGA design could sustain acceleration of more threads of Minimap2 with more kernels; for this, the implementation has to be updated to the Vitis environment, since SDAccel instances have been deprecated during the length of this Masters, and a new multi-FPGA AWS Instance has to be created; the increased cost must be evaluated;

- Tile sizes could be doubled by allocating all the BRAM available on the device to a single kernel to confirm if it could reduce data transfers, increase accuracy, and remove the PCIe competition;

- Traceback support could be added to the design to increase the acceleration, mainly by reducing the size of the read data (from traceback pointers to CIGAR strings); this implies in changing the Verilog RTL description with a probable increase in hardware area in the programmable logic;

- A second affine function could be added to the hardware design to properly mirror Minimap2's alignment scores, and increase the compared accuracy.

REFERENCES

ADEWALE, B. A. Will long-read sequencing technologies replace short-read sequencing technologies in the next 10 years? *African Journal of Laboratory Medicine*, v. 9, November 2020.

ALSER, M. et al. Accelerating genome analysis: A primer on an ongoing journey. *IEEE Micro*, Institute of Electrical and Electronics Engineers (IEEE), v. 40, n. 5, p. 65–75, September 2020. ISSN 1937-4143.

ALTSCHUL, S. F. et al. Basic local alignment search tool. *Journal of Molecular Biology*, v. 215, n. 3, p. 403–410, October 1990. ISSN 0022-2836.

AMARASINGHE, S. L. et al. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, v. 21, n. 30, February 2020.

AMAZON EC2 F1 Instances. 2022. ⟨https://aws.amazon.com/ec2/instance-types/f1/?nc1=h_ls⟩. Accessed: 2022-06-28.

AMAZON EC2 Instance Types. 2022. ⟨https://aws.amazon.com/ec2/instance-types/?nc1=h_ls⟩. Accessed: 2022-06-28.

AMAZON Web Services. 2022. ⟨https://aws.amazon.com/?nc1=h_ls⟩. Accessed: 2022-06-28.

ANTIPOV, D. et al. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, v. 32, n. 7, p. 1009–1015, 11 2015. ISSN 1367-4803.

BASE Quality Score Recalibration (BQSR). 2020. ⟨https://github.com/broadinstitute/gatk-docs/blob/master/gatk3-methods-and-algorithms/Base_Quality_Score_Recalibration_(BQSR).md⟩. Accessed: 2022-06-28.

BURROWS-WHEELER Aligner. 2010. ⟨http://bio-bwa.sourceforge.net/⟩. Accessed: 2022-06-28.

CHAISSON, M. J.; TESLER, G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC Bioinformatics*, v. 13, September 2012.

CHI, K. R. The year of sequencing. *Nature Methods*, v. 5, p. 11–14, January 2008.

CMAKE. 2022. ⟨https://cmake.org/⟩. Accessed: 2022-06-28.

CNNSCOREVARIANTS. 2020. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037226672-CNNScoreVariants⟩. Accessed: 2022-06-28.

COSTER, W. D. et al. Structural variants identified by oxford nanopore promethion sequencing of the human genome. *Genome Research*, v. 29, June 2019. ISSN 1178-1187.

DARWIN-WGA. 2019. ⟨https://github.com/gsneha26/Darwin-WGA⟩. Accessed: 2022-06-28.

DEVELOPING on AWS F1 with SDAccel and RTL Kernels - Part 1 of 4. 2022. ⟨https://www.xilinx.com/video/software/developing-on-aws-f1-with-sdaccel-and-rtl-kernels-part1.html⟩. Accessed: 2022-06-28.

EUROPEAN Nucleotide Archive. 2022. ⟨https://www.ebi.ac.uk/ena/browser/home⟩. Accessed: 2022-06-28.

FENG, Z. et al. Accelerating long read alignment on three processors. In: *Proceedings of the 48th International Conference on Parallel Processing*. New York, NY, USA: Association for Computing Machinery, 2019. (ICPP 2019). ISBN 9781450362955.

FERRAGINA, P.; MANZINI, G. Opportunistic data structures with applications. In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. [S.l.: s.n.], 2000. p. 390–398. ISSN 0272-5428.

FILTERVARIANTTRANCHES. 2019. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037227632-FilterVariantTranches⟩. Accessed: 2022-06-28.

FRITH, M. C.; HAMADA, M.; HORTON, P. Parameters for accurate genome alignment. *BMC bioinformatics*, v. 11, n. 80, February 2010.

FUJIKI, D. et al. Seedex: A genome sequencing accelerator for optimal alignments in subminimal space. In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. [S.l.: s.n.], 2020. p. 937–950.

FUNCOTATOR. 2022. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037224432-Funcotator⟩. Accessed: 2022-06-28.

GADSBøLL, K. et al. Current use of noninvasive prenatal testing in europe, australia and the usa: A graphical presentation. *Acta Obstetricia et Gynecologica Scandinavica*, v. 99, n. 6, p. 722–730, 2020.

GENERA. 2022. ⟨https://www.genera.com.br/⟩. Accessed: 2022-06-28.

GENOME Analysis Toolkit. 2022. ⟨https://gatk.broadinstitute.org/hc/en-us⟩. Accessed: 2022-06-28.

GEORGAKOPOULOS-SOARES, I. et al. Emt factors and metabolic pathways in cancer. *Frontiers in Oncology*, v. 10, n. 80, 2020. ISSN 2234-943X.

GNU gprof. 1998. ⟨https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html⟩. Accessed: 2022-06-28.

GOOGLE Cloud. 2022. ⟨https://cloud.google.com/⟩. Accessed: 2022-06-28.

GOTOH, O. Optimal sequence alignment allowing for long gaps. *Bulletin of Mathematical Biology*, v. 52, p. 359–373, May 1990.

GOYAL, A. et al. Ultra-fast next generation human genome sequencing data processing using dragen™ bio-it processor for precision medicine. *Open Journal of Genetics*, v. 7, p. 9–19, March 2017.

GRCH38. 2013. ⟨https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26⟩. Accessed: 2022-06-28.

GUO, L. et al. 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)*. [S.l.: s.n.], 2019. p. 127–135. ISSN 2576-2621.

HAPLOTYPECALLER. 2022. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037225632-HaplotypeCaller⟩. Accessed: 2022-06-28.

HENG, J.; HENG, H. H. Karyotype coding: The creation and maintenance of system information for complexity and biodiversity. *Biosystems*, v. 208, p. 104476, October 2021. ISSN 0303-2647.

HENNESSY, J. L.; PATTERSON, D. A. A new golden age for computer architecture. *Communications of the ACM*, v. 62, n. 2, p. 48–60, 2019.

HUAWEI CLOUD. 2022. ⟨https://www.huaweicloud.com/intl/en-us/⟩. Accessed: 2022-06-28.

HUMAN 54x Dataset. 2014. ⟨http://datasets.pacb.com/2014/Human54x/fast.html⟩. Accessed: 2022-06-28.

HUMAN Genome Project Information Archive. 2003. ⟨https://web.ornl.gov/sci/techresources/Human_Genome/index.shtml⟩. Accessed: 2022-06-28.

ILLUMINA. 2022. ⟨https://www.illumina.com/⟩. Accessed: 2022-06-28.

KAPLAN, R.; YAVITS, L.; GINOSAR, R. Rassa: Resistive prealignment accelerator for approximate dna long read mapping. *IEEE Micro*, v. 39, n. 4, p. 44–54, July 2019. ISSN 1937-4143.

KOLIOGEORGI, K. et al. Dataflow acceleration of smith-waterman with traceback for high throughput next generation sequencing. In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. [S.l.: s.n.], 2019. p. 74–80. ISSN 1946-1488.

KUSHNICK, T. Thompson & thompson genetics in medicine. *JAMA*, v. 267, n. 15, p. 2115–2115, April 1992. ISSN 0098-7484.

LANGMEAD, B.; SALZBERG, S. L. Fast gapped-read alignment with bowtie 2. *Nature Methods*, v. 9, p. 357–359, March 2012.

LASZLO, A. H. et al. Decoding long nanopore sequencing reads of natural dna. *Nature Biotechnology*, v. 32, p. 829–833, June 2014.

LI, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, v. 34, n. 18, p. 3094–3100, September 2018. ISSN 1367-4803.

LIAO, Y.-L. et al. Adaptively banded smith-waterman algorithm for long reads and its hardware accelerator. In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. [S.l.: s.n.], 2018. p. 1–9. ISSN 2160-052X.

LIU, T. et al. A benchmarking study of sars-cov-2 whole-genome sequencing protocols using covid-19 patient samples. *iScience*, v. 24, n. 8, p. 102892, 2021. ISSN 2589-0042.

MANTERE, T.; KERSTEN, S.; HOISCHEN, A. Long-read sequencing emerging in medical genetics. *Frontiers in Genetics*, v. 10, p. 426, 2019. ISSN 1664-8021.

MAXFIELD, C. M. Chapter 3 - the origin of fpgas. In: MAXFIELD, C. M. (Ed.). *The Design Warrior's Guide to FPGAs*. Burlington: Newnes, 2004. p. 25–56. ISBN 978-0-7506-7604-5.

MINIMAP2-ACCELERATION. 2021. ⟨https://github.com/UCLA-VAST/minimap2-acceleration⟩. Accessed: 2022-06-28.

MUTEXES. 2022. ⟨https://www.gnu.org/software/libc/manual/html_node/ISO-C-Mutexes.html⟩. Accessed: 2022-06-28.

NCBI. 2022. ⟨https://www.ncbi.nlm.nih.gov/⟩. Accessed: 2022-06-28.

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, v. 48, n. 3, p. 443–453, 1970. ISSN 0022-2836.

ONO, Y.; ASAI, K.; HAMADA, M. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, v. 29, n. 1, p. 119–121, January 2013. ISSN 1367-4803.

OPENCL Reference Pages. 2011. ⟨https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml/⟩. Accessed: 2022-06-28.

ORTEU, A.; JIGGINS, C. D. The genomics of coloration provides insights into adaptive evolution. *Nature Reviews Genetics*, v. 21, 2020. ISSN 1471-0064.

OXFORD NANOPORE Technologies. 2022. ⟨https://nanoporetech.com/⟩. Accessed: 2022-06-28.

PACBIO. 2022. ⟨https://www.pacb.com/⟩. Accessed: 2022-06-28.

PALAZZO, A. F.; GREGORY, T. R. The case for junk dna. *PLOS Genetics*, Public Library of Science, v. 10, n. 5, p. 1–8, May 2014.

PBSV. 2022. ⟨https://github.com/PacificBiosciences/pbsv⟩. Accessed: 2022-06-28.

PFEUFER, V.; SCHULZE, M. Genetics/dna sequencing: Laser fluorescence powers sequencing advances. *Laser Focus World*, January 2015. Accessed: 2022-06-28.

POPLIN, R. et al. A universal snp and small-indel variant caller using deep neural networks. *Nature Biotechnology*, v. 36, November 2018. ISSN 1546-1696.

PTHREAD_SELF. 2021. ⟨https://man7.org/linux/man-pages/man3/pthread_self.3.html⟩. Accessed: 2022-06-28.

REUTER, J. A.; SPACEK, D. V.; SNYDER, M. P. High-throughput sequencing technologies. *Molecular cell*, v. 58, n. 4, p. 586–597, 2015.

RUN: ERR2585114. 2018. ⟨https://www.ebi.ac.uk/ena/browser/view/ERR2585114?show=reads⟩. Accessed: 2022-06-28.

SAMURA, O. Update on noninvasive prenatal testing: A review based on current worldwide research. *Journal of Obstetrics and Gynaecology Research*, v. 46, n. 8, p. 1246–1254, 2020.

SANGER, F.; NICKLEN, S.; COULSON, A. R. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, v. 74, n. 12, December 1977. ISSN 5463–5467.

SDACCEL Environment. 2019. ⟨https://www.xilinx.com/htmldocs/xilinx2019_1/sdaccel_doc/index.html⟩. Accessed: 2022-06-28.

SEGMENTATION Faults. 2022. ⟨https://matrixzj.github.io/documentations/memory_segfaults.html⟩. Accessed: 2022-06-28.

SEQUENCE Alignment/Map Format Specification. 2021. ⟨http://samtools.github.io/hts-specs/SAMv1.pdf⟩. Accessed: 2022-06-28.

SMITH, T. F.; WATERMAN, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology*, v. 147, n. 1, p. 195–197, 1981. ISSN 0022-2836.

ŠOŠIC, I. et al. Fast and sensitive mapping of nanopore sequencing reads with graphmap. *Nature Communications*, v. 7, March 2016. ISSN 11307.

ŠOŠIC, M.; ŠIKIC, M. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, v. 33, p. 1394–1395, May 2017. ISSN 1394-1395.

SRX9063500: PacBio SMRT whole genome sequencing of Sri Lankan Tamil H. sapiens. 2019. ⟨https://www.ncbi.nlm.nih.gov/sra/SRX9063500[accn]⟩. Accessed: 2022-06-28.

SUZUKI, H.; KASAHARA, M. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC bioinformatics*, v. 19, n. 45, February 2018.

TENG, C. et al. Accelerating the base-level alignment step of dna assembling in minimap2 algorithm using fpga. In: *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*. [S.l.: s.n.], 2021. p. 1–4. ISSN 2473-4667.

THIEL, V. et al. Mechanisms and enzymes involved in sars coronavirus genome expression. *Journal of General Virology*, Microbiology Society, v. 84, n. 9, p. 2305–2315, 2003. ISSN 1465-2099.

TMUX. 2022. ⟨https://github.com/tmux/tmux⟩. Accessed: 2022-06-28.

TURAKHIA, Y.; BEJERANO, G.; DALLY, W. J. Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly. *SIGPLAN Not.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 2, p. 199–213, February 2018. ISSN 0362-1340.

TURAKHIA, Y. et al. Darwin-wga: A co-processor provides increased sensitivity in whole genome alignments with high speedup. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. [S.l.: s.n.], 2019. p. 359–372. ISSN 2378-203X.

VITIS Unified Software Platform. 2022. ⟨https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html⟩. Accessed: 2022-06-28.

VMACCEL. 2022. ⟨https://www.vmaccel.com/⟩. Accessed: 2022-06-28.

WANG, X. et al. When fpga meets cloud: A first look at performance. *IEEE Transactions on Cloud Computing*, v. 10, p. 1344–1357, May 2020.

APPENDIX A

This appendix presents the programming options adopted to integrate GACT-X's host lines into Minimap2's original source code. A new C++ file "gactx.cpp" was created and added to Minimap2's file and to the compilation object list. It contains the functions:

- **"load_file_to_memory"** loads the xclbin binary file to boot the kernel in the FPGA;
- **"fpga_configuration_and_setup"** selects a Xilinx platform from the ones available; selects a target device; creates context, command queue, program, and kernel variables; defines memory bank mapping; and creates input and output buffers;
- **"fpga_shutdown_and_cleanup"** releases memory objects and other global variables;
- **"gactx_align"** configures tile size, tile overlap, and alignment scores; sends alignment input sequences; programs the tile algorithm; reads the tile outputs; encodes the final CIGAR string; and updates Minimap2's alignment results.

The compilation was changed to using CMake (CMAKE, 2022) with the Xilinx's compiler "xcpp", and inclusion of directories, libraries, and flags necessary for running the host's code. OpenCL variables cl_context, cl_command_queue, cl_program, cl_kernel, cl_mem_ext_ptr_t, cl_mem, cl_event, as well as the pointers to the host's memory were initialized as global variables so that they can be shared among the multiple threads in the multi-threading option of Minimap2.

cl_kernel, cl_mem_ext_ptr_t, cl_mem, and host memory pointers are initialized as arrays of length (NUM_KERNELS), which is a #define that can be set as 1 or 2 by the developer, and indicates the number of kernels being utilized in the circumstance. A similar global int array "kernel_expanding" is used to indicate whether each kernel is free or occupied with 0 or 1 respectively.

The aligning process in Minimap2 is set up in the “mm_align1” function in “align.c”. It is where ksw calls are made for each sub-sequence between two anchors. This was substituted with a complete right-extend from the first anchor using GACT-X, whenever both of the input sequences are longer than 1,000 nucleotides.

To couple multiple CPU threads with 1 or 2 kernels, a FIFO queue (global array variable) was created. Each thread that reaches the alignment stage goes to the end of the queue, and waits until it is the first in line and there is a kernel available. This is done using “pthread_self” (PTHREAD_SELF, 2021) as the thread identifier. Mutex (MUTEXES, 2022) is used to protect lines that edit the FIFO queue, to avoid conflicting memory access. The function “gactx_align” is called after the thread has picked its corresponding kernel and left the queue.

When running 2 kernels in parallel, there can be conflict in the PCIe channel (e.g. when two threads send data to the FPGA at the same time). This can result in Segmentation Faults (SEGMENTATION..., 2022). The cl_event global variable is used to synchronize these transfers.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 31 de Agosto de 2022

Assinatura do autor: Carolina Teng

Assinatura do orientador:

Catalogação-na-publicação

,

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine / Volte e preencha o campo Autor -- versão corr. -- São Paulo, 2022.
132 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.CIRCUITOS FPGA 2.COMPUTAÇÃO EM NUVEM 3.BIOINFORMÁTICA 4.GENÔMICA 5.ALGORITMOS I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

CAROLINA TENG

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine

Corrected version

São Paulo
2022

CAROLINA TENG

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine

Corrected version

Dissertation presented to the Polytechnic School of University of São Paulo for obtainment of the Title of Master of Science.

São Paulo
2022

CAROLINA TENG

Accelerating the alignment phase of Minimap2 genome assembly algorithm Using GACT-X in a commercial Cloud FPGA machine

Corrected version

Dissertation presented to the Polytechnic School of University of São Paulo for obtainment of the Title of Master of Science.

Area of Concentration:

Microelectronics (Graduate Program in Electrical Engineering)

Advisor:

Fernando Josepetti Fonseca

São Paulo
2022

ACKNOWLEDGMENTS

This author is extremely grateful first and foremost to Professor Wang Jiang Chau, for accompanying, teaching, and advising her throughout the entirety of her Master's degree. The amount of learning acquired in this trajectory is of immeasurable value, from technical subjects to introduction to what it is to have scientific thinking. Thanks to Professor Wang for the friendship, tutoring, and great dedication, without which the quality and detail present in this project and in this text would not have been obtained. The shared excitement for solving bugs, relief for achieving good results, and exhaustion from writing until late will never be forgotten and will be greatly missed.

Thanks to the author's colleague Renan Weege Achjian, who shares the same interests in Bioinformatics and FPGA projects, who have helped and followed this author with many issues in the development stages, and have also cheered for all the accomplishments on the way. It has been the best companionship and teamwork one could ask for, and hopefully many more projects and events will be shared in the future.

Thanks to Mauricio Perez for maintaining the Wiener server from Department of Electronic Systems Engineering, and for helping this author in many technical issues. Thanks to Professor Carlos Menck for providing access to the Seal server from *Instituto de Ciências Biomédicas*. Thanks to the qualification board members Ricardo Pires and Fernando Josepetti Fonseca for reviewing the qualification text and suggesting improvements and corrections.

Finally, thanks to the author's family and friends for all the support.

ABSTRACT

TENG, C. **Accelerating the alignment phase of Minimap2 genome assembly algorithm using GACT-X in a commercial Cloud FPGA machine**. 2022. Dissertation (Masters) - Polytechnic School, University of São Paulo, São Paulo, 2022.

Genetic sequencing can provide crucial information in medicine and in biology studies. The technologies developed in the field are advancing rapidly and the current third-generation of genome sequencers have significant improvements over the second-generation. In parallel to that, sequencing throughput has been increasing at an exponential rate, which, coupled with price reduction, has resulted in a leap of generation of genomic data to be processed. Transistor technology is reaching its fundamental limits, and Moore's Law is becoming obsolete, so other alternatives are required to efficiently process such an amount of data.

Long-reads from the third generation of sequencers are shown to be an emerging type of genetic data, with average lengths of thousands of nucleotides each. State-of-the-Art algorithm Minimap2 is able to assemble these reads into the genome that was sampled, but it is a computationally-intensive process: for the human genome size with sufficient coverage, running times can reach up to dozens of CPU hours. Hardware acceleration has been proposed as an effort to make Minimap2 more efficient, but up to the present moment, only one of its main bottlenecks, the chaining step, has been successfully accelerated on FPGA. No efficient solution has been proposed for the aligning step, implemented as the ksw function. GACT-X is a Cloud FPGA design that performs a banded SWG alignment with fixed memory, suitable for any size of input.

GACT-X with tiles of size 4,000 can be 2x faster than ksw when aligning long sequences. Replacing the alignment function ksw in Minimap2 with GACT-X on a Cloud hybrid system can provide up to 1.41x acceleration on the entire execution to the software counterpart, with comparable accuracy for data that have high similarity to the reference genome. This dissertation presents all the relevant background information, the development stages and methods, the results achieved on three different datasets, and the proposed future work on this acceleration project.

Keywords: Cloud Computing, Minimap2, Field Programmable Gate Arrays, Smith-Waterman-Gotoh, Co-processors, Acceleration, Genomics.

RESUMO

TENG, C. ***Accelerating the alignment phase of Minimap2 genome assembly algorithm using GACT-X in a commercial Cloud FPGA machine***. 2022. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2022.

O sequenciamento genético pode forneccer informações cruciais em medicina e em estudos de biologia. As tecnologias desenvolvidas na área estão avançando rapidamente e a atual terceira-geração de sequenciadores de genoma possuem melhorias significantes sobre a segunda-geração. Paralelamente a isso, a taxa de sequenciamento vem aumentando exponencialmente, o que, aliado à redução de preços, resultou em um salto de geração de dados genômicos a serem processados. A tecnologia de transistores está atingindo seus limites fundamentais, e a Lei de Moore está se tornando obsoleta, então outras alternativas são necessárias para processar tal quantidade de dados.

Long-reads da terceira geração de sequenciadores são um tipo emergente de dados genéticos, com comprimentos médios de milhares de nucleotídeos cada. O algoritmo do Estado-da-Arte Minimap2 é capaz de montar essas *reads* de volta ao genoma que foi amostrado, mas é um processo computacionalmente intensivo: para o tamanho do genoma humano com cobertura suficiente, os tempos de execução podem chegar a dezenas de horas de CPU. Aceleraçao em hardware foi proposta como uma aplicação para tornar o Minimap2 mais eficiente, mas até o presente momento, apenas um de seus principais gargalos, a etapa de *chaining*, foi acelerada com sucesso em FPGA. Nenhuma solução eficiente foi proposta para a etapa de alinhamento, implementada como a função ksw. O GACT-X é um design de FPGA em nuvem que executa o alinhamento de SWG em banda, com consumo de memória fixo, adequado para qualquer tamanho de entrada.

O GACT-X com *tiles* de tamanho 4.000 pode ser 2x mais rápido que o ksw ao alinhar sequências longas. Substituir a função de alinhamento ksw no Minimap2 pelo GACT-X em um sistema híbrido na nuvem pode proporcionar aceleração de até 1,41x sobre toda a execução do software, com precisão comparável para dados que têm alta similaridade com o genoma de referência. Esta dissertação apresenta todas as informações básicas relevantes, as etapas e os métodos desenvolvimento, os resultados alcançados em três conjuntos de dados diferentes e o trabalhos futuros propostos para este projeto de aceleração.

Palavras-Chave – Computação na Nuvem, Minimap2, Arranjo de Porta Programável em Campo, Smith-Waterman-Gotoh, Co-processadores, Aceleração, Genômica.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AFI	Amazon FPGA Image
AMI	Amazon Machine Image
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
AWS	Amazon Web Services
AXI	Advanced eXtendible Interface
BAM	Binary (Sequence) Alignment Map
BRAM	Block RAM
BSW	Banded SW
BWA	Burrows-Wheeler Aligner
CIGAR	Compact Idiosyncratic Gapped Alignment Report
CLB	Configurable Logic Block
CL	Custom Logic
CLR	Continuous Long Read
CMOS	Complementary Metal-Oxide-Semiconductor
CNV	Copy Number Variation
CPU	Central Process Unit
CUDA	Compute Unified Device Architecture
DDR	Dual Data Rate
DFE	Data-Flow Engine
DMA	Direct Memory Access
DNA	DeoxyriboNucleic Acid
DP	Dynamic Programming
DSA	Domain-Specific Architecture
ENA	European Nucleotide Archive
FCCM	Field-programmable Custom Computing Machines
FF	Flip-Flop
FIFO	First In, First Out
FPL	Field Programmable Logic
GATK	Genome Analysis ToolKit
GPU	Graphics Processing Unit
GRCh38	Genome Reference Consortium human build 38
HDL	Hardware Description Language
HGP	Human Genome Project
HLS	High Level Synthesis
hsa	homo sapiens organism code
IOB	Input/Output Block
I/O	Input/Output
ISA	Instruction Set Architecture
KNL	KNights Landing
LUT	Look-Up Table
MEM	Maximal Exact Match

MM	Memory Mapped
MPSoC	Multi-Processor System-on-Chip
MUX	MUltipleXer
NCBI	National Center for Biotechnology Information
NUMA	Non-Uniform Memory Access
NW	Needleman-Wunsch
ONT	Oxford Nanopore Technologies
OpenCL	Open Computing Language
OS	Operating System
PacBio	Pacific Biosciences
PCIe	Peripheral Component Interconnect express
PCR	Polymerase Chain Reaction
PE	Processing Element
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computer
RNA	RiboNucleic Acid
RTL	Register Transfer Level
SAM	Sequence Alignment Map
SD	Secure Digital
SD	Standard Deviation
SDK	Software Development Kit
SIMD	Single Instruction, Multiple Data
SK	Suzuki-Kasahara
SLR	Super Logic Region
SMRT	Single Molecule Real-Time
SNV	Single Nucleotide Variant
SRAM	Static RAM
SSE	Streaming SIMD Extensions
SV	Structural Variant
SW	Smith-Waterman
sw	software
SWG	Smith-Waterman-Gotoh
USD	United States Dollar
VCF	Variant Call Format
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WC	Watson-Crick
WGA	Whole Genome Alignment
XML	eXtensible Markup Language

CONTENTS

1 INTRODUCTION

Genetic sequencing can provide information in medicine for a wide variety of uses. In preventative medicine, some genetic variations are shown to raise the risk for certain diseases, such as breast cancer, heart diseases, and type II diabetes, for which therapies or preventative strategies are available. Genetic sequencing can also be used to identify current or future genetic diseases, such as muscular dystrophy and Waardenburg syndrome. This information can help with life planning and earlier treatment of symptoms. Another use of genetic sequencing is to help asses whether a person is a carrier of variants that might cause disease in their children, but does not affect themselves, such as cystic fibrosis, fragile X syndrome, and sickle cell anemia (KUSHNICK, 1992).

Collecting and analyzing genetic data is also crucial in biology studies, such as in biodiversity (HENG; HENG, 2021), evolution (ORTEU; JIGGINS, 2020) and metabolic pathways (GEORGAKOPOULOS-SOARES et al., 2020). It has even become a commodity of public interest, providing people insights on their ancestry and personal phenotypes (GENERA, 2022).

Current sequencing technology is not capable of reading straightaway a complete human genome strand, so each DNA (deoxyribonucleic acid) molecule in the sample needs to be cleaved into many much smaller sequences, called fragments. The sequenced fragments are then called reads. Each read can also be understood as the equivalent sequence translated into nucleotide bases represented by characters (A, T, C, G) for computer processing. Depending on the technology used in the sequencing process, there can be two types of reads: short-reads, with lengths of a few hundreds of bases (ILLUMINA, 2022), and long-reads, with many thousands of bases on average (PACBIO, 2022)(OXFORD. . . , 2022).

Currently, the so-called second-generation of sequencing technology produces short-reads with high throughput, dominates the market (ADEWALE, 2020) and is expected to stay prevalent for the next years. However, only the emerging third-generation of sequencers that produce long-reads is able to identify long alterations in the DNA (MAN-

TERE; KERSTEN; HOISCHEN, 2019). The slow transition to long-reads is explained mainly by their higher sequencing cost compared to short-reads. Nowadays hybrid long- and short-reads solutions for genome assembly are implemented in needed situations (ANTIPOV et al., 2015).

Several algorithms and programs have been developed to assemble reads back into the original complete DNA sequence (LI, 2018)(BURROWS-WHEELER..., 2010)(LANGMEAD; SALZBERG, 2012). Some of the assembling algorithms map the reads to a genome reference, which is used as a guide (GRCH38, 2013). First, approximate mapping positions are identified for the read in the reference. Then, the reads are aligned to the reference's mapping region to pinpoint the variations in the genome that has been sequenced.

Some classic algorithms, such as the Smith-Waterman-Gotoh (SWG) algorithm (SMITH; WATERMAN, 1981)(GOTOH, 1990), use dynamic-programming to align two character strings. It was published in 1981-1990 and is still used to this day in read assembly programs, but with many heuristics and transformations added on, such as calculating only a portion of the matrix where the optimal alignment is more probable to be (FUJIKI et al., 2020)(LIAO et al., 2018).

Read assembly is currently considered a major bottleneck in the entire genome analysis pipeline (ALSER et al., 2020), that includes laboratory sampling, sequencing, processing and annotating. Sequencing technologies have been increasing their output capacity in terms of number of reads at an exponential rate (REUTER; SPACEK; SNYDER, 2015), whereas computational power has been slowly reaching the limits of transistor technology (HENNESSY; PATTERSON, 2019). Read assembly is also the bottleneck of the genome processing pipeline (GENOME..., 2022), which corresponds to the technical activities involved in genetic sequencing.

Although the cost of sequencing DNA is a more significant impediment for the technology's diffusion in clinical settings, the speed of acquiring and interpreting genomic information is crucial in certain applications. For instance, prenatal testing is performed on women during pregnancy to assess whether the fetus could be born with a genetic condition or birth defect, which can be helpful to determine the management of the pregnancy and delivery (SAMURA, 2020; GADSBøLL et al., 2020). Pathogen genomics can be used in diagnosing infections, investigating outbreaks and describing transmission patterns, and has been highly present in the latest COVID-19 pandemic (LIU et al., 2021; THIEL et al., 2003).

In order to improve the computational performance in software-based algorithms, one alternative is designing specialized hardware for the assembly task; particularly with application specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). ASICs are more costly due to customized manufacturing; only a high-volume production would justify its adoption. FPGAs, on the other hand, are configurable devices that can surpass software performance without requiring a high manufacturing capital.

Several FPGA acceleration articles can be found in the read assembly acceleration literature, and some have achieved improvements in processing time (KOLIOGEORGI et al., 2019)(FUJIKI et al., 2020)(GUO et al., 2019). Given the variety of new algorithms and tools that are developed in the field every year, and the steady evolution of sequencing technology that now produces considerably longer reads, it is crucial for the hardware research to constantly adapt to the new changes.

1.1 Motivation

Many programs, such Bowtie2 (LANGMEAD; SALZBERG, 2012) have been developed to efficiently assemble short-reads at the second generation technology. However, their strategies are often not suitable to process long-reads, having their performance hampered. Hardware acceleration that has been proposed for alignment in these algorithms on a few hundred nucleotides (KOLIOGEORGI et al., 2019) (FUJIKI et al., 2020) can't support sequences of many thousands of nucleotides. This is because the SWG algorithm with banding has two scaling aspects: the number of stored backtracking pointers grows with linear proportion to the inputs' lengths, and the cumulative alignment scores used in the wave-front expansion increase with the number of matching nucleotides.

BWA-MEM (BURROWS-WHEELER..., 2010) and Minimap2 (LI, 2018) are examples of programs/algorithms proposed to fill the software gap, the latter being 50 times faster than the former one, besides having better mapping accuracy than most other programs; therefore, Minimap2 is one of the current State-of-the-Art algorithms for assembling long-reads. Minimap2 has a chaining algorithm that takes advantage of the higher load of information carried by long-reads to find approximate mapping positions with better performance and accuracy. It also uses a transformed version of the SWG algorithm, proposed by Suzuki and Kasahara (SUZUKI; KASAHARA, 2018), for the alignment (or extending) step, that limits data size in Streaming SIMD (Single Instruction, Multiple Data) Extensions (SSE) vector instructions to optimize parallelization of computation of cells.

Still, mapping one human sample of reads to a reference using Minimap2 is very time consuming, taking dozens of Central Processing Unit (CPU) hours on an Intel Xeon processor, which further increases the already expensive genome testing budget. This is because the human genome is really long (approximately 3.2 GB of data) and read information need to be 8-17 times larger in order to get enough map coverage for reliable clinical standard (i.e. to statistically cover sequencing errors and mapping heuristics, and to identify heterozygous states) (AMARASINGHE et al., 2020). Some laboratories even need to invest in powerful and power hungry compute clusters just to meet the computational demands.

Previous studies have determined that the chaining and extending steps of Minimap2 are its run-time bottlenecks, taking together around 70% of the total execution time. Focused on this, authors of such works have already successfully accelerated Minimap2's chaining step on FPGA and Graphic Processing Unit (GPU) (GUO et al., 2019). Others

have also implemented the extending step on CPU, GPU and KNL (Knights Landing) (FENG et al., 2019), with an acceleration of 3.9x times with 128 CUDA (Compute Unified Device Architecture) streams with 512 threads each in the GPU case; but achieved lower performance per thread in the hardware designs compared to software. To the author's knowledge, no work in the literature has successfully accelerated Minimap2's extending step on an FPGA.

Designing an application as hardware is more complex than programming it as software. In addition, the design of an FPGA accelerator takes a longer time, and is usually more expensive than programming a GPU accelerator. So it is important to assess which application can achieve a better performance. According to the authors of (GUO et al., 2019), FPGA may achieve a better performance in several applications, as in the case of Minimap2, for the following reasons:

- When the algorithm includes a great amount of checking and filtering options, many control and branch conditions are required, almost at the same rate as the integer and float arithmetics. On FPGAs, these control logic will add to the pipeline without having a significant impact on throughput;
- Tasks with irregular-width integer type might be more suitable for FPGAs;
- GPUs need to unpack the data structure, adding instructions to some applications, whereas FPGAs' inputs are kept in BRAM (Block Random-Access Memory) and used immediately with combinatorial logic;
- GPUs' performance relies on parallelism, but read assembling, although parallelizable along reads, has an extensive processing pipeline for each read.

The extending step of Minimap2 is based on a banded SWG algorithm optimized for wave-front parallelization of cells with SSE vector instructions, that consumes time and memory in linear proportion to the input's lengths. The chaining algorithm of Minimap2 breaks the sequence pairs into many sub-sequences that are sent separately to the extending function. With this, the extending module is prepared to deal with input data that has an average length of a few hundred bases, but that sometimes can also be very long. This variation of sizes requires a new hardware design that can compensate for two issues: first, short sequences won't be worth sending to co-processors separately, because even if they can be processed faster in the FPGA, the time spent transferring the data alone will be higher than the processing time taken by the software (TENG et al., 2021);

second, long sequences can extrapolate the hardware's resources that calculate banded SWG matrices.

Darwin (TURAKHIA; BEJERANO; DALLY, 2018) was published at about the same time as Minimap2 and is a software-hardware hybrid system that also assembles long-reads, achieving up to 183.8x speedup over Edlib (ŠOŠIC; ŠIKIC, 2017). Its initial filtering stage runs on software, presenting a high RAM (Random-Access Memory) consumption (30-64GB of RAM for the human genome), which makes Darwin efficient only for sequencing cases with references of limited size. Their extending step GACT, which also calculates the SWG matrix, has been accelerated on an FPGA and was adapted to work with inputs of any length, since they limit BRAM consumption in hardware by dividing the matrix into tiles of fixed size.

The same authors of Darwin also published the software-hardware hybrid whole genome aligner Darwin-WGA that came with an improved extending architecture GACT-X (TURAKHIA et al., 2019). It is not a read assembler, but a single long stream aligner. It would be possible to adapt Darwin-WGA to run as a read assembler by transferring the GACT-X module into Darwin's software support (since both of them use the D-SOFT filtering algorithm) after taking care of the RAM consumption issue. GACT-X is 2x faster than GACT and was developed on the Amazon Web Services (AWS) Cloud platform (AMAZON..., 2022c), so it can be easily replicated for other uses. The re-purposing of Darwin-WGA's GACT-X module, designed on an FPGA, could potentially accelerate Minimap2's extending bottleneck with a Cloud hybrid architecture.

In recent years, Cloud FPGAs (AMAZON..., 2022a) (HUAWEI..., 2022) (VMACCEL, 2022) have become a compelling alternative to reduce the initial cost of hardware implementation, by charging a smaller value per time of usage. Cloud FPGAs work the same way as Cloud computers: a big cluster of processors is physically stored and managed by a company that offers their clients scalable storage and computing capacity, charging in proportion to their usage demand. Some companies like Amazon (AMAZON..., 2022c) offer other advantages to developers. For instance, their FPGA comes with a Shell that wraps any kernel to fit the FPGA's I/O (Input/Output) resources, and projects developed on their platform can be encrypted and advertised on their marketplace, with all intellectual property secured. Therefore, an acceleration scheme can be tested with reduced implementation overhead; only after the implementation has shown to be successful, the designer may decide to move to a customized board implementation, or to rely on the Amazon platform for business or research.

Although Cloud FPGA providers announce large accelerations for applications when compared to software implementations (for instance, Amazon claims that AWS FPGA Instances can accelerate compute-bound applications up to 100x), the real acceleration is limited by virtualized PCIe (Peripheral Component Interconnect Express) transfer's throughput and latency, which is dependent on the driver's technology and whether virtual machines are adding extra virtual interrupts. Previous work (WANG et al., 2020) has measured the performance gap between Cloud FPGAs and physical implementation PCIe transfers and indicated that it is the limiting bottleneck for communication-intensive workloads, which in their examples did not perform over 2x faster than the software counterparts. Therefore, the adoption of Cloud FPGAs for Minimap2 must be analyzed and carefully considered.

1.2 Objectives

This Master's dissertation presents the development steps of a software-hardware hybrid system that accelerates the State-of-the-Art program Minimap2's aligning step by using the Cloud FPGA module GACT-X from Darwin-WGA in substitution to the Suzuki-Kasahara's algorithm. The specific objectives are:

- **Data acquisition.** The selection of a set of human genome data suitable to the experiments proposed in this dissertation is needed. Human sequencing data can be downloaded from many different open source repositories, such as the United State's National Center for Biotechnology Information (NCBI) (NCBI, 2022) and the European Nucleotide Archive (ENA) (EUROPEAN..., 2022); reads data can also be generated with simulating tools such as PBSIM (ONO; ASAI; HAMADA, 2013). The consensus human genome reference is updated from time to time and its versions are submitted at NCBI (GRCH38, 2013).

- **Evaluation of Minimap2.** First, Minimap2's processing time bottlenecks need to be determined for long-read datasets that have different read length distributions. Minimap2's performance running on AWS' CPU machines need to be measured and used as a reference to assess the acceleration. Minimap2's mapping and aligning accuracies need to be collected for the same purpose. Finally, some internal data aspects need to be measured, to identify potential issues when integrating with the FPGA design.

- **Adapting GACT-X and Evaluating the Implementation.** The original GACT-X module published on GitHub must be adapted with changes in the host for alignment of different pairs of sequences produced by the chaining step of Minimap2's software. The data transfer must be analyzed in order to verify the best sequence format to be adopted. GACT-X's Verilog files may need to be altered to exploit the best performance for the implementation.

- **Integration.** Minimap2's compiler should be merged with GACT-X's compiler, and the accelerator's host needs to be included in Minimap2's code for an integrated design. The integration has to be compatible with the multi-threading capacity of Minimap2 with proper synchronization between multiple kernels and multiple cores in the host. With this, it is possible to measure the total acceleration.

- **Measuring acceleration.** Execution time measurements must be made for the GACT-X module and for the hybrid software-hardware accelerated system as well,

in order to analyze the optimization options and difficulties. The total acceleration and the acceleration by read length are to be measured for analysis.

- **Measuring accuracy.** Comparing the alignment accuracy is important to assure the proposed accelerated solution is acceptable. The alignment accuracy is impacted by the heuristics applied to accelerate the calculation of the SWG scores. GACT-X and Minimap2 use different banding algorithms and both also break the pairs of sequences into smaller sub-sequences with different methods, chaining for Minimap2 and tile for GACT-X.

1.3 Text Organization

This dissertation is divided into 7 main chapters. The introduction to the context of this work has been presented in this chapter. Chapter 2 probes the human genome, genome sequencing, and FPGA concepts and fields. Chapter 3 lightly introduces works that are related to the one proposed, presenting their main strategies and results. Chapter 4 is more theoretical and explains in detail and examples the main algorithms used in this project. Chapter 5 shows the first steps of collecting and generating data and quantitatively studying the State-of-the-Art program Minimap2. Chapter 6 describes the steps of adaptation and integration performed on the hybrid Cloud design GACT-X to be used to accelerate Minimap2, and the final accuracy and speed measurements are presented in the end. Chapter 7 summarizes all the results obtained in each step and proposes some future work options.

2 BACKGROUND

Bioinformatics is inherently a multidisciplinary field, that uses algorithms and digital processing to help solve biological problems. This chapter gives a broad introduction to each relevant topic involved, with concepts from both fields. It starts with an overview of the human genome and its relevance in medicine. Then sequencing technologies are briefly introduced, allowing readers to understand the data that is being dealt with, the differences between long-reads and short-reads, the GATK best practices' steps for the genome processing pipeline, and the current reference-guided read assembly *seed-and-extend* strategy. FPGAs are described in the last subsection; this can help understand how they differ from general processors and ASICs, and how they are able to accelerate software counterparts. At last the AWS Cloud FPGA Instance used in this project is referenced and detailed, followed the description of the OpenCL (Open Computing Language) toolset.

2.1 Overview of the Human Genome

DNA is a polymeric nucleic acid macro molecule, composed of three types of unities: a sugar of five carbons (desoxirribose), one base containing nitrogen, and a phosphate group. The bases can be of type Adenine (A), Guanine (G), Thymine (T) and Cytosine (C). Any of them links to the desoxirribose by the nitrogen atom, and to a phosphate group, forming a corresponding nucleotide (Figure 1). Poly-nucleotide chains form a double helix structure, in which one ribbon is the Watson-Crick (WC) complement of the other (Figure 2). For the purpose of genome sequencing, the actual molecule configuration is not relevant and the nucleotides are abstracted to the characters A, C, T, and G.

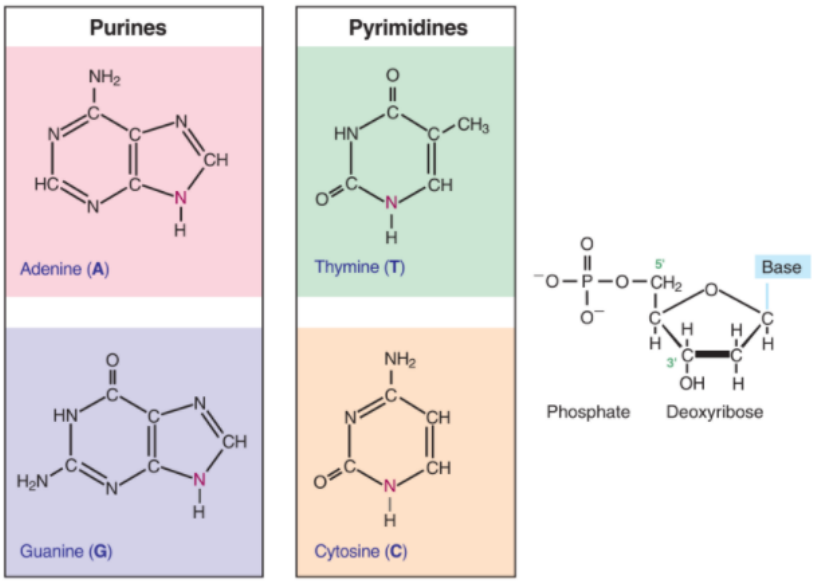

Figure 1: Lewis structures of the four types of DNA nucleotides (KUSHNICK, 1992)

On the left, DNA bases can be Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). Each base links to the deoxyribose, which is connected to a phosphate group (on the right), by the nitrogen in magenta, to form the corresponding nucleotides.

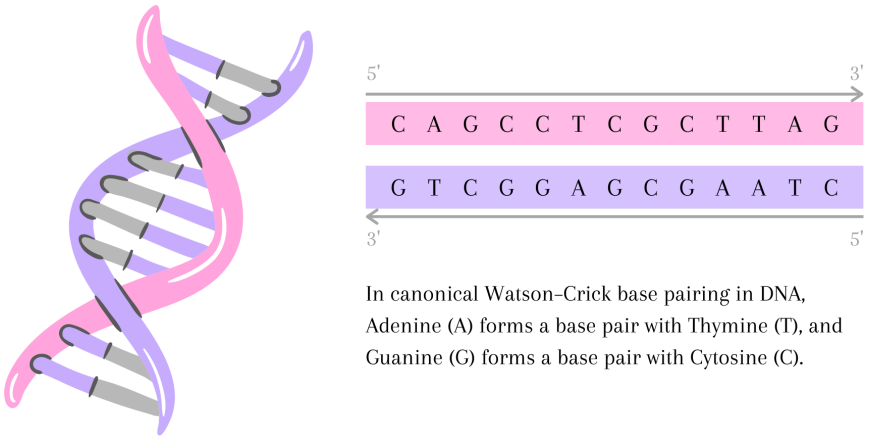

Figure 2: Canonical Watson-Crick base pairing in DNA

The strands are conventionally read in the direction 5' to 3', which is determined by the phosphodiester 5'-3' bonds between desoxirribose adjacents. One strand is the inverted Watson-Crick complement of the other. This complementability enables efficient and correct repair of DNA damage.

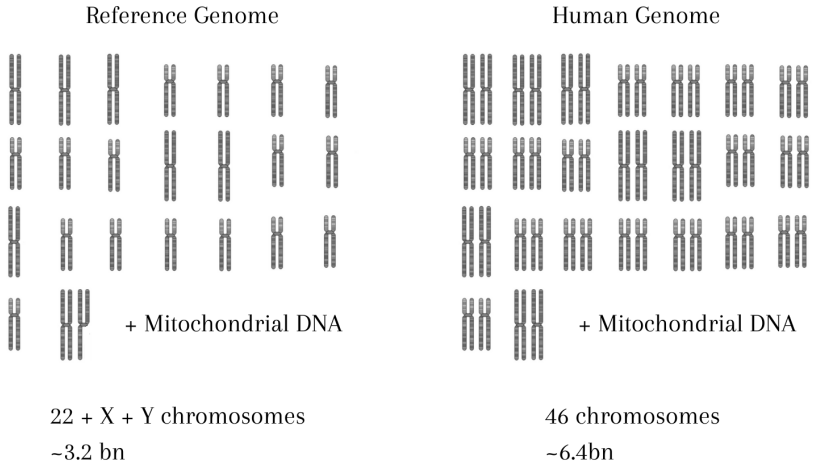

Figure 3: Human genome reference composition vs. somatic cell composition

On the left, the human genome reference has the 22 types of autosome human chromosomes, the 2 types of sex chromosomes, and mitochondrial DNA. On the right, human somatic cells' autosome chromosomes have two copies of each gene and both of them can express and generate a product, except in special cases of allelic imbalance.

Every somatic cell (diploid cell) of the body carries it's copy of the human genome, which has around 6.2 billion nucleotides. Given that humans are diploid, and receive 23 chromosomes from each parent, the established reference to the human genome (more in Subsection 2.1.1) has around 3.2 billion nucleotides containing all 24 types of chromosomes (Figure 3). The first human genome reference was obtained by an international collaborative research effort called The Human Genome Project (HGP) (HUMAN..., 2003), which started in 1990 and completed in 2003. The reference is updated as better sequencing technologies emerge and more research is done. The latest version, Genome Reference Consortium Human Build 38 (GRCh38), was published by NCBI in 2013. The references are presented in one of the DNA strands with direction 5' to 3'.

There are around 30,000 genes (functional units of genetic information) in the human genome. They can be responsible for the production of proteins or functional RNAs (ribonucleic acids). Only less than 1.5% of the human genome codifies proteins and it is believed that around 5% of the genome influences or determines gene expression patterns during development or in different tissues. It is still heavily debated whether the remaining portion of the genome could also provide relevant signals for the genome's functions (PALAZZO; GREGORY, 2014). These sub-sequences can contribute in an interconnected manner to the phenotypes of an organism. Genes appear in the DNA sequence in different densities per chromosome, as shown in Figure 4.

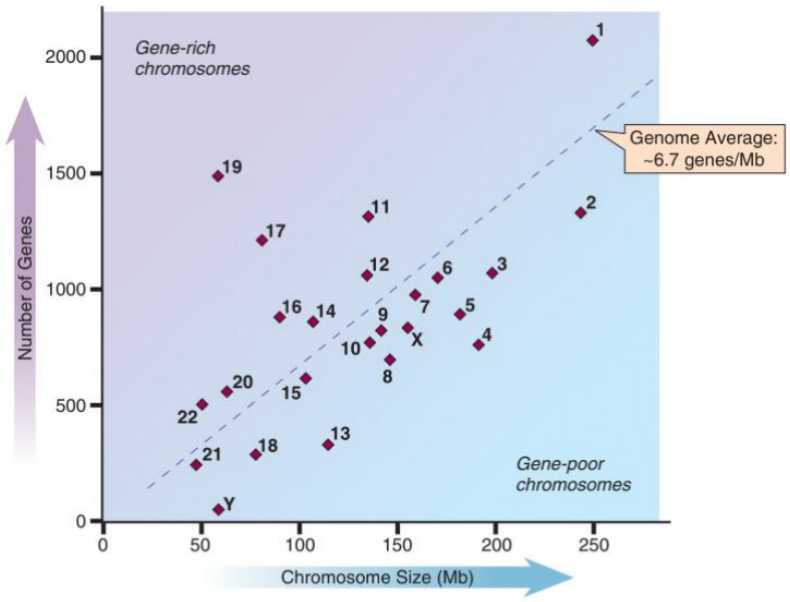

Figure 4: Human gene concentration graph for each chromosome (KUSHNICK, 1992)

The genome has on average 6.7 encoder genes for every mega-base; chromosomes under this line are called gene-poor chromosomes and above this line are called gene-rich chromosomes.

Analysis of the genome sequence shows that about half of it consists of unique DNA copies, which means that they only appear once (or a few times) throughout the whole genome. Most of the unique DNA copies are short sequences (a couple of kilo-bases or less) and are scattered in the genome. The other half of the DNA sequence consists of repetitive DNA that shows up hundreds of thousands of times, identically or with small changes, throughout the genome. They contribute to maintaining the chromosome's structures (e.g. centromere and telomer), are an important source of variation between different individuals, and can be responsible for up to one every 500 genetic diseases.

Given the number of individuals in our species, it is expected that every base-pair in the human genome varies in someone somewhere in the world. There's about 0.5% of genome sequence variation between any two individuals chosen at random. The majority of these differences consists of insertions or deletions of short stretches, number or copies of repetitive elements, or inversion of order of sequences in a certain position.

Genetic diseases can be categorized into three types. Chromosomal disorders, caused by excess or lack of localized genes, affect around 7 out of 1,000 humans born alive and are responsible for around half of all spontaneous abortions that happen in the first three months of pregnancy. Monogenic diseases are caused by mutations in individual genes and follow the autosomal recessive, autosomal dominant, or linked to X patterns. It is a rare

Figure 5: Frequency of genetic disorder types in the population and common examples

disorder, but causes serious disturbances in 1 out of 300 humans born alive and prevails in 1 out of 50 people throughout life. Multi-factorial diseases with complex heritage result from the combination of several gene variations and environmental factors. It is a prevalent genetic disorder and affects 5% of the pediatric population and more than 60% of the population in general (Figure 5).

2.1.1 The Human Genome Reference

Genome references are created as synthetic hybrids (an archetype of the most common variants) aimed to represent a common standard in a species. The purpose is to make it easier to identify which variations are commonly observed in a population and which variations are more unique. Humans are diploid organisms, which means that they carry two copies of each autosome chromosome, one copy originating from each parent. Genome references, however, are haploid, representing each type of chromosome only once. This requires additional steps for identifying heterozygous states (situations where a person carries different variants in the same position on both chromosomes).

The primary assembly of a genome reference contains the assembled chromosomes, the unlocalized sequences (known to belong to a specific chromosome but with unknown order or orientation) and unplaced sequences (with unknown chromosome). The latest human genome GRCh38 came out in 2013 (GRCH38, 2013). In the ".fna" format file, the primary assembly has 193 sub-sequences (24 chromosomes + 127 unplaced + 42 unlocalized), and was separated from the rest of the file for use in this dissertation.

2.2 Genome Sequencing and Genome Analysis

The Human Genome Project (HUMAN..., 2003) prompted the development of cheaper DNA sequencing technologies, with the purpose of obtaining reads: pieces of ordered genetic coding collected from a human sample material. It was a 20 year conjoint research effort of many scientists around the world that cost around 3 billion USD (United States Dollar) in total. At the end of the project, the cost to sequence a human genome was reduced to 100 million USD, using traditional Sanger sequencing.

Sanger sequencing (SANGER; NICKLEN; COULSON, 1977) follows the steps: chain-termination PCR (polymerase chain reaction) is applied to a DNA fragment, generating multiple copies of it that are randomly terminated at different lengths, and with the last nucleotide emitting a fluorescent label corresponding to its base; the copies are subjected to gel electrophoresis: by applying an electric current, since DNA is negatively charged, it moves to one direction with speed determined by its size; the final arrangement of the gel matrix can be translated into the DNA fragment's sequence. The original manual Sanger sequencing method performed four PCR reactions, one for each base type, resulting in fours columns, as shown in Figure 6. Automated Sanger sequencing performs only one PCR amplification for all base types.

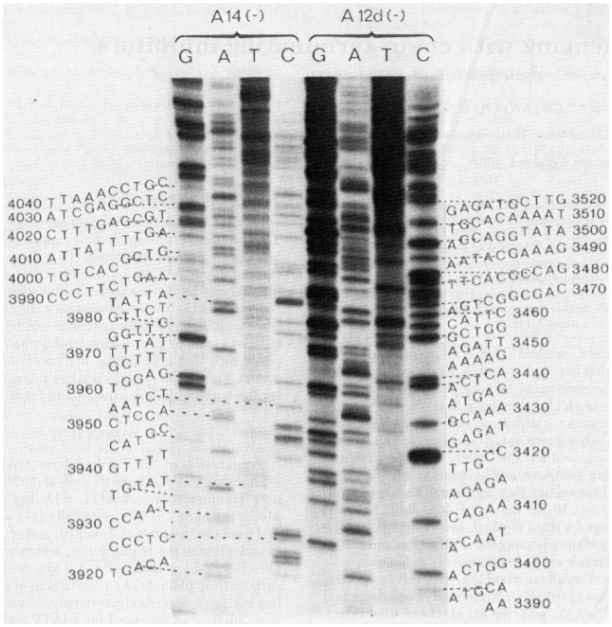

Figure 6: Sanger sequencing gel electrophoresis (SANGER; NICKLEN; COULSON, 1977)

The sequence is written from left to right and upwards, beside each corresponding band.

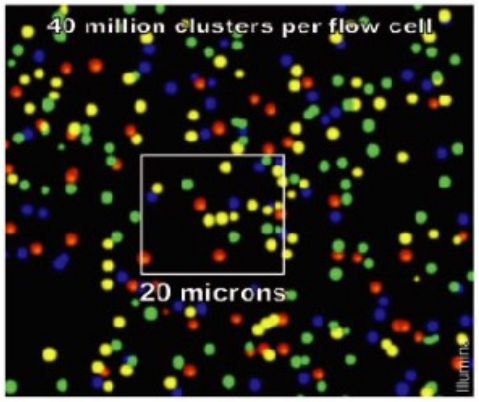

Figure 7: Illumina sequencer DNA clusters captured by a microscope (CHI, 2008)

Each color corresponds to a base type being read from a cluster at a given moment.

To really understand functions of the human genome, thousands of genomes need to be sequenced. This was only made possible with the introduction of the second-generation sequencers. Illumina sequencing (ILLUMINA, 2022) currently takes around 90% of the sequencing market (ADEWALE, 2020). It uses flow-cells: microscope slides with channels inside. The sequencing process follows the steps: the sample's library with the reads is expanded using normal PCR amplification; adapter sequences are added to both ends of the reads, these adapter sequences correspond to primer and capture sequences; the DNA is denatured and captured by complement sequences that are fixed in the flow-cell; DNA polymerase fills the complementary strand on top of the fixed capture sequence and the original sequence is washed off; the captured sequences are copied many times until a cluster is formed with many copies of the same sequence; fluorescent terminators are added to the flow cell in cycles: when incorporated to the sequence, they emit a specific frequency for the base type; a powerful microscope takes pictures of the clusters and sequences the reads one base at a cycle (Figure 7).

In Illumina sequencing, the read lengths are limited because not every strand in a cluster can be captured in each cycle, so as sequencing progresses, more and more strands lag behind, affecting the fluorescent color of the cluster and reducing accuracy. This technology achieves high throughput due to the ability to read millions of clusters of sequences and many different samples simultaneously.

The third-generation of sequencers introduced long-reads, many orders of magnitude longer than in the second-generation (more details in Subsection 2.2.1). One of them is from Oxford Nanopore Technology (ONT) (OXFORD. . . , 2022). They use 18 nm nanopores embedded in lipid membranes to sequence DNA and RNA. The single stranded molecules are forced to thread individually through the pores and each base changes the

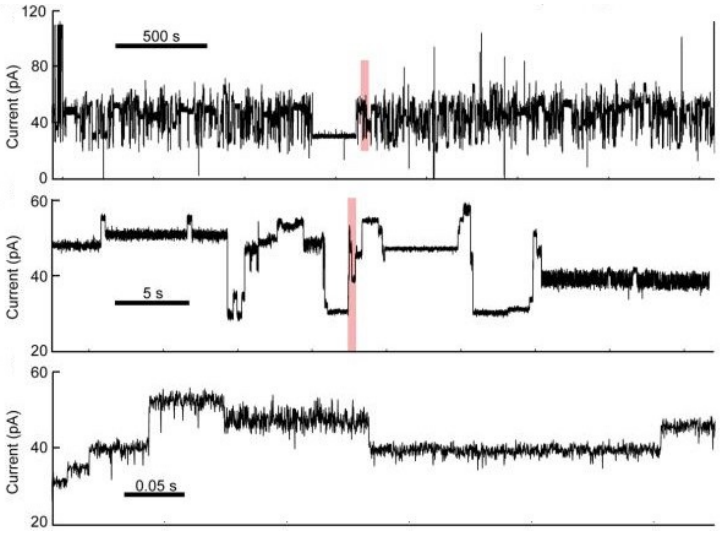

Figure 8: ONT sequencer Ion current (LASZLO et al., 2014)

Each consecutive graph is a zoom-in of the red section in the previous graph.

electric current that is being measured (Figure 8). However, a pore contains around 6 bases at a time, affecting the measurement and increasing the error rate, but this is alleviated by error correction methods ($< 5\%$) (AMARASINGHE et al., 2020). The reads produced can be really long (up to hundreds of Mega-bases). The sequencer is really portable and can be directly plugged into a personal computer.

Pacific Biosciences (PacBio) (PACBIO, 2022) sequencers are also from the third-generation, with long-reads. They use nanometer wells (Figure 9) with DNA polymerase at the bottom. A camera underneath captures videos that register a fluorescence spike when a new base is incorporated. The fluorescent component is able to leave after a time, dropping the intensity of light. The error rates are similar to ONT, but are random, which makes it possible to generate consensus sequences. For that, the same strand is connected end-to-end as a circle and is re-sequenced many times to remove sequencing errors. With this, the achieved accuracy can be higher than the ones achieved with Illumina technology (AMARASINGHE et al., 2020).

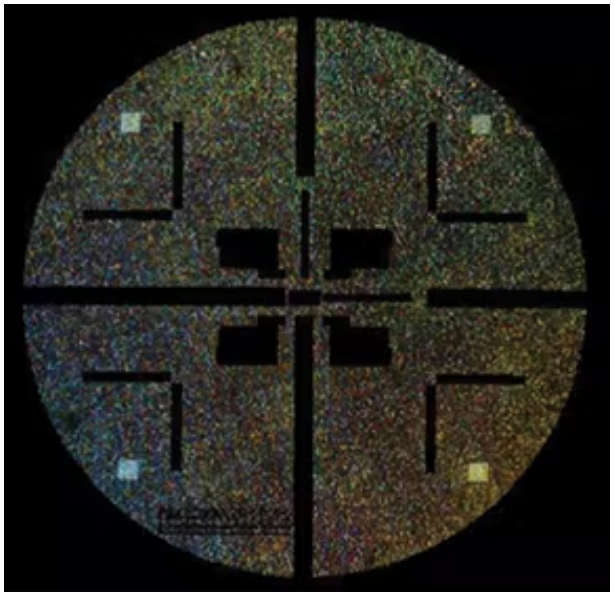

Figure 9: PacBio sequencer nanometer wells (PFEUFER; SCHULZE, 2015)

2.2.1 Short-Reads vs. Long-Reads

The second-generation of sequencing technology leveraged the clinical use of genetic testing by steeply reducing the cost to sequence a human genome or exome (fraction of the genome that codifies for genes). Read sequences from this generation commonly have only a few hundreds of bases, and so, are called short-reads. Even nowadays, short-reads are still known for being some of the most cost-effective, accurate, and popular types of genetic data, even after the introduction of the third-generation of sequencing technology (ADEWALE, 2020).

The third-generation came along with long-reads, that have average length of kilobases, which solved many sequences in the human genome reference that short-reads were not able to cover. Firstly, short-reads are unable to, or are really ineffective at, detecting structural variants (SVs), which are insertions, deletions, duplications, inversions, or translocations in the genome that affect more than 50 nucleotide bases. Each human genome has $>$ 20,000 SVs and additional thousands of indels (insertion–deletion mutations with $<$ 50 base-pairs or bp), and most of them have remained undetected until long-reads were introduced. This poses a serious problem since SVs account for the greatest number of divergent bases across human genomes.

Secondly, short-read sequencing technologies often require the DNA to be fragmented and subjected to PCR amplification, which introduces GC-content coverage bias (high GC content can affect the efficiency of PCR due to the tendency of these templates to

Table 2: Comparison between short-read and long-read data

	Short-Reads	**Long-Reads**
	Second Generation	*Third Generation*
Technologies	Illumina, BGI, Thermo Fisher	PacBio, ONT
Length	up to 600 bases	up to hundreds of kilo-bases
Error rate	$< 0.1\%$	$< 5\%$
Cost (human genome)	USD 942	USD 1500
Assembly algorithms	BWA-MEM, Bowtie2, SNAP, Minimap2	Minimap2, blasr-mc, BWA-MEM

fold into complex secondary structures) and hampers the detection of base modifications, such as methylation (when a methyl group is added to a base, changing the DNA's activity without changing the sequence) (AMARASINGHE et al., 2020). Long-reads have also shown to be essential at detecting copy number variations (CNVs) (regions with multiple copies of short sequences), at phasing alleles (assigning gene variants to paternal or maternal chromosomes), and at differentiating pseudo-genes (sequences that resemble functional genes but can't produce functional proteins) (MANTERE; KERSTEN; HOISCHEN, 2019).

Long-reads were known for having a large sequencing error rate compared to short-reads, but a recent article published in 2020 (AMARASINGHE et al., 2020) showed that error correction strategies (such as the one mentioned for the PacBio technology in Section 2.2) reduced it considerably, closing the gap to short-reads. Short-reads now have an error rate $< 0.1\%$ and long-reads $< 5\%$. One of the factors that are slowing the adoption of long-read sequencers is the higher cost to sample a human genome (USD 942 using short-reads and USD 1500 using long-reads in 2020 (ADEWALE, 2020)) (see comparison between short- and long-reads in Table 2). Some researchers argue that it also takes time for the scientific community to become adapted to the new technologies from the third-generation (ADEWALE, 2020).

2.2.2 The GATK Pipeline

The next step after collecting genomic reads is processing the data to acquire useful information, such as the specific variants in the sequenced individual's genome. An extensive pipeline of tools is used in this step, which is currently considered the bottleneck of the complete genome analysis pipeline (ALSER et al., 2020). The most conventionalized processing pipelines come from the Genome Analysis Toolkit (GATK) (GENOME..., 2022), directed by the GATK Best Practices for different ends, with the workflows designed mainly for Illumina short-reads. This subsection will skip through the processing tools in GATK4 and also comment on changes that would be applied to process long-reads

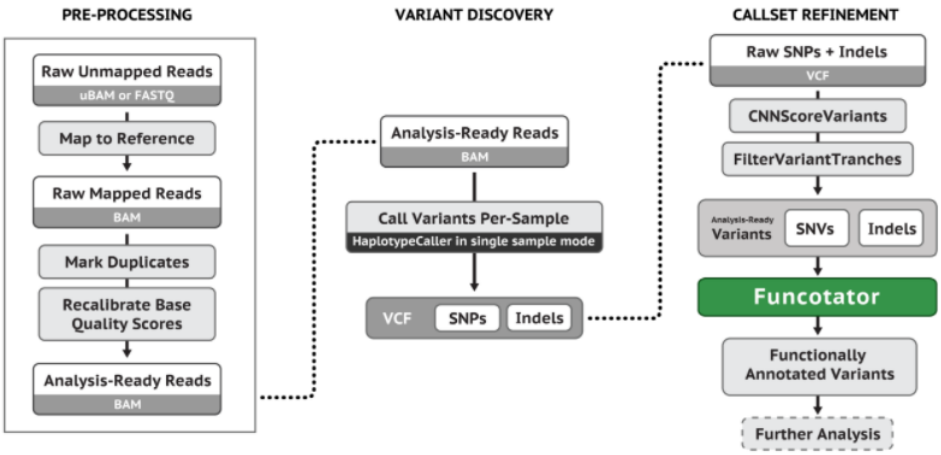

Figure 10: GATK4's pipeline for variant calling (GENOME. . . , 2022)

```
@S1_1
AATCCCTGGCTTCAGCCGGGGTTTGGACACACGTTTGGAGGGAGGGACAGCGAATCAAGAAGGGTTATGAAAGGATATGGACTAAGTGATTGAGAA
+S1_1
.++*../(,.-(--$###%,+'##'*-##$$*#&-,"('##'*,%"&(+%-*&%+#$%-..'&,+-$")(*("+)"-,.+,(#&*&*-.+.//,..
```

Figure 11: Example of a read in ".fastq" format

The read sequence and read quality strings were pruned for clarity, showing only the first line of characters.

from the third-generation. The processing pipeline is divided into three main sections: pre-processing, variant discovery, and post-processing.

The description is based on the generic germline short variant per-sample calling pipeline, referred to as Germline Single-Sample Data in GATK4 (Figure 10). Here, germline refers to alignment to a conventionalized genome reference, and single-sample data refers to the analysis made on an individual genome.

Pre-processing

The pipeline's input is composed of raw unmapped reads produced by the sequencers, that usually come in format ".fastq". Each read in the ".fastq" file has 4 lines: the first and third lines are headers, the second line is the nucleotide sequence, and the fourth line is the quality of each base of that sequence. The quality score is related to the probability of the corresponding base call being incorrect; the lower the probability, the higher the quality score. The scores are encrypted with ASCII (American Standard Code for Information Interchange) codes. For the example, in Figure 11, "@S1_1" and "+S1_1" are headers, and the quality codes for the four initial nucleotides AATC, are .++* respectively, which means that their P_error values are 0.05012, 0.10000, 0.10000, and 0.12589.

The reads are mapped to a reference genome to produce a SAM (Sequence Alignment

Map) file, or its binary version BAM, with mapping information of each read. The purpose of mapping reads to a reference genome is to "rebuild" the original DNA sample, from which the molecules had to be cleaved into much shorter sub-sequences. In the processing pipeline, read assembly is the most computationally intensive step. For Illumina short-reads, GATK recommends the BWA-MEM (BURROWS-WHEELER..., 2010) assembler tool. For long-reads, Minimap2 (LI, 2018) has better accuracy and speed.

The output file after the mapping step is of extension ".bam" or ".sam". The first lines (194 for the primary assembly reference) are headers. After them, for each read, the records display read name, read sequence, read quality, alignment information, custom tags, mapped chromosome, start coordinate, alignment quality, and the Compact Idiosyncratic Gapped Alignment Report (CIGAR) string. The alignment flag (Figure 12), the mapped chromosome, and the start coordinate are the mapping results for the read. Figure 13 shows an example of part of a ".sam" file. The first 5 lines are still part of the header of the file. Then the first read S1_1 is mapped with the forward direction (flag 0) to the position 164,654,461 of the RefSeq NC_000001.11, with mapping quality 60. The CIGAR string starts with 5M1D2M1I (explained later), the read sequence starts with AATCCCTGG, and the sequencing base quality starts with .++*../(,.

The CIGAR strings are alignment structures between the reads and the region to which they were mapped on the reference. The CIGAR string contains pairs — number operator — that represent the alignment path between reference and read. The number indicate the count of sequential occurrences of the operator. The operators can be: "M" for match or mismatch; "I" for insertion to the read; "D" for deletion from the read; and "S" for soft-clipping, which results from semi-global or local alignments that lose the edges of the sequences. Less often used operators are "N", used to differentiate introns

Bit		Description
1	0x1	template having multiple segments in sequencing
2	0x2	each segment properly aligned according to the aligner
4	0x4	segment unmapped
8	0x8	next segment in the template unmapped
16	0x10	SEQ being reverse complemented
32	0x20	SEQ of the next segment in the template being reverse complemented
64	0x40	the first segment in the template
128	0x80	the last segment in the template
256	0x100	secondary alignment
512	0x200	not passing filters, such as platform/vendor quality controls
1024	0x400	PCR or optical duplicate
2048	0x800	supplementary alignment

Figure 12: SAM file alignment flags (SEQUENCE..., 2021)

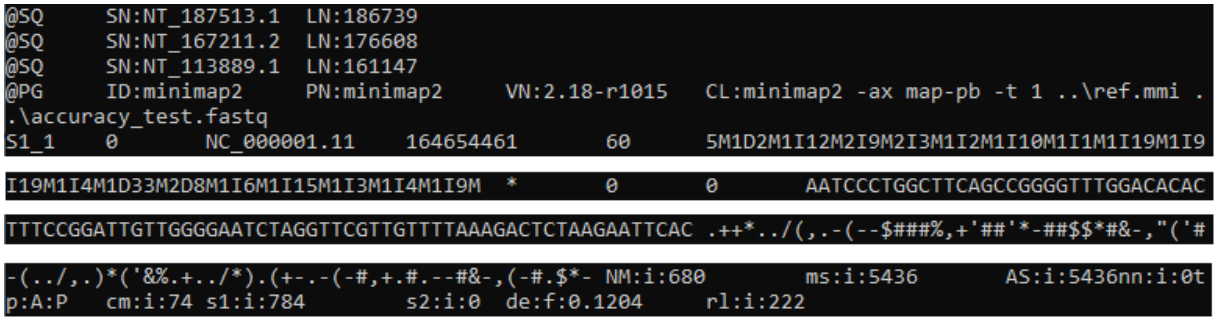

Figure 13: Example of last headers and first alignment report in a SAM file

This is a screenshot of part of a SAM file after the first header lines until, including, the first mapping report. Part of the CIGAR, read sequence, and read quality strings were pruned for clarity.

from deletions; and "H", used for hard-clipping, which works similarly to soft-clipping, but the read string also loses the clipped edges. Section 4.1 presents the genome sequence alignment process and some CIGAR examples.

The last step of pre-processing, which is more specifically for Illumina reads, marks duplicates that may emerge from the replication step during library preparation. This is done by verifying if two or more reads have the same orientation, mapping position and length. After marking duplicates, the base quality scores are re-calibrated using the BQSR tool (BASE..., 2020), which detects the systematic errors created by the sequencer using a machine learning model, to compensate for the tendency of sequencers to overestimate quality scores. PacBio and ONT long-read sequencing technologies do not use PCR amplification, so this step is not required.

Section 2.2.3 will dive more into the pre-processing step of the genome analysis pipeline, as it is the area of focus of this work.

Variant Discovery

This step identifies genomic variations in the sequenced individual using the tool HaplotypeCaller (HAPLOTYPECALLER, 2022). It reassembles the reads in regions presenting variation signs using the *de novo* process, which glues sequences suffix-to-prefix instead of aligning them with a reference; and then calls the variants of each base position. HaplotypeCaller has a very high sensitivity, meaning that it detects many variants that can sometimes be irrelevant. The output file format is VCF (Variant Call Format), which contains variants in each line and columns with chromosome, position, identifier, reference sequence, list of alternative alleles, quality score, a filtration flag, description of the variation, and other information. PacBio suggests using pbsv (PBSV, 2022) to call structural variants joined with DeepVariant (POPLIN et al., 2018) to call

small variants.

Post-Processing

The germline analysis pipeline is focused on identifying variants from the individual that are not common for the species. The undesired variants can be filtered out with the CNNScoreVariants (CNNSCOREVARIANTS, 2020) and the FilterVariantTranches (FILTERVARIANTTRANCHES, 2019) tools. CNNScoreVariants is a trained convolutional neural network model. With relevant variants identified, the funcotator (FUNCtional annOTATOR) (FUNCOTATOR, 2022) links them to their respective functions, using a set of data sources provided by the user.

2.2.3 Reference Guided Read Assembly

There are several ways to re-assemble reads back into the complete genome strand. Two of the most common methods are the *de novo* assembly and the reference-guided assembly. The *de novo* assembly method matches suffix to prefix of different reads, expanding the sequence in a scaffolding way into many long sequences called *contigs*. Genome references themselves, including the human genome reference, and short genomes are assembled with this method. The second method is mapping and aligning the reads to a pre-established reference for the species. This method demands considerably less computation and is more commonly used for assembling long genomes that have a published reference to follow. The reference-guided assembly method is extensively used in clinical sampling of human genomes and exomes.

Reference-guided assembly relies on the similarity between genomes of the same species (about 99.5% for humans). It can generate reference biases; for example, variants that are predominant in an ethnicity might not be represented in the reference. The post-processing step can be tailored to these situations and filter out variations that are actually expected for an individual's ancestry. The reference-guided method also relies on statistical inference. Clinical applications require at least 8 times coverage for each base for an acceptable accuracy (AMARASINGHE et al., 2020). This should cover random sequencing and alignment errors and help identify heterozygous states, where a different variant is inherited from each of the biological parents.

Figure 14 illustrates a reference-guided assembly. The reference sequence is on the top line in red, while the read samples are in gray in following lines. The coverage is represented by each line and each column is linked to the nucleotide of a specific position

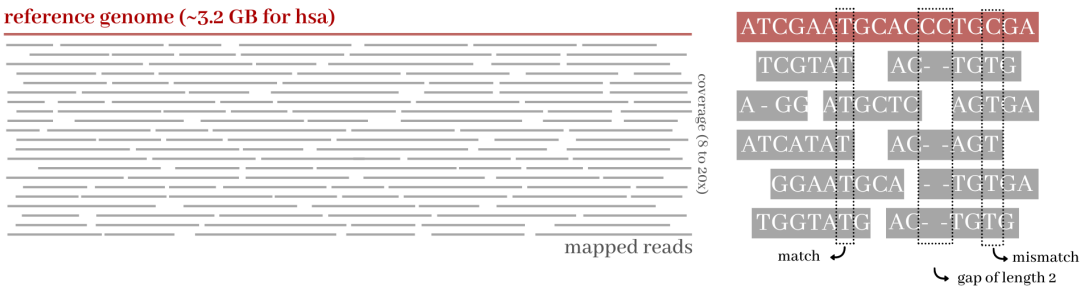

Figure 14: Illustrated reference-guided assembly and statistical inference for base-call

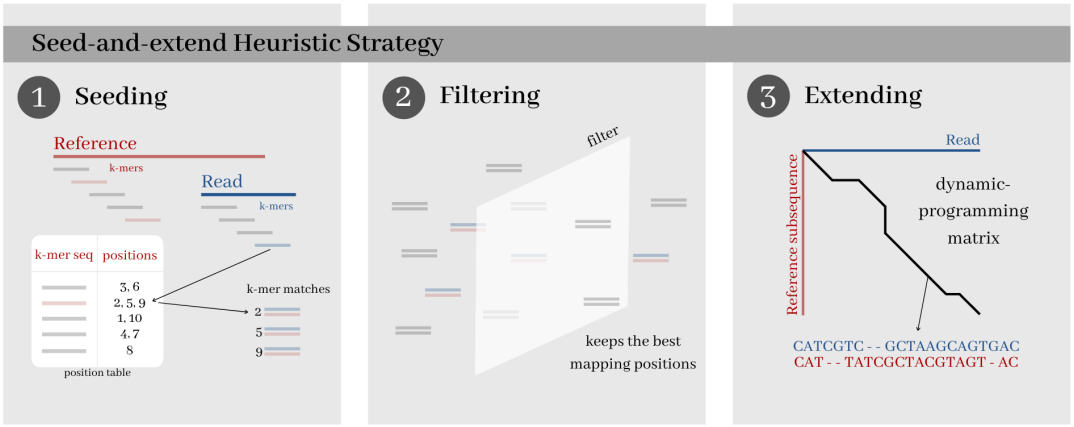

Figure 15: The seed-and-extend read assembly strategy

In the seeding stage, the algorithm finds exact or non-exact matches of very short sequences between the read and the reference. Indexing strategies are applied to the reference to accelerate the seeding stage. Because the seeding stage can result in a great number of false positives, often an intermediate filtering stage is required to reduce the number of selected items. On the extending stage, the Smith-Waterman-Gotoh algorithm (described in Section 4.1.3) is used to align the entire read sequence to the reference region.

in the reference. Variants are determined by dissimilarities between the sequenced sample and the reference genome. In the zoomed figure on the right, under the nucleotides "CC" in the reference, there are two columns missing, with dashes in the reads. That would be identified as a deletion in the sequenced genome. There is also a column where the reference genome has the nucleotide "C" whereas the reads have the nucleotide "T", which would be identified as a homozygous SNV (single nucleotide variant). The variant would be classified as homozygous because all the reads in the coverage have this SNV; if it was heterozygous, around half of them would have the SNV.

There are many methods to map and align reads to a reference. The most successful method has been the *seed-and-extend* approach (Figure 15). It was pioneered by BLAST (ALTSCHUL et al., 1990) and it is used by almost all read assembly tools currently.

Some examples of seeding strategies are: searching for identical matches of fixed length scanning through fixed intervals, used in Bowtie2 (LANGMEAD; SALZBERG, 2012) and Minimap2 (LI, 2018) (more in Section 4.2.2); finding maximal exact matches (MEMs), the longest exact matches between read and reference, used in BWA-MEM (BURROWS-WHEELER..., 2010); non-identical seeds can also be used by mapping algorithms.

Indexing the reference is essential for finding the seeds in short time. Each seeding method can be better suited for different indexation methods. The FM-Index (FERRAGINA; MANZINI, 2000) is a memory efficient indexation method used in BWA-MEM and Bowtie2 that sorts all the rotations of the sequence, and indexes each type of nucleotide in the first and last columns. The hash table indexing method is used in Minimap2; it converts seed sequences into an index for a table containing all positions for that seed in the reference (more in Section 4.2.1).

The seeding process can end up pointing to many regions to align in the reference. Some techniques can be applied to filter out the false positive seeds. Filtering for short-reads is harder because they carry little extra information that would not result already in a complete extension of the read. For example, the work described in Section 3.2 is at the same time a filtering step, that calculates the alignment between read and mapped region in the reference using a very narrow band; and an extending step, because most of the results of these alignments correspond to the optimal alignment. For long-reads, filtering can be applied by extending a small region near the seed and evaluating the score obtained before deciding to extend the entire read (TURAKHIA; BEJERANO; DALLY, 2018), or by grouping the parallel seeds from the same read to find its best mapping position in the reference as in Minimap2 (LI, 2018) (more in Section 4.2.3).

Finally, with the one or few remaining mapping positions, the algorithm needs to align the read to the mapped region in the reference to identify potential variants. For this alignment, a matrix is adopted, with the reference and query (read) sequences positioned in the matrix' axes. The alignments that are performed can be global (when both sequences are fully aligned to each other), which is preferred when they are similar and have similar lengths; local (only a portion of the sequences is aligned), used to find regions of similarity between two sequences; and semi-global (alignment is extended from a fixed point between the two sequences and stops when the similarity ends), commonly used in the *seed-and-extend* method.

The Smith-Waterman-Gotoh (SWG) algorithm has quadratic time and memory complexities in relation to the input's lengths. Because of that, the assembly tools usually do

not calculate the entire SWG matrices for this step. Since the alignment path is expected to stay around the anti-diagonal of the matrices, different banding techniques can be used to limit computation to cells around the anti-diagonals, reducing the complexities to a linear relation; they can adopt fixed bands or dynamic bands. Fixed bands limit the calculation to cells in a fixed area of the matrix (FUJIKI et al., 2020). Dynamic bands try to follow the deviation of the alignment path (LI, 2018). Other algorithms compute overlapping quadrants of the matrix, named tiles, one at a time, limiting the memory usage to a constant value, independent of the input's lengths (TURAKHIA et al., 2019).

Chapter 4.2 describes the specific algorithms used in the seeding, filtering and extending steps of Minimap2. Chapter 4.3 describes a hardware implementation of the alignment process that is taken in the extending stage. These two are combined in this work to provide an acceleration for Minimap2.

2.3 FPGA Acceleration

Currently, almost every 32-bit and 64-bit processor uses reduced instruction set computer (RISC) instructions to communicate software and hardware. It is a type of instruction set architecture (ISA), where instructions are typically as simple as microinstructions, being executed directly by the hardware. In RISC, recently executed instructions are stored in a fast memory called cache, since they are more likely to be reused. This architecture adds extensive flexibility to general purpose processors, but also increases considerably the number of clock cycles required to perform the same task as an ASIC.

The computers' most basic components are the transistors. Their purpose is to reliably and accurately control electric currents, used to switch or maintain voltage representing zeroes or ones in binary logic. Specific combinations of transistors in a circuit can create logic gates for conjunction, disjunction, and negation. Combined logic gates can perform all sorts of more complex calculations. Transistors are fabricated with a semiconductor like silicon, treated with elements that give it an electron emitting or electron absorbing characteristic. Arranging these types of semiconductors in layers gives transistors the ability to control the current propagating between its terminals based on the voltage applied to its gate.

In 1965, Gordon Moore first predicted that the transistor density on processors would double every year, and in 1975, he revised it to be every two years. His prediction was so precise that it became known as Moore's Law. From around 2000, Moore's Law began to slow down, and by 2018 there was a 15-fold gap between the predictions and the technology at that time. This gap is only going to increase as CMOS (complementary metal-oxide-semiconductor) technology approaches fundamental limits, and the costs to deal with this skyrocket (Figure 16).

In parallel to that, Robert Dennard stated that as transistor density increased, power consumption per transistor would decrease, allowing power consumption per mm^2 to stay nearly constant. With power density increasing, "Dennard scaling" also began to slow in 2007 and became almost nonexistent by 2012 (Figure 17).

The slowdown of Moore's Law and the end of Dennard scaling induced architects to exploit better data parallelism. They developed a branch prediction strategy to keep modern processor cores' pipelines full. However, the average misprediction rate of 19% (on Intel Core i7 in this example) made the energy efficiency of modern processors even worse, as additional energy is needed to restore the initial state.

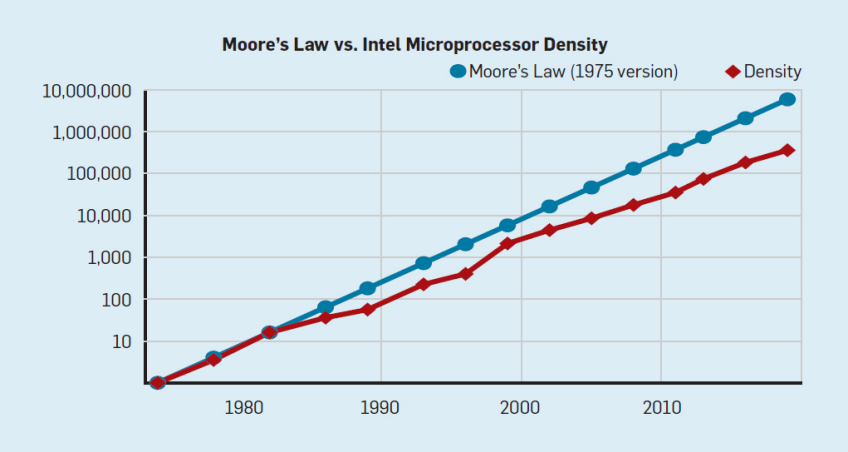

Figure 16: Moore's Law vs. Density (HENNESSY; PATTERSON, 2019)

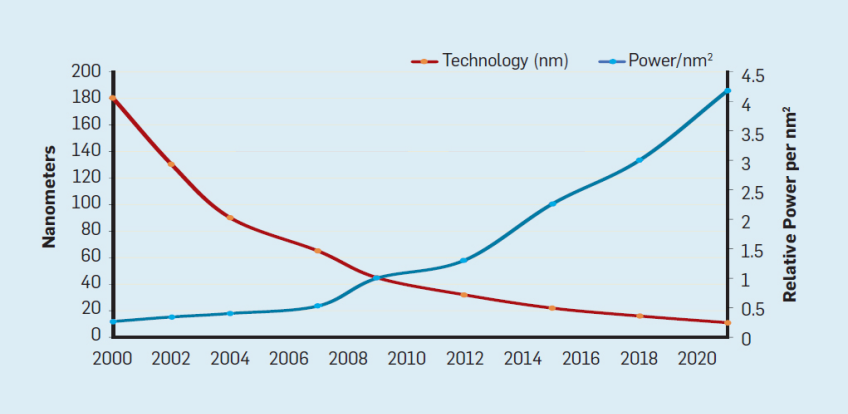

Figure 17: Transistors' length vs. power/nm^2 (HENNESSY; PATTERSON, 2019)

The multi-core (more than a single CPU) approach allowed the programmer to set the best use of thread parallelism for each implementation, but did not solve the energy consumption problem, since every active core consumes power whether or not it is contributing to boost the performance. Also, according to Amdahl's Law, the speedup from parallel computing is limited by the portion of the program that is sequential.

Quantum effects on extremely small transistors lead to high thermal dissipation and CMOS technology is approaching its fundamental limits (i.e. reaching lengths equivalent to a few atoms), but there's still a growing need for computational power. Genomics is an example of it: the Illumina NovaSeq 6000 system can sequence about 48 human whole genomes at 30x genome coverage in about two days. However, analyzing (performing assembling and variant calling to) the sequencing data of a single human genome requires over 32 CPU hours on an Intel Xeon processor, 23 of which are spent on read assembly (GOYAL et al., 2017).

Domain-specific architectures (DSAs) are one of the best solutions to this. They are architectures tailored to a specific problem domain, which in this case would be genome assembly. Some examples of DSAs are GPUs and neural network processors. DSAs can exploit a more efficient type of parallelism for the domain, can make more effective strategies for memory access, can use less precision when adequate, and are energy efficient. FPGAs are programmable hardware used on DSA development that constantly appear on genome alignment acceleration research and were argued to be the most efficient option for the genomics domain (GUO et al., 2019).

ASICs are able to deliver a higher performance when compared to CPUs or DSAs (GPUs and FPGAs), because their operating frequency can be adjusted and optimized to the longest path of the circuit, becoming over 3 times higher than the operating frequency of FPGAs (KAPLAN; YAVITS; GINOSAR, 2019); and because the components can be distributed on the chip in the most efficient manner, under same technology. However, the fabrication cost for ASICs is high, requiring a high volume production to become viable. Since the genomics field is evolving extremely fast, and new algorithms are published every year, it is not prudent to adopt a fixed solution.

2.3.1 Field Programmable Gate Array

Before explaining FPGAs, some basic electronic components need to be introduced. Lookup Tables (LUTs) are tables that produce an output based on the input values. They act like logic gate circuits and, in great numbers, can produce any combinational logic function and even perform very complex computations. LUTs can be looked at as being very small RAMs that are loaded with data when configuring the FPGA. The input would represent a storage address and the output would be the stored data. Lookup tables reduce significantly computation time as they can produce the same results of a circuit that has many logic gates. They are also components that attribute a re-configurable aspect for FPGAs.

Flip-flops (FFs) are logic circuits that work as 1 bit memories. Clock pulses can keep or alter their output value, depending on the value set on the input. A series of flip-flops can constitute a register that also has similar characteristics to memories. A latch is a circuit that registers data if the input has ever been at a high value, keeping output high after this occurs, and only going back to zero if reset is activated. Multiplexers (MUXes) are able to switch the output to one of its multiple inputs through control signals.

FPGAs are, in a simplified way, ICs that contain configurable blocks of logic and

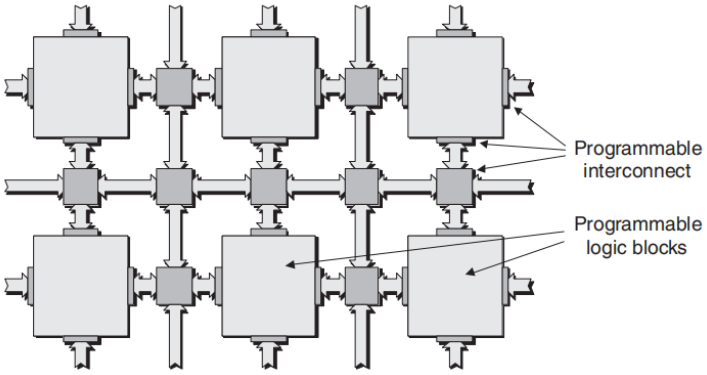

Figure 18: Simplified illustration of an FPGA architecture (MAXFIELD, 2004)

Switch Matrices are the keys arranged in rows and columns that interconnect CLBs and connect them to the inputs and outputs of the FPGA.

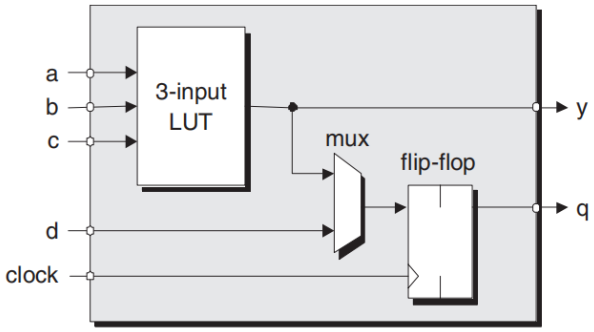

Figure 19: Key elements of a configurable logic block in an FPGA (MAXFIELD, 2004)

configurable interconnects between them (Figure 18). In the illustration in Figure 19, each configurable block is formed by three key elements: a 3-input lookup table, a register that could act as a flip-flop or a latch, and a multiplexer. A 3-input function can be loaded in the LUT, the multiplexer could accept the output from the LUT or another input from the logic block, and the register could be configured to act as a FF or a latch.

Another important aspect of FPGAs is how the logic is loaded into the board. Most FPGAs use, besides LUTs for configurable logic, SRAM (static RAM) configuration cells to hold control signals for interconnections, which is a fast re-configurable method, with a downside of requiring configuration in every system power-up. Another issue with this method is the hindrance to protect the intellectual property, since the configuration file is commonly stored in an external non-volatile memory. Bit-stream encryption can be applied with additional circuitry to avoid this, or Cloud implementations can block access from the end user to private components.

Several modern FPGAs available in the market include in their fabric embedded Block

RAMs (BRAMs) and high-speed Input/Output Blocks (IOB) that make the interface between internal blocks of the FPGA and its external pins. Some FPGAs come with embedded processor cores. Considering FPGAs adopt volatile SRAM for configuration, booting with an external micro-controller is required to load the memory value into the FPGA every time it is initialized. It is also possible to reconfigure the FPGA while it is running a project with a complex system.

The FPGA project developer generates the circuit description at behavior level, at RTL (register transfer level) with a hardware description language (HDL) or with a block diagram graphic description. Most common HDLs are Verilog and VHDL (VHSIC (Very High Speed Integrated Circuit) HDL). Some tools like VIVADO, from Xillinx, can translate high level programming languages such as C and C++ into HDL, process known as high-level synthesis.

On one hand, it is considerably easier and faster to design FPGA projects than ASIC projects, since no complex backend design steps (e.g. layout generation and layout optimization) is needed, and the reprogrammability aspect allows the design to promptly adapt to the market changes. On the other hand, FPGA projects are limited by on-chip resources and mostly don't reach the same level of ASIC performance. Manufacturing costs per chip are lower for ASICs in large production runs, but are prohibitively high for small reproductions. FPGAs also have high manufacturing costs, but they are shared by the high number of users or purchasers.

Designing an FPGA accelerator takes a longer time, and is usually more expensive than programming a GPU accelerator, although it may deliver, in general, faster processing. GPUs are considered still efficient, when the related application present strong data parallelism. According to the authors of (GUO et al., 2019), FPGAs may achieve better performance than GPUs in several applications, as in the one considered in this dissertation.

2.3.2 Cloud FPGAs (the AWS F1 Instance)

FPGA designs can be implemented in a physical board or, nowadays, even in the Cloud. Genomic tools, specially read assembly tools, deal with great amounts of data, requiring processors to have higher volatile and non-volatile memory capacities. FPGA boards with high resource count can be very costly and inaccessible for development projects. Even if the board is accessible, developers must deal with the hardware-software interface configurations and with system maintenance for new platform support tools.

Cloud computing services can reduce the initial cost by charging a smaller price for usage time. Designs that are implemented in popular Cloud servers, such as Amazon Web Services (AMAZON..., 2022c) and Google Cloud (GOOGLE..., 2022); and that use more unique processors, such as FPGAs and GPUs; have been increasingly used in the research environment, because they can be more easily replicated and more people can avail the results. Considering the reasons above, this project was developed for, and was implemented on, a Cloud machine containing many CPUs and an FPGA.

Amazon Web Services (AWS) is a Cloud platform that offers technological services through the web. AWS provides on-demand delivery of IT resources over the internet with pay-as-you-go pricing. On-demand Instances are more suitable for short-term, irregular workloads that cannot be interrupted, which is typical for development and research projects. A wide variety of Instance types are available, with different number of CPUs, RAM and storage capacities, and network and data transfer bandwidths (AMAZON..., 2022b).

Some Instance types offer accelerated computing with hardware integration. The F1 Instances are the only AWS Instances that contain FPGAs. There are f1.2xlarge, f1.4xlarge and f1.16xlarge Instances, in increasing performance potential, with their respective resources listed in Table 3. For the purposes of this dissertation, the f1.2xlarge will suffice.

The f1.2xlarge Instance has a server with eight CPU Cores and one Xilinx UltraScale+ VU9P FPGA in a separate board. Since this Instance is the one used in this dissertation, from this point on, it will be referred simply as AWS F1, unless otherwise explicitly

Table 3: Resources available on AWS F1 Instances

Instance	FPGAs	vCPU	Mem (GiB)	SSD (GB)	Network Performance (Gbps)
f1.2xlarge	1	8	122	470	10
f1.4xlarge	2	16	244	940	10
f1.16xlarge	8	64	976	4 x 940	25

stated. The server is equipped with 4 DDR4 (Double Data Rate) channels, providing a bandwidth of 4x16 GiB/s, each accessing a 72-bit wide ECC-protected memory (48 Gb/s bandwidth). The interface with the host is done with dedicated PCIe Gen3x16 connection (30 Gb/s bandwidth).

The advantages of AWS F1 Instances with respect to FPGAs are that they already come integrated to a CPU host server with fast interface and drivers installed; AWS FPGAs are pre-loaded with a Shell that wraps the kernels (HDL synthesized hardware in FPGA that run algorithms and are called via OpenCL functions) to fit the FPGA's I/O range, presenting a standard AXI (Advanced eXtensible Interface) to the kernels. The VU9P FPGAs are very powerful, and have more SRAM capacity than FPGAs currently available at the author's research institution.

The project's development environment is SDAccel (SDACCEL..., 2019) (default changed to Vitis in 2020 (VITIS..., 2022)). On the host CPU, the custom application (written in C/C++) interacts with the FPGA by using the OpenCL API (Application Programming Interface). OpenCL Runtime manages and services the requests sent to the FPGA. The drivers handle the PCIe transfers between the host and devices. The FPGA comes preloaded with the necessary logic to DMA (Direct Memory Access) the data in local DDR memory. The custom kernels read this data, process it, and write the results back to DDR, using standard AXI-4 (Figure 20).

The steps for hardware development in AWS are:

- Create SDAccel/Vitis kernels from C/C++, OpenCL, or RTL models; the resulting containers (".xo" files) contain kernel XML (eXtensible Markup Language) metadata, RTL files, and Vivado IP project;

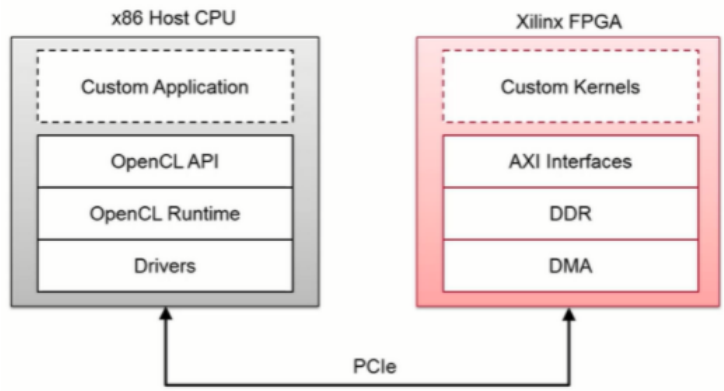

Figure 20: CPU-FPGA interconnection in AWS F1 Instances (DEVELOPING..., 2022)

- Compile the platform (the SDAccel/Vitis compiler links the kernels and other required hardware components), instantiate the kernels and the F1 Shell, generate DDR interfaces and interconnect logic, make all the necessary connections, run synthesis and place&route, resulting in a ".xclbin" binary file;
- Create the encrypted Amazon FPGA Image (AFI), which is stored by an AWS back-end service.

VU9P is a stacked silicon device, meaning that a silicon interposer connects 2 or more FPGA dies (Super Logic Regions). SLR is a slice of a device containing a subset of its resources. VU9P has 3 SLRs, and each has access to one or more DDR interfaces. The Shell includes the PCIe link and hardware necessary to transfer data between the host and kernels in the CL (Custom Logic) region (Figure 21). The Shell's clock runs at a fixed 250 MHz. The CL supports clock frequency of up to 500 MHz.

The kernels access DDRs via AXI4 MM (Memory Mapped) that have 512 bit busses clocked at 250 MHz. Each DDR is accessible by a maximum of 16 AXI4 busses, so an F1 FPGA can have at most 64 kernels. The DDR global memory has an inherent latency overhead when accessed, both for host via PCIe, and for kernel via DDR memory controllers. One cycle of data transfer takes in total 4 DDR transfers.

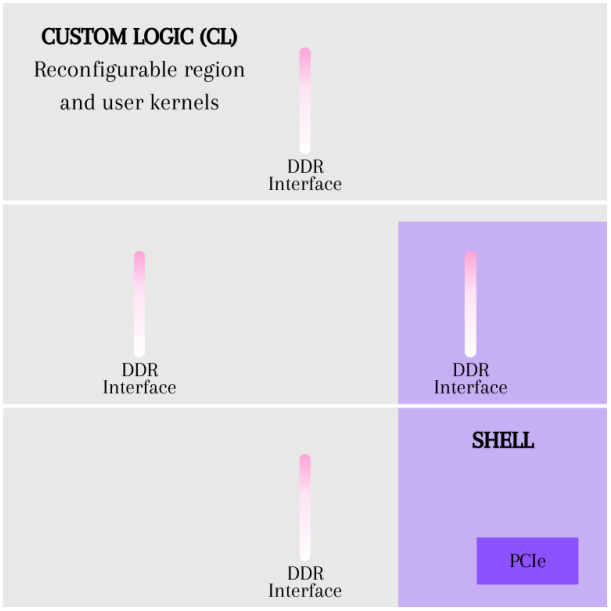

Figure 21: The AWS F1 Instance's VU9P FPGA with 3 SLRs, 4 DDRs, and AWS Shell

2.3.3 The OpenCL Tool-Set

OpenCL is a tool-set that augments C and C++ languages with data types, data structures, and functions for building parallel programs to run on high-performance processors (OPENCL..., 2011). It is a standardized interface that allows developers to program different devices without having to learn multiple languages.

In 2008, the company Apple was selling very popular consumer electronic products (e.g. iPhone, iPad, iPod, and MAC). They were built using devices from third party companies, that benefited from this with the rise in market share and developer interest. Apple faced the need for a common interface that suited any device, so together with its vendors, Apple formed the OpenCL Working Group, one of many in the Khronos Group (a consortium of companies for graphics advancement).

OpenCL targets platforms as the one presented in Figure 20, which describes a hardware implementation in FPGA, but the device instances could be instead, for example, a set of GPUs running concurrent code. A custom application in C/C++ runs in the host, along with the acceleration platform's execution commands and its device's functional code. In the case of GPUs, this code corresponds to C/C++ blocks, while for FPGAs it consists of HDL compiled binaries. OpenCL is a tool-set with a large number of specific commands; the implementation's host in this research practices the following steps with their corresponding commands/functions:

- **Create a platform structure:** with the software development kit (SDK) previously installed on the machine, the vendor is informed (e.g. Xillinx, Nvidia, etc.) in this step; the cl_platform_id data type and the clGetPlatformIDs and clGetPlatformInfo functions are used;
- **Create a device structure:** linked devices from the specified vendor are identified, and one or more are chosen for the application; the cl_device_id data type and the clGetDeviceIDs and clGetDeviceInfo functions are used;
- **Create a context:** devices that work together are grouped into a context, that will have a command queue designated to it; the cl_context data type and the clCreateContext function are used;
- **Create a command queue:** various types of commands can be en-queued to the device with this data structure, from transferring buffer objects to executing kernels; the queue also synchronizes commands by using event flags, and by deter-

mining whether the commands are ordered by FIFO (first in, first out) or not; the cl_command_queue data type and the clCreateCommandQueue function are used;

- **Create a program:** programs are containers of kernels; both of them store executable code, but kernel represents a single function whereas program can represent more than one function; at this step, the binary file with instructions to build logic in the FPGA is read and loaded in the memory; the cl_program data type and the clCreateProgramWithBinary, clBuildProgram, and clGetProgramBuildInfo functions are used;

- **Create kernels:** kernels can be associated to a specific memory bank, such as one of the DDRs from the AWS FPGA (see Section 2.3.2); the cl_kernel and cl_mem_ext_ptr_t data types and the clCreateKernel function are used;

- **Create buffers:** one or more buffers (regions of a memory used to temporarily store data) are reserved in the device for it to read or write input and output data; this step requires maximum data size and memory bank information; the cl_mem data type and the clCreateBuffer and clEnqueueMapBuffer functions are used;

- **Transfer data:** data is written into or read from the buffers by the host; this is a command order sent to the command queue; the clEnqueueWriteBuffer and clEnqueueMapBuffer functions are used;

- **Set the kernel arguments:** any I/O configured kernel argument is set at this stage; the clSetKernelArg function is used;

- **Execute the kernels:** this is a kernel execution command order sent to the command queue; the clEnqueueTask function is used.

3 RELATED WORK

Thousands of relevant works related to genomic read assembling can be found in the literature. This literature review is focused on papers that use hardware designs (e.g. GPU, FPGA, KNL, and ASIC) to achieve some improvement with respect to their software counterparts. The first two works accelerate short-read assembly algorithms and the other works are related to long-read assembly algorithms.

3.1 SWG on FPGA for Short-Reads (KOLIOGEORGI et al., 2019)

This work consists of accelerating the short-read assembly program Bowtie2's alignment step with FPGA implementation. In Bowtie2, the SWG step takes 60% of the execution time, being 56% for matrix-fill and 4% for traceback. A communication overhead on the hardware-software co-designed architectures was identified and addressed.

The wave-front parallelism is explored by an array of PEs (Processing Elements), with the same length as the read sequence (average 270 bp for their dataset), that fills the matrices in a skewed pattern in $n + m - 1$ steps (Figure 22). The matrices are stored on on-chip memory for use by the traceback step, which is also implemented in hardware.

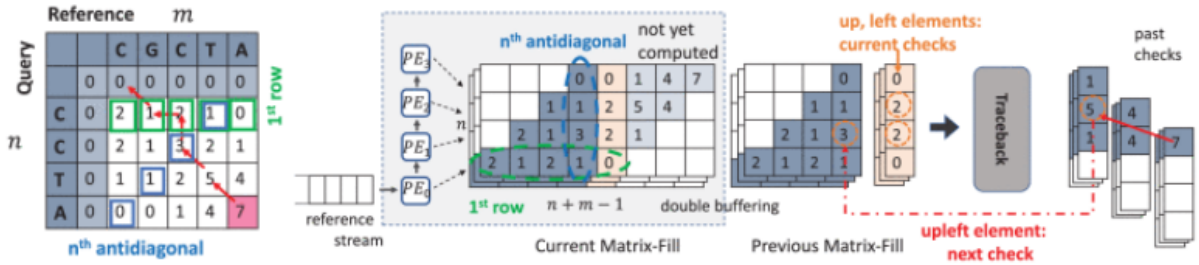

Figure 22: Related work - data-flow for SWG (KOLIOGEORGI et al., 2019)

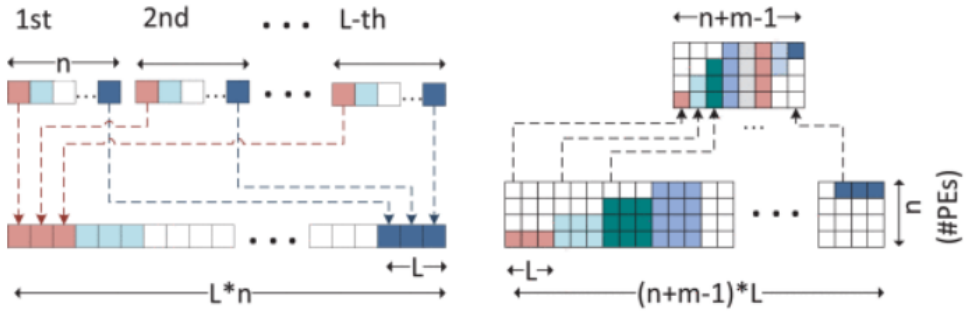

Figure 23: Related work - interleaving read sequences (KOLIOGEORGI et al., 2019)

Interleaving of L read sequences causes a skewed memory pattern for storing score matrices in BRAM.

In their architecture, they noticed that the computation of a cell value requires a chain of multiplexers, which introduces a latency L between consecutive diagonals. To alleviate this, L reads are interleaved in a round robin manner (Figure 23). Double buffering was also implemented to avoid halting matrix-fill operation while streaming data to traceback.

The design was implemented on Xilinx VU9P Ultrascale FPGAs running at 200 MHz and MAX5C DFE (data-flow engine). Compared to Bowtie2 with SIMD optimization running in an Intel Xeon E5-2658A processor at 2.2 GHz, the accelerator achieved 18x speed-up and the integrated system had 35% performance gain. This design is only suited for aligning short-reads, since the number of PEs is directly linked to the length of the query.

3.2 SeedEx: BSW on FPGA for Short-Reads (FUJIKI et al., 2020)

SeedEx is a co-processor implemented on an FPGA that performs seeding expansion with narrow banded SW (BSW) as a filtering algorithm. First, they noticed that in the BWA-MEM algorithm, more than 98% of the seed expansions required a band $w \leq 10$ for short-reads. So they developed a narrow banded accelerator with a three step optimality check to find and rerun the non-optimal expansions on the host.

The first step calculates the theoretical highest score (upper-bound score) and compares it with the score obtained within the band. Thresholds $S1$ and $S2$ are calculated for the smaller region and the bigger region outside the band respectively (difference is due to length asymmetry between query and reference sub-sequence) (Figure 24). The highest score would be the seed's score subtracted by a gap with the band's length and added to a match in the remaining sequence. If the score is under $S1$, it is automatically

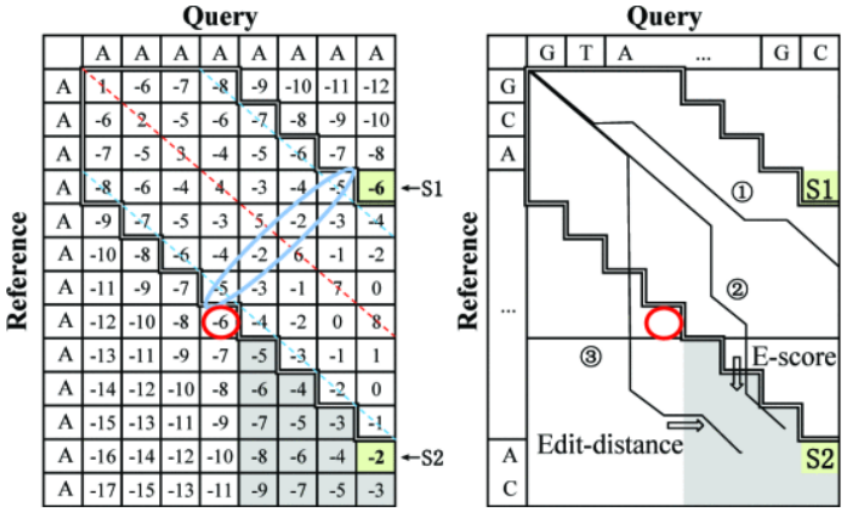

Figure 24: Related work - SeedEx' three step optimality check (FUJIKI et al., 2020)

Thresholds $S1$ and $S2$ from 2 regions outside the band where optimal alignment could occur. On the left is the matrix of thresholds. On the right are the three possible optimal alignment paths with scores higher than $S1$.

sent to rerun on the host. If it is higher than $S2$, optimality is guaranteed. If it is between $S1$ and $S2$, further checking is required.

It is proven by contradiction that if the optimal score is outside the band, then it must be in the shaded region in Figure 24. There are two ways of reaching this region, from above or from the side. The second step performs an E-score check by dislocating the border cells out of the band and performing matches for all subsequent alignments. The maximum value resulting from these interactions is the threshold of acceptance of the band's alignment.

Finally, in the third step they perform the Edit-distance check where they perform a seed expansion with the lower gap penalty and edit-distance scoring on the shaded region using $S1$. The best resulting score is an optimistic threshold for alignments coming into the gray area. If the alignment score in the band passes both E-score and Edit-distance thresholds, then it is optimal.

The design was implemented on AWS F1 instance (f1.2xlarge). They chose a band size of 41, which resulted in a thresholding passing rate of 71.76% and a rerun rate of 1.81%.

12 BSW cores generated the narrow-band and E-check scores. 4 Edit machines performed the Edit-distance check's expansion. With perfect prefetching and appropriate buddering, the memory access time was completely hidden. They achieved 43.9 M seed extensions/s and 1.5 M reads/s (the latest was coupled with seeding acceleration), a 1.3x

speed-up over BWA-MEM and BWA-MEM2.

SeedEx could be repurposed as an accelerator of a filtering step for long-read assembly (it would be equivalent to the BSW modules in Darwin-WGA, see 3.8), but can't be used for long-read alignment for two reasons: the traceback memory requirement would be prohibitive, and long-reads would hardly be limited to narrow bands.

3.3 RASSA: ASIC for Finding Mapping Positions (KAPLAN; YAVITS; GINOSAR, 2019)

RASSA consists of an accelerator for the filtering or pre-processing step of genome assembly. A circuit of resistive memories called *memristors* stores the reference genome and, at the same time, compares it in parallel with chunks (100-200 bp) of the read sequence to measure the Hamming distance between them (Figure 25). A threshold of around 50% determines the mapping position(s) of the read (Figure 26).

RASSA achieved 16-77x speedup over Minimap2 with higher sensitivity. While Minimap2 only mapped about 20% of all reads from their dataset, RASSA always mapped more than 72% of the reads with false positives between 6.9% and 39.2%. However, RASSA's tested circuit only supported small genomes (< 31.5 Mbps), and there was no evaluation for human genome size performance.

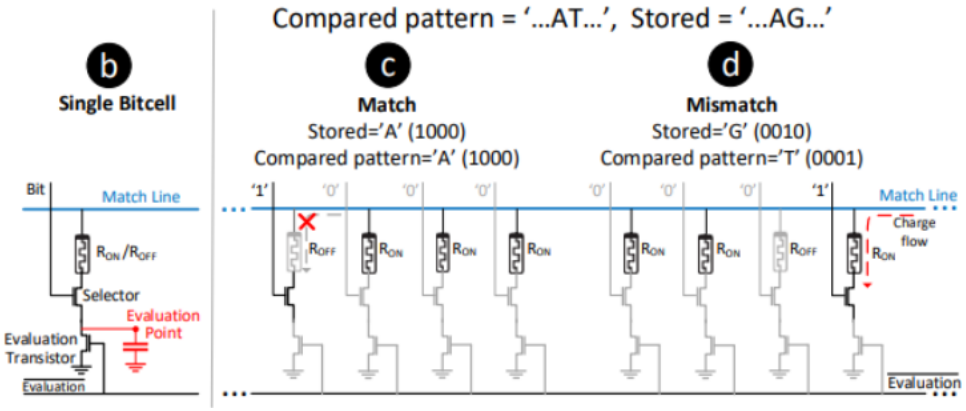

Figure 25: Related work - circuit of memristors (KAPLAN; YAVITS; GINOSAR, 2019)

b) Single RASSA bitcell. **c)** 'A' base from reference stored in-memory compared to 'A' base from read resulting in no charge loss. **d)** mismatch between 'G' and 'T', resulting in match line voltage reduction.

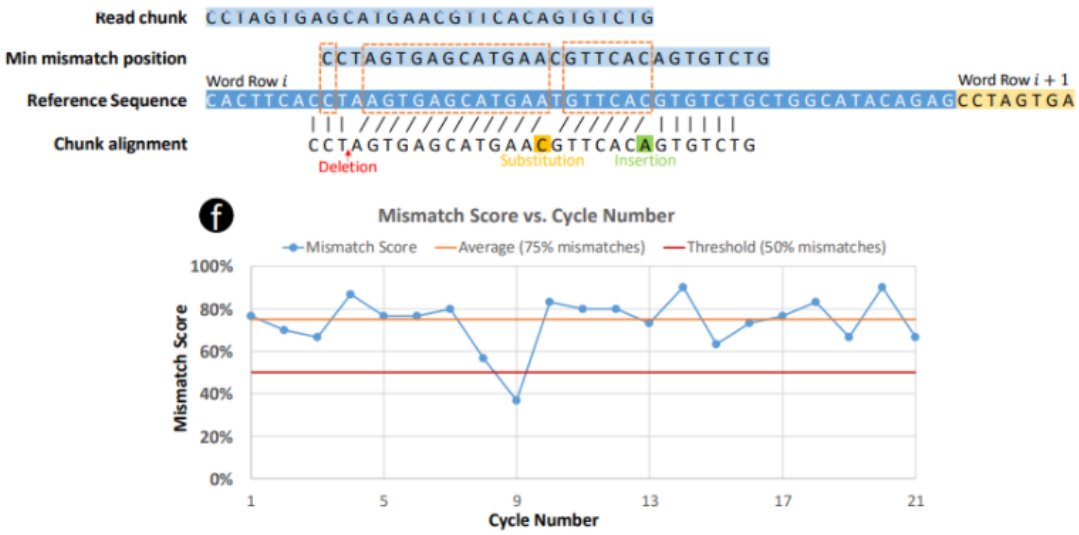

Figure 26: Related work - mapping threshold (KAPLAN; YAVITS; GINOSAR, 2019)

Read is scanned through the reference and when it hits a potential mapping position, the mismatch rate drops under a 50% threshold.

3.4 Minimap2's Chaining Step on FPGA and GPU (GUO et al., 2019)

This project aimed to accelerate the chaining step of Minimap2 in 3 processors: CPU, GPU and FPGA, and compared the results. For the CPU, they binded a thread to a designed core, allocated data of tasks in a neighbor NUMA (non-uniform memory access) node, and implemented SIMD so that the weights between eight pairs of anchors can be computed concurrently. With this, they achieved almost linear speedup to the number of cores (13.9x with 14 cores).

For hardware implementation, they changed the order of the operation sequence because of a loop-carried dependency. Instead of comparing the current anchor with N previous anchors to find a maximum, they compare it with N later anchors and update the temporary value of each of them (Figure 27). They also dispatch a data batch so that, at any clock cycle, every PE can fetch input data. This solves the problem of different input sizes from different tasks (Figure 28).

The architecture was implemented on an AWS F1 Instance, running at 250 MHz with 8 PEs. PCIe bandwidth was the limiting factor. For long-reads, the FPGA accelerator is 277x faster than a single-thread software, 28x faster than 14-core optimized software and 4x faster than their GPU implementation. Since the chaining step is one of the bottlenecks for Minimap2's runtime on long-reads, with this acceleration the bottleneck

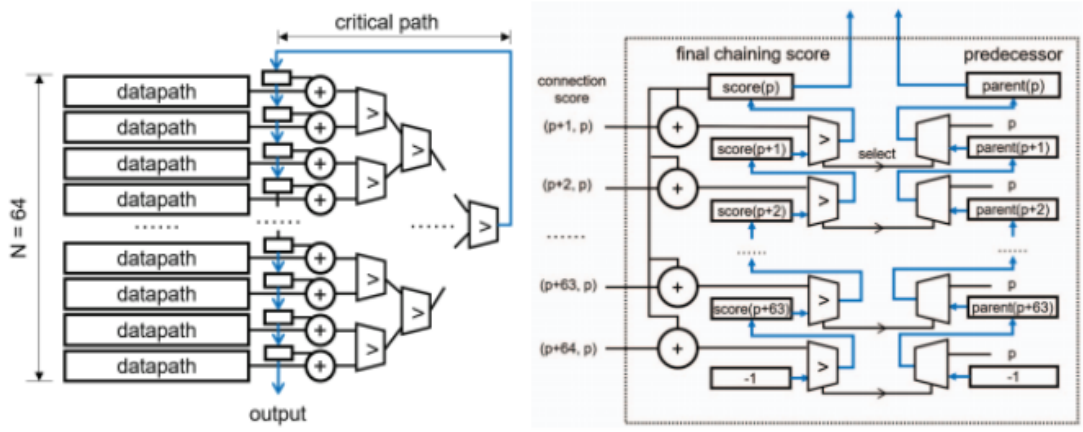

Figure 27: Related work - reordered operation sequence (GUO et al., 2019)

On the left, the original algorithm with a long critical path. On the right, the reordered operation sequence reduces the critical path.

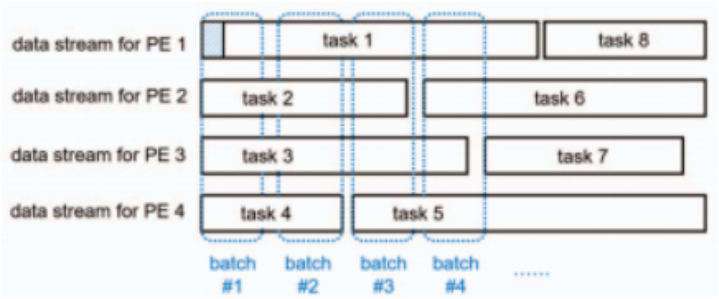

Figure 28: Related work - fine-grained task dispatching scheme (GUO et al., 2019)

is passed to the extending step. This project has not been fully integrated into Minimap2, but the separate module is available on GitHub (MINIMAP2-ACCELERATION, 2021).

3.5 Minimap2's Extending Step on CPU, GPU and KNL (FENG et al., 2019)

This article implemented acceleration techniques on Minimap2's base-level alignment step for long-reads on CPU, GPU and KNL. For PacBio simulated dataset on the human genome, this step consumes 65.42% of the time. They noticed that the Suzuki-Kasahara transformation uses linear arrays to store the matrices, reducing memory usage, but introducing intra-loop data dependency. They proposed a new memory layout with another coordinate transformation that does not alter the memory requirement (Figure 29). The vector load procedure was reduced to a single load instruction (Figure 30).

On their setup, the CPU was running with 20 cores, GPU with 5120 and KNL with

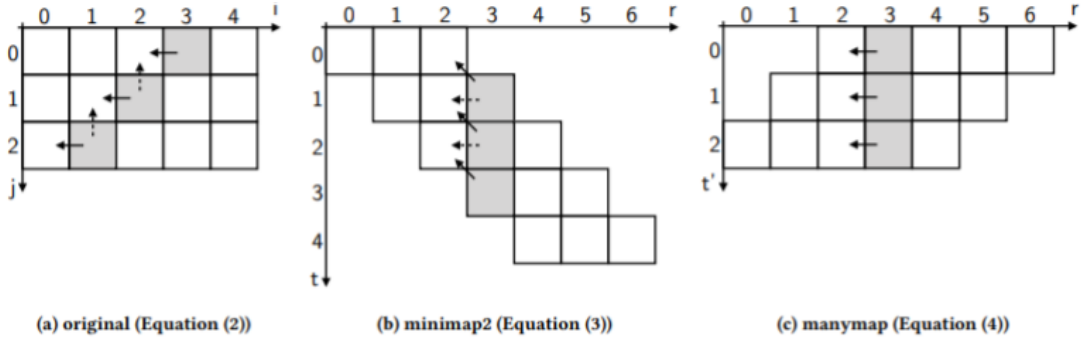

Figure 29: Related work - optimized memory layout for SWG (FENG et al., 2019)

```
xt  = _mm_load_si128(&X[t]);  // xt  = x[r-1][t..t+15]
x1  = _mm_srli_si128(xt, 15);
xt  = _mm_slli_si128(xt, 1);
xt  = _mm_or_si128(xt, tmp);  // xt  = x[r-1][t-1..t+14]
tmp = x1;                     // tmp = x[r-1][t+15]
```

(a) minimap2

```
xt = _mm_loadu_si128(&X[t - r + qlen]);
```

(b) manymap

Figure 30: Related work - vector load reduced to a single instruction (FENG et al., 2019)

64. For matrix fill and traceback, CPU was 1.3 to 4.5 times faster than Minimap2, GPU was 3.2 times faster and KNL was 3.9 times faster. They concluded that, although KNL and GPU both outperformed CPU, a high-end server CPU was still the most efficient platform due to the others' low single thread performance and occupancy issue.

3.6 Darwin: A Hybrid Design for Long-Read Assembly (TURAKHIA; BEJERANO; DALLY, 2018)

Darwin presented the deployment of a complete genome assembly algorithm that is optimized for hardware acceleration. It is divided in two parts: Diagonal-band Seed Overlapping based Filtration Technique (D-SOFT) for the seeding and filtering steps, and Genome Alignment using Constant memory Traceback (GACT) for a second filtering stage and the sequence alignment step. With D-SOFT implemented on CPU and GACT implemented on an Arria 10 FPGA, clocked at 150 MHz, Darwin achieves up to 183.8x speedup over GraphMap (ŠOŠIC et al., 2016) with a single thread on a dual socket Intel Xeon E5-2658 processor (2.2 GHz).

The reference is first indexed with a seed position table, where seed hits are stored

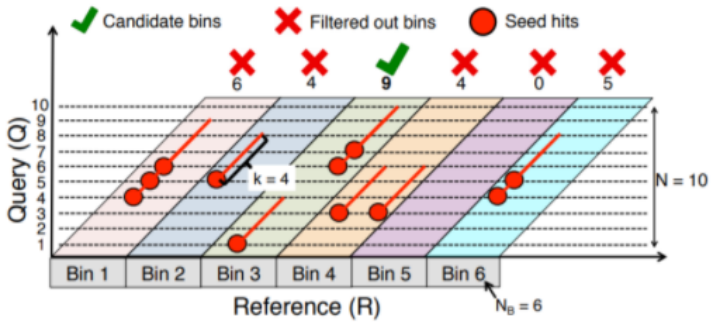

Figure 31: Related work - D-SOFT mapping (TURAKHIA; BEJERANO; DALLY, 2018)

In the example, $k = 4$, $BinSize = 10$, and $h = 8$.

sequentially, which enables faster and fewer memory accesses. High frequency seeds that occur more than $32 \times |R|/4^k$ times, where $|R|$ is the size of the reference, are discarded. Then, in D-SOFT, the reference is divided into bins of size 128. Seeds of size $k = 15$ are collected from a read with $stride = 1$ between start and end. When matched to the position table, the hits are computed so that each bin has a count of unique bases that account for seed hits that may overlap. When the count for a bin first exceeds a threshold $h = 13$, the last hit in the bin is added to the candidate positions list.

Figure 31 shows an example with a reference with 6 bins of 10 nucleotides each. Each of the query's 4-mers, with stride (or dislocation) of 1 is searched in the position table. The first k-mer finds a match in Bin 3, illustrated by a red circle and a line spanning the k-mer's length. When a match is found, the Bin updates its count, without considering overlapping bases that have already been counted in previous k-mers. After iterating through all the query's 4-mers, the last hit from the Bins with counts that exceeded the threshold of 8 are sent as seeds for the next stage. In the example, only Bin 3 passed the filtration process.

GACT performs a second filtration stage and the extending step for the seeds that passed. It computes sub-squares of the SWG matrix, called tiles, with fixed size $T = 320$ and overlap border $O = 128$, creating sequentially a band around the anti-diagonal of the matrix. The first tile starts from the expansion point; it is the only one that starts the traceback from the highest score and it is bigger than the other tiles ($T = 384$) because it works as a second filtering stage. The expansion will only proceed if the first tile's score surpasses $h_{tile} = 90$. The traceback in each tile proceeds until the overlap boundary is reached. The other tiles are clipped in the position where the previous tile stopped backtracking and perform their traceback from the bottom-right cell. Figure 32 presents an left-extension example from the seeding point in green, with $T = 4$ and $O = 1$.

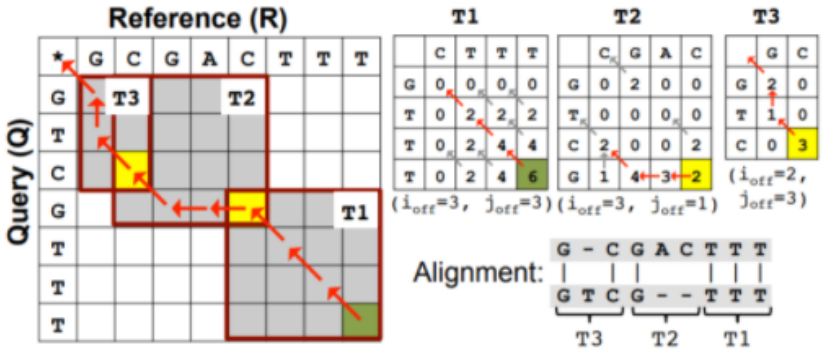

Figure 32: Related work - GACT extension (TURAKHIA; BEJERANO; DALLY, 2018)

In the example, $T = 4$ and $O = 1$.

The memory requirement is fixed at $O(T^2)$, ideal for hardware implementation with limited resources, and the authors affirm that GACT gives optimal results empirically. The hardware design is based on a systolic array. An array of $N_{PE} = 32$ processing elements (PEs) computes blocks of N_{PE} rows from the tile, exploiting the wave-front parallelism of the matrix. In each block, each PE holds its corresponding nucleotide from the query sequence and the reference sequence is streamed through the array. In each clock cycle, each PE computes the three scores and the traceback pointer. Figure 33 shows an example with $N_{PE} = 4$ and $T = 9$.

They synthesized 4 GACT arrays with traceback support (limited by on-chip memory — each PE required 2 KB SRAM, which equals 64 KB SRAM per array), and 36 GACT arrays that do not perform traceback and only compute the first tile's score. The peak throughput was 1.3 M tiles/s. The reported 183.8x speed-up over GraphMap was in relation to single-threaded software, but the hardware design takes multiple kernels. A

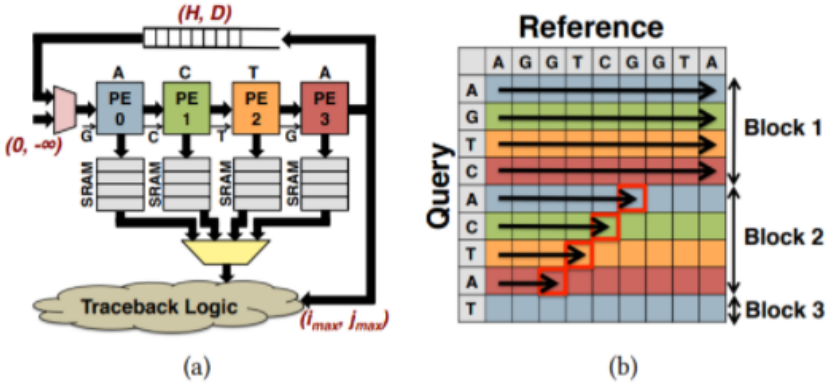

Figure 33: Related work - GACT architecture (TURAKHIA; BEJERANO; DALLY, 2018)

(a) Systolic architecture of GACT with $N_{PE} = 4$. **(b)** Matrix fill scheme for $T = 9$.

one to one comparison should have had a reduced speed-up.

Darwin was published about the same time as Minimap2, and the latter achieved significantly better accuracy and also better performance in comparison to GraphMap. There has not been any performance evaluation of Darwin in relation to Minimap2. D-SOFT was developed to achieve best performance in hardware implementation and, although the authors have simulated a fully accelerated Darwin ASIC, which would provide a great speed-up, in the current implemented FPGA design, D-SOFT runs on software and consumes prohibitive amount of RAM for long references, such as the human genome.

3.7 Improved GACT Algorithm Using BSW (LIAO et al., 2018)

This article presents an improved version of Darwin's GACT accelerator. The authors replaced the tiles with bands without affecting accuracy empirically and achieved a 2.5x speed-up in hardware (Figure 34). They also developed a dynamic programming algorithm for band overlapping. The algorithm consists of searching for regions with high match rates that are more likely to be in the optimal alignment, and overlapping on these instances. Traceback was performed on the hardware.

They used a systolic array of PEs, with length equal to the band, $L = 512$ and $B = N_{PE} = 128$. Score cells were treated as 12-bit integers. Compared to GACT, they reduced the total required clock cycles to 40%. They implemented the accelerator on an Intel 40nm Stratix IV FPGA, EP4SGX230KF40C2, running at 125 MHz, with RIFFA data communication through PCIe interface. No open-source code has been provided by the authors for eventual third part improvements. Later, the authors of Darwin also published an improved architecture that bands GACT and achieves higher speed-up, which will be presented in the next Section 3.8.

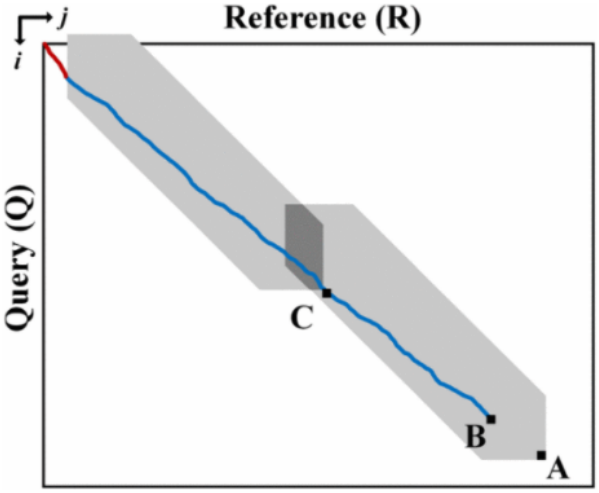

Figure 34: Related work - GACT adapted with bands (LIAO et al., 2018)

3.8 Darwin-WGA: A Hybrid Design for Whole Genome Alignment (TURAKHIA et al., 2019)

The authors of Darwin developed a whole genome alignment (WGA) hybrid design by adapting the D-SOFT and the GACT algorithms used in Darwin. WGA is a method that aligns two complete genomes. It is used for identification and prediction of functional elements (genes and regulatory sequences), for deducing the evolutionary relationship between species, and for ancestral genome reconstruction. Darwin-WGA uses an adapted D-SOFT algorithm for seeding, BSW for gaped filtering, and the improved GACT-X design for aligning. The main differences in Darwin-WGA compared to Darwin are: the query is a long genome sequence, instead of many shorter reads; aligned sequences have a significantly lower similarity for originating from different species; longer indels are expected in the result.

The adaptations aimed at increasing sensitivity, so that true positives could pass on through the filtering stage. D-SOFT uses spaced seed patterns instead of perfect matches to find seeds. A BSW filtering stage, implemented on an FPGA, is added to the design, which also increases the algorithm's sensitivity by allowing gaps in the filtration. Finally, longer gaps require wider tiles to show up in the final alignments, but GACT has a quadratic consumption of on-chip SRAM in relation to tile size. To linearize this ratio, the X-drop algorithm was added to create a dynamic band around the alignment path in the tiles, naming the new algorithm GACT-X. GACT-X is further detailed in Chapter 4.3.

In their architecture, the software runs D-SOFT, configures the arrays, controls the execution of tiles, and reconstructs the alignments from traceback pointers. 50 BSW arrays and 2 GACT-X arrays are implemented on an AWS FPGA, operating at 150 MHz. GACT-X achieved 4.6 k tiles/s (14.6 MB/s) throughput, which is over 4x speed-up over GACT, using 2x less memory. Although Darwin-WGA is not a read assembler, the GACT-X module in it computes the SWG alignment for arbitrarily long sequences, which can be repurposed to accelerate read assembling algorithms.

3.9 Minimap2's Extending Step on FPGA (TENG et al., 2021)

This is a work related to the author's under-graduation project, accomplished with her colleagues Renan Weege Achjian and Caio da Costa Braga, which consisted on accelerating the extending step of Minimap2, using a Xilinx MPSoC Zynq Ultra96v1 board, containing an FPGA and an Arm processor. First, the extending step was identified as Minimap2's main bottleneck when mapping short-reads (58.6% of the total execution time).

One of ksw_extd2_sse41's main loops, commented as "gap left-alignment" in Minimap2's original code, was synthesized into VHDL, using the VIVADO High Level Synthesis (HLS) tool. The SSE instructions were changed to regular C and pragma commands were added to optimize the logic description. From the RTL files, an IP block was generated and the block design was completed in VIVADO to generate the bit-stream.

The host was loaded into the Arm processor, using PYNQ OS (operating system) and Jupyter interface. The intermediate data was stored on an SD (Secure Digital) card inserted in the board. This design only consumed 3% of FFs and 11% of LUTs. The required clock cycles were reduced 155x compared to software. However, the acceleration scheme did not provide performance increase to the ksw function, due to a data transfer overhead that took 99.9% of the time taken by the whole design.

4 ASSEMBLY AND ALIGNMENT ALGORITHMS

This chapter describes the key algorithms that form the basis of the acceleration project of this work: SWG, Minimap2, and GACT-X. The first and last algorithms perform sequence alignment; and the second performs read-to-reference assembly, which also includes a sequence alignment stage. SWG is a classic biological sequence alignment algorithm used in almost any read assembly program, usually with added heuristics to improve performance and reduce memory consumption. The SWG's predecessor is the algorithm Needleman-Wunsch. Minimap2 is the State-of-the-Art algorithm for assembling long-reads to a genome reference. GACT-X has the same function as Minimap2's extending step, and is an FPGA design suited for alignment of long sequences. GACT-X is implemented on an AWS Instance and is going to be used in this project to accelerate Minimap2's extending step.

4.1 Biological Sequence Alignment

All the following methods are based on matrices. The examples given are simple for didactic reasons, but real short and long-reads can measure up to 600 bp and hundreds of kbp respectively.

4.1.1 The Needleman-Wunsch Algorithm

Saul Needleman and Christian Wunsch first published a method to find the best alignments between two sequences using a matrix based dynamic programming algorithm in 1970 (NEEDLEMAN; WUNSCH, 1970). Their method allowed comparisons involving matches, when the letters are identical in the same position of the alignment; mismatches, when the letters are different in the same position; and gaps, when either one of the sequences have an insertion or a deletion of a nucleotide or sub-sequence. In other words, it is a metric that represents the minimum number of "mutational events" required to

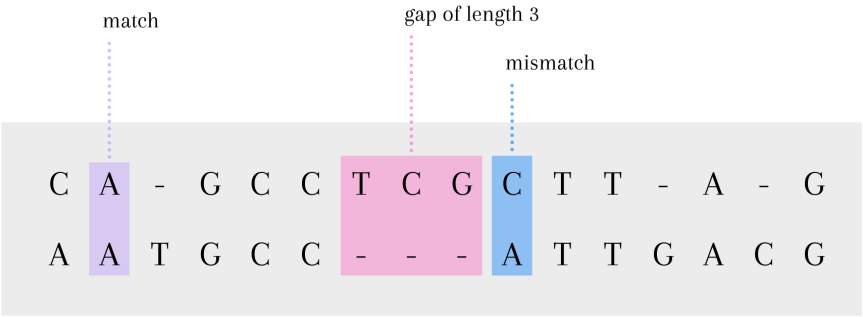

Figure 35: Example of an alignment between two DNA sequences

Alignment between sequences “CAGCCTCGCTTAG” and “AATGCCATTGACG”. Each position of the alignment has to correspond to a match, a mismatch, or a gap. Gaps can also be called indels, insertions or deletions, depending on the situation.

convert one sequence into another. Take the example of an alignment in Figure 35.

Given a matrix S, the axes of the Needleman-Wunsch matrix correspond, respectively, to the query and the reference being aligned, $A = a_1a_2...a_n$ and $B = b_1b_2...b_m$. Matches and mismatches are compensated with function $s(a_i, b_j)$ which has a positive value when it is a match and a negative value when it is a mismatch. Gaps are penalized with a constant w. In dynamic-programming fashion, each cell contains the best score up to that position and the best alignment interaction(s) between every pair of bases that resulted in this score. The equation 4.1 shows how each cell of matrix S is computed.

$$S_{ij} = max\{S_{i-1,j-1} + s(a_i, b_j), S_{i-k,j} - w, S_{i,j-1} - w\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \quad (4.1)$$

As an example, take the alignment between $A = ATGACTCTCAGAC$ (reference) and $B = ATCTCGAGTGAGC$ (query) in Figure 36. A match score is 1, a mismatch penalty is 0.4 and a gap penalty is 0.4. First column and row are filled as gaps, since it’s the only possible interaction between an empty sequence and a non-empty one. The filled Needleman-Wunsch (NW) matrix is shown in the figure. The arrows indicate the origin(s) of the maximum score for each cell and the backtracking generates the best alignment path(s) as shown in purple.

The backtrack pointers indicate that there is a match/mismatch when the path is diagonal, an insertion when the path is vertical, and a deletion when the path is horizontal (when the reference is in the x axis). The computation in $S_{1,1}$ results in 1 due to the match of “A”s, which scores +1 from the upper diagonal cell, resulting in a value larger than the ones coming from above or from the left. After all cells are calculated, a traceback phase,

S		A	T	C	T	C	G	A	G	T	G	A	G	C
	0	-0.4	-0.8	-1.2	-1.6	-2	-2.4	-2.8	-3.2	-3.6	-4	-4.4	-4.8	-5.2
A	-0.4	1	0.6	0.2	-0.2	-0.6	-1	-1.4	-1.8	-2.2	-2.6	-3	-3.4	-3.8
T	-0.8	0.6	2	1.6	1.2	0.8	0.4	0	-0.4	-0.8	-1.2	-1.6	-2	-2.4
G	-1.2	-2	1.6	1.6	1.2	0.8	1.8	1.4	1	0.6	0.2	-0.2	-0.6	-1
A	-1.6	-2	1.2	1.2	1.2	0.8	1.4	2.8	2.4	2	1.6	1.2	0.8	0.4
C	-2	-0.6	0.8	2.2	1.8	2.2	1.8	2.4	2.4	2	1.6	1.2	0.8	1.8
T	-2.4	-1	0.4	1.8	3.2	2.8	2.4	2	2	3.4	3	2.6	2.2	1.8
C	-2.8	-1.4	0	1.4	2.8	4.2	3.8	3.4	3	3	3	2.6	2.2	3.2
T	-3.2	-1.8	-0.4	1	2.4	3.8	3.8	3.4	3	4	3.6	3.2	2.8	2.8
C	-3.6	-2.2	-0.8	0.6	2	3.4	3.4	3.4	3	3.6	3.6	3.2	2.8	3.8
A	-4	-2.6	-1.2	2	1.6	3	3	4.4	4	3.6	3.2	4.6	4.2	3.8
G	-4.4	-3	-1.6	-2	1.2	2.6	4	4	5.4	5	4.6	4.2	5.6	5.2
A	-4.8	3.4	-2	-0.6	0.8	2.2	3.6	5	5	5	4.6	5.6	5.2	5.2
C	-5.2	-3.8	-2.4	-1	0.4	1.8	3.2	4.6	4.6	4.6	4.6	5.2	5.2	6.2

Figure 36: Example of a Needleman-Wunsch global alignment matrix

Needleman-Wunsch global alignment matrix between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$, $MismatchScore = 0.4$ and $GapScore = 0.4$.

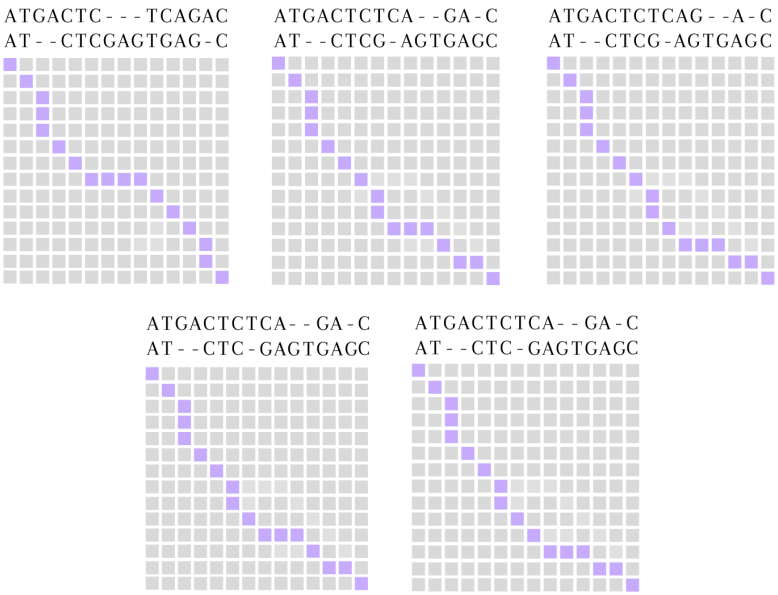

Figure 37: Needleman-Wunsch optimal alignment paths found in the example

Best alignments produced by NW global alignment between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$, $MismatchScore = 0.4$ and $GapScore = 0.4$.

starting from the last cell up until it reaches the origin, marks the possible interactions and finds all the best paths for the alignment. There were in total 5 alignments that obtained the $MaximumScore = 6.2$, as shown in Figure 37. The caption shows the alignment

and the graph shows the feasible path in the matrix that represents the alignment. The CIGAR string for the first alignment would be "2M2D3M3I4M1D1M".

4.1.2 The Smith-Waterman Algorithm

Later in 1981, Temple Smith and Michael Waterman had noticed that the frequency of occurrence of short insertions or deletions in nature was length-dependent, so they added an affine function to represent the gap score (SMITH; WATERMAN, 1981). They have also adapted the method to compute local alignment, which is useful in applications, such as searching for similarities between two sequences that have different sizes or are from different phylogenies.

Now gaps are penalized by the affine function $W_k = a + bk$, where the constant a represents the gap opening score, the multiplier b represents the gap extension score, and the variable k represents the length of the gap. Each cell then has to take into consideration gaps coming from all previous cells in the same line and column. The new equation below shows how each cell of matrix S is computed.

$$S_{ij} = max\{S_{i-1,j-1} + s(a_i, b_j), max_{k \leq 1}\{S_{i-k,j} - W_k\}, max_{l \leq 1}\{S_{i,j-1} - W_l\}\} \\ , l \leq i \leq n \text{ and } l \leq j \leq m \tag{4.2}$$

Using the same example, but aligning two sequences with an affine gap cost: A =ATGACTCTCAGAC (reference), B = ATCTCGAGTGAGC (query), match score is 1, mismatch penalty is 0.4, gap opening score is 1 and gap extension score is 0.3. Figure 38 shows the final Smith-Waterman matrix. $S_{1,2}$, for example, evaluates to -0.3 because the maximum value comes from $S_{1,1}$ scoring $1 - 1.3 = -0.3$, while, from $S_{1,0}$ the score is $-1.3 - 1.3 - 0.3 = -2.9$, from the upper diagonal the score is $-1.3 - 1.0 = -2.3$, and from the top the score is $-1.6 - 1.3 = -2.9$.

This example produced one best alignment with $Score = 3.8$, shown below. The CIGAR string for this alignment would be "2M2D3M3I4M1D1M".

A	T	G	A	C	T	C	-	-	-	T	C	A	G	A	C
A	T	-	-	C	T	C	G	A	G	T	G	A	G	-	C

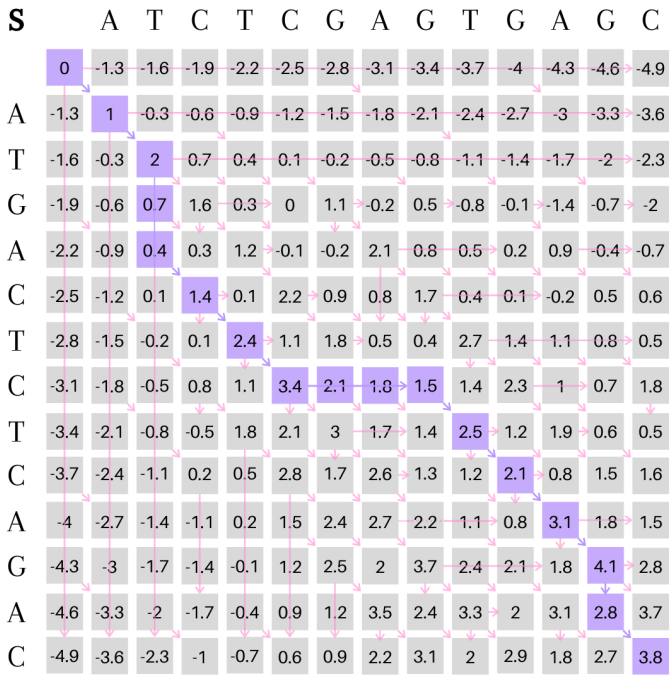

Figure 38: Example of a Smith-Waterman global alignment matrix

Smith-Waterman global alignment matrix between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$, $MismatchScore = 0.4$, $GapOpeningScore = 1$ and $GapExtensionScore = 0.3$.

To turn this process into a local alignment between two sequences, a new arrow pointing to the origin without penalty or reward has to be created (illustrated as a circle in Figure 39). Therefore all the values in the matrix are going to be positive or zero, since the score in the origin is zero. The backtracking should also start from the positions in the table with the maximum score (the purple scores in the matrix), instead of starting from the last cell, and end if it encounters an origin arrow. This will remove the requirement for the alignment to stretch from the beginning to the end of each sequence.

The algorithm found two solutions for best local alignments with $Score = 4.7$, for which, below, the strings are shown:

T	C	T	C	-	A	G		T	C	T	C	A	G	A
T	C	T	C	G	A	G		T	C	T	C	-	G	A

There is also the semi-global alignment, that is frequently used in read mapping algorithms that apply the seed-and-extend strategy. The alignment starts at the origin, that would be the seeding region, and extends until the max score in the matrix, so no origin arrow is used.

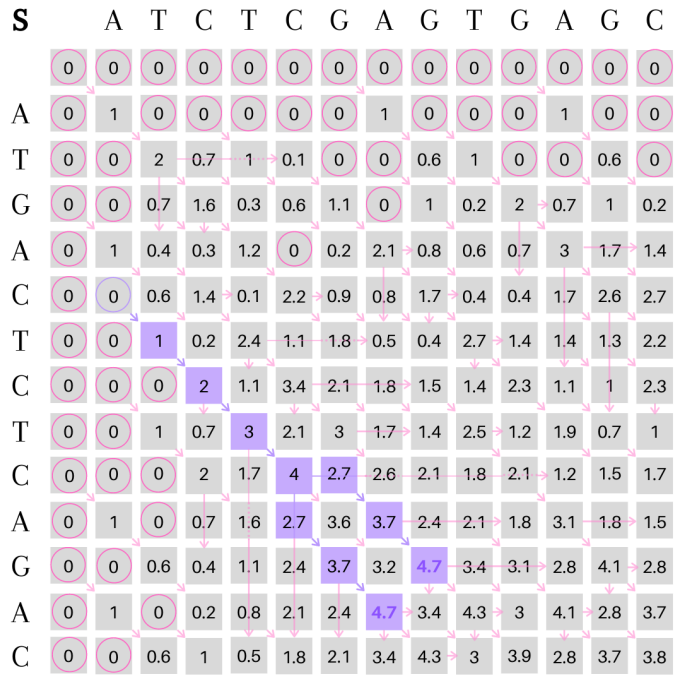

Figure 39: Example of a Smith-Waterman local alignment matrix

Smith-Waterman local alignment matrix between sequences ATGACTCTCAGAC (reference) and ATCTCGAGTGAGC (query), using $MatchScore = 1$,$MismatchScore = 0.4$, $GapOpeningScore = 1$ and $GapExtensionScore = 0.3$.

Although the SW Algorithm generated alignments that were more consistent biologically, it increased considerably the time complexity compared to the NW solution, from $O(MN)$ to $O(MN(M+N))$, where M and N are the lengths of the sequences being aligned. Also, the affine gap method frequently excluded very long gaps that could be generated from a single mutational event, such as crossing-over or transposition of a movable element. Logarithmic, quadratic, or other concave functions can be implemented in the SW algorithm to alleviate this issue, but will further increase the complexity of it.

4.1.3 The Smith-Waterman-Gotoh Algorithm

Osamu Gotoh came up with an elegant solution to the problems mentioned in the previous section and published his algorithm in 1990 (GOTOH, 1990). By adding matrices for each gap type, he could perform more levels of dynamic programming, eliminating the need to check the gap score coming from each previous cell in every iteration. His method could also extend to a piece-wise linear gap-weighting function that could approximate concave curves without increasing the complexity extensively.

For an example with one affine gap function, two extra matrices are added, one for

the horizontal gap score H and another for the vertical gap score V. The cells of the 3 matrices are filled simultaneously, starting from H and V. Cells in H and V only need to compute the best score between opening a new gap from the main matrix S, or expanding the gap from its neighbor. S needs to get the maximum score between a diagonal path from its neighbor, or the scores computed in matrices H and V in the same cell position. The summary of the algorithm is shown in the equations below.

$$\begin{aligned} H_{ij} &= max\{S_{i-1,j} - a, H_{i-1,j} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\ V_{ij} &= max\{S_{i,j-1} - a, V_{i,j-1} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\ S_{ij} &= max\{S_{i-1,j-1} + s(a_i, b_j), H_{ij}, V_{ij}\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \end{aligned} \tag{4.3}$$

Adding two more matrices will increase the cumulative memory requirement to store the traceback pointers by 3 times, but will reduce the time complexity in relation to the SW algorithm from cubic back to quadratic. This is because, with SWG, each cell will only have to consider scores from previous neighboring cells (considering "neighbor" cells from other matrices), instead of all previous cells in the same row/column as in the SW algorithm.

Taking the same example and aligning two sequences with an affine gap cost: A =ATGACTCTCAGAG (reference), B = ATCTCGAGTGAGC (query), match score is 1, mismatch penalty is 0.4, gap opening score is 1 and gap extension score is 0.3. Figure 40 shows the final Smith-Waterman-Gotoh matrices. The first matrix V computes the vertical gaps, the second matrix H computes the horizontal gaps, and the third matrix is the main matrix S. As in the case of $S_{1,2}$ seen in Section 4.1.2 from Figure 38, it also evaluates to -0.3, since it is the maximum among the values obtained from the upper diagonal cell $S_{0,1}$, $-1.3 - 1 = -2.3$, from $V_{1,2}$, -2.9, and $H_{1,2}$, -0.3. On their turn, $V_{1,2}$ is obtained from the max from $S_{0,2}$, $-1.6 - 1.3 = -2.9$, and from $V_{0,2}$, $-\infty - 0.3 = -\infty$. Similarly, $H_{1,2}$ is obtained from $S_{1,1}$, $1 - 1.3 = -0.3$, and from $H_{1,1}$, $-2.6 - 0.3 = -2.9$.

Both the SW and the SWG Algorithms will generate the same matrix S and compute the same alignments, but each with a different strategy. With the SWG Algorithm, any number of affine functions L can be added to approximate a desired curve. The time complexity is now $O(MN(L + C))$, where C is a constant.

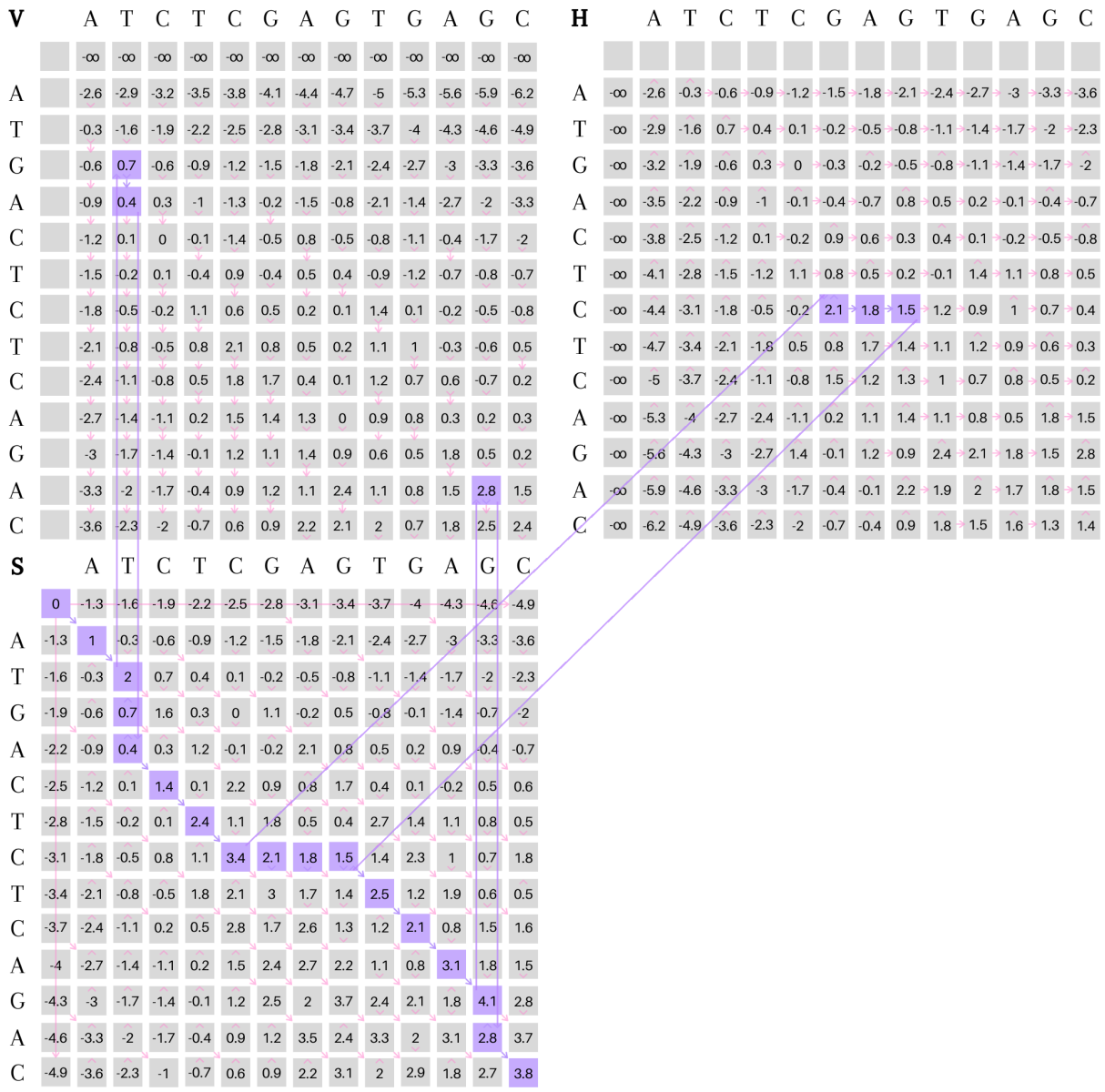

Figure 40: Example with the Smith-Waterman-Gotoh global alignment matrices

Smith-Waterman-Gotoh global alignment matrices between sequences ATGACTCTCAGAC (query) and ATCTCGAGTGAGC (reference), using $MatchScore = 1$, $MismatchScore = 0.4$, $GapOpeningScore = 1$ and $GapExtensionScore = 0.3$.

4.2 Minimap2: Assembly Algorithm for Long-Reads

Minimap2 (LI, 2018) is a State-of-the-Art algorithm formulated by Heng Li and published in 2018 in Bioinformatics. This section will present Minimap2's heuristic strategies to assemble genomes from short- and long-reads, achieving better accuracy and time performance for long-reads than other previous works.

Minimap2 was developed to tackle the problem of assembling long-reads produced by SMRT (Single Molecule Real-Time) sequencing technology and ONT. These technologies can produce reads longer than 10 kbp, at the cost of higher error rate ($\sim 15\%$), but error correction methods have reduced this rate to $< 5\%$. In general, it is not feasible for such data to be processed by mainstream short-read assemblers, mainly due to memory and higher error rate impediments.

The new strategies adopted in Minimap2 allowed it to be over 30 times faster than traditional long-read assemblers, with higher accuracy: Figure 41 (a) shows that, for a given mapping quality threshold (x axis), Minimap2 achieves the highest percentage of mapped reads (y axis).

The algorithms that had been developed specifically to assemble long-reads, such as BLASR (CHAISSON; TESLER, 2012) and BWA-MEM, would often perform five times slower than the ones developed for short-reads, when processing the same amount of data. Minimap2 running on short-reads performs 3 to 4 times faster than mainstream short-read assemblers, such as Bowtie2, and other long-read assemblers, such as BWA-MEM, but it is 1.3 times slower than SNAP. However, it is more accurate than Bowtie2 and SNAP and less accurate than BWA-MEM. The accuracy can be seen in Figure 41 (b), where, Minimap2 shows a second higher percentage of mapped reads, given a mapping quality threshold. The author in (LI, 2018) affirmed that Minimap2 showed to be worse than BWA-MEM in accuracy because the last one tries to locally align a read in a small region close to its mate.

Although Minimap2 uses the typical index and seed-and-extend procedure (see the method in Chapter 2.2.3), each of these steps has had a new improvement that was proven to be more effective. The next topics will present each one of these new adaptations.

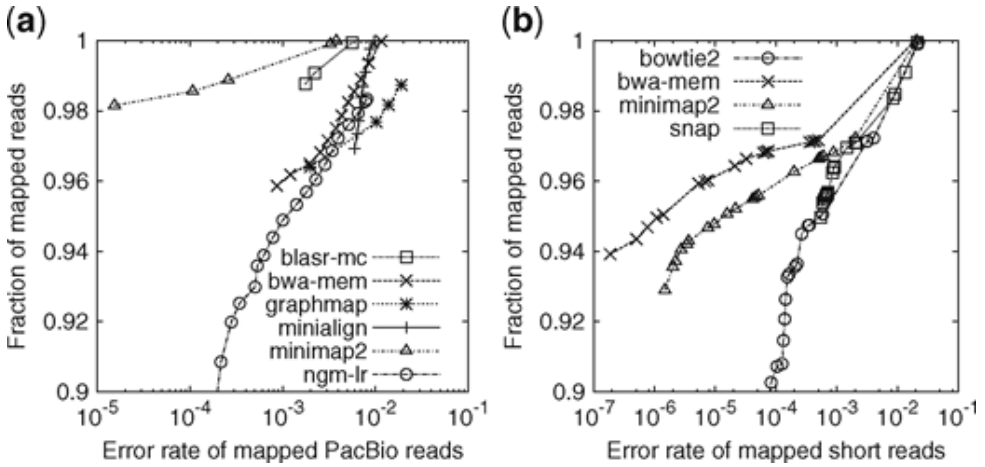

Figure 41: Comparison of mapping performances between read assemblers (LI, 2018)

A read was considered correctly mapped when it overlapped the true interval by at least 10%. **(a)** Long-read assembly evaluation. 33088 $\geq$ 1000 bp reads were simulated. **(b)** Short-read assembly evaluation. 10 million pairs of 150 bp reads were simulated.

4.2.1 Hash Table Indexing

Mainstream aligners often use a full-text index, such as suffix array or FM-index (FERRAGINA; MANZINI, 2000), to index reference sequences. This allows higher seed uniqueness which will reduce the number of unsuccessful extensions. Minimap2 uses a table for indexing fixed length code. The fixed length allows more efficient computation: query seeds with multiple hits can be skipped without affecting the final accuracy, compensating for the advantage that seed uniqueness offers. The author ended up concluding that a hash table is the ideal data structure for mapping long noisy sequences.

On Minimap2, a k-mer is a code sequence of length k. The k-mers are collected from the reference during indexing stage, and later on from the reads during mapping stage, through the same function. A window of size k scans through the target sequence with w steps. The hash function from eq. 4.4 is applied to all these k-mers and their respective inverted Watson-Crick complements (see Figure 2). The sub-sequence or the complement that has the smallest hash is taken as a minimizer and is added to the set of minimizers M (Figure 42). Minimap2 uses $k = 15$ and $w = 5$, but these can be changed by the user.

$$\varphi(s) = \varphi(a_1) \times 4^{k-1} + \varphi(a_2) \times 4^{k-2} + \cdots + \varphi(a_k) \tag{4.4}$$

If $\varphi(A) = 0$, $\varphi(C) = 1$, $\varphi(G) = 2$ and $\varphi(T) = 3$ are adopted, the hash function will always map a k-mer to a distinct 2k-bit integer. Only poly-A will always get zero.

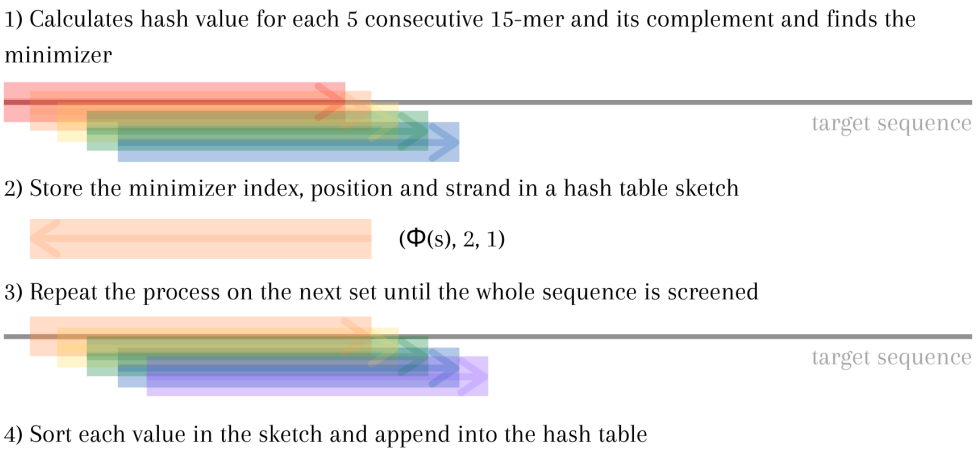

Figure 42: Minimap2's algorithm for collecting minimizers and indexing

In this example, the minimizer is the reverse complement of the orange k-mer. The minimizer index, calculated with function $\varphi(s)$; the choice of minimizer in the set (2); and the reverse complement flag "1" are stored.

Because of this, Minimap2 uses $\varphi' = h \cdot \varphi$, where h is an invertible integer hash function on $[0, 4^k)$. In short, a hash table is a dictionary where the key is the minimizer hash and the value is a set of target sequence index, position of minimizer, and sequence. Each value list is sorted before being added to the hash table to reduce heap allocations and cache misses.

4.2.2 Perfect Match Seeding

For each query sequence (referring to read sequences), Minimap2 takes query minimizers as seeds and finds exact matches, called anchors, in the reference, using the previously indexed hash table (Figure 43). From the figure, it can be noticed that a seed in the read may be anchored in different parts of reference. This step is often used in other assemblers but in Minimap2, the seeds have fixed length and are matched to the reference through the hash table. The seed is skipped when it has too many occurrences in the reference.

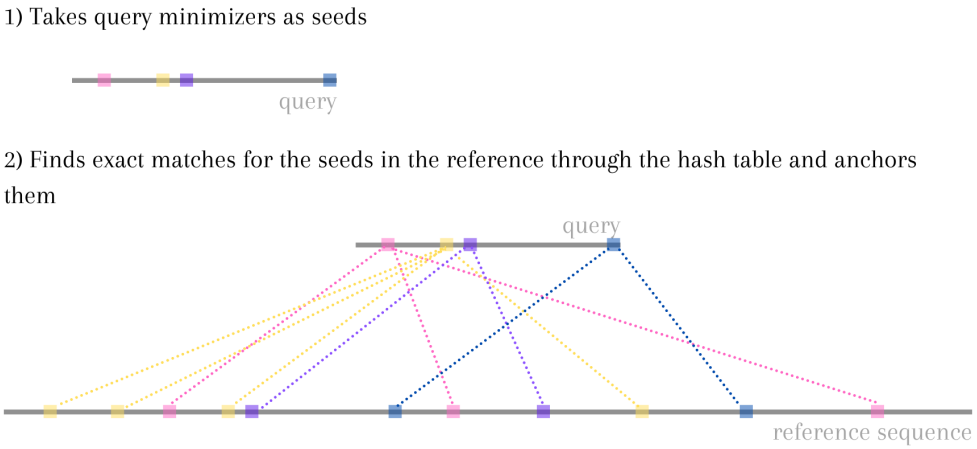

Figure 43: Minimap2's seeding and anchoring steps

4.2.3 Chaining for a Filtering Stage

All sets of collinear anchors are grouped into chains (Figure 44). An anchor is a 3-tuple (x, y, w) that indicates that interval $[x - w + 1, x]$ on the reference matches interval $[y - w + 1, y]$ on the query. The chaining score up to the $i - th$ anchor in the sorted chain is given by the function 4.5. The function can be computed with dynamic programming.

$$f(i) = max\{\{f(j) + \alpha(j, i) - \beta(j, i)\}, w_i\} \tag{4.5}$$

In function 4.5, $\alpha(j, i) = min\{min\{y_i - y_j, x_i - x_j\}, w_i\}$ is the number of matching bases between the two anchors. $\beta(j, i) = \gamma_c((y_i - y_j) - (x_i - x_j))$ is the gap cost and equals to ∞ when $y_j \geq y_i$ or when the distance between two anchors is higher than G. A

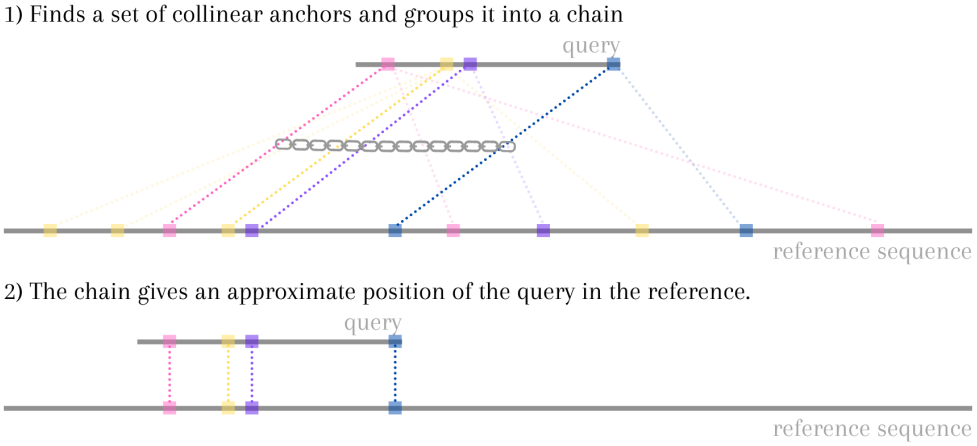

Figure 44: Minimap2's chaining step

gap of length l for the average seed length $\bar{w}$ costs:

$$\begin{aligned} \gamma_c(l) &= 0.01 \cdot \bar{w} \cdot |l| + 0.5 \cdot log_2|l| \ (l \neq 0) \\ \gamma_c(l) &= 0 \ (l = 0) \end{aligned} \tag{4.6}$$

For N anchors, chaining has complexity $O(N^2)$, but is able to take a generic gap cost function, is simple to implement, and is not associated with a large constant like in other algorithms. Minimap2 applies an heuristic strategy to accelerate this step. It limits the number of iterations to $h = 50$, since it is unlikely that an anchor i chained with j has better score when chained against j's predecessors. This reduces the complexity to $O(hN)$ and, according to Minimap2's author, it can almost always find the optimal chain, and even when it fails, the optimal chain is often close.

As mentioned before, the chaining process is a dynamic-programming algorithm, so during chaining, there is a process of backtracking that appends to a chain new anchors in order that provide the best score. When backtracking, anchors are marked as 'used', so that no anchor is used in more than one chain. Sometimes chains can have significant or complete overlaps due to repeats in the reference. To identify the primary chains, all chains are sorted according to their scores, and an empty list Q is created. For each query, if the chain overlaps with a chain in Q by 50% or more, the chain is marked as secondary; otherwise, it is added to Q so, in the end, Q contains all the primary chains.

Although many chains can be detected, usually each read should only be mapped to one place in the reference. The best primary chain is picked for this purpose to be aligned in the next stage. At the end, each read with its associated chain will be mapped to the reference, with each chain composed of several anchors. Sub-sequences between adjacent anchors are called in this dissertation as anchor-separated sub-sequences, and are individually aligned in the Smith-Waterman-Gotoh algorithm, as to be seen in the next section. The sub-sequence in the chain that extends to the right, from the leftest anchor to the end of the sequences is called anchor-extended sub-sequence, and will be used at Chapter 6.1.2 in the GACT-X's implementation.

4.2.4 SSE Optimized Alignment

After getting the primary chains, Minimap2 uses the dynamic-programming-based algorithm Smith-Waterman-Gotoh, with the Suzuki-Kasahara formulation (SUZUKI; KASAHARA, 2018), to perform global alignment of anchor-separated sub-sequences in a chain, and semi-global alignment for the head and tail sequence pairs. In this way, the final result is a local alignment that maps the read on the reference and aligns the read sequence to a reference sub-sequence.

Minimap2 implements its alignment with a 2-piece affine gap cost, as shown in eq. 4.7. In this way, long insertions and deletions, that often occur in complex genomes such as the human genome, can be recovered.

$$\gamma_a = min\{q + |l| \cdot e, \tilde{\mathrm{q}} + |l| \cdot \tilde{\mathrm{e}}\} \tag{4.7}$$

The Suzuki-Kasahara Formulation

Hajime Suzuki and Masahiro Kasahara published in 2018 a reformulation of the Smith-Waterman-Gotoh Algorithm (SUZUKI; KASAHARA, 2018), allowing it to be the fastest SSE SWG implementation, with a 2.1 fold higher performance than that of the fastest implementation for the semi-global alignment of long-reads. They noticed that, as the read length increases, the values of the cells in the dynamic-programming matrix increase. Traditionally, SSE implementations could achieve 16-way parallelization for short sequences, but only 4-way parallelization for long sequences, when the peak alignment score reached 32,767, which was turning the alignment step into a significant bottleneck. This happens because SSE vector instructions are vectors of fixed length with data that have limited size; if the data surpasses the limit value, more cells are needed to represent it, reducing the parallelization capacity of the vector instructions.

Difference Recurrence Relation

Instead of calculating and storing all the values in a matrix, they proposed a "difference recurrence relation", where the differences between the neighboring cells are stored, not the integral value of each cell. In this way, they could guarantee that the values would have a limited range and would be able to be stored as an 8-bit integer, enabling 16-way SSE vectorization.

First consider the following small change to the SWG formulation:

$$
\begin{aligned}
S_{ij} &= max\{S_{i-1,j-1} + s(a_i, b_j), H_{i-1,j} - b, V_{i,j-1} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\
H_{ij} &= max\{S_{ij} - a, H_{i-1,j} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\
V_{ij} &= max\{S_{ij} - a, V_{i,j-1} - b\}, 1 \leq i \leq n \text{ and } 1 \leq j \leq m
\end{aligned} \tag{4.8}
$$

With this, four difference matrices are created, as shown in eq. 4.9 and Figure 45.

$$
\begin{aligned}
\Delta H_{ij} &= S_{ij} - S_{i-1,j}, (i \geq 1) \\
\Delta V_{ij} &= S_{ij} - S_{i,j-1}, (j \geq 1) \\
\Delta H'_{ij} &= H_{ij} - S_{ij} \\
\Delta V'_{ij} &= V_{ij} - S_{ij}
\end{aligned} \tag{4.9}
$$

A new intermediate variable I is introduced to represent the diagonal difference $S_{ij} - S_{i-1,j-1}$. The new recurrences then become:

$$
\begin{aligned}
I_{ij} &= max\{s(a_{i-1}, b_{j-1}), \Delta H'_{i-1,j} + \Delta V_{i-1,j} - b, \Delta V'_{i,j-1} + \Delta H_{i,j-1} - b\} \\
\Delta H_{ij} &= A_{ij} - \Delta V_{i-1,j} \\
\Delta V_{ij} &= A_{ij} - \Delta H_{i,j-1} \\
\Delta H'_{ij} &= max\{-a, \Delta H'_{i-1,j} - \Delta H_{ij} - b\} \\
\Delta V'_{ij} &= max\{-a, \Delta V'_{i,j-1} - \Delta V_{ij} - b\}
\end{aligned} \tag{4.10}
$$

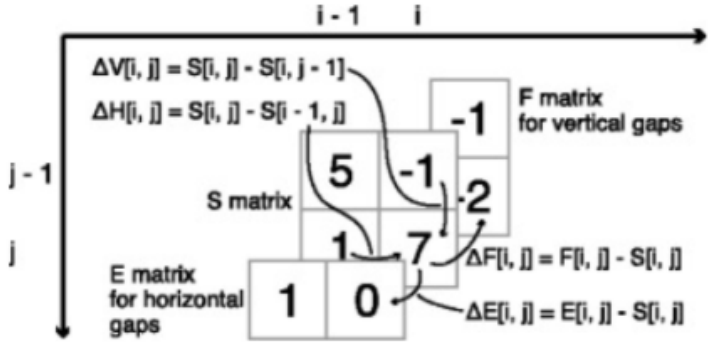

Figure 45: The Suzuki-Kasahara transformation (SUZUKI; KASAHARA, 2018)

In the first transformation of the Suzuki-Kasahara algorithm, the new values ΔH, ΔV, $\Delta H'$ and $\Delta V'$ are stored in four new matrices instead of the original values S, H and V.

All the values in the cells are now limited by:

$$
\begin{aligned}
-a - b \leq \Delta H \leq M + a + b \\
-a - b \leq \Delta V \leq M + a + b \\
-a \leq \Delta H' \leq 0 \\
-a \leq \Delta V' \leq 0
\end{aligned}
$$

, where M is the maximum value of $s(x, y)$, or the match score in nucleotide alignment.

The row-column coordinate is transformed into the diagonal-antidiagonal coordinate by letting $r \leftarrow i + j$ and $t \leftarrow i$. In this way, cells with the same antidiagonal index r are independent of each other, allowing them to fully vectorize the computation of all cells on the same diagonal. Using smaller integer types also reduces the memory requirements. Now all the matrices go through one more transformation:

$$
\begin{aligned}
A_{G_{ij}} &= A_{ij} + 2a + 2b \\
\Delta H_{G_{ij}} &= \Delta H_{ij} + a + b \\
\Delta V_{G_{ij}} &= \Delta V_{ij} + a + b \\
\Delta H'_{G_{ij}} &= \Delta H'_{ij} + \Delta V_{ij} + 2a + b \\
\Delta V'_{G_{ij}} &= \Delta V'_{ij} + \Delta H_{ij} + 2a + b \\
s_G(x, y) &= s(x, y) + 2a + 2b
\end{aligned}
\tag{4.11}
$$

The new recurrences then become:

$$
\begin{aligned}
I_{G_{ij}} &= max\{s_G(a_{i-1}, b_{j-1}), \Delta H'_{G_{i-1,j}}, \Delta V'_{G_{i,j-1}}\} \\
\Delta H_{G_{ij}} &= A_{G_{ij}} - \Delta V_{G_{i-1,j}} \\
\Delta V_{G_{ij}} &= A_{G_{ij}} - \Delta H_{G_{i,j-1}} \\
\Delta H'_{G_{ij}} &= max\{A_{G_{ij}}, \Delta H'_{G_{i-1,j}} + a\} - \Delta H_{G_{i,j-1}} \\
\Delta V'_{G_{ij}} &= max\{A_{G_{ij}}, \Delta V'_{G_{i,j-1}} + a\} - \Delta V_{G_{i-1,j}}
\end{aligned}
\tag{4.12}
$$

Finally, the values can be stored as an array of unsigned integers:

$$
0 \leq \Delta H_G, \Delta V_G, \Delta H'_G, \Delta V'_G \leq M + 2a + 2b
$$

The critical path, or the longest operation dependency chain, is reduced to 4 from 8 in the first difference recurrences transformation, and from 5 in the non-difference semi-global alignment algorithm.

Adaptive Banded Dynamic-Programming Algorithm

In the SK (Suzuki-Kasahara) algorithm, an adaptive banded dynamic-programming strategy is used, so that only part of the cells, specifically those around the antidiagonal, is calculated, since it is the region where the alignment between two somewhat similar sequences is expected to be. A forefront vector of constant width iteratively moves right and down and forms a band, trying to move away from the cells with lower scores. The band is divided into blocks containing vectors calculated in 32 successive updates each (Figure 46).

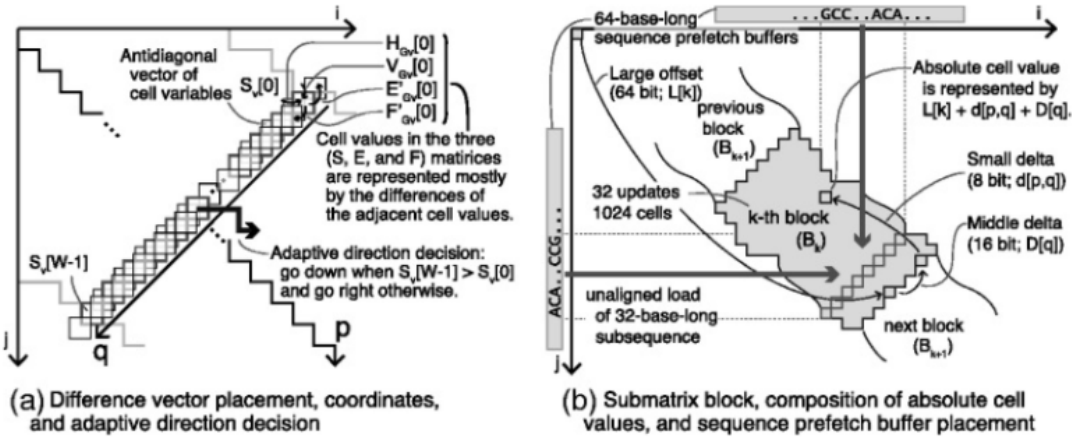

Figure 46: Suzuki-Kasahara's banding strategy (SUZUKI; KASAHARA, 2018)

(a) In the new coordinates, $p = i + j$ corresponds to the position of the vectors, and q is a local position within a vector. The advancing direction (right or down) is determined by the smallest difference between values of cells $S_v[0]$ and $S_v[W-1]$. **(b)** Data structure: each block has 32 vectors and is indexed as k. $L(k)$ is an offset integer that helps calculating the absolute value of a cell in the block.

The Z-drop heuristic

To avoid computing global alignment of unrelated sub-sequences, in query and reference, between two adjacent anchors (e.g. when there is a short inversion), Minimap2 computes an accumulative alignment score and breaks the alignment when the score drops too fast in the anti-diagonal direction. The condition follows eq. 4.13.

$$S(i', j') - S(i, j) > Z + e \cdot |(i - i') - (j - j')| \tag{4.13}$$

, where e is the gap extension cost and Z is an arbitrary threshold. The Z-drop strategy was first used in BWA-MEM and is similar to the X-drop algorithm implemented in BLAST (ALTSCHUL et al., 1990), but allows the presence of a single long gap.

When the alignment is broken, Minimap2 performs local alignment in the same region but with one sub-sequence reverse complemented. This may help identify short inversions that were missed during the chaining process.

4.3 GACT-X: an FPGA Accelerated SWG Implementation with Fixed Memory Usage (TURAKHIA et al., 2019)

As seen in Section 3.8, GACT-X is a hardware aligner, implemented by its developers on an AWS Cloud FPGA, that was replicated in this work to accelerate Minimap2. GACT-X computes the SWG matrix with limited traceback memory for any input lengths. The memory usage is limited because the computation is performed in tiles of fixed size (instead of on a long band as in other alignment algorithms); one tile overlaps a previous one by a certain number of bases to expand the alignment, forming a scaffold band on the SWG matrix. GACT-X differs from GACT from Darwin (Section 3.6) by computing cells within an X-drop band (to be clarified next) in each tile, further reducing on-chip SRAM usage, and allowing tiles to be considerably wider; and by using a different overlapping strategy.

GACT-X's expansion starts from the anchoring point and stops when the last tile's max score is negative or zero, or when the alignment has reached the border of query or target sequences. The overlap system is similar to GACT's, but if the max score cell lies outside the overlap boundary, the next tile's origin will be placed on that cell. In Figure 47, tile T1 performs left-extension from the anchor, and tiles T2 and T3 perform right-extension. T3's origin is placed on the intersection of the alignment path from T2 and its overlap border, so the alignment tail produced by T2 in green is ignored. The blue area in the zoomed image represents the X-drop band in T2.

X-drop is the concept of interrupting further computation of cells or further expansion

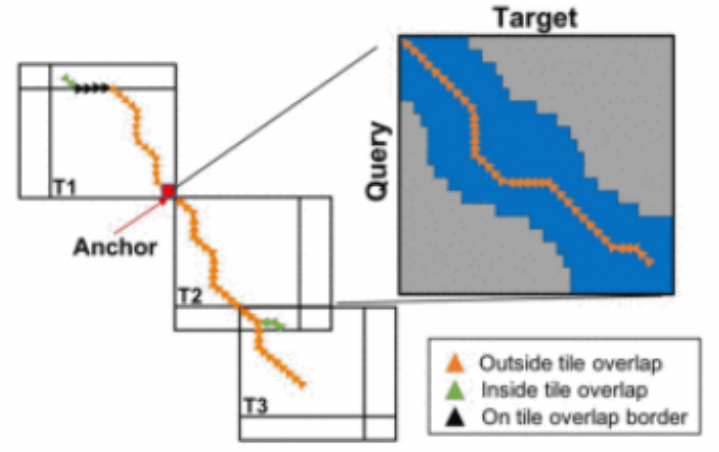

Figure 47: GACT-X tile overlap algorithm (TURAKHIA et al., 2019)

of alignment if the new score drops below a threshold of X (Y in GACT-X) in comparison to the maximum score registered. It is a broad definition because the term X-drop is used in different algorithms for different purposes. Minimap2 uses X-drop to interrupt global-alignment extension of sequences between anchors if they are too dissimilar (see Subsection 4.2.4). GACT-X uses X-drop to create a dynamic band around the alignment path inside of tiles.

In each tile, the matrix is filled row-wise while tracking the maximum score V_{max} of the tile. Stripes with $N_{PE} = 32$ rows are calculated with wave-front parallelism. The computation in a stripe is terminated when all the scores in a column fall below $(V_{max} - Y)$, Y being a given threshold. The next stripe also begins computation from the first column with all cells exceeding $(V_{max} - Y)$, generating a dynamic band around the alignment (Figure 48).

The processor implementation follows approximately the same architecture as in Darwin. The host transfers the target and the query sequences to the DRAM. N_{PE} query characters are loaded to the PEs, and target elements are streamed in a systolic fashion. A fixed memory of 1 BRAM bank per PE is allocated to store the 4-bit traceback pointers sequentially — 2 bits for the main matrix and 2 bits for gap matrices. Start and stop positions of each stripe are stored in separate BRAMs (Figure 49).

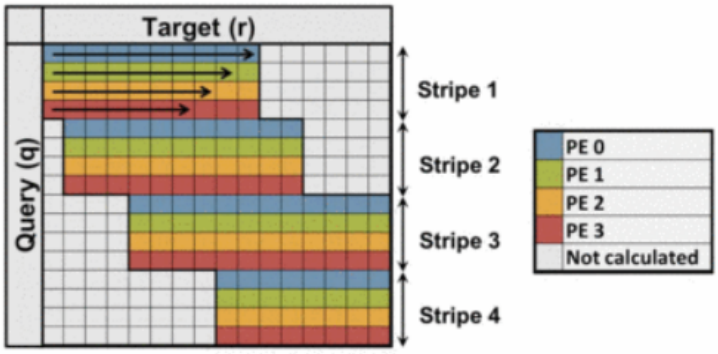

Figure 48: GATC-X tile with X-drop banding (TURAKHIA et al., 2019)

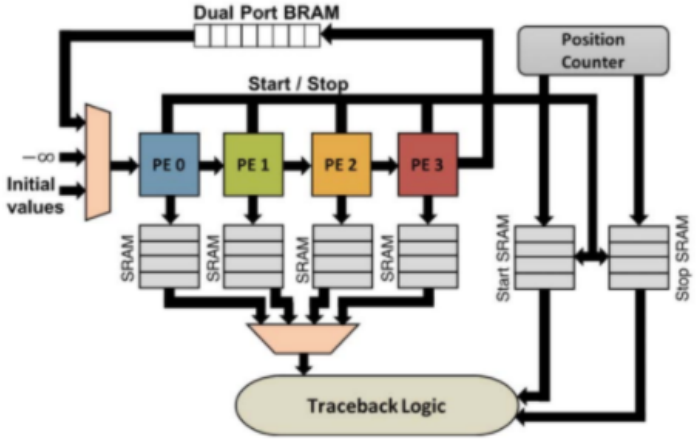

Figure 49: GACT-X systolic array with 4 PEs (TURAKHIA et al., 2019)

5 FIRST STEPS FOR PREPARATION

This chapter describes a methodical analysis of Minimap2, carried out to help develop the acceleration strategy. The objective is to obtain and apply the algorithm on different datasets (long-read sequences) and assess the algorithm's characteristics assembling human genomes. First, it shows how simulated and real human input data were collected. Then, it presents the execution time evaluation for the different datasets. Some graphic results for mapping quality assessment and alignment characteristics are presented.

5.1 Simulating Data with PBSIM

PBSIM, a model-based simulator (ONO; ASAI; HAMADA, 2013), collects stretches of a reference sequence, mimicking the read length distribution; and adds variants and sequencing errors. The GRCh38 PacBio continuous long reads (CLRs) were generated with the PBSIM tool, based on the GRCh38 reference, with error profile sampled from file 'm131017_060208_42213_*.1.*' downloaded at (HUMAN..., 2014). This group of reads will be referred as Simulated PacBio reads. Simulated reads are useful for evaluating the mapping accuracy, due to lack of truth in real data. Since the original reference is known, the simulated reads can be mapped in Minimap2 to check its mapping accuracy. Minimap2's author simulated PacBio data with this same process to compare their mapping accuracy with other tools (LI, 2018).

PBSIM simulates differences introduced to reads by analyzing alignments performed on real PacBio reads with respect to reference sequences using LAST (FRITH; HAMADA; HORTON, 2010) ($match = 1$, $mismatch = -2$, $gap_{opening} = -1$ and $gap_{extension} = -1$). Length distribution is log-normal; average accuracy over the read length has normal distribution with parameters given by the user; single molecule errors are stochastic (random); coverage depth is nearly uniform. Since PacBio nucleotide insertions have higher probability of being the same as their following neighbors, half of inserted nucleotides are the same as their following nucleotides and the other half is random.

PBSIM outputs simulated reads in FASTQ format, and read origins in MAF format, for each of the 193 sequences in the GRCh38 reference's Primary Assembly. The MAF files can be used to evaluate the mapping accuracy, since it describes where the reads came from. The algorithm was executed and the 193 ".fastq" and 193 ".maf" files were merged into one of each, to facilitate the file manipulation, resulting in 121GB of CLRs. Figure 50 shows PBSIM's log report; the default coverage depth is 20; the read length thresholds were 100 to 25,000; the mean length, deviation, mean accuracy, and accuracy deviation were sampled from the given file. Some simulated reads had lengths outside of the threshold determined, and were filtered out, resulting in the distribution in Table 4.

Over seven million reads were generated in total, which corresponds to a 20x coverage depth. The read lengths mean, standard deviation (SD), minimum, and maximum are respectively 8310, 106.26, 100, and 24,988. The reads' similarity rate to the reference strand is on average 0.85, with 0.00017 SD. Substitution, insertion, and deletion rates are 0.015, 0.090, and 0.046 respectively. Note that, on the sample given, there were considerably more insertions than deletions.

```
:::: Simulation parameters :::

Simulated by fastq sampling.

prefix : sd
data-type : CLR
depth : 20.000000
length-mean : (sampling FASTQ)
length-sd : (sampling FASTQ)
length-min : 100
length-max : 25000
accuracy-mean : (sampling FASTQ)
accuracy-sd : (sampling FASTQ)
accuracy-min : 0.750000
accuracy-max : 1.000000
difference-ratio : 10:60:30

:::: FASTQ stats ::::

file name : m131017_060208_42213_c100579642550000001823095604021496_s1_p0.1.subreads.fastq

:: all reads ::
read num. : 23365
read total length : 195771725
read min length : 35
read max length : 33422

:: filtered reads ::
read num. : 23001
read total length : 191169143
read min length : 104
read max length : 24988
read length mean (SD) : 8311.340507 (5448.702016)
read accuracy mean (SD) : 0.846338 (0.026921)
```

Figure 50: PBSIM simulation parameters and statistics from the ".fastq" sample used

Table 4: Statistics of CLRs generated with PBSIM in sampling mode

number of reads	coverage depth	read length				read accuracy		rates		
		mean	SD	min	max	mean	SD	substitution	insertion	deletion
7,460,510	20	8310	106.26	100	24,988	0.85	0.00017	0.015	0.090	0.046

5.2 Collecting Real Reads from Genomic Databases

Real datasets, obtained from real human samples, can give insights on real runtime and accuracy behaviors of Minimap2. The datasets were chosen by size, which is expected to be over the minimum coverage for clinical applications ($\geq 8\times$), and by current and known sequencing technologies. The first datasets encountered are from ethnicities that lack representation in the most renowned databases, therefore it is expected that they might diverge more from the genome reference used. This divergence is better for evaluating assembly algorithms because they expose them to more situational adversities. Other factor that should be considered when choosing the datasets is potential issues in the library preparation, for example, contamination of foreign DNA, but this can only be said after the pre-processing step of the genome analysis pipeline.

The first dataset was collected from real human ONT PromethION reads at the European Nucleotide Archive (RUN..., 2018), from a project that sequenced and produced the Yoruba reference genome NA19240, using 5 PromethION flowcells (COSTER et al., 2019). Some researchers are working on producing regional genome references to reduce the bias from the GRCh38 reference. However, the best practices still recommend using the later, while these projects mature with more sequencing data. The reads have on average 16,900 bases according to ENA's base and read counts (28,528,692,209 and 1,688,000).

The second dataset was collected from real PacBio Sequel II reads at NCBI (SRX9063500..., 2019). The reads were collected from a Sri Lankan individual. The reads have an average length of 13,329 bases according to NCBI's base and read counts (52.2G and 3,916,231).

A script that collects the read length histograms of the simulated and the two real datasets was written in Python, and the measurements are displayed in Figure 51. The average read length alone sometimes can't provide the whole information. For example, the real PacBio dataset and the real ONT dataset have close mean lengths (13,329 and 16,900), but their length distribution is drastically different. It is important to observe that the last length interval to the right corresponds to the accumulated frequency of all reads larger than 30,000. In the case of real ONT dataset, the last item does not correspond to a peak, but to a flat low frequency distribution.

The length distribution is going to be relevant when assessing the performance, as seen in Section 5.3.

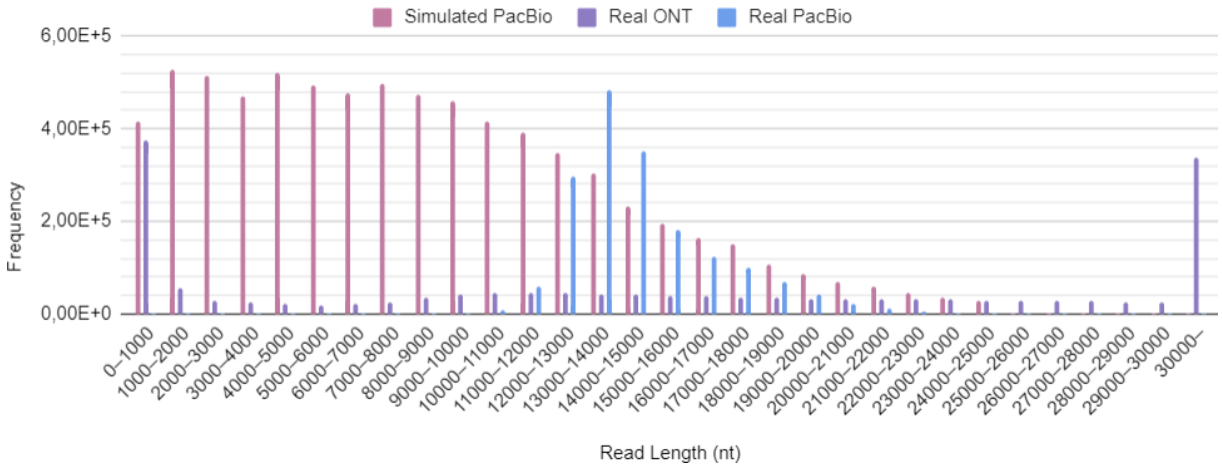

Figure 51: Simulated PacBio, real ONT and real PacBio read length histograms

5.3 Profiling Minimap2's Execution Time

It was already presented previously that assembly algorithms are computationally demanding and constitute the bottleneck of the genome analysis pipeline. Before accelerating an algorithm, some measurements on the software's throughput and its runtime bottlenecks should be taken.

Profiling Minimap2 with the three read sets presented in previous Sections 5.1 and 5.2 were performed on a Dell PowerEdge R910 server[1], running Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-201-generic x86_64), with further details in Figure 52 (information was obtained with the instruction "lscpu" on the command prompt). All three read sets were mapped to the Primary Assembly of the GRCh38 reference. To save time, the minimizer ".mmi" reference index was previously saved and used to substitute the reference ".fasta" file.

The datasets were mapped using 40 threads to collect Minimap2's throughput and internal data information presented in Section 5.4. Minimap2 was executed to output the alignment in ".sam" format and with optimization for ONT or PacBio reads. The gprof Linux tool (GNU..., 1998) was used to profile the execution time, and each execution was done with one thread. Since only one thread was being used for profiling, only the first 200,000 reads of each dataset were selected for use, otherwise the execution would take too long. The results are presented in Table 5.

[1]Professor Carlos Menck from *Instituto de Ciências Biomédicas* at University of São Paulo granted access to the Bioinformatics Seal server, with 4 Intel® Xeon® CPU E7-4870, 80 cores and 504GB of memory.

```
cteng@seal:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   44 bits physical, 48 bits virtual
CPU(s):                          80
On-line CPU(s) list:             0-79
Thread(s) per core:              2
Core(s) per socket:              10
Socket(s):                       4
NUMA node(s):                    4
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           47
Model name:                      Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:                        2
CPU MHz:                         2394.129
BogoMIPS:                        4788.25
Virtualization:                  VT-x
L1d cache:                       1.3 MiB
L1i cache:                       1.3 MiB
L2 cache:                        10 MiB
L3 cache:                        120 MiB
NUMA node0 CPU(s):               0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,76
NUMA node1 CPU(s):               1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61,65,69,73,77
NUMA node2 CPU(s):               2,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,66,70,74,78
NUMA node3 CPU(s):               3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63,67,71,75,79
```

Figure 52: ICB Seal server's CPU information

Table 5: Minimap2's throughput and bottlenecks running on three datasets

	Average Length (nt)	Real Time (hours) *	CPU Time (hours) *	Dataset Size (Gbases)	Throughput (kbases/s)	Chaining Time **	Extending Time **
Simulated PacBio Reads	8,300	2:46	42:41	61.99	403.46	8%	49%
Real PacBio Reads	13,300	4:55	88:21	52.20	164.11	50%	25%
Real ONT Reads	16,900	2:14	34:27	28.53	230.00	27%	42%

* 40 threads, complete dataset
** 1 thread, first 200,000 reads

The algorithm's speed and profile could have been influenced by input data characteristics, such as genome reference length, read length distribution, and similarity between reference and reads. Throughput was calculated based on the datasets' sizes and the CPU time. The simulated PacBio reads showed the highest throughput, but also the lowest average read length. Real PacBio reads had the lowest throughput and highest chaining time. More analyses presented in Section 5.4 can also help explain these differences.

The Minimap2's functions ksw_extd2_sse41 (referred to as ksw in this dissertation) and mm_chain_dp, responsible for the base-level alignment and chaining steps respectively, took together between 57% and 75% of the execution time. Other functions in order took 10% or less of the execution time. It is possible to conclude that the aligning and chaining steps are the bottlenecks of Minimap2's performance when assembling long-reads. Previous works reinforce this conclusion (GUO et al., 2019)(FENG et al., 2019) or add that these are also the same bottlenecks of Minimap2 on short-read execution (TENG et al., 2021).

5.4 Analyzing Minimap2's Intermediate Processing Data

5.4.1 Indel Sizes, Alignment Deviation, and Mapping Quality

Another important preparation step before accelerating Minimap2 is assessing the mapping accuracy and the characteristics of the alignments produced by it. Mapping quality can be measured by the proportion of reads that have been mapped to the reference and, particularly for the simulated data, mapping quality can also be assessed by the proportion of reads that were mapped near the original position. Other information pieces, such as indel sizes and deviation from anti-diagonal, help assess how the banding algorithm in Minimap2 behaves with respect to different long-read profiles. For that, two Python scripts were written to obtain statistical data from the outputs of simulated and real data.

The first set of data refers to histograms of indel sizes as shown in Figure 53. The indel sizes are directly collected from the numbers preceding the "I" or "D" alignment flags in the CIGAR strings reported in the SAM files. For all three datasets, the graphs showed that most of the indels are of a single nucleotide, and that the frequency decreases steeply, the longer the indels.

A second set of data refers to histograms of minimum fixed half band required to find the same alignment for the read as found in the Suzuki-Kasahara algorithm calls

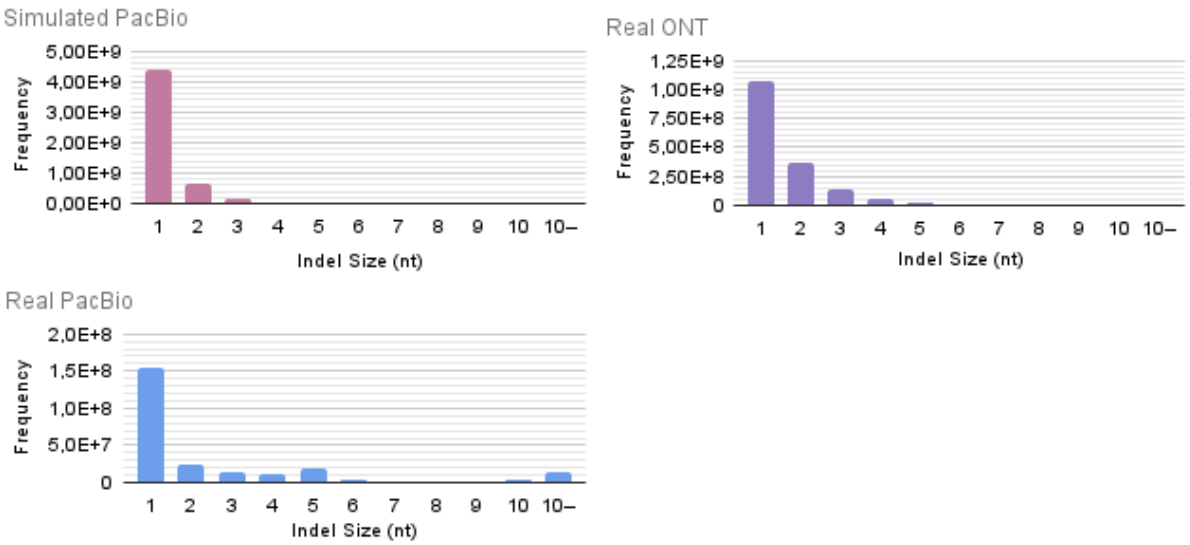

Figure 53: Indel size histograms from CIGAR strings reported on the SAM files

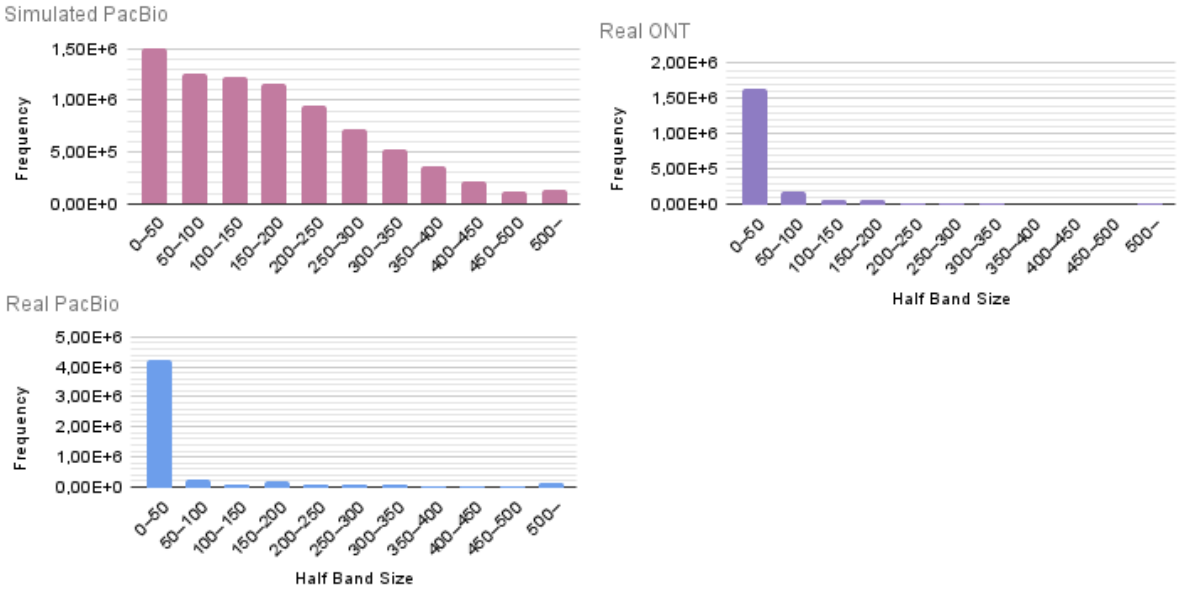

Figure 54: Histograms of deviation from the anti-diagonal from CIGAR strings

coupled with the chaining partitioning, and it is shown in Figure 54. The minimum half-band represents how much the resulting alignment has deviated from the anti-diagonal; it is calculated by adding cumulative insertions and deletions in the CIGAR string while canceling each other. It is important to note that, for Minimap2, this deviation for reads does not represent the deviation from the anti-diagonal in the SK matrices, to be presented in the next section, since the reads are irregularly divided by anchors.

For the simulated data, the deviation was significant up to a number of 500 from the anti-diagonal; this could have resulted from the higher insertion rate, compared to the deletion rate, as shown in Table 4. On the other hand, a great portion of the alignments stayed in a range of 50 from the anti-diagonal for real ONT and real PacBio cases, meaning a tighter band can be adopted for better performance.

One second script collects histograms for mapping overlap and histograms for SAM alignment flag reports. The mapping overlap can only be obtained from simulated reads, due to lack of truth in real data. It measures how much Minimap2's alignment overlaps with the original position of the read, calculated in percentage of the read's length. With Minimap2 running on the simulated PacBio reads, 99.45% of the reads were mapped to the correct RefSeq. From these, over 99.6% of the reads were mapped with an overlap greater than 90% to the expected mapping region.

The histograms on SAM flags, shown in Figure 55, were obtained directly from the SAM file, and the main report refers to the items "mapped" and "4" (see Figure 12 at

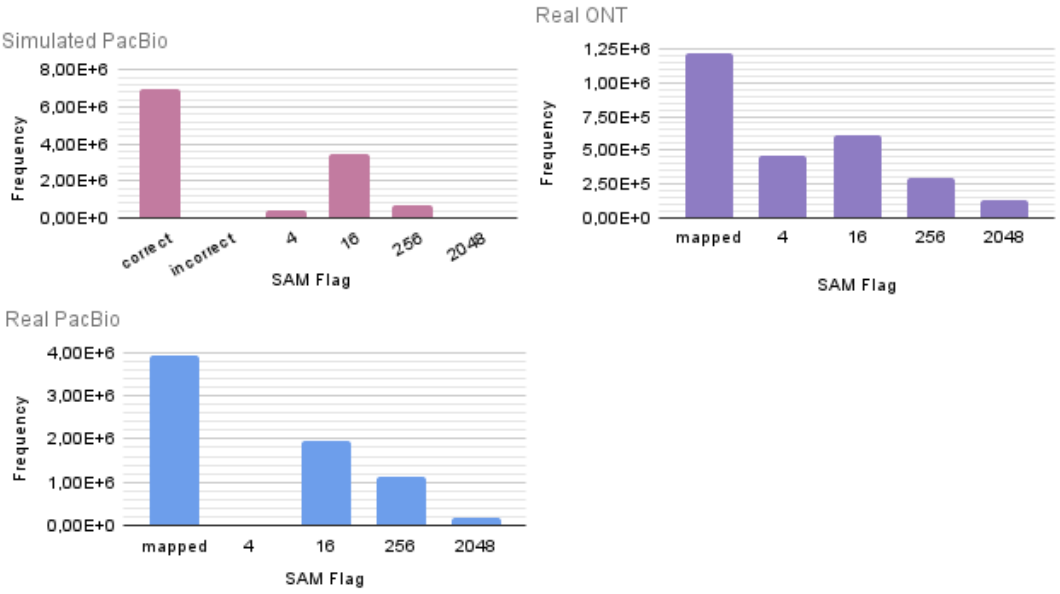

Figure 55: Map report histograms from the SAM flags

page 42), which corresponds to reads that could not be mapped to the reference with Minimap2. For the simulated data, the "mapped" column is separated into "correct" and "incorrect", since it possible to evaluate if the read was mapped to the correct reference sub-sequence, as discussed in Section 2.1.1. For simulated PacBio, real ONT and real PacBio reads, 6%, 27.5% and 0.04% of reads were not mapped, respectively.

With these analyses, it is possible to infer that Minimap2 has a good mapping accuracy and can map an acceptable portion of the real data. Further investigation is required to explain the high rate of unmapped reads in the real ONT dataset. The indel size histograms and the deviation from anti-diagonal histograms can be used to tune the accelerator's banding strategy in a way that does not harm the alignment accuracy.

5.4.2 Uniformity and Distribution of Anchors

Minimap2's original code was also edited to collect some other important internal data information, able to attest Minimap2's capability of generating uniform processing data, particularly regarding adjacent anchors. Initially, as shown in Figure 56, query and target length histograms were obtained, corresponding to the lengths of anchor-separated sub-sequences, in read and reference streams, respectively. The first three histograms refer to the sub-sequences in reads, while the other three refer to the sub-sequences in the reference.

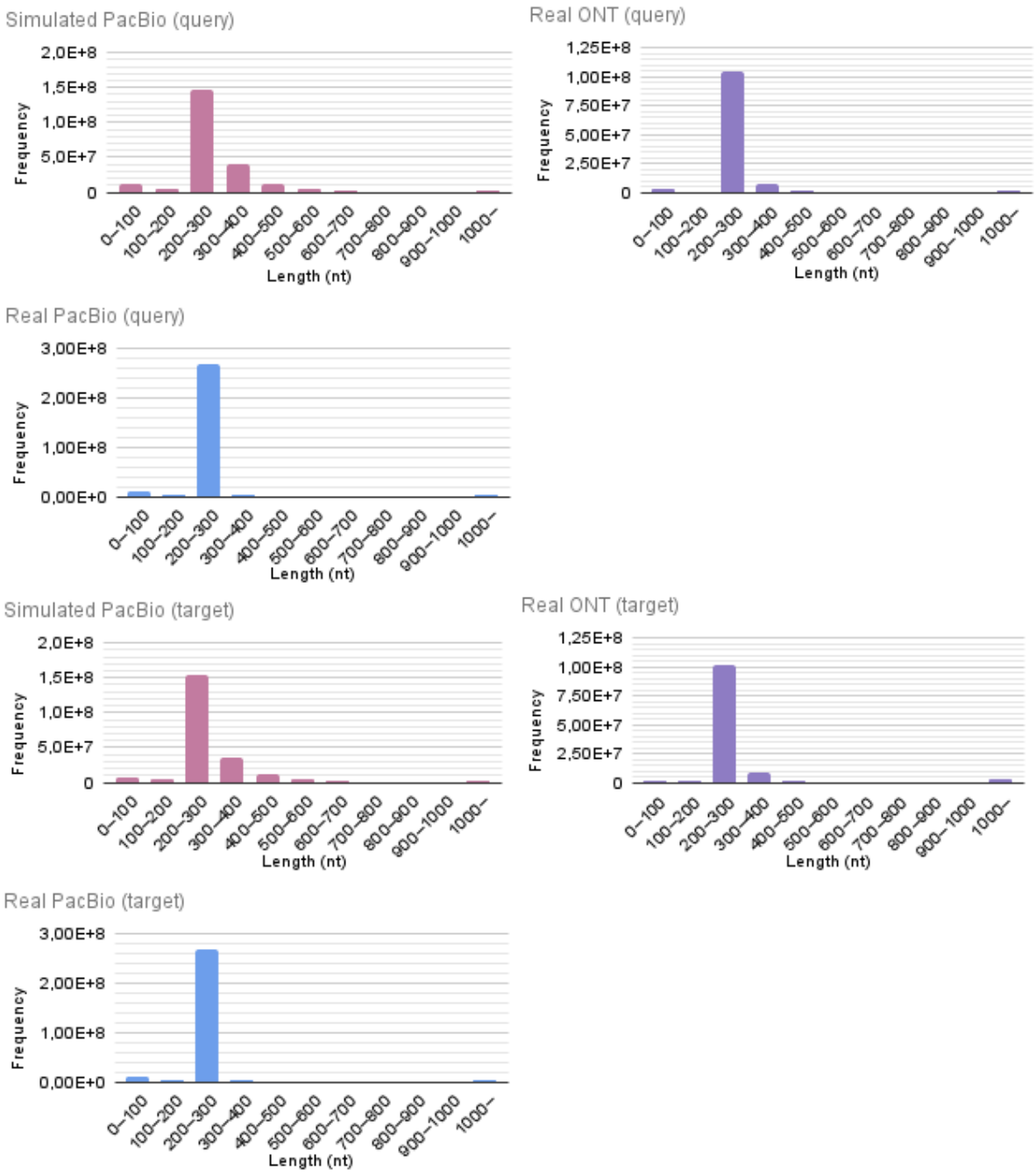

Figure 56: Anchor-separated query and target length histograms

Minimap2's intermediate data analysis for all the three datasets showed that the anchoring and chaining processes divided most of the reads into sub-sequences quite uniformly, with sizes between 200 and 300 nucleotides, as one can see in the peaks of the graphs. However, some sub-sequences were a little larger, and a few remained with lengths over 1000 nucleotides, meaning the need of larger SK matrices. As expected, the histograms for query and target for each dataset were very similar, since the chaining algorithm searches for more parallel anchors, meaning that the distances between consecutive

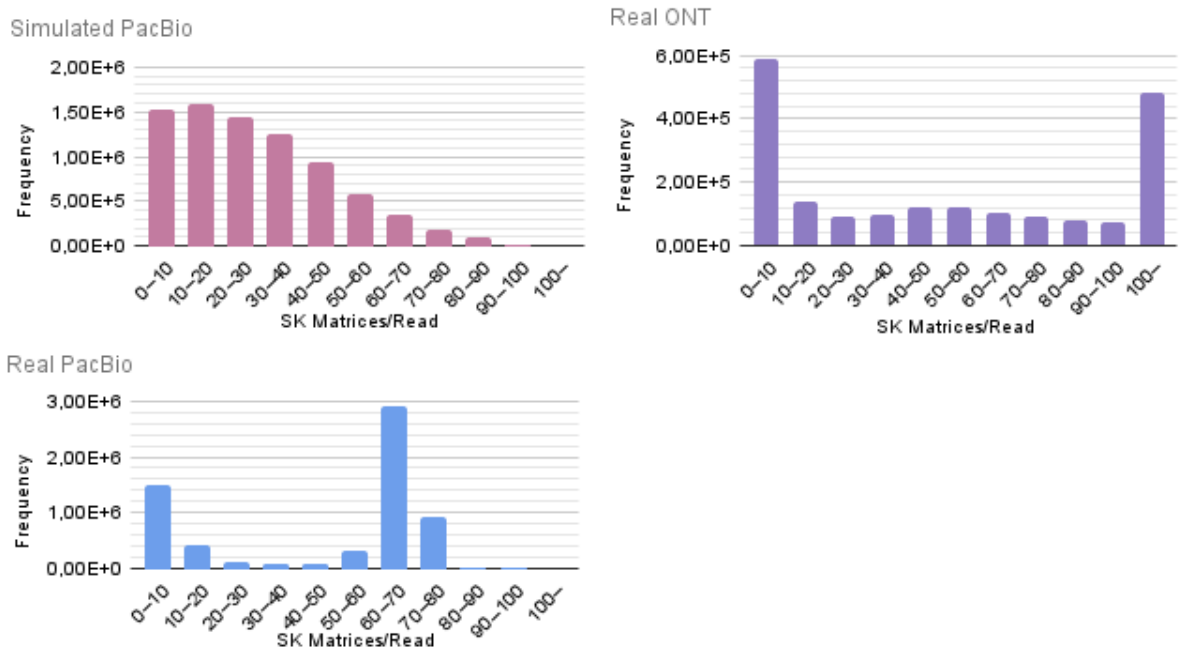

Figure 57: Histograms of number of SK matrices per read

anchors in the read and in the reference are similar.

Another form of verifying the uniformity is through the histogram with the number of SK matrices per read, i.e. the number of ksw calls to anchor-separated sub-sequences in each read, which is shown in Figure 57. The distribution seems to be different for the three datasets, but they can be better understood side-by-side with the read length histograms shown in Figure 51. The graphs for the simulated data and the real ONT data follow the curve of their read length histograms, indicating that the seed distribution was homogeneous across reads. If the sizes of anchor-separated sub-sequences are similar for most chains in a dataset, it is expected that the larger the read, the larger the number of sub-sequences. For the simulated reads, the SK Matrices/Read decreases steadily since the frequency also decreases when the read size diminishes. For the real ONT dataset, the frequency of small sized reads is high, then dropping and staying constant for varied larger sized reads; the same occurs for the SK Matrices/Read numbers. Only the real PacBio dataset did not show exactly this behavior because small SK Matrices/Read occurred frequently although there was no frequency of short reads in the dataset, indicating that some longer sub-sequences could appear.

Finally, SK deviation from anti-diagonal histogram is shown in Figure 58. The deviation works the same way as the previous deviation from anti-diagonal, shown in Figure 54, but it is for alignments between adjacent anchors. For simulated reads and real ONT reads, the deviation from the anti-diagonal in each SK matrix was more compressed to

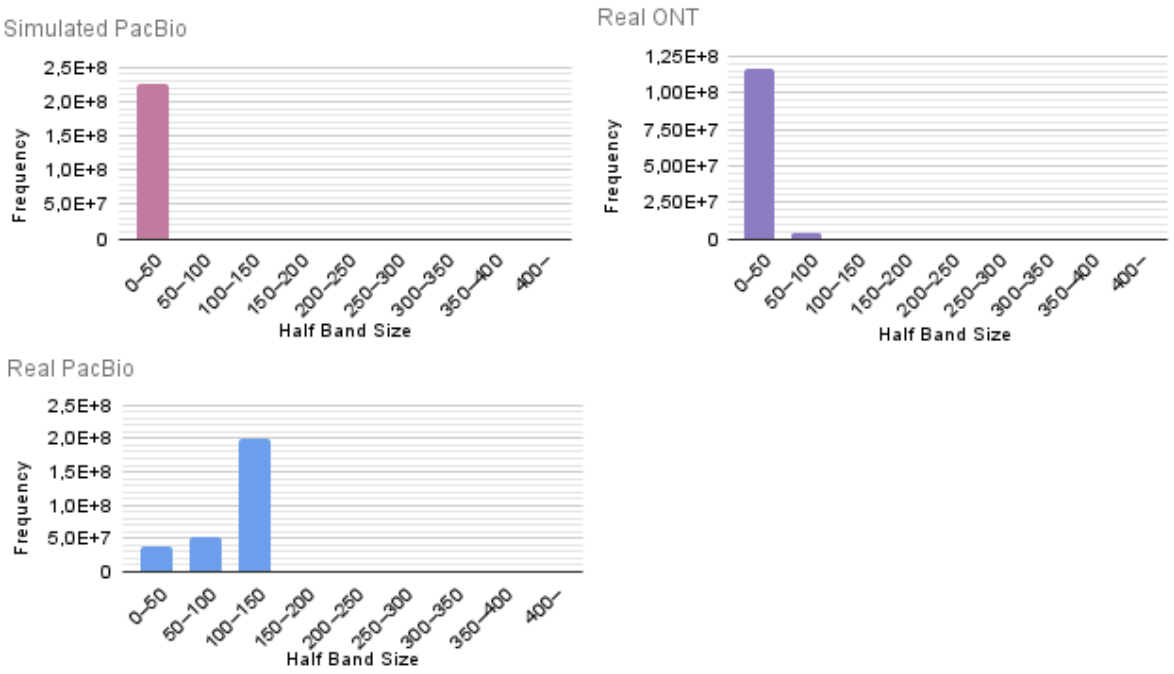

Figure 58: Histograms of deviation from the anti-diagonal in SK matrices

under 50, but for real PacBio reads, most of the matrices had a deviation of 100 to 150. This indicates that applying a fixed narrow band, such as the one described in Section 3.2, could be inefficient for some datasets in anchor-separated sub-sequences.

6 MINIMAP2 WITH AWS FPGA ACCELERATED GACT-X: ADAPTATION, INTEGRATION AND RESULTS

This chapter describes the process of adapting, installing, and integrating the GACT-X module in the Cloud AWS FPGA Instance to the software implementation of Minimap2. The module has as inputs the sequences generated in Minimap2 after the seeding and chaining steps, that are separately aligned to the reference in the ksw function, and as outputs the alignment results. The first section describes specific adaptations done to GACT-X's host and Verilog codes to improve its performance and turn it compatible with Minimap2. Then it shows an integration option like in the original Minimap2 that has anchor-separated sub-sequences, and results on performance are presented. It also presents a new adaptation to the integration method, aimed to overcome problems observed in the results in the first option; in this new version, anchor-extended sub-sequences are used from the chaining step; results in performance and accuracy are presented. A second section presents the integration process between Minimap2 and GACT-X, compatible with the multi-kernel and multi-threading capabilities, using OpenCL synchronization methods; total acceleration measurements are collected for all combinations of number of threads and number of kernels possible in the F1 Instance.

6.1 Adapting GACT-X for Better Performance

Darwin-WGA was published on GitHub (DARWIN-WGA, 2019). The entire algorithm, as well as the separated modules BSW and GACT-X, can be mirrored and implemented on AWS. Only the GACT-X module was used for this project, as mentioned in Section 4.3.

First, an AWS Instance was created with the FPGA Developer AMI (Amazon Machine Image), version 1.4.1, in region US West (Oregon). Currently, AWS in Brazil does not provide any Amazon FPGA instances. As mentioned before, the chosen instance type was

f1.2xlarge, the smallest one that has an FPGA, enough for our analysis purposes. This AWS instance is structured in its host-device environment as seen in Section 2.3.2. From Windows, Puttygen was used to access the terminal and WinSCP was used to manage files. tmux (TMUX, 2022), a terminal multiplexer for controlling terminals even if detached from a screen, was installed to run long jobs remotely. The aws-fpga library was swapped to an older version, since GACT-X was developed on the SDAccel environment and, from 2020 on, the default environment changed to Vitis. A bucket was created to keep the Amazon FPGA Images. The VIVADO tool version was set up to version 2017.4.

Part of software in OpenCL for GACT-X at the host had to be adapted for this implementation. Since Darwin-WGA is a whole genome aligner, originally it was developed to send the two entire genomes to be aligned to the DDR, looping the pairs of positions to be extended from them. For read mapping, as performed in Minimap2, every pair of reference sub-sequence and read is streamed to the DDR to get the extending results. The process of creating and managing tiles was not implemented in the original GACT-X host, and had to be added as well.

Figure 59 shows the fluxogram for transferring data to the FPGA with the tile algorithm. The inputs correspond to many pairs of read and reference sub-sequences, which were written in a separate file after the seeding and chaining steps in Minimap2 (left of the figure). In the example, the loop cycles 5 times, since there were 5 tiles in total. The sequence of steps for the software-hardware interaction follows:

- Each pair is transferred to the FPGA into the ref_seq and query_seq buffers (see item **1)** in the figure);
- A loop sets the tile's starting index with variables ref_offset and query_offset, and the tile's size with variables ref_len and query_len (see item **2)** in the figure);
- The FPGA kernel or kernels calculate the alignment for the tile and write in h_tile_output the alignment score, max score index with variables ref_pos and query_pos, and the number of traceback pointers in the band; the traceback pointers are written in h_tb_output.

The alignment parameters were changed to have about the same proportions as the default in Minimap2 (Table 6). The match, mismatch, and gap scores are 2, -4, -4, -2 for Minimap2, so they were set as 10, -20, -30, -10 for the adapted GACT-X. The difference in the gap opening score is due to different interpretations on whether an extension occurs when opening a gap, but the resulting score is the same. GACT-X

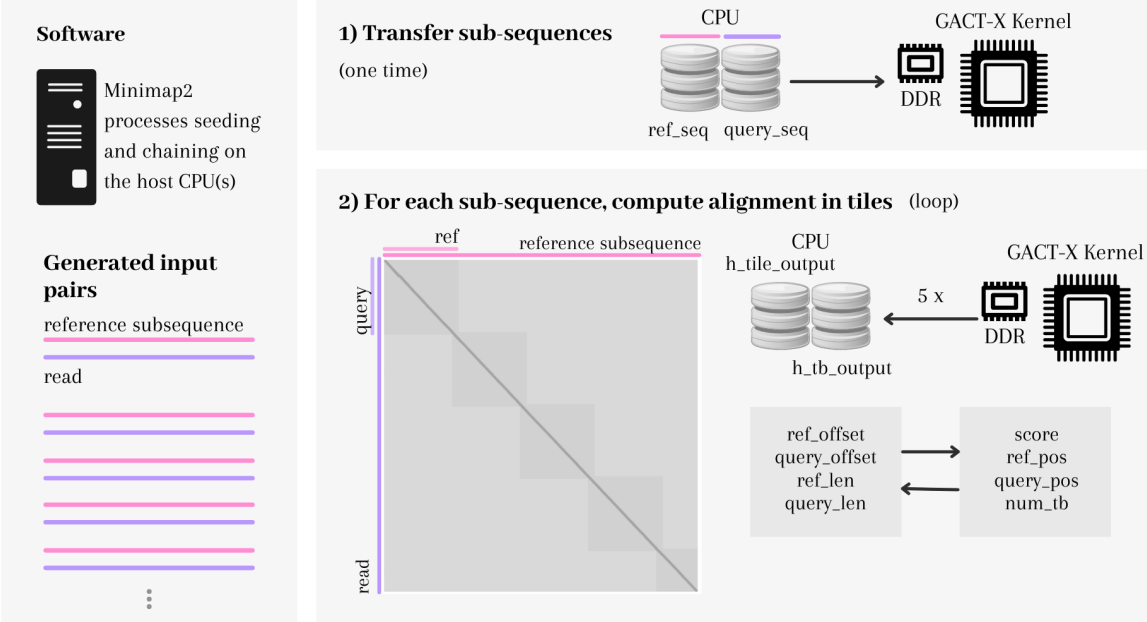

Figure 59: Example of GACT-X's host-FPGA data transfer fluxogram

Table 6: Minimap2 and GACT-X's alignment parameters

		match	mismatch	1st function		2nd function		Y-drop
				gap open	gap extend	gap open	gap extend	
Minimap2		2	-4	-4	-2	-24	-1	-
GACT-X	**original**	91 to 100	-31 to -125	-430	-31	-	-	9530
	adapted	10	-20	-30	-10	-	-	943

originally uses substitution matrices for match and mismatch scores, where different pairs of nucleotides result in different scores. GACT-X only supports one gap function; only the Minimap2's affine gap function with higher slope and lower translocation was used in GACT-X because it picks the more frequent short indels. The Y-drop value was adjusted according to GACT-X's new gap function, allowing the presence of a same size of gap in the band as originally allowed.

6.1.1 GACT-X with Anchor-Separated Sub-Sequences from Minimap2

The initial strategy to accelerate the extending step was to send, individually, every anchor-separated sub-sequence that lay between two anchors to be aligned in the FPGA, using global alignment. This is the strategy used originally by Minimap2, that at the same time reduces sequence lengths and avails the matching information produced by the chaining process.

The sequence of steps in the host for the software-hardware interaction described in OpenCL (see commands in Section 2.3.3) is:

- The input files are uploaded to the AWS Instance;
- The host creates input and output buffers in the DDR with sufficient space for any input length in the dataset; for that, the clCreateBuffer command is used;
- The pairs of anchor-separated sub-sequences are collected from the input files, and sent to the buffers in the FPGA through the clEnqueueWriteBuffer command;
- The arguments to the compute kernel (e.g. tile beginning and end positions, alignment scores, and buffers) are set through the clSetKernelArg command;
- The kernel is executed through the clEnqueueTask command;
- The output buffers are read back in through the clEnqueueMapBuffer command and the results are processed to recover the alignment from the direction pointers (traceback);
- The alignments for each pair of sequences are written into an output file.

The simulated PacBio dataset was adopted for this testing phase, and only its first 100,000 reads were used, due to the high execution time. The real ONT and PacBio datasets are not used in this development stage, but the final design's performance was measured on them and will be presented later. The datasets contain some millions of input reads each, but it is reasonable to consider that the sequencing technologies produce uniformly distributed read sizes, therefore, 100,000 samples can be considered representative of the whole dataset.

The query and target anchor-separated sub-sequences that are aligned in Minimap2, generated after the chaining step, were collected. The input length histograms have been presented in Fig. 56 for all three datasets, showing the anchor-separated sub-sequence sizes concentrated between 200 and 400 bases.

This Minimap2-GACT-X implementation showed performance problems, as they can be seen in Figure 60, which presents average execution times in using Minimap2's ksw function and a GACT-X kernel for the alignment step. The results for the simulated PacBio dataset showed that GACT-X performed considerably worse than ksw from Minimap2 in this design, that mostly aligns short sequences (although originated from long-reads). The performances for ksw and GACT-X are similar only for sequences larger than

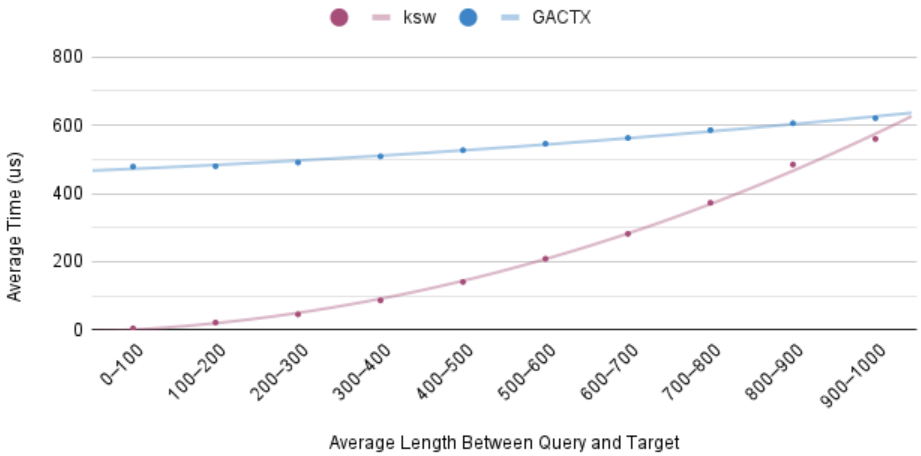

Figure 60: Average processing times per length for anchor-separated sub-sequences

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

1,000 bases. This is because GACT-X's processing time starts with a shifted constant that corresponds to the data transfer time from host to FPGA and back (see item **1)** in Figure 59), for every anchor-separated sub-sequence, creating a heavy transfer latency that undermines the gains obtained with a faster alignment performed in the kernel. For the real ONT an PacBio datasets, the transfer latency is expected to be similar, since the average lengths of sub-sequences, shown in the histograms of Figure 56, are highly concentrated in the 200-300 bases range.

6.1.2 GACT-X with Anchor-Extended Sub-sequences from Minimap2

For a more efficient solution, instead of aligning each anchor-separated sub-sequence, the strategy changed to considering, in each read, the anchor-extended sub-sequences, i.e., the portion from the first anchor to the end of the sequence and its corresponding reference sub-sequence, expanding the alignment in a semi-global fashion. The software-hardware interaction follows the fluxogram of Figure 59, however the item **1)** is altered to a reduced number of transfers of longer sub-sequences.

The experiments were also performed on the first 100,000 reads from the simulated PacBio dataset. In order to confirm the effects of the new transfer, Figure 61 was generated. It shows that, for the simulated PacBio dataset, the new transferred data have lengths to be aligned increased in one to two orders of magnitude. As expected, the histogram follows the shape of the one presented in Fig. 51 for the reads, therefore, it is expected that the real ONT and PacBio datasets also present increased lengths in their

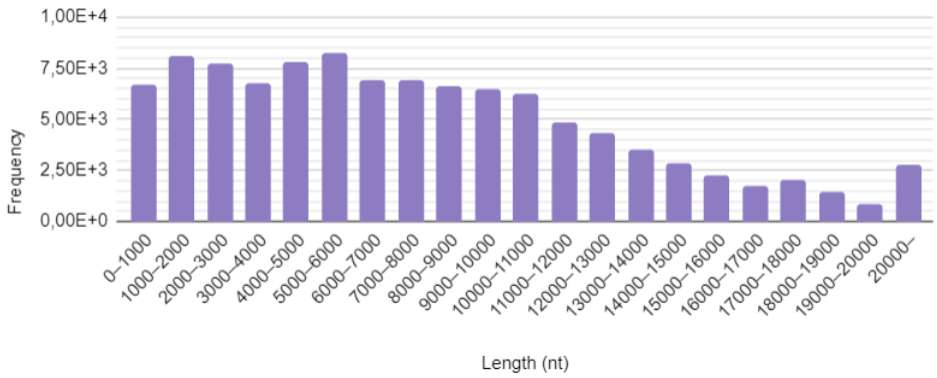

Figure 61: Anchor-extended query-target average length histogram

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

transferred sub-sequences.

By increasing the lengths of the transferred sub-sequences, the tile processing had to be activated, and the heuristics described in Section 4.3 had to be implemented due to limitations in hardware resources. In the original Verilog files, used to build the binary code of the kernels, the tile size is set up to be at most $2,048$, determined by wires in internal modules of size $log_2(2,048) = 11$. If the host forces an alignment between longer sequences, either the result is going to turn out incorrect, or the execution will potentially freeze, requiring forced interrupt, cleaning and reloading the AFI into the FPGA.

Each tile requires a new data transfer cycle, as at item **2)** in Figure 59, so it was expected that, as tile size increases, fewer tiles would be required in each expansion, and better the overall performance would become. In order to test the behavior of the hardware-software implementation, its performance was measured under different tile sizes. The max tile size value in the Verilog files was changed to $8,192$ and a new AFI was generated.

The performances for the new strategy were measured with varying tile sizes and are shown in Figures 62 and 63. The first one presents the execution times with respect to the many ksw calls in software along with the ones of GACT-X, for different sub-sequence sizes. The second one refers to the total execution time for the ksw function in Minimap2 and for GACT-X.

Regarding Figure 62, as expected, GACT-X's performance improved as tile sizes increased from 1,000 to peak at 4,000, considering the same size of the input data. However, after that, the performance started worsening, and it is suspected that it is due to BRAM saturation, since direction pointer storage is linearly cumulative and increases with tile

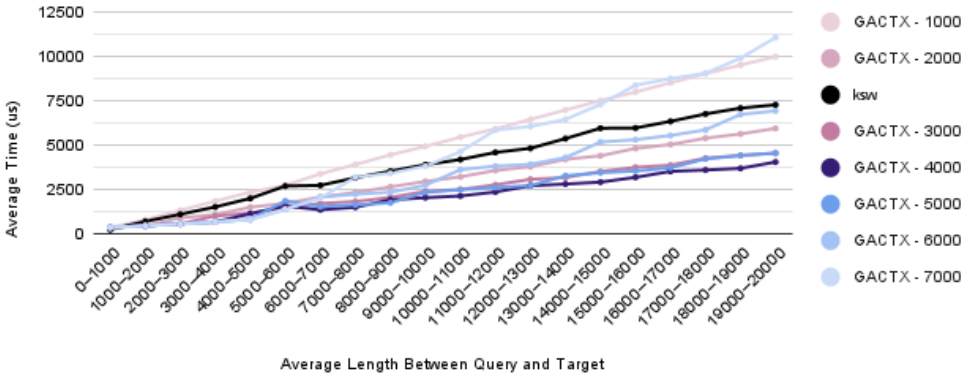

Figure 62: Average processing times per length for anchor-extended sub-sequences

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

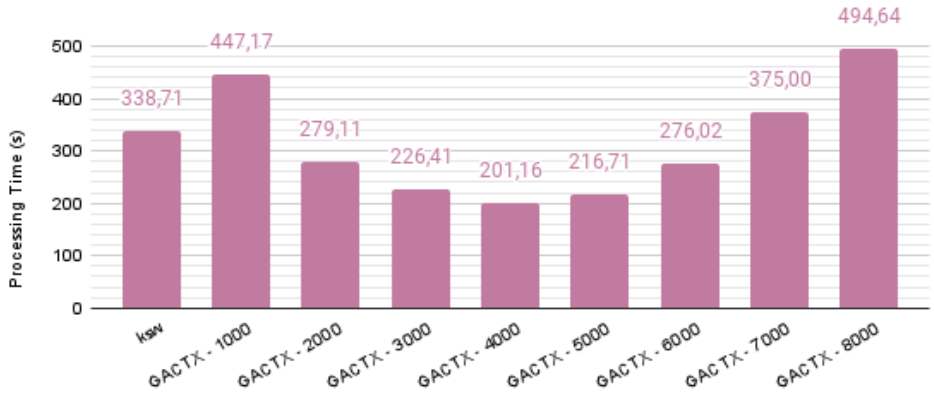

Figure 63: Total processing times for ksw and GACT-X

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

size. Still, it was already possible to observe an acceleration compared to ksw's performance. As for the real ONT and PacBio datasets, it is expected also a steady increase of performance for larger tiles, since the transfer time will be similarly reduced; due to the BRAM limitation, tile of size 4,000 should be the one with highest performance improvement too.

Figure 63 shows that GACT-X with tile size of 4,000 achieved the largest speed-up of 1.68x compared to Minimap2's ksw software execution time. Other changes can further improve this number, and are discussed later in this section and in Section 7.

GACT-X's accuracy was also evaluated against Minimap2's ksw function in a range of tile sizes. Figure 64 shows two sets of results: a) from the alignments, it was possible to compare the alignment scores obtained from the Suzuki-Kasahara algorithm, coupled with chaining divisions, with the alignment scores obtained from the GACT-X algorithm;

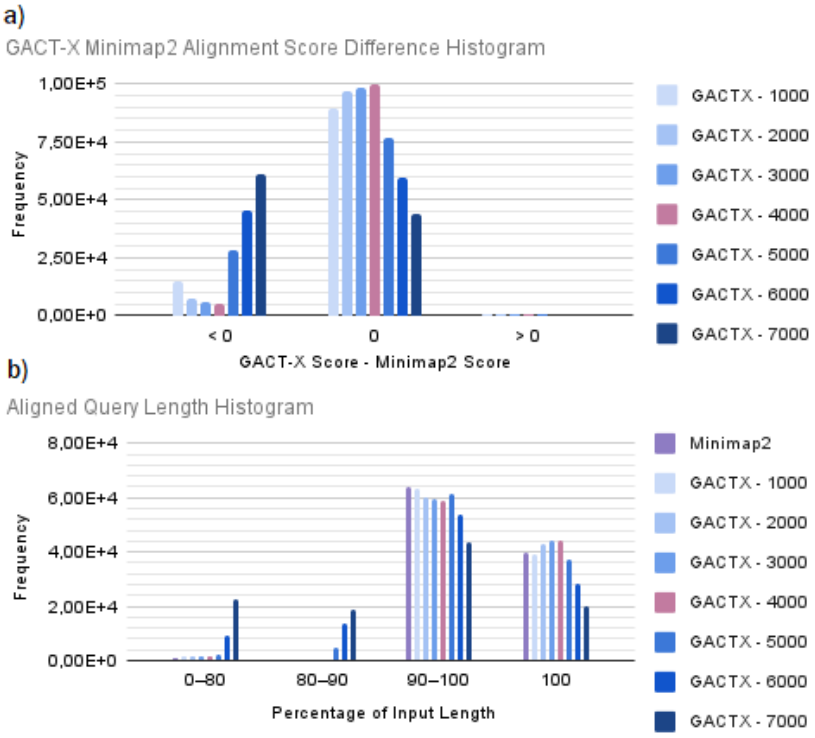

Figure 64: Histograms of GACT-X and Minimap2's score and query percentage differences

The first 100,000 simulated PacBio long-reads were aligned with Minimap2 to the GRCh38 genome reference.

b) the percentage of aligned inputs of query sequences.

The first histogram shows the proportion of alignments where GACT-X had a worse (< 0), equal (0), and better (> 0) score in relation to ksw from Minimap2, for varying tile sizes. 95.42% of alignments found with GACT-X with tile $4,000$ had the exact same score as the ones produced in Minimap2, even with chaining information hidden from GACT-X. It means that GACT-X with tile $4,000$ can provide a accuracy at the same level of the ksw algorithm. The cases with tiles of 5,000 bases or more show lower accuracy and also are the cases where the performance is low, as seen in Figures 62 and 63. Again, it is believed that BRAM saturation causes some data dis-alignment, leading to wrong results. The small percentage of alignments' lower scores may be explained by these factors: the tile heuristic adds uncertainty in the borders of each tile; GACT-X does not support a second gap function, so frequently loses longer gaps that should be expected to appear.

The second histogram corresponds to the percentage of query sub-sequences that were used in each alignment in Minimap2's ksw (with global alignment with anchor-separated sub-sequences and semi-global alignment after the last anchor) and in each alignment for GACT-X with different tile sizes (considering a semi-global alignment from the first anchor

to the end of the sequences). When an input sequence is aligned, it may happen that the max score, which is where the backtrack starts in semi-global alignment (see Section 4.1.2), does not occur in the last (right-down) cell, and only part of the sequence is used. Ideally, the query sequence should be aligned entirely to reduce waste. By the histogram of Figure 64, 60.97% of Minimap2's tracked alignments consumed between 90% and 100% of the corresponding sequence, and 37.62% consumed the entire query sequence. For GACT-X with tile $4,000$, the percentages were 55.21% and 43.04% respectively, indicating a better usage. The accuracy for tiles from $5,000$ on decreased steeply, which could also be linked to BRAM saturation.

Given that the tests with tile size of 4,000 showed the best performances in both accuracy and speed for the simulated data, the same measurements were made with the two real datasets with this same configuration, and are presented in Tables 7 and 8.

Table 7 shows that GACT-X's accuracy was very similar to Minimap2's on the simulated PacBio data, considerably worse on the real ONT data (41.21% of the alignments produced had lower scores), and dissimilar for the real PacBio data, presenting high rate of both worse (14.90%) and better (18.28%) alignments. Table 8 reinforces GACT-X's similar accuracy performances for the PacBio datasets, with similar ratio of aligned query lengths, but worse for the real ONT dataset, with lower ratio of aligned query lengths. With these results and the alignment reports presented in Figure 55, it is possible to induce that the real ONT dataset has a significant disparity to the genome reference used, which could be affecting the mapping and alignment accuracies of both of the algorithms.

The partial results obtained to this point have provided a strong indication that it is possible to accelerate Minimap2's extending step, even with SSE optimization, by using a software plus hardware hybrid architecture. GACT-X's tile heuristic allows alignment be-

Table 7: Minimap2 and GACT-X's alignment score differences for three datasets

GACT-X Minimap2 Score Difference	< 0	**0**	> 0
simulated PacBio	3.90%	95.42%	0.68%
real ONT	41.21%	47.79%	11.00%
real PacBio	14.90%	66.83%	18.28%

Table 8: Minimap2 and GACT-X's aligned percentage of query sequences for three datasets

	Percentage Aligned	**0-80**	**80-90**	**90-100**	**100**
simulated PacBio	**Minimap2's ksw**	1.06%	0.22%	60.97%	37.62%
	GACT-X 4,000	1.52%	0.23%	55.21%	43.04%
real ONT	**Minimap2's ksw**	21.56%	2.59%	73.57%	2.29%
	GACT-X 4,000	36.24%	3.64%	58.47%	1.64%
real PacBio	**Minimap2's ksw**	18.42%	0.76%	13.15%	67.67%
	GACT-X 4,000	12.01%	2.71%	19.59%	65.69%

tween arbitrarily long sequences, which was an issue that long-read mapping faced on limited FPGA resources. GACT-X's Y-drop heuristic mimics Minimap2's fixed bandwidth heuristic, both decreasing memory and time complexities of the SWG algorithm from quadratic to linear. GACT-X's accuracy remained satisfactory compared to Minimap2's (considering alignment scores) and GACT-X was able to align a similar percentage of the read sequences on the PacBio datasets. GACT-X's accuracy is aggravated in comparison to Minimap2 on the real ONT dataset that seems to have lower similarity to the genome reference.

Satisfactory acceleration and accuracy has occurred as far as the tile size is not larger than 4,000 bases, limitation imposed by the allocated BRAM size for one kernel. The use of larger tiles may be desirable since it could further improve the performance, likely maintaining the accuracy. With the FPGA device in the AWS F1 Instance, it is possible to double the tile size, since just less than half of the BRAM available is consumed by one GACT-X kernel. This ratio is explored in the next section by implementing 2 parallel GACT-X modules in the same device. Bigger tiles can be accomplished with the use of more expensive FPGA platforms with larger BRAM resources.

6.2 Integrating GACT-X into Minimap2 with Multi-kernel and Multi-threading

With the results obtained in Section 6.1, GACT-X was integrated to Minimap2 on the host, and a series of experiments were made for the software-hardware hybrid implementation. The tile size was set to 4,000 following the results in Section 6.1.2. The hybrid system accommodates the multi-threading and multi-kernel capabilities of the system.

In order to set the host, a new C++ file "gactx.cpp" was created and added to Minimap2's source to accommodate the hardware's host lines. It contains the functions to configure and load the binary file, initiate the acceleration context, align with GACT-X, and clean the system. Detailed description of the OpenCL code implementation can be found in Appendix A.

Minimap2's original implementation allows multi-threading. Each thread performs the sequential processing of seeding, chaining, and aligning one read at a time (process shown in Figures 43, 44 and 46); threads can be distributed to cores in the host server for concurrent processing. OpenCL supports multi-kernels in the same FPGA, as many as the resources can fit.

Generally speaking, any thread could be assigned to any kernel in order to optimize concurrent processing. However, in Minimap2, only a limited form of concurrency in using the kernels related to GACT-X is possible, because of the sequential aspect mentioned in the previous paragraph:

- The number of threads need to correspond to at least the number of kernels, otherwise the excess kernels will be idle;
- A kernel cannot carry the execution of different threads interchangeably.

The restrictions above require that, whenever there are more threads than kernels running, and the corresponding sequences generated by the chaining steps are ready for alignment processing, a waiting list must be used in order to access an available/free kernel. On the other hand, in situations of a single thread run (single core), whatever the number of kernels, only one may be used.

The AWS f1.2xlarge Instance allows the implementation with up to 64 kernels and 8 cores, as explained in Section 2.3.2. This project was tested with up to 2 GACT-X kernels and 8 Minimap2 cores, due to the number of BRAMs available in the FPGA device, that

could only support 2 GACT-X modules with tiles of size 4,000. This test size is enough to give a good insight on the problem, as it will be seen later in this section, and there will be discussion on how to use it with a larger number of modules.

The analyses were made with the 100,000 first reads from the simulated PacBio, and the real ONT and PacBio datasets. Two types of run were executed for time measurements. In both of them, the measurements were made with all combinations of 1 to 8 software (sw) threads and 1 to 2 kernels.

The first run is for the comparison between the hybrid Minimap2 with GACT-X implementation, and Minimap2 in software with ksw-based alignment. This run was to measure the total execution time, displayed by Minimap2 in the command prompt at the end of the execution; regarding the GACT-X kernel, command queue was used to stream commands to the FPGA and, right away, execution proceeded to execute the next line without waiting for the command to finish, except in synchronization points. The results are presented in Table 9.

In the table, all three datasets, as seen in column 1, are processed in the three implementations for Minimap2 in software, Minimap2 with GACT-X implemented in 1 kernel, and Minimap2 with GACT-X implemented in 2 kernels, as shown in column 2. Results are shown for increasing number of threads, which were tried up to 8, as listed in columns 4 to 11. The total execution times in seconds were measured for each test instance. Two types of acceleration were computed: a) the acceleration obtained for each implementation with respect to the software, for the same number of threads (set for each column); b) the integrated system's thread acceleration, which is the acceleration for increasing the num-

Table 9: Execution times and acceleration in the Minimap2-GACT-X integrated system

	Number of threads (sw)		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
simulated PacBio	**software**	**total execution (s)**	759.157	390.198	266.926	206.255	197.863	186.727	177.508	169.927
	1 kernel	**total execution (s)**	539.56	299.76	233.88	208.93	203.74	253.84	409.03	453.26
		acceleration	1.41	1.30	1.14	0.99	0.97	0.74	0.43	0.37
		thread acceleration	1.00	1.80	2.31	2.58	2.65	2.13	1.32	1.19
	2 kernels	**total execution (s)**	538.996	289.557	211.837	198.700	203.366	228.516	273.066	313.150
		acceleration	1.41	1.35	1.26	1.04	0.97	0.82	0.65	0.54
		thread acceleration	1.00	1.86	2.54	2.71	2.65	2.36	1.97	1.72
real ONT	**software**	**total execution (s)**	3,105.50	1,559.59	1,056.77	806.37	774.30	719.29	684.98	646.35
	1 kernel	**total execution (s)**	2,243.69	1,168.48	837.14	712.91	653.68	660.44	937.68	1,258.13
		acceleration	1.38	1.33	1.26	1.13	1.18	1.09	0.73	0.51
		thread acceleration	1.00	1.92	2.68	3.15	3.43	3.40	2.39	1.78
	2 kernels	**total execution (s)**	2,197.265	1,126.070	794.728	638.109	586.025	576.393	672.990	858.846
		acceleration	1.41	1.38	1.33	1.26	1.32	1.25	1.02	0.75
		thread acceleration	1.00	1.95	2.76	3.44	3.75	3.81	3.26	2.56
real PacBio	**software**	**total execution (s)**	4,255.21	2,132.71	1,431.38	1,083.29	1,037.82	965.57	904.33	858.31
	1 kernel	**total execution (s)**	3,570.25	1,813.39	1,243.76	1,003.40	919.39	889.68	1,077.63	1,575.97
		acceleration	1.19	1.18	1.15	1.08	1.13	1.09	0.84	0.54
		thread acceleration	1.00	1.97	2.87	3.56	3.88	4.01	3.31	2.27
	2 kernels	**total execution (s)**	3,470.799	1,765.341	1,239.144	994.346	885.107	831.761	855.454	1,117.282
		acceleration	1.23	1.21	1.16	1.09	1.17	1.16	1.06	0.77
		thread acceleration	1.00	1.97	2.80	3.49	3.92	4.17	4.06	3.11

ber of threads, with respect to the single threaded execution, for the software-hardware integrated implementation.

The execution times of the software-only implementation for all datasets have shown a steady decrease for increasing number of threads, as it is expected (it can be seen in line 2, for example, for the simulated PacBio dataset); the relation is not linear due to the increase in management complexity for larger number of threads. The execution times for the integrated software-hardware systems also decreased in the initial addition of threads, as expected, but stabilized and increased again for about 5 or 6 threads or more, whatever the number of kernels, reflecting in reduced thread acceleration (see, for example, lines 3 and 6, for simulated PacBio dataset, and can be observed clearly through the figures on thread acceleration in lines 5 and 8).

Another issue is that data with 7 and 8 threads had their execution times exploded, being inconsistent with the change rate for the other measured times, and making them unreliable for analysis (therefore, they will not be considered in later discussions); considering there are 8 available cores for processing, probably two of them are used for system management and conflicts with the GACT-X's host lines.

Although the integrated software-hardware systems show decreased time execution with increased number of threads, that occurs in a lower rate than for the full software implementation. This can be seen through the acceleration numbers in lines 3, 6, 10, 13, 17 and 20. These results show that, for the integrated implementations built in this work, the higher number of threads do not help much. In fact, a more detailed study on the bottlenecks must be provided, what is done in next paragraphs.

The table shows that the use of 2 kernels improved the execution time if compared to the single kernel case, as expected, but this trend changes after 5 threads. Besides that, the reduced time with 3 or more threads was limited, with the maximum close to 10%. Also, as commented before, for 1 or 2 threads, the availability of 2 kernels does not help. It is expected that, with more available kernels, the workload from a larger number of threads can be alleviated, with better acceleration, but that would require increased hardware resources and costs.

The highest achieved acceleration therefore was of 1.41x with 1 thread for simulated PacBio with 1 and 2 kernels, and real ONT with 2 kernels, and second highest was 1.35x for simulated PacBio with 1 thread and 1 kernel, and real ONT with 2 threads and 2 kernels. It is worth to mention that this study with GACT-X's acceleration only refers to the alignment phase in Minimap2; the chaining step may also be hardware accelerated

(GUO et al., 2019), improving the total performance of Minimap2.

Although the results in Table 9 showed some expected trends, several expectations were not accomplished:

- The acceleration decreased as threads outnumbered kernels, performing worse than the software counterpart when the gap is too big;
- Acceleration with 2 kernels was only slightly better than with one kernel, when a speed-up of 2x might be expected.

The suspicion for the above behavior is the performance limitation due to competition of the PCIe transfer channel. In order to verify it, a second run of experiments were made; time measurements were taken with different tools for each situation, in order to deal with the actions in separate. Every kernel processing, data transfer and waiting in line instance has had the time measured and accumulated, respectively. These instances in each thread were measured independent of any other of Minimap2's execution that could occur concurrently. Therefore, the total accumulated time cannot be directly correlated to the total execution time, shown in Table 9, except for the case of a single thread, which is totally sequential.

The kernel processing times were measured using the function "clGetEventProfiling-Info" and START and END event flags. The data transfer times, which includes the times to transfer sequences and tile information, as can be seen by the items 1 and 2, respectively, in Figure 59, are measured similarly. The time spent by the threads waiting in line for a kernel was measured using "clock_t". Every writing, reading, and kernel event had to be completed for the execution of the code to proceed to the next line by setting the "blocking_write" argument as CL_TRUE, or by using "clWaitForEvents", so that the measurements could be taken.

The results are displayed in Table 10. For each of the three datasets, shown in column 1, the two integrated implementations, with 1 and 2 kernels, were considered, as displayed in column 2. For each implementation, the case of kernel processing, data transfer and in line times are shown in column 3, referring to different thread numbers, from 1 to 8, as indicated in columns 4 to 11.

Considering the kernel processing time reflects the computation of the total amount of the inputs, which is constant throughout the different kernel implementations, the results have shown to be as expected: for all datasets, it remained quite constant comparing 1 kernel with 2 kernels cases. Also, considering any particular implementation for a dataset, the

Table 10: Processing, transferring, and waiting times in the Minimap2-GACT-X system

	Number of threads (sw)		1	2	3	4	5	6	7	8
simulated PacBio	1 kernel time (s)	processing	94.14	94.62	95.16	96.16	96.71	98.11	132.26	150.40
		data transfer	26.90	28.81	32.07	33.24	34.13	37.59	152.30	150.51
		time in line	0.21	83.11	318.66	845.77	1,789.56	3,377.48	13,990.11	21,404.43
	2 kernels time (s)	processing	94.20	94.99	95.43	95.89	96.49	110.83	129.61	141.75
		data transfer	27.00	34.77	37.87	37.25	47.99	146.70	165.48	166.65
		time in line	0.15	0.24	44.49	186.04	619.47	3,813.98	7,592.47	12,094.00
real ONT	1 kernel time (s)	processing	322.29	323.02	324.33	326.42	327.82	331.18	388.75	476.91
		data transfer	12.20	48.54	51.66	53.72	59.88	75.27	361.88	503.51
		time in line	0.15	148.91	544.15	1,310.52	2,548.16	4,926.27	24,567.76	50,482.29
	2 kernels time (s)	processing	322.66	324.16	324.98	326.04	327.69	346.86	376.96	420.51
		data transfer	42.80	53.15	61.25	60.11	86.25	227.04	347.46	476.46
		time in line	0.24	0.27	54.48	242.43	802.66	4,210.88	11,777.99	25,100.46
real PacBio	1 kernel time (s)	processing	255.12	255.52	257.62	259.56	261.43	264.64	324.07	465.84
		data transfer	53.28	57.38	61.37	65.27	74.19	91.04	383.26	723.37
		time in line	0.20	63.13	219.84	499.16	962.39	1,909.10	16,175.72	53,765.80
	2 kernels time (s)	processing	255.79	257.62	258.97	259.90	261.93	284.62	333.82	388.27
		data transfer	53.64	64.29	70.02	70.59	91.89	275.96	452.71	683.43
		time in line	0.10	0.41	15.78	68.70	228.42	2,063.87	8,240.65	23,054.56

kernel processing times remained relatively constant among different number of threads. With 6 or more threads, an increasing deviation occurs; since the OpenCL's clEnqueue-Task command is a macro, probably the management of large numbers of threads start to affect the measurements of its time span.

For all datasets, the data transfer time increased with the number of threads (for a fixed implementation), and with the number of kernels (for a fixed number of threads); since the measurement is made on OpenCL commands, which involves transfer channel liberation, probably a channel access latency component was included.

With respect to the time in line, the consistency for all datasets can be observed in the table, by spotting the single thread case, which has a sequential nature; independent of the implementation (1 or 2 kernels), there will always be a kernel available and the observed waiting time is close to zero. In the case of 2 threads, for a 2 kernels implementation, the same occurs; however, for a 1 kernel implementation, a second thread has to wait for the kernel to be liberated from the computation of a first thread, therefore, the waiting time in line is larger, for example, 83.11 seconds for the simulated PacBio dataset.

Another observation regarding the time in line is that, for all datasets and implementations, it increases with the number of threads. That indicates that the kernels are getting more occupied and less available to take new jobs and clear the waiting queue. The management complexity for the threads, what affects the data dispatching time, is specially critic for 7 or 8 threads, probably due to the superposition with the operating system execution. This time in line effect has a strong impact on the total execution time, leading to the figures observed in Table 9.

The increase in time in line has indicated that 1 or 2 kernels were not always available for the threads, becoming the bottleneck of the integrated system. Besides that, not all availability of a second kernel is occupied, indicating a dispute in the PCIe interface. Still, the waiting time for the 2 kernels implementation, for all datasets, showed to be, for the varied number of threads, significantly shorter than the corresponding ones in the 1 kernel implementation. For example, comparison can be made between lines 4 and 7, 10 and 13, 16 and 19. This was expected since for 2 kernels implementation, chances of kernel availability is increased; this suggests that a larger number of kernels (therefore, more hardware resources) can bring better results as the number of threads also increases.

What the results tell about optimization:

- As expected, according to (WANG et al., 2020), PCIe turns out to be a bottleneck if the transfer rate is high. The traffic was reduced in this research by manipulating the data in the host, basically using anchor-extended sub-sequences instead of anchor-separated ones;
- Another improvement option would be addressing twice the number of BRAMs to a kernel, allowing the tile size to be twice as large, which would further reduce the transfer rate; this would allow fitting only one kernel in the device, but would also eliminate the PCIe channel conflict, and could potentially achieve a better performance than 2 parallel kernels in a device;
- The number of kernels should correspond to the number of software threads for the integrated system's best performance; if multi-threading is wished, a corresponding increment on hardware resources must accompany it;
- Further improvement, including hardware changes, is adding a traceback logic to the kernel (changing the Verilog implementation), which would reduce the data transfer sizes.

7 CONCLUSION AND FUTURE WORK

This project started with the identification of genome sequencing data as tending to migrate to longer reads (seen in the third-generation of sequencers), as they present several crucial advantages over short-reads from the previous generations. The computing stage has been identified as being the genome analysis pipeline's bottleneck, mainly because of the advances of highly parallelized sequencing of data which provides a great amount of genomic data to be processed, and because of the slow-down of computing power advances. One option for resolving this bottleneck is the development of DSAs, that use advantages of alternative architectures to target specific domains.

Minimap2 was identified as the State-of-the-Art algorithm for assembly of long-reads, and some works were already made on several types of processors, such as GPUs, FPGAs, and KNL to accelerated one of its main bottlenecks, the chaining or the aligning stages. No work that has successfully accelerated Minimap2's aligning step on an FPGA has been identified on the literature, which culminates on the proposition of this project.

GACT-X, an FPGA Cloud design for the SWG algorithm, was considered to be a good option for accelerating Minimap2's aligning step running on long-reads because of its limited memory consumption characteristic, which solves banded SWG's inherent linear complexity to the inputs' lengths.

Through the development of this work, the following conclusions could be drawn:

- There are genome data and tools available in public repositories to be used. Two sets of real human genome long-reads with high coverage from PacBio and ONT sequencing technologies were obtained from the NCBI and ENA databases. Another set of reads was generated with the PBSIM tool, used to simulate PacBio long-reads. These datasets were considered to be a reasonable sample for the experiments done in this research. The datasets presented different read length distributions. Simulated reads concentrated on shorter to medium lengths with average of 8,300 nucleotides; real PacBio reads concentrated on medium lengths with average of

13,300; and real ONT reads had a homogeneous distribution of lengths up to 100,000 nucleotides, with average of 16,900;

- An analysis of the Minimap2 algorithm running the datasets was important to understand its dynamics for later changes in the alignment or mapping step. Minimap2's accuracy and speed performances varied for the three datasets. The highest throughput of 403.46 kbases/s was for the simulated reads, that also had the lowest average length. The lowest throughput of 164.11 kbases/s was for the real PacBio reads, where the profiling indicated that it had the chaining step as a more significant bottleneck (50%) than the extending step (25%), whereas the other data had an inverted proportion. The highest frequencies of indels that showed in the final alignments were of a single nucleotide, although simulated reads caused an abnormal deviation from the alignment matrix's anti-diagonal, justified by the higher ratio of insertions. PacBio data presented high mapping rates, whereas 27.5% of ONT reads were unmapped;

- Minimap2's chaining process divided the read-reference pairs of sequences into many short (200-300 nt) sub-sequences. A direct substitution of the ksw alignment function in Minimap2 by a hybrid hardware implementation, aligning anchor-separated sub-sequences, resulted in a low performance due to data transfer latency. An alternative sequence execution by the aligner was developed and tested. Aligning longer anchor-extended sub-sequences resolved this (low performance) issue;

- GACT-X's alignment parameters were adapted to produce the most similar results to Minimap2, changing the Y-drop threshold to keep the maximum gap size in the band as in the original values. Internal wires defined in the Verilog code were increased to accommodate bigger tiles, and with tests on the simulated data, it was found that tiles of size 4,000 provided the best speed and accuracy performances, and such a configuration would be adopted in the Minimap2 GACT-X integrated solution. Experiments showed that, for part of the simulated data, GACT-X was 1.68x faster than ksw. With simulated data, 95.42% of the alignments had the same score as in Minimap2; real PacBio data presented a more dissimilar accuracy, with high rate of higher and lower scores; real ONT data had a considerable worse alignment (41.21%). Minimap2 and GACT-X aligned a similar proportion of the reads for the PacBio datasets, but interrupted early the alignment of many ONT reads;

- A Minimap2 GACT-X integrated system was developed to support multi-threading

and multi-kernel. Up to 8 threads and 2 kernels were implemented; the number of kernels was limited by the device's BRAM resources. The highest acceleration of 1.41x was observed on simulated and real ONT runs on 1 thread, and of 1.23x on real PacBio data (this lower rate can be linked to the lower occupation of the extending step observed in profiling); the general Minimap2 performance can be improved by hardware accelerating the chaining step;

- Detailed measurements for the kernel processing, data transfer and thread waiting execution times were performed in order to understand the limitations on the Minimap2-GACT-X integrated system. Acceleration decreased with more threads and could be explained by two factors: the kernels were not always available to process incoming data, increasing the time of threads waiting in line; multi-kernel generated conflict of data transfer in the PCIe channel. The first factor indicates that a larger amount of kernels would be needed for a larger number of threads; the second factor could be eased by implementing multi-devices (FPGAs) with one kernel in each having a dedicated PCIe channel.

Considering the results obtained in Chapter 6 and the analysis made on them, some future developments could be carried on to improve these results:

- A multi-FPGA design could sustain acceleration of more threads of Minimap2 with more kernels; for this, the implementation has to be updated to the Vitis environment, since SDAccel instances have been deprecated during the length of this Masters, and a new multi-FPGA AWS Instance has to be created; the increased cost must be evaluated;

- Tile sizes could be doubled by allocating all the BRAM available on the device to a single kernel to confirm if it could reduce data transfers, increase accuracy, and remove the PCIe competition;

- Traceback support could be added to the design to increase the acceleration, mainly by reducing the size of the read data (from traceback pointers to CIGAR strings); this implies in changing the Verilog RTL description with a probable increase in hardware area in the programmable logic;

- A second affine function could be added to the hardware design to properly mirror Minimap2's alignment scores, and increase the compared accuracy.

REFERENCES

ADEWALE, B. A. Will long-read sequencing technologies replace short-read sequencing technologies in the next 10 years? *African Journal of Laboratory Medicine*, v. 9, November 2020.

ALSER, M. et al. Accelerating genome analysis: A primer on an ongoing journey. *IEEE Micro*, Institute of Electrical and Electronics Engineers (IEEE), v. 40, n. 5, p. 65–75, September 2020. ISSN 1937-4143.

ALTSCHUL, S. F. et al. Basic local alignment search tool. *Journal of Molecular Biology*, v. 215, n. 3, p. 403–410, October 1990. ISSN 0022-2836.

AMARASINGHE, S. L. et al. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, v. 21, n. 30, February 2020.

AMAZON EC2 F1 Instances. 2022. ⟨https://aws.amazon.com/ec2/instance-types/f1/?nc1=h_ls⟩. Accessed: 2022-06-28.

AMAZON EC2 Instance Types. 2022. ⟨https://aws.amazon.com/ec2/instance-types/?nc1=h_ls⟩. Accessed: 2022-06-28.

AMAZON Web Services. 2022. ⟨https://aws.amazon.com/?nc1=h_ls⟩. Accessed: 2022-06-28.

ANTIPOV, D. et al. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, v. 32, n. 7, p. 1009–1015, 11 2015. ISSN 1367-4803.

BASE Quality Score Recalibration (BQSR). 2020. ⟨https://github.com/broadinstitute/gatk-docs/blob/master/gatk3-methods-and-algorithms/Base_Quality_Score_Recalibration_(BQSR).md⟩. Accessed: 2022-06-28.

BURROWS-WHEELER Aligner. 2010. ⟨http://bio-bwa.sourceforge.net/⟩. Accessed: 2022-06-28.

CHAISSON, M. J.; TESLER, G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC Bioinformatics*, v. 13, September 2012.

CHI, K. R. The year of sequencing. *Nature Methods*, v. 5, p. 11–14, January 2008.

CMAKE. 2022. ⟨https://cmake.org/⟩. Accessed: 2022-06-28.

CNNSCOREVARIANTS. 2020. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037226672-CNNScoreVariants⟩. Accessed: 2022-06-28.

COSTER, W. D. et al. Structural variants identified by oxford nanopore promethion sequencing of the human genome. *Genome Research*, v. 29, June 2019. ISSN 1178-1187.

DARWIN-WGA. 2019. ⟨https://github.com/gsneha26/Darwin-WGA⟩. Accessed: 2022-06-28.

DEVELOPING on AWS F1 with SDAccel and RTL Kernels - Part 1 of 4. 2022. ⟨https://www.xilinx.com/video/software/developing-on-aws-f1-with-sdaccel-and-rtl-kernels-part1.html⟩. Accessed: 2022-06-28.

EUROPEAN Nucleotide Archive. 2022. ⟨https://www.ebi.ac.uk/ena/browser/home⟩. Accessed: 2022-06-28.

FENG, Z. et al. Accelerating long read alignment on three processors. In: *Proceedings of the 48th International Conference on Parallel Processing*. New York, NY, USA: Association for Computing Machinery, 2019. (ICPP 2019). ISBN 9781450362955.

FERRAGINA, P.; MANZINI, G. Opportunistic data structures with applications. In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. [S.l.: s.n.], 2000. p. 390–398. ISSN 0272-5428.

FILTERVARIANTTRANCHES. 2019. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037227632-FilterVariantTranches⟩. Accessed: 2022-06-28.

FRITH, M. C.; HAMADA, M.; HORTON, P. Parameters for accurate genome alignment. *BMC bioinformatics*, v. 11, n. 80, February 2010.

FUJIKI, D. et al. Seedex: A genome sequencing accelerator for optimal alignments in subminimal space. In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. [S.l.: s.n.], 2020. p. 937–950.

FUNCOTATOR. 2022. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037224432-Funcotator⟩. Accessed: 2022-06-28.

GADSBøLL, K. et al. Current use of noninvasive prenatal testing in europe, australia and the usa: A graphical presentation. *Acta Obstetricia et Gynecologica Scandinavica*, v. 99, n. 6, p. 722–730, 2020.

GENERA. 2022. ⟨https://www.genera.com.br/⟩. Accessed: 2022-06-28.

GENOME Analysis Toolkit. 2022. ⟨https://gatk.broadinstitute.org/hc/en-us⟩. Accessed: 2022-06-28.

GEORGAKOPOULOS-SOARES, I. et al. Emt factors and metabolic pathways in cancer. *Frontiers in Oncology*, v. 10, n. 80, 2020. ISSN 2234-943X.

GNU gprof. 1998. ⟨https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html⟩. Accessed: 2022-06-28.

GOOGLE Cloud. 2022. ⟨https://cloud.google.com/⟩. Accessed: 2022-06-28.

GOTOH, O. Optimal sequence alignment allowing for long gaps. *Bulletin of Mathematical Biology*, v. 52, p. 359–373, May 1990.

GOYAL, A. et al. Ultra-fast next generation human genome sequencing data processing using dragen™ bio-it processor for precision medicine. *Open Journal of Genetics*, v. 7, p. 9–19, March 2017.

GRCH38. 2013. ⟨https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26⟩. Accessed: 2022-06-28.

GUO, L. et al. 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)*. [S.l.: s.n.], 2019. p. 127–135. ISSN 2576-2621.

HAPLOTYPECALLER. 2022. ⟨https://gatk.broadinstitute.org/hc/en-us/articles/360037225632-HaplotypeCaller⟩. Accessed: 2022-06-28.

HENG, J.; HENG, H. H. Karyotype coding: The creation and maintenance of system information for complexity and biodiversity. *Biosystems*, v. 208, p. 104476, October 2021. ISSN 0303-2647.

HENNESSY, J. L.; PATTERSON, D. A. A new golden age for computer architecture. *Communications of the ACM*, v. 62, n. 2, p. 48–60, 2019.

HUAWEI CLOUD. 2022. ⟨https://www.huaweicloud.com/intl/en-us/⟩. Accessed: 2022-06-28.

HUMAN 54x Dataset. 2014. ⟨http://datasets.pacb.com/2014/Human54x/fast.html⟩. Accessed: 2022-06-28.

HUMAN Genome Project Information Archive. 2003. ⟨https://web.ornl.gov/sci/techresources/Human_Genome/index.shtml⟩. Accessed: 2022-06-28.

ILLUMINA. 2022. ⟨https://www.illumina.com/⟩. Accessed: 2022-06-28.

KAPLAN, R.; YAVITS, L.; GINOSAR, R. Rassa: Resistive prealignment accelerator for approximate dna long read mapping. *IEEE Micro*, v. 39, n. 4, p. 44–54, July 2019. ISSN 1937-4143.

KOLIOGEORGI, K. et al. Dataflow acceleration of smith-waterman with traceback for high throughput next generation sequencing. In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. [S.l.: s.n.], 2019. p. 74–80. ISSN 1946-1488.

KUSHNICK, T. Thompson & thompson genetics in medicine. *JAMA*, v. 267, n. 15, p. 2115–2115, April 1992. ISSN 0098-7484.

LANGMEAD, B.; SALZBERG, S. L. Fast gapped-read alignment with bowtie 2. *Nature Methods*, v. 9, p. 357–359, March 2012.

LASZLO, A. H. et al. Decoding long nanopore sequencing reads of natural dna. *Nature Biotechnology*, v. 32, p. 829–833, June 2014.

LI, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, v. 34, n. 18, p. 3094–3100, September 2018. ISSN 1367-4803.

LIAO, Y.-L. et al. Adaptively banded smith-waterman algorithm for long reads and its hardware accelerator. In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. [S.l.: s.n.], 2018. p. 1–9. ISSN 2160-052X.

LIU, T. et al. A benchmarking study of sars-cov-2 whole-genome sequencing protocols using covid-19 patient samples. *iScience*, v. 24, n. 8, p. 102892, 2021. ISSN 2589-0042.

MANTERE, T.; KERSTEN, S.; HOISCHEN, A. Long-read sequencing emerging in medical genetics. *Frontiers in Genetics*, v. 10, p. 426, 2019. ISSN 1664-8021.

MAXFIELD, C. M. Chapter 3 - the origin of fpgas. In: MAXFIELD, C. M. (Ed.). *The Design Warrior's Guide to FPGAs*. Burlington: Newnes, 2004. p. 25–56. ISBN 978-0-7506-7604-5.

MINIMAP2-ACCELERATION. 2021. ⟨https://github.com/UCLA-VAST/minimap2-acceleration⟩. Accessed: 2022-06-28.

MUTEXES. 2022. ⟨https://www.gnu.org/software/libc/manual/html_node/ISO-C-Mutexes.html⟩. Accessed: 2022-06-28.

NCBI. 2022. ⟨https://www.ncbi.nlm.nih.gov/⟩. Accessed: 2022-06-28.

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, v. 48, n. 3, p. 443–453, 1970. ISSN 0022-2836.

ONO, Y.; ASAI, K.; HAMADA, M. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, v. 29, n. 1, p. 119–121, January 2013. ISSN 1367-4803.

OPENCL Reference Pages. 2011. ⟨https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml/⟩. Accessed: 2022-06-28.

ORTEU, A.; JIGGINS, C. D. The genomics of coloration provides insights into adaptive evolution. *Nature Reviews Genetics*, v. 21, 2020. ISSN 1471-0064.

OXFORD NANOPORE Technologies. 2022. ⟨https://nanoporetech.com/⟩. Accessed: 2022-06-28.

PACBIO. 2022. ⟨https://www.pacb.com/⟩. Accessed: 2022-06-28.

PALAZZO, A. F.; GREGORY, T. R. The case for junk dna. *PLOS Genetics*, Public Library of Science, v. 10, n. 5, p. 1–8, May 2014.

PBSV. 2022. ⟨https://github.com/PacificBiosciences/pbsv⟩. Accessed: 2022-06-28.

PFEUFER, V.; SCHULZE, M. Genetics/dna sequencing: Laser fluorescence powers sequencing advances. *Laser Focus World*, January 2015. Accessed: 2022-06-28.

POPLIN, R. et al. A universal snp and small-indel variant caller using deep neural networks. *Nature Biotechnology*, v. 36, November 2018. ISSN 1546-1696.

PTHREAD_SELF. 2021. ⟨https://man7.org/linux/man-pages/man3/pthread_self.3.html⟩. Accessed: 2022-06-28.

REUTER, J. A.; SPACEK, D. V.; SNYDER, M. P. High-throughput sequencing technologies. *Molecular cell*, v. 58, n. 4, p. 586–597, 2015.

RUN: ERR2585114. 2018. ⟨https://www.ebi.ac.uk/ena/browser/view/ERR2585114?show=reads⟩. Accessed: 2022-06-28.

SAMURA, O. Update on noninvasive prenatal testing: A review based on current worldwide research. *Journal of Obstetrics and Gynaecology Research*, v. 46, n. 8, p. 1246–1254, 2020.

SANGER, F.; NICKLEN, S.; COULSON, A. R. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, v. 74, n. 12, December 1977. ISSN 5463–5467.

SDACCEL Environment. 2019. ⟨https://www.xilinx.com/htmldocs/xilinx2019_1/sdaccel_doc/index.html⟩. Accessed: 2022-06-28.

SEGMENTATION Faults. 2022. ⟨https://matrixzj.github.io/documentations/memory_segfaults.html⟩. Accessed: 2022-06-28.

SEQUENCE Alignment/Map Format Specification. 2021. ⟨http://samtools.github.io/hts-specs/SAMv1.pdf⟩. Accessed: 2022-06-28.

SMITH, T. F.; WATERMAN, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology*, v. 147, n. 1, p. 195–197, 1981. ISSN 0022-2836.

ŠOŠIC, I. et al. Fast and sensitive mapping of nanopore sequencing reads with graphmap. *Nature Communications*, v. 7, March 2016. ISSN 11307.

ŠOŠIC, M.; ŠIKIC, M. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, v. 33, p. 1394–1395, May 2017. ISSN 1394-1395.

SRX9063500: PacBio SMRT whole genome sequencing of Sri Lankan Tamil H. sapiens. 2019. ⟨https://www.ncbi.nlm.nih.gov/sra/SRX9063500[accn]⟩. Accessed: 2022-06-28.

SUZUKI, H.; KASAHARA, M. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC bioinformatics*, v. 19, n. 45, February 2018.

TENG, C. et al. Accelerating the base-level alignment step of dna assembling in minimap2 algorithm using fpga. In: *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*. [S.l.: s.n.], 2021. p. 1–4. ISSN 2473-4667.

THIEL, V. et al. Mechanisms and enzymes involved in sars coronavirus genome expression. *Journal of General Virology*, Microbiology Society, v. 84, n. 9, p. 2305–2315, 2003. ISSN 1465-2099.

TMUX. 2022. ⟨https://github.com/tmux/tmux⟩. Accessed: 2022-06-28.

TURAKHIA, Y.; BEJERANO, G.; DALLY, W. J. Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly. *SIGPLAN Not.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 2, p. 199–213, February 2018. ISSN 0362-1340.

TURAKHIA, Y. et al. Darwin-wga: A co-processor provides increased sensitivity in whole genome alignments with high speedup. In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. [S.l.: s.n.], 2019. p. 359–372. ISSN 2378-203X.

VITIS Unified Software Platform. 2022. ⟨https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html⟩. Accessed: 2022-06-28.

VMACCEL. 2022. ⟨https://www.vmaccel.com/⟩. Accessed: 2022-06-28.

WANG, X. et al. When fpga meets cloud: A first look at performance. *IEEE Transactions on Cloud Computing*, v. 10, p. 1344–1357, May 2020.

APPENDIX A

This appendix presents the programming options adopted to integrate GACT-X's host lines into Minimap2's original source code. A new C++ file "gactx.cpp" was created and added to Minimap2's file and to the compilation object list. It contains the functions:

- **"load_file_to_memory"** loads the xclbin binary file to boot the kernel in the FPGA;
- **"fpga_configuration_and_setup"** selects a Xilinx platform from the ones available; selects a target device; creates context, command queue, program, and kernel variables; defines memory bank mapping; and creates input and output buffers;
- **"fpga_shutdown_and_cleanup"** releases memory objects and other global variables;
- **"gactx_align"** configures tile size, tile overlap, and alignment scores; sends alignment input sequences; programs the tile algorithm; reads the tile outputs; encodes the final CIGAR string; and updates Minimap2's alignment results.

The compilation was changed to using CMake (CMAKE, 2022) with the Xilinx's compiler "xcpp", and inclusion of directories, libraries, and flags necessary for running the host's code. OpenCL variables cl_context, cl_command_queue, cl_program, cl_kernel, cl_mem_ext_ptr_t, cl_mem, cl_event, as well as the pointers to the host's memory were initialized as global variables so that they can be shared among the multiple threads in the multi-threading option of Minimap2.

cl_kernel, cl_mem_ext_ptr_t, cl_mem, and host memory pointers are initialized as arrays of length (NUM_KERNELS), which is a #define that can be set as 1 or 2 by the developer, and indicates the number of kernels being utilized in the circumstance. A similar global int array "kernel_expanding" is used to indicate whether each kernel is free or occupied with 0 or 1 respectively.

The aligning process in Minimap2 is set up in the “mm_align1” function in “align.c”. It is where ksw calls are made for each sub-sequence between two anchors. This was substituted with a complete right-extend from the first anchor using GACT-X, whenever both of the input sequences are longer than 1,000 nucleotides.

To couple multiple CPU threads with 1 or 2 kernels, a FIFO queue (global array variable) was created. Each thread that reaches the alignment stage goes to the end of the queue, and waits until it is the first in line and there is a kernel available. This is done using “pthread_self” (PTHREAD_SELF, 2021) as the thread identifier. Mutex (MUTEXES, 2022) is used to protect lines that edit the FIFO queue, to avoid conflicting memory access. The function “gactx_align” is called after the thread has picked its corresponding kernel and left the queue.

When running 2 kernels in parallel, there can be conflict in the PCIe channel (e.g. when two threads send data to the FPGA at the same time). This can result in Segmentation Faults (SEGMENTATION..., 2022). The cl_event global variable is used to synchronize these transfers.