Accelerated Systems for Portable DNA Sequencing

by

Harisankar Sadasivan

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2023

Doctoral Committee:

Professor Satish Narayanasamy, Chair
Professor David Blaauw
Associate Professor Reetuparna Das
Professor Scott Mahlke

Harisankar Sadasivan
hariss@umich.edu
ORCID iD: 0000-0002-2832-458X

To my family and teachers

ACKNOWLEDGMENTS

I am grateful to a lot of people with whom I have interacted throughout the course of my program. Advice, lessons, interactions, and support from a lot of people inside and outside the University of Michigan Ann Arbor have contributed to my research.

First and foremost, I would like to thank my advisor, Prof. Satish Narayanasamy, for his guidance, support, and encouragement throughout my research. His expertise and insights have been invaluable in shaping my work.

I'm grateful to Prof. David Blaauw and Prof. Reetuparna Das for co-advising me during the first few years of Ph.D. Their insights helped me prioritize and explore different strategies for sequence mapping. I would like to express my gratitude to Prof. Jenn Wiens who provided valuable insights on applying machine learning for my fourth chapter. Additionally, I'm also thankful to Prof. Scott Mahlke, Prof. Jason Mars and Prof. Ronald Dreslinski for their constructive feedback during various stages of my PhD.

I'm also thankful to a few members of the UM Medical School including Prof. Robert P. Dickson, Dr. John Erb-Downward, Dr. Rishi Chanderraj, and Piyush Ranjan for introducing us to nanopore sequencing for medical diagnostics.

UM and UM-CSE has several good resources for students, some of which I am grateful to have utilized during my Ph.D program. The Graduate Programs Office helped me plan my program. The Departmental Computing Organization helped me manage systems and networks. UM-ITS helped me plan my cloud instance usage.

I was fortunate to work with some amazing people at NVIDIA during my internship. I am thankful to Dr. Johnny Israeli, Dr. Ajay Tirumala, Dr. Eric Dawson, Dr. Mehrzad Samadi, Daniel Stiffler, Milos Maric and Vishanth Iyer from NVIDIA for their recommendations on performance

optimizations and CUDA best practices. I am also thankful to Oded Green, Lotfi Slim, Harry Clifford and Mehrzad Samadi from NVIDIA for providing information on related work in healthcare on GPUs.

I would like to particularly thank our lab alumni Dr. Subarno Banerjee and Dr. Arun Subramaniyan for providing me constant guidance and support.

I would like to thank Tim Dunn for his contributions to the second chapter of this dissertation. I also enjoyed working with Dr. Jack Wadden and Kush Goliya while conducting experiments in our wet lab situated in Engineering Research Building-2. I also thank Kuan-Yu Chen from EECS for collaborating with me on synthesizing our ASIC, and Yichen Gu and Ashwin Bhat for being there in my brainstorming sessions during the first year of my Ph.D. program.

I am grateful to our funding sources including D. Dan and Betty Kahn Foundation, National Science Foundation (NSF award 2030454), NVIDIA Corporation, U Michigan Ann Arbor, and Google Cloud Platform. I also thank the NIGMS Human Genetic Cell Repository at the Coriell Institute for Medical Research for providing us with the NA12878 cell lines/DNA samples. We thank ONT for the developer access to Guppy and Read Until.

Finally, I owe a debt of gratitude to Prof. Chandra Sekhar Seelamantula and Prof. Aparna P who mentored me during the various stages of pursuing my undergraduate degree.

I would like to thank my dear friends who make my Ann Arbor memories colorful. I enjoyed the time I spent at IEEE-HKN doing intellectual activities and voluntary community service.

Finally, I am blessed to have a wonderful family support me throughout my program. My spouse Gayatri Nair, brother Vinu Sankar Sadasivan, mother Rekha Rajagopal, father Sadasivan Thrilokan, and grandmother Girija Nair have cheered and prayed for me from different parts of the globe. I dedicate this dissertation to them.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

The MinION is a revolutionary handheld DNA sequencer that is inexpensive, portable, and can perform real-time sequencing. MinION is increasingly used in Precision Medicine applications. However, the MinION lacks portable compute power. This thesis introduces two clinical applications of the MinION and identifies and solves performance bottlenecks through hardware and software solutions to enable portable microbial diagnostics. Finally, we discuss how our accelerated solutions will fit on a laptop with a GPU.

More than 99% of DNA fragments in a typical human sample are non-target (human), which may be skipped in real-time using theMinION's Read Until feature. We analyze the performance of the Read Until pipeline in detecting target microbial species for targeted viral pathogen detection and microbiome abundance estimation. We find new sources of performance bottlenecks (basecaller in the classification of a fragment) that are not addressed by past genomics accelerators. While SquiggleFilter and DTWax are our solutions for targeted viral pathogen detection, RawMap is for untargeted microbiome analysis. We also discuss accelerating the bottleneck step in the DNA mapping software (Minimap2) used in all of MinION's sequencing workflows.

For targeted virus detection, we discuss SquiggleFilter which is a portable and programmable hardware-software solution that directly analyzes MinION's raw squiggles and filters everything except target viral DNA fragments. SquiggleFilter avoids the expensive basecalling step and uses hardware accelerated subsequence Dynamic Time Warping (sDTW). We show that our 14.3W 13.25mm^2 accelerator has 274$\times$ greater throughput and 3481$\times$ lower latency than existing GPU-

based solutions while consuming half the power. DTWax overcomes the on-chip memory limitations of SquiggleFilter by optimizing its high-accuracy sDTW algorithm for GPUs resulting in a $\sim$1.92X sequencing speedup and $\sim$3.64X compute speedup: costup.

For the untargeted classification and analysis of microbiome, we discuss RawMap which is a machine-learning-based smart and efficient solution that reduces sequencing time and cost by $\sim$24% and computing cost by $\sim$22%. We also discuss how RawMap may be used as an alternative to the RT-PCR test for viral load quantification of SARS-CoV-2.

Minimap2 is a software used in all MinION workflows. minimap2-accelerated (mm2-ax) is a heterogeneous design for sequence mapping where minimap2's bottleneck step is sped up on the GPU with bit-exact output. mm2-ax on an NVIDIA A100 GPU improves the bottleneck step (chaining) with 4.07 - 1.93X speedup: costup over a SIMD baseline.

Finally, we envision a portable solution to microbial diagnostics with a laptop connected to the MinION. DTWax can perform targeted virus detection on the GPU and RawMap does untargeted microbiome classification on the CPU. Post-sequencing tasks like basecalling, alignment (using mm2-ax) and variant calling use the GPU.

CHAPTER 1

Introduction

1.1 Long read sequencing

Genomic technologies are revolutionizing clinical diagnostics and public health. By 2025, Genomics is expected to produce 40 exabytes of data (20X of that of YouTube)[3]. Apart from being used to track the origin, transmission and evolution of infectious agents like SARS-CoV-2, genomics is used in various diagnostics tests including microbiome analysis, tumor analysis, childhood and rare disorders and non-invasive prenatal screening[4].

Sequencing has been becoming exponentially cheaper over the several years[5]. A $100 solution to sequence a human genome is right around the corner[6]. Among the available sequencing technologies, long read sequencing is gaining more popularity with improved raw read accuracy, reduced end-to-end sequencing times, lower costs of adoption, and ease of portability [7, 8]. Longer reads help span highly repetitive regions in the genome which short reads cannot. This helps applications like denovo assembly and structural variant calling [7, 9]. A recent study used long read sequencing to showcase the world's fastest blood-to-variants workflow for genetic diagnosis at the point-of-care [8]. This further underlines the emerging significance of long read sequencing.

1.2 Oxford Nanopore Technology's MinION sequencer

Oxford Nanopore Technology (ONT) is a company that manufactures MinION DNA/RNA long-read sequencers. MinION is attractive because of many factors:

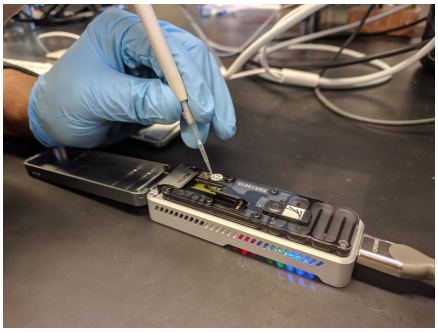

Figure 1.1: MinION sequencer in our laboratory.

Long reads: MinION sequencers are capable of measuring long strands of DNA or reads, and can theoretically sequence any strand, regardless of length. The current world record stands at over 4 million bases [10].

Cost: The MinION only costs $1,000, and offers affordable specimen preparation kits ($100/use) and flow cells ($125/use assuming 4× re-use). In comparison, it costs $80,000-$100,000 to purchase even the most affordable "Next Generation Sequencing" machines.

Real-time: MinION sequencers provide real-time, streaming output from the device. Streaming signal output enables on-the-fly secondary analyses, and the ability to stop sequencing as soon as the desired coverage (redundancy in sequencing a certain region of the genome) is reached. Read Until is a feature that lets us instruct the MinION to eject a read if we think it is not a target and selectively sequence only the target read. As shown in Fig.1.2, when a DNA fragment passes through the nanopore, it may be analyzed in real-time to classify it as belonging to the target DNA of interest or not. If it is a target, we completely sequence it and if it is a non-target, Read Until enables us to instruct the MinION to reverse the potential on the pore and eject the fragment. Read Until is a feature that is useful in our clinical applications of interest where more than 99% of the DNA is human and ejecting them would help save sequencing time and cost.

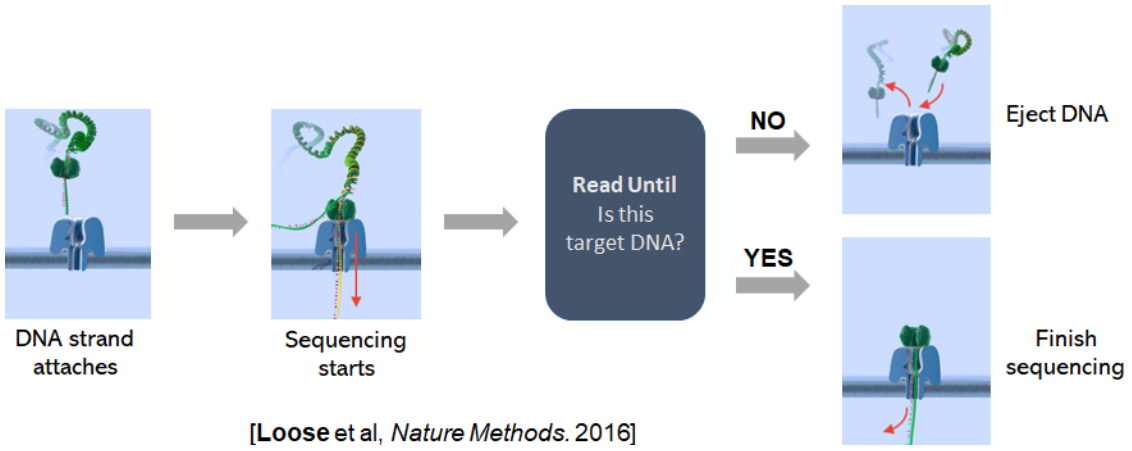

Figure 1.2: Read Until selectively sequences only target DNA read

Portability: A key feature that sets the MinION sequencer apart from all other sequencers in terms of wet-lab, sequencing and compute. However, MinION lacks the portable compute power that is required for real-time sequencing using Read Until.

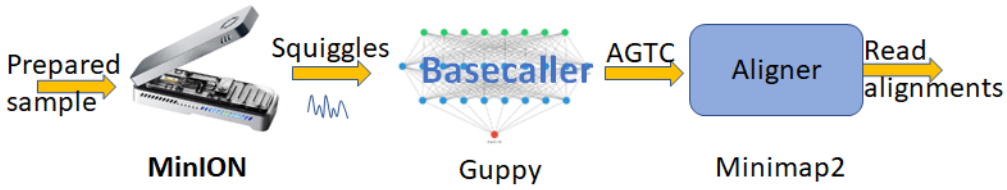

Figure 1.3: A basic MinION sequencing pipeline

As an artifact of the wet-lab process (DNA extraction and library preparation), DNA in the input sample gets fragmented. We call these DNA fragments, reads. A MinION has 512 channels, each of which can sequence reads approximately at the rate of 450 bases per second. A nanopore (a nano-scale channel) senses the DNA molecule that passes through the pore by measuring the characteristic disruptions in electric current density. Decoding this noisy but characteristic output signal of the MinION (squiggle) helps us understand the DNA base (A, G, T, or C). This decoding of squiggles to bases is performed by the basecaller as shown in Fig. 1.3. Guppy is an ONT-proprietory basecaller based on a deep neural network that runs on a GPU but is slow and inaccurate for some of the real-time tasks we describe in the upcoming chapters.

The output of the basecaller is basecalled reads. These reads are read by an aligner. ONT

recommends Minimap2 as an aligner. Minimap2 does approximate string matching to find the approximate region of match for the query string on a target reference genome and can also perform base-level alignment. As we explain later on, Minimap2 also has a performance bottleneck due to the irregularity of workload and lack of parallelism in existing implementations.

ONT MinION has a variety of applications as listed on their website, some of which includes microbiology, cancer, population genomics, infectious diseases and human genomics. We develop efficient hardware-software solutions for performance bottlenecks in applications with MinION including viral pathogen detection and human microbiome estimation. As a final contribution, we accelerate Minimap2 which is used in all MinION workflows.

1.3 Motivation and Organization

We introduce two clinical applications of the MinION sequencer– viral pathogen detection and microbiome abundance estimation. The Read Until feature of the MinION may be used to save time and cost by filtering out the high fraction of non-target human DNA reads (up to 99%). However, we find new sources of performance bottlenecks (basecaller in the classification of a read) that are not addressed by past genomics accelerators. This problem is further amplified by the sequencing throughput expected to grow by 100X in the near future.

Chapter 2 (SquiggleFilter) talks about a portable virus detector– a novel and programmable hardware accelerated dynamic time warping (DTW) based filter that directly analyzes MinION's raw squiggles and filters everything except target viral reads, thereby avoiding the expensive base-calling step. However, SquiggleFilter can only identify one microbial species of size 100Kb at a time due to its limited on-chip memory. This limits us to infectious viruses (almost all are less than 100Kb in size).

To overcome SquiggleFilter's problem of limited on-chip memory, **Chapter 3** talks about optimizing the SquiggleFilter algorithm for GPUs[11]. GPUs are more easily programmable, scalable, and more widely available.

For our second application of microbiome abundance estimation, we need to be able to identify several large bacterial genomes with each bacterial genome being in the order of millions of bases. **Chapter 4** discusses an efficient machine learning-based Read Until filter where we do not need to program in the reference genome. RawMap is a microbial species-agnostic (untargeted) Read Until classifier for filtering non-target human reads. RawMap is offered as a plug-and-play CPU solution to existing Read Until pipelines and we demonstrate sequencing time and cost savings with RawMap in microbiome abundance estimation. We also discuss how RawMap may be used as an alternative to the RT-PCR test for viral load quantification of SARS-CoV-2.

Chapter 5 is our attempt at identifying and solving the performance bottleneck of Minimap2, the state-of-the-art aligner used for almost all long-read processing workflows including the two clinical applications under our consideration, Whole Genome Sequencing, targeted gene panel sequencing, and in-operation-room decision making. We transform Minimap2's bottleneck step to expose better parallelism and accelerate it on the GPU with bit-exact output.

CHAPTER 2

SquiggleFilter: An Accelerator for Portable Virus Detection

2.1 Introduction: Virus Detection for Pandemic Preparedness

The COVID-19 pandemic caused by the SARS-CoV-2 virus continues on a global scale. Today, diagnostic tests are widely available to detect SARS-CoV-2. Most of these tests involve some form of Polymerase Chain Reaction (PCR), a common technique for exponentially amplifying DNA/RNA. In order to detect a virus such as SARS-CoV-2, custom "primers" are first designed and manufactured which will only attach to and amplify specific regions of DNA/RNA in the target virus's genome. After PCR, the virus's presence or absence can be determined based on whether the amplification was successful or not.

A significant shortcoming of the current approach is that PCR primers are targeted to a specific virus. Custom primer design is a complex, error-prone, and time-consuming process [12] [13]. Even though SARS-CoV-2's RNA was sequenced in early January 2020, validated SARS-CoV-2 specific PCR primers took several months to develop [13] [14]. Lack of mass testing capability in the early stages of SARS-CoV-2 made it difficult to detect and control its spread, leading to a catastrophic pandemic. While we now have adequate testing capability for SARS-CoV-2, it is not unlikely for another novel virus like SARS-CoV-2 or its variants to emerge in the near future [15], and if it does, we need to be prepared with adequate testing infrastructure in place to detect and control its spread in the early stages.

We envision a programmable virus detector (one that constructs whole viral genomes) that can be deployed worldwide. As soon as an emerging novel virus is discovered and sequenced, the reference genome of the novel virus would be distributed to all the devices, instantly turning them into targeted detectors.

Our solution uses Oxford Nanopore Technologies' (ONT) MinION Mk1B (henceforth, referred to as the MinION). We replace targeted PCR with universal PCR [16], which amplifies *all* DNA/RNA. Thus, it avoids the problem of custom PCR primer design and deployment mentioned earlier. However, this introduces a different problem, as up to 99.99% of the DNA/RNA in a typical biological specimen (e.g. saliva) is non-viral [17] (non-target) and most belongs to the host. Amplifying all DNA/RNA preserves this ratio, resulting in the vast majority of sequencing and computing time and cost stemming from processing non-target DNA/RNA.

In order to solve this needle-in-a-haystack problem, we use Read Until [18]. As reads (DNA/RNA fragments) are sequenced, they need to be analyzed in real-time. As soon as the computer classifies that the read is non-viral, the sequencer is instructed to eject it, which saves the time and cost of sequencing non-viral reads (greater than 99% of all reads). Unfiltered viral reads are used to construct the whole virus genome using reference-guided assembly (alignment and variant calling).

The MinION, however, does not have any on-board computing power to perform such secondary analysis. In this paper, we analyze the performance of the Read Until bioinformatics pipeline for efficiently sequencing viral pathogens, and realize a portable computing solution that can be integrated with MinION.

We discover new performance bottlenecks that are not addressed by past genomics accelerators [19, 20, 21, 22, 23, 24, 25, 26]. In particular, we find that the Deep Neural Network (DNN) basecaller (software that translates MinION's electrical squiggles to AGTC bases) dominates the computing time (96%). The aligner and variant caller, which have been the targets of recent accelerator research, constitute a much smaller fraction of compute. We also find that a current edge GPU is inadequate to keep up with the throughput of the MinION. Also, its high latency in classifying a read prevents us from taking advantage of the latency-critical Read Until feature of

MinION.

Converting squiggles to bases using a compute-intensive basecaller, and then aligning to check if a read belongs to the target virus is needlessly expensive for classifying it. Instead, we skip the basecaller altogether by directly comparing each read's squiggles to the precomputed expected signal profile of the target virus's entire reference genome (the "reference squiggle"). By skipping the compute-intensive basecaller step, we improve efficiency significantly.

We present SquiggleFilter, a hardware/software co-designed filter which identifies non-target reads by directly comparing the real-time measured squiggles to the target virus's precomputed reference squiggle. A classification decision is made based on the degree of match. We develop a custom subsequence dynamic time warping (sDTW) algorithm [27] to perform this classification. It includes solutions that improve accuracy by adaptively examining longer read prefix lengths when needed. It also includes customizations that result in area efficient hardware.

sDTW-based SquiggleFilter is significantly more efficient than a DNN-based basecaller, and its regular compute-bound characteristic makes it amenable for hardware acceleration. sDTW is a dynamic programming algorithm [28] whose complexity is proportional to the product of the length of the reference (R) and query (Q). Its regular memory access pattern allows us to build a fast and space efficient 1D systolic array accelerator for sDTW with a constant number of processing elements. Fortunately, we find that almost all epidemic viruses have genome references of length 50,000 (R) bases or smaller (see Figure 2.9) [29]. As a result, our accelerator can easily complete the classification in $\sim$2R cycles (forward and backward of reference strand), and still meet the strict latency requirement for leveraging Read Until.

Our work makes the following contributions:

- we demonstrate that basecalling is the computational bottleneck in the virus sequencing pipeline. Read alignment and variant calling – targets for prior accelerator work – are not the bottleneck.
- we identify direct squiggle alignment (first proposed in [30]) as a more efficient alternative to basecalling and alignment when enriching low-concentration viral specimens with Read

Until.

- we propose multi-stage sDTW and several modifications to vanilla sDTW to realize an accurate and efficient hardware accelerator.
- we co-design a sDTW hardware accelerator to filter non-viral reads, for variable read prefix and almost all infectious viral genome lengths
- we demonstrate that this hardware, unlike current approaches, will enable Read Until to scale with rapidly increasing nanopore sequencing throughput
- we quantify accuracy and efficiency of our classifier using real-world metagenomic datasets, including datasets collected from our wet-lab experiments for Read Until.

Results: We design an edge device with compute capabilities similar to a Jetson Xavier System-on-Chip [31] consisting of SquiggleFilter, an edge GPU, and an 8-core ARM processor. We show that our proposed SquiggleFilter can accurately distinguish target viral DNA/RNA from background human DNA/RNA. We evaluate accuracy using non-contagious lambda phage virus data sequenced in our own lab. In terms of efficiency, we show that our SquiggleFilter accelerator has 274× higher throughput than the conventional software pipeline (using a basecaller) on an edge GPU while only consuming an area of 13.25mm^2 and power of 14.31W. SquiggleFilter's throughput is 233.65M samples/s, which far exceeds the maximum throughput of 2.05M samples/s on a MinION [32], and is adequate to handle up to a 114× increase in MinION's throughput in the future. The latency for classifying any read is 0.043ms, which is insignificant to Read Until decision's critical path.

2.2 Background: State-of-the-art Virus Detectors

2.2.1 Need for a Virus Detector

While SARS-CoV-2 was discovered – and its RNA genome sequenced – by early January 2020, it was not until several months later that mass testing was available worldwide. Figure 2.1 shows

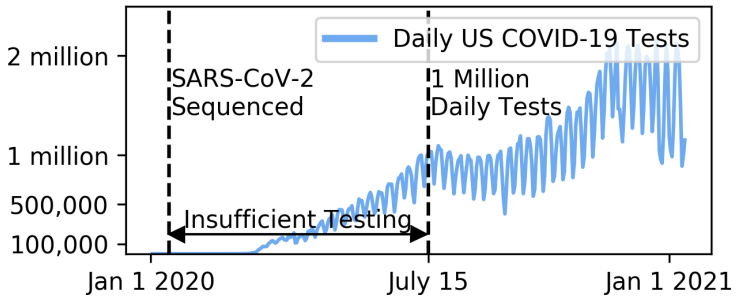

Figure 2.1: Progression of US COVID-19 testing [1]

the steady increase in daily COVID-19 tests performed within the United States [1]. A widely established global testing infrastructure would have helped control the spread of the virus early on, possibly saving hundreds of thousands of lives.

Given the increasing frequency of viral outbreaks, experts are concerned that it is only a matter of time before a new virus threatens the globe [15]. Thus, we need a virus testing technology that can be widely deployed *ahead-of-time*, and reprogrammed to detect and identify mutations in novel viruses as soon as they emerge.

In this work, we focus on controlling the spread of novel infectious viruses in their early stages, as soon as they are discovered and sequenced. Our goal is to enable a universal rapid test that can determine the whole genome of a target virus using reference-guided assembly. Targeting a specific virus enables us to make significant optimizations that help us reduce time and cost of sequencing and compute.

2.2.2 State-of-the-art Virus Detectors

Table 2.1 lists commonly used tests and ONT-based sequencing solutions for SARS-CoV-2. None of the methods except direct RNA or DNA sequencing are programmable, and therefore, are not effective in controlling the pandemic in its early stages. Antigen (paper) tests detect specific surface proteins on the virus. They are cheap, portable, and fast. However, they have low sensitivity and can only detect viruses present at high concentrations.

Molecular tests identify specific regions of interest in a virus's genome and amplify this DNA

Tests	Diagnostic Power	Programmable	Time (min)	Cost ($)
Antigen-based test				
Paper [33]	presence		15	5
Non-sequencing molecular test				
RT-LAMP [34][35]	presence		60	15
RT-PCR [36]	presence		120-240	<10
Sequencing based molecular test (30× coverage)				
ARTIC [37][36]	98 targets		305	100
LamPORE [38]	3 targets		<65	-NA-
RNA: 1% virus	whole genome	✓	240	110
0.1% virus [39]	whole genome	✓	1206	190
DNA: 1% virus	whole genome	✓	320	105
0.1% virus [40]	whole genome	✓	470	120

Table 2.1: A comparison of popular commercial and ONT sequencing-based virus detectors for SARS-CoV-2.

if present in the specimen. Polymerase Chain Reaction (PCR) is a common technique used for amplification. It has high sensitivity [41] but requires thermal cycling, which can be slow and expensive. LAMP (Loop Mediated Isothermal Amplification) is a more recent technology that obviates the need for a thermal cycler, but its primers are more complicated to design than PCR.

If amplification was successful (i.e., target DNA is present), it can be detected using fluorometry or colorimetry. Most clinical tests for SARS-CoV-2 stop here. However, by sequencing the amplified specimen, we can assemble portions of virus's genome, depending on the number of targets amplified. ARTIC and LamPORE [38] amplify 98 and 3 genes respectively, and then use ONT's nanopore sequencing.

Current solutions for virus detection use multiplex primer sets specific to a virus. Primer design is a complex, error-prone and time-consuming process [12] [13]. Thus, they are not an effective solution for early pandemic control. The COVID-19 pandemic highlights this problem, where designing and distributing target-specific primers was challenging, especially when supply chains broke amidst the pandemic.

An alternative to developing custom primers is to directly sequence the specimen following amplification with universal primers, which non-selectively amplify all DNA. This amplification

step is required to increase the quantity of DNA, which greatly reduces average capture time (the time required for a DNA strand to enter a nanopore) and therefore sequencing time. The wet-lab protocol followed, Sequence Independent Single Primer Amplification (SISPA) [43, 44], is universal and hence can be used on all RNA viruses. SISPA has four major steps: (1) RNA extraction, (2) complementary DNA generation, (3) PCR amplification, and (4) final sequencing specimen preparation.

A significant hurdle to SISPA-based sequencing is that following amplification, the specimen contains the genetic material of the target virus among a sea of human and bacterial DNA/RNA. The proportion of target virus DNA/RNA can be as low as 0.01% percent [17]. As a result, the time and cost of sequencing and data processing for this approach is significantly greater than that of custom primer-based solutions.

If this cost barrier can be overcome, this approach would enable detection of novel viruses without requiring months to develop and distribute virus-specific primers. Read Until can greatly increase the efficiency of sequencing by filtering out non-target reads using the virus's reference genome. Current Read Until approaches are limited by insufficient throughput, but our hardware accelerated SquiggleFilter ensures the future scalability of Read Until on higher throughput sequencers.

2.2.3 Insufficient Compute Power for Read Until

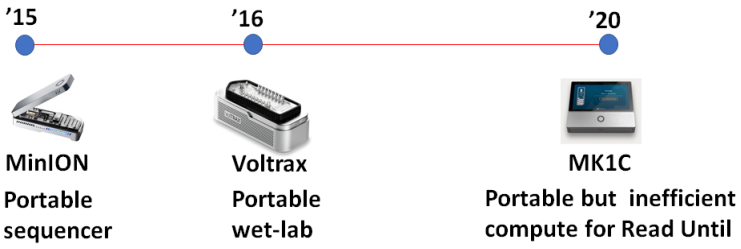

Figure 2.2: Sequencing and wet-lab is portable. Compute, though portable, is insufficient for Read Until.

Although the MinION has its advantages, it has no inbuilt computing power to make Read Until decisions. A slow read classification results in wasted sequencing time. We additionally find that commodity GPUs are undesirable in terms of both throughput, latency and power.

2.3 Compute Bottlenecks in Portable Virus Detection

Our goal is to build a cost and time efficient sequencing pipeline for determining the whole genome of a targeted virus, but without using custom primers for target amplification. We seek to reduce time and cost using the Read Until feature of Oxford Nanopore (ONT)'s palm-sized MinION sequencer.

To this end, we constructed a software pipeline using state-of-the-art bioinformatics tools and analyzed its performance. Our profiling results expose new performance bottlenecks that are different from those targeted in past accelerators for human genome sequencing [19, 20, 21, 22, 23, 24, 25, 26].

2.3.1 Bioinformatics Pipeline

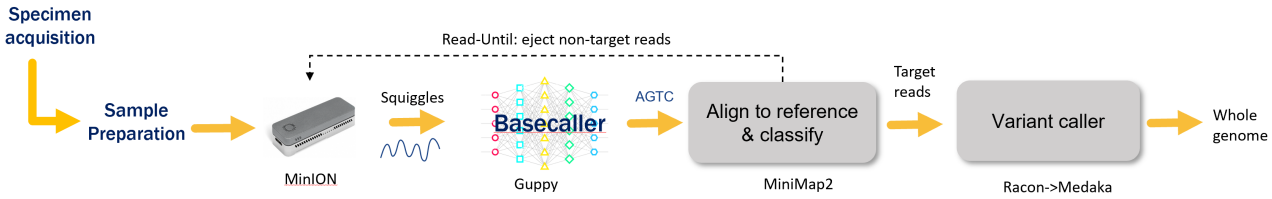

Figure 2.3: A Read Until pipeline for targeted reference-guided assembly of a virus genome.

The MinION sequencer measures electrical current signals that represent the bases (A, G, T, C) moving through each pore, recording approximately 10 samples for each base. All the active pores (up to 512 in the MinION) concurrently produce squiggles for the reads flowing through them. These squiggles can be analyzed in real-time as the reads flow through the pores.

Figure 2.3 illustrates the analysis pipeline for the squiggles. A *basecaller* translates squiggles into bases. The latest basecallers (such as ONT's Guppy [45]) use compute-intensive DNNs, which

must be large and deep to attain state-of-the-art accuracy. Guppy processes reads in chunks of 2000 samples, and uses five bidirectional LSTM layers for encoding followed by a custom CTC (Connectionist Temporal Classification) decoder. ONT provides two versions of its basecaller: a high-accuracy version (Guppy), and another that trades off accuracy for performance (Guppy-lite).

In our Read Until pipeline, squiggles of a read are basecalled in real-time. After a short prefix of a read has been basecalled, it is then processed by an aligner (MiniMap2 [46]) that aligns the read to the target's reference genome. If a good alignment is found, then the read is classified as a target and passed on to the next stage. Otherwise, a signal is sent to the MinION device, instructing it to eject the non-target read from further sequencing. Thus, the critical computing path for Read Until includes both the basecaller and aligner.

The target reads are collected and analyzed by a variant caller (Racon [47] followed by Medaka[48]). We seek to cover every position in the reference genome by 30 reads (30× coverage). The variant caller analyzes the reads piled up at each reference genome location, and identifies any genomic differences ("variants") between the sequenced and reference viruses. As the variant caller is not involved in Read Until decisions, it is off the critical path.

2.3.2 Performance Bottlenecks

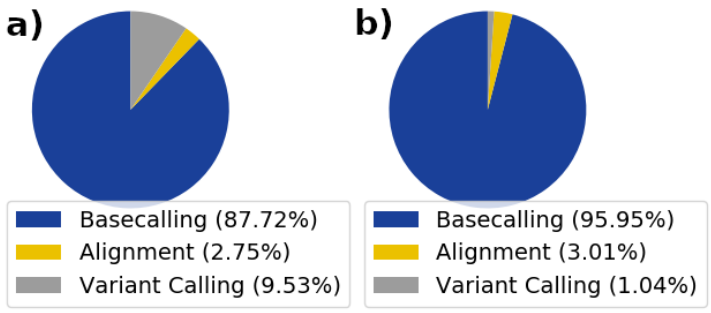

Figure 2.4: Basecalling is the bottleneck in a Read Until assembly of a SARS-CoV2 genome from specimens with **a)** 1%, and **b)** 0.1% viral reads.

Figure 4.3 shows the performance bottlenecks of the bioinformatics pipeline (Section 2.3) used to assemble the whole SARS-CoV2 genome, evaluated on the CPU and GPU in Table 2.3. The results

are shown for two representative biological specimens, one where the target viral reads constitute 1% of all the reads, and the other 0.1%.

We observe that a large fraction of computing time (96%) goes towards basecalling. This is in spite of using the more efficient, but less accurate, Guppy-lite.

Compute spent towards aligning (MiniMap2) and variant calling (Racon and Medaka) constitutes significantly smaller fraction, especially for specimens with low viral load (0.1%). In contrast, prior work on genomics accelerators targeted aligners and variant callers used for reference-guided assembly of human DNA [19, 20, 21, 22, 23, 24, 25, 26]. There are several reasons for this significant difference, discussed next.

All the reads are aligned to a target viral genome to classify them as target or non-target. This alignment step, however, is significantly less compute intensive compared to aligning to a human genome, because viral genomes are much shorter (≈30,000 bases) than human DNA (3 billion bases).

Only a small fraction of target reads (1% to 0.1%) need to be processed for reference-guided assembly of a viral genome. Therefore, the variant caller is invoked only for a small fraction of sequenced reads. Also, given that viral genomes are shorter, we find that the variant caller does not consume much compute resources. Furthermore, the variant caller is not on the critical path for using Read Until, as it is not required for classifying reads.

We find that even a 250W Titan GPU has barely enough basecalling throughput (with low accuracy Guppy-lite) to keep up with a MinION's maximum sequencing throughput. An edge GPU (e.g., Jetson Xavier's) is several times slower than that, and therefore it cannot process all the sequenced reads in real-time to exploit the latency sensitive Read Until feature.

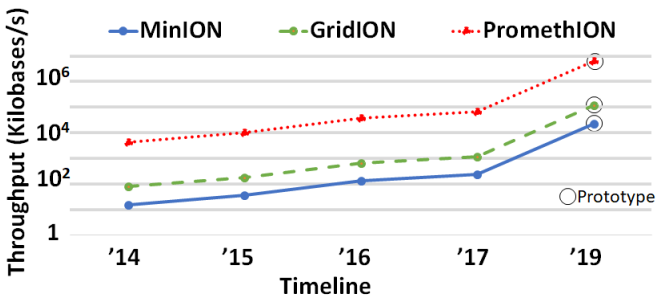

Figure 2.5: Sequencing throughput is increasing exponentially [2].

Sequencing throughput, however, continues to grow, as shown in Figure 2.5. Oxford Nanopore Technologies (ONT)'s GridION is only slightly larger, but has 5× the sequencing throughput of a MinION. ONT announced in 2019 that they are working with MinION prototypes that provide 16× sequencing throughput of MinION devices available in the market today. Within the next few years, they plan to release a production flowcell with 100× greater throughput [49].

Currently, the MinION does not have any on-board compute capability. Our goal is to map all the secondary compute analysis onto an edge system-on-chip so that it can be integrated with the MinION. We address this growing computing need with our small, low-power accelerated SquiggleFilter, which greatly reduces the basecalling and alignment computation required for non-target reads.

2.4 SquiggleFilter: A Squiggle-level Targeted Filter using Dynamic Time Warping

As discussed in Section 2.3, classifying a read being sequenced by analyzing its short prefix as target or not, in real-time, is the compute bottleneck. Additionally, basecalling for this classification consumes the most compute time.

Instead of using a basecaller (DNNs) and MiniMap2 aligner to classify a read's prefix, we discuss SquiggleFilter's algorithm that directly aligns each read's electrical signals (query) to the target viral genome's precomputed electrical signal (reference). As a majority of the reads are

non-targets, we reduce latency and save much of the work done in basecalling and aligning these non-target reads.

SquiggleFilter aligns the query squiggle with a precomputed reference squiggle of the viral genome using a variant of the dynamic time warping (DTW) algorithm [50]. Recent work has eschewed sDTW due to it's $\Theta(NM)$ complexity [51, 52, 53, 54], but we demonstrate that since both queries (read prefixes) and virus genomes are short, it is a practical solution for viral read enrichment. We further demonstrate its effectiveness on real sequencing data for a SARS-CoV-2 specimen.

Finally, we propose multi-stage sDTW filtering to improve efficiency, and discuss several improvements to conventional sDTW that help realize an efficient hardware accelerator.

2.4.1 Constructing the Reference Squiggle

In order to align raw signals to a reference genome, the known sequence of bases must first be converted to an expected current profile [55, 30, 56]. As a strand of DNA passes through a nanopore, the current measured is affected by 5-6 adjacent bases simultaneously. A lookup table is provided by ONT which contains the expected current (in pA) for every possible combination of six bases ("6-mer") [57]. This conversion is demonstrated in Figure 2.6, after which the expected signal is normalized using the mean and standard deviation.

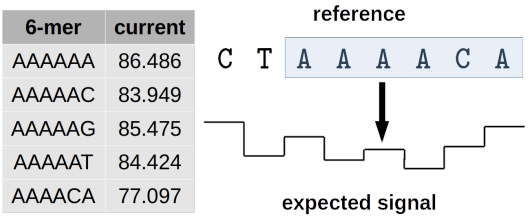

Figure 2.6: Aligning reference bases to expected currents.

2.4.2 Normalizing Query Squiggles

Figure 2.7a shows a contrived minimal example of multiple raw nanopore signals corresponding to the same sequence of bases. Due to a variable rate of DNA/RNA translocation through the

nanopore, these signals are out-of-sync (transitions between current levels do not occur simultaneously). Using Dynamic Time Warping (discussed next) solves this issue, and signals are aligned to the expected signal profile (shown in red in Figure 2.7b). Slight differences in applied bias voltages at each nanopore cause the measured currents to differ significantly, which is why normalization within each read is additionally helpful (Figure 2.7c).

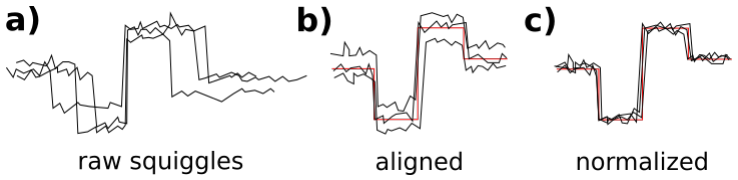

Figure 2.7: **a)** Three raw current measurements ("squiggles") for the same sequence of bases. We then show squiggles aligned to the expected signal **b)** without, and **c)** with normalization.

2.4.3 Subsequence Dynamic Time Warping

Dynamic Time Warping (DTW) is a dynamic programming algorithm which is commonly used to align out-of-sync signals [27, 58]. Our filter applies subsequence DTW (sDTW), a slight modification of standard DTW which allows the entire query signal to align to any small portion of the reference, rather than forcing end-to-end alignment of both sequences.

The original sDTW algorithm works as follows for subsequence query Q of length N, reference sequence R of length M, and scoring matrix S:

```
def sDTW(Q,R):
  S = zeros(N,M)
  S[0,0] = (Q[0]-R[0])^2
  for i in range(1,N):
    S[i,0] = S[i-1,0] + (Q[i]-R[0])^2
  for i in range(1,N):
    for j in range(1,M):
      S[i,j] = (Q[i]-R[j])^2 + min(S[i-1,j-1], S[i,j-1], S[i-1,j])
  return min(S[N,:])
```

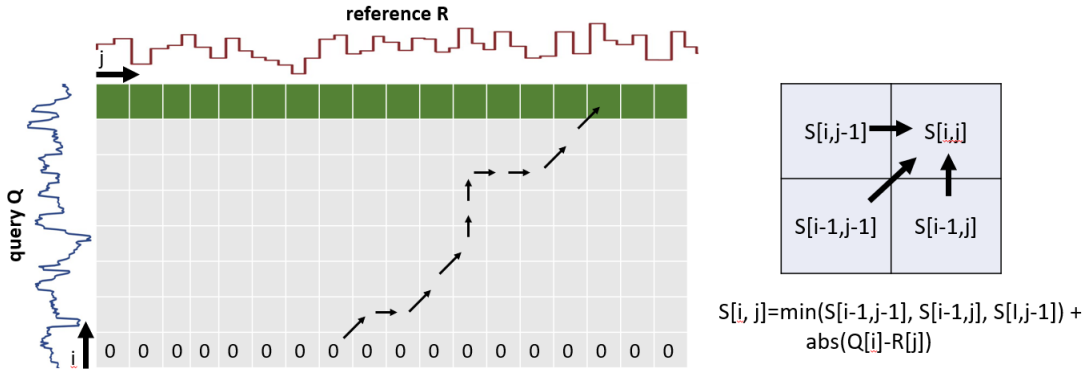

Figure 2.8: Dynamic time warping algorithm.

The above algorithm dynamically computes all possible alignments of the query Q to reference R (keeping only the best ones) while allowing arbitrary many-to-one or one-to-many mappings between the two signal profiles. It is illustrated in Figure 2.8. Matrix S records a running tally of the net squared differences between the two signals (using the best alignment of $Q[0:i]$). At the end, $S[N, j]$ (highlighted top row in Figure 2.8) contains the alignment cost of Q to a subsequence of the reference $R[x:j]$, where x is the start of the best alignment ending at j. The minimum value in this row corresponds to the least squared difference in signal between alignments of the signal to the reference, and thus the cost of the optimal alignment.

2.4.4 sDTW for Virus Detection

The majority of viruses which are responsible for human epidemics have relatively small single-stranded RNA genomes [29], as is demonstrated in Figure 2.9.

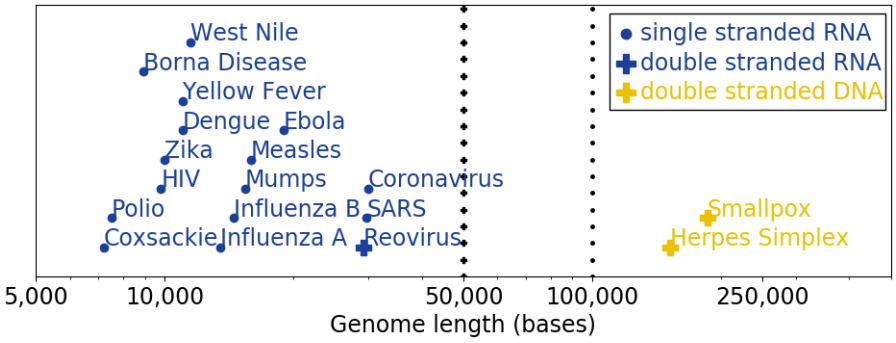

Figure 2.9: Epidemic virus genome lengths.

The two notable exceptions are *smallpox* and *herpes simplex*, which have larger and more chemically stable double-stranded DNA genomes. Because most viruses have small genomes, we design our filter to operate on viruses with single-stranded genomes of length less than 100,000 bases. Equivalently, the filter works on viruses with double-stranded genomes less than 50,000 bases long. At such short reference genome lengths, it is computationally feasible to compare reads to the entire reference genome for filtering. This would not be a feasible solution for complex organisms such as humans, with genomes approximately 3 billion base pairs long.

2.4.5 sDTW is an Effective Filter

We seek to design a solution that is capable of detecting all strains of a particular viral species. It is therefore important that our filter is tolerant to variants in the sequenced genome relative to the reference genome used by our filter. We found that reference-guided filtering can be accurate regardless of viral strain, since the number of mutations between different strains is low. Table 2.2 presents the number of single base mutations between an assembled virus genome for several known SARS-CoV-2 strains, relative to the original Wuhan reference assembly [59]. No insertions or deletions were observed. Strains were defined using NextStrain's [60] classification of all sequenced SARS-CoV-2 genomes into groups of shared ancestors, or "clades", and data was sourced from the GISAID database [61].

Clade	Mut.	GISAID ID	Lab of Origin	Country
19A	23	593737	SE Area Lab Services	Australia
19B	18	614393	Bouake CHU Lab	Ivory Coast
20A	22	644615	Dept. Clinical Microbiology	Belgium
20B	17	602902	NHLS-IALCH	South Africa
20C	17	582807	Public Health Agency	Sweden

Table 2.2: There are few mutations between SARS-CoV-2 strains, relative to the Wuhan reference genome.

Since there are only a handful of mutations between various SARS-CoV-2 strains, the final sDTW alignment cost will not be significantly impacted. This cost is used to determine whether

a given read aligns to the viral reference genome by comparing it to a constant threshold. If the alignment cost exceeds the chosen threshold, then the squiggle did not match well with any subsequence of the reference genome's expected current profile, and the read can be discarded. Figure 2.10 shows that a static threshold can be used to distinguish between viral and human DNA fragments (discarding reads above the threshold and keeping reads below the threshold) even when only a few thousand signals have been captured. Due to the slight overlap in final alignment costs, some reads will be incorrectly classified when using a static threshold.

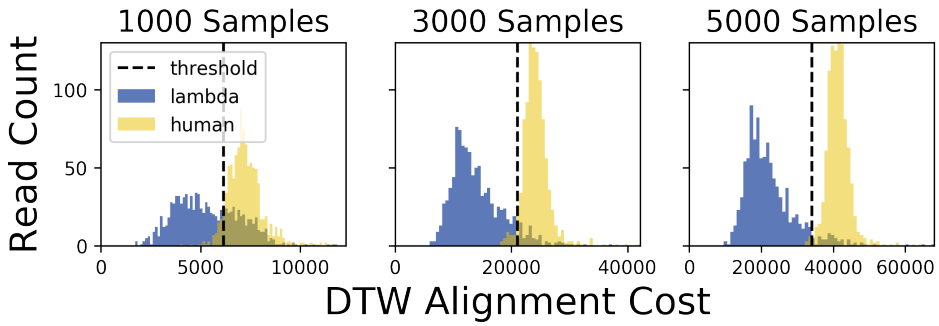

Figure 2.10: sDTW cost distributions for reads of 3 prefix lengths, aligned to the lambda phage genome.

2.4.6 Multi-stage sDTW Filtering

We observed that as a read's sequenced prefix length increases, the sDTW alignment cost is more accurately able to distinguish between target and non-target DNA (there is a decrease in overlap between cost distributions in Figure 2.10). However, waiting to make a Read Until decision increases the proportion of non-target DNA sequenced.

Therefore, instead of a single-stage filter that chooses a constant read length and threshold, we can filter in multiple stages. The first stage examines a shorter read length (e.g. 1000 samples), but chooses a less aggressive threshold that may let many non-target reads through. Non-target reads filtered and ejected using Read Until at this stage would be very short. If a read is retained, it is sequenced further. The second stage then examines the longer read prefix (e.g. 5000 samples), and filters using a more aggressive threshold. Intermediate results can be stored to avoid recomputation.

In this way, several stages enable the classifier to filter a majority of non-target reads after seeing only a short prefix. Only reads with initial low-confidence are sequenced more before a decision is made. We have designed our hardware accelerator with this (optional) capability.

2.4.7 sDTW Algorithm Improvements

We propose several modifications to sDTW which help improve either our accelerator's efficiency or accuracy of non-target read filtering.

Absolute Difference: We reduce hardware area and avoid multiplication by using the Manhattan distance instead of Euclidean (`abs(Q[i]-R[i])` instead of `(Q[i]-R[j])`^{2}).

Integer Normalization: Our solution uses 8-bit fixed point arithmetic during normalization, with no significant impact to classification accuracy (see Figure 2.17).

No Reference Deletions: Since the MinION averages 10 samples per base pair, it is unnecessary during sDTW computation for a single squiggle value to be able to align to multiple bases. We removed the possibility of reference deletions entirely from our dynamic programming computation, so that `S[i,j] = abs(Q[i]-R[j]) + min(S[i-1,j-1], S[i-1,j])`.

Match Bonus: This final modification improves filtering accuracy. We found that reads with higher average translocation rates generally have higher alignment costs. To ensure sDTW alignment costs solely represent quality of alignment and are independent of translocation rate, we implemented a "match bonus" that rewards reads for matching additional reference bases, reducing the alignment cost for each matching base by a constant (10) scaled by the number of signals aligned to the previous reference base (thresholded to 10).

2.4.8 Need for an Accelerator

Despite the reduction in computation when compared to basecalling, sDTW alignment is still too slow to run on commodity hardware. sDTW alignment does avoid expensive floating point operations, instead requiring 8-bit integer comparisons and additions/subtractions. sDTW also has a smaller memory footprint (60,000 reference bases) compared to Guppy-lite (284,000 weights)

when filtering SARS-CoV-2. Despite memory and operation complexity advantages, however, the number of operations required for sDTW (1,400 million) is greater than that of Guppy-lite (141 million). This is still more efficient than Guppy (2,412 million). In order to meet current and future MinION device requirements for Read Until, it is necessary to design an accelerator.

2.5 Accelerated SquiggleFilter

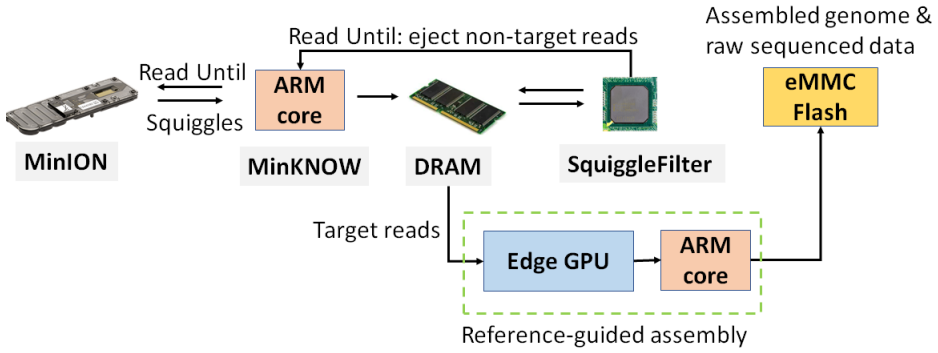

Figure 2.11: System-on-Chip design with the accelerated hardware filter on ASIC integrated with NVIDIA GPU and 8-core ARM v8.2 64-bit CPU

We present a System-on-Chip for reference-guided assembly of target viruses, shown in Figure 2.11. Its capabilities are similar to a Nvidia Jetson TX2, except for our SquiggleFilter accelerator. Our SquiggleFilter accelerator classifies and filters non-target reads, which constitute greater than 99% of all reads in most biological specimens. Thus, a large fraction of computing identified in Section 2.3 is handled by our SquiggleFilter accelerator. Furthermore, our accelerator enables low latency read classification, allowing us to use Read Until to eject non-target reads after sequencing only a short prefix.

Target reads (and any false positives) are processed off of Read Until's critical path. Only these small fraction of reads need to be basecalled, aligned, and variant called. We find that we can perform these computations on an edge GPU (basecaller) and ARM processor (aligner and variant caller), and still construct the whole viral genome in approximately 10 minutes. Unfiltered non-target reads (false positives due to sDTW algorithm) will fail to align to the viral reference genome

after basecalling, and so they will be discarded without affecting the accuracy of conventional reference-guided assembly. The final assembled genome and raw sequencing data is written to a 32GB eMMC 5.1 flash memory, which is sufficient to store one day's worth of sequencing data.

We now present the 1D systolic array based SquiggleFilter accelerator for our squiggle-level classification algorithm discussed in Section 2.4. It can be programmed to target any novel viral genomes less than 100K bases. It supports variable query length. That is, it can classify read prefixes of different lengths, and thereby supports multi-stage filtering. The size of the systolic arrays and buffers are derived from our analysis of real-world metagenomic data.

2.5.1 SquiggleFilter Design

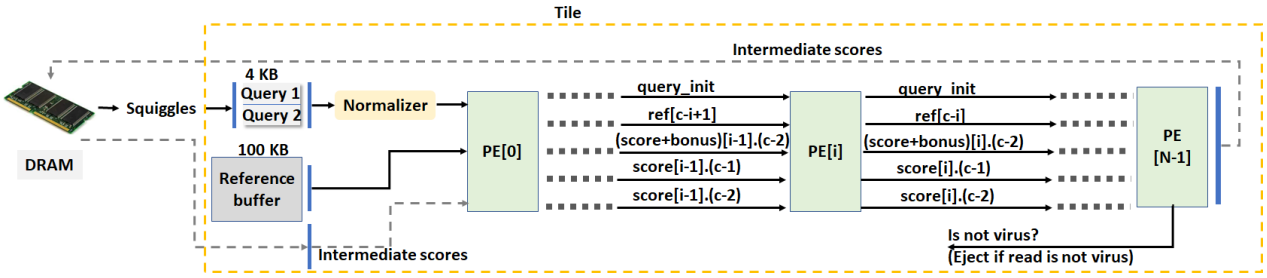

Figure 2.12: SquiggleFilter Tile. N=2000 PEs are connected with streaming inputs and outputs. The last PE determines the classification by comparing its cost to a threshold every cycle. c is the cycle and i is the PE index.

SquiggleFilter consists of 5 independent tiles (one tile is shown in Figure 2.12). Each can be individually power-gated based on desired filtering throughput. This number was chosen to meet the expected $100\times$ future increase in sequencing throughput. Each read is assigned to an available tile for classification. As a read is sequenced, squiggles from a MinION R9.4.1 flow cell are streamed into DRAM in real-time. From there, squiggles are fetched into a tile's query buffers. Two ping-pong query buffers enable simultaneous squiggle loading and normalization. Once the desired length of read prefix has been sequenced, the raw squiggles of a query are normalized and then stored across the processing elements connected in a 1D systolic array.

Each tile also stores a copy of the precomputed reference signal (loaded from flash during

an initialization phase) in a reference buffer. The reference samples are then streamed into the systolic array. The entire sDTW matrix is computed in a wavefront parallel manner as described in Section 2.4.7. The final PE determines the final minimum alignment cost, and sends a control signal to the MinION to eject the read if the final cost exceeds a predetermined threshold. Non-ejected reads are sequenced in full and stored in memory.

The number of cycles required to classify a new read is the read prefix length (2000 samples) plus the reference genome length (60,000 samples for SARS-CoV-2).

Reference Buffer: We chose to use a separate buffer (100 KB) for each tile, even though all the reference buffers across the tiles store the same information (viral genome's reference squiggles). This allows us to reduce access latency and provide sustained throughput to each tile with just one read port. The area cost of duplicating the references is negligible, as reference buffers constitute only 6.98% of total tile area.

Furthermore, our design is independent of reference length and limited only by the reference buffer size provisioned. By loading a new precomputed reference signal onto the on-board flash, SquiggleFilter can easily be reprogrammed to detect a novel virus.

Variable Query Length: As discussed in Section 2.4.6, there exists a trade off between classification accuracy and sequencing length of queries. We find (Section 2.7.4) that read prefix length of 2000 samples yields the most savings using Read Until, when we use a single threshold. Therefore, we use a 1D systolic array of size 2000 PEs.

Our SquiggleFilter design can handle variable read prefix lengths that are multiples of 2000 squiggle samples. To support query lengths longer than 2000 samples and multi-stage filtering, we configure the last PE such that it can optionally write the sDTW costs every cycle to DRAM. This consumes significant memory bandwidth. However, it enables sDTW computation to continue if greater classification accuracy by analyzing a longer prefix is desired. These intermediate costs are then loaded from DRAM and used to initialize the PEs (similar to initial normalized query) prior to computing the costs for a 4000-sample prefix length.

2.5.2 Processing Element

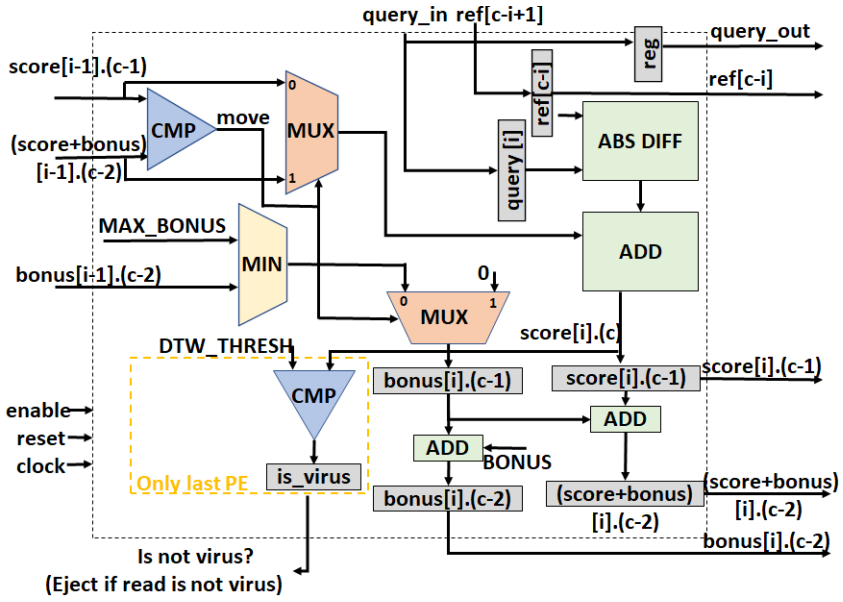

Figure 2.13: SquiggleFilter Processing Element.

Each PE computes a cell in the sDTW matrix every cycle, using the final algorithm described in Section 2.4.7. At cycle c, each PE (Figure 2.13) checks for the minimum among its previous neighbor's $c-1$ and $c-2$ cycle's outputs, modified by a bonus which rewards matching new reference bases. This minimum is then added to the absolute difference of the current query and reference values. Each PE stores the resulting costs and bonuses from its last two cycles for the next PE. Additionally, the last PE contains logic to compare its cost to a predefined threshold which determines whether or not to eject the read. This threshold can be reprogrammed on the SquiggleFilter based on software analysis of the target strain, but we have found it to be relatively robust across species and sequencing runs. Each PE is 1203μm^2 and requires 1.92mW when synthesized for a 28nm TSMC chip.

2.5.3 Normalizer

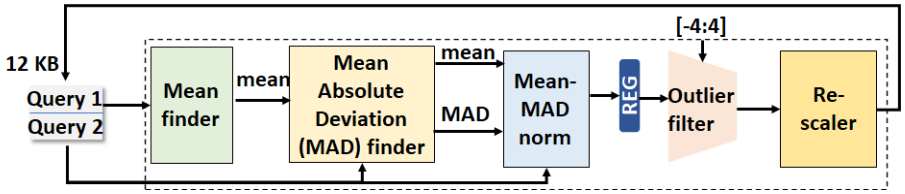

Figure 2.14: SquiggleFilter Normalizer.

Normalization rescales the raw signals in order to improve classification accuracy when performing sDTW [62], as discussed in Section 2.4.2. The normalizer, shown in Figure 2.14, is a query preprocessor which streams in 10-bit samples from the query buffer for accumulation. After every $n = 2000$ samples, the normalizer updates the mean and Mean Absolute Deviation (MAD), defined as follows:

$$\text{mean} = \bar{x} = \sum_{i=1}^{n} \frac{x_i}{n} \qquad MAD = \sum_{i=1}^{n} \frac{|x_i - \bar{x}|}{n}$$

Thereafter, the streamed-in samples are transformed with mean-MAD normalization. The output normalized value is filtered for outliers and then re-scaled to a reduced precision 8-bit integer which is then fed to the tiles for sDTW classification. We find that 8 bits of precision is sufficient for accurate classification (Figure 2.17). For efficiency, we do not convert the ADC sample to floating point, but instead use fixed-point values in the range $[-4, 4]$.

2.6 Implementation

Human DNA datasets containing MinION R9.4 and R9.4.1 flow cells were obtained from the Nanopore Whole-Genome Sequencing Consortium [63] and the ONT Open Datasets [64]. The SARS-CoV-2 dataset contains raw MinION R9.4.1 data available from the Cadde Centre [65]. We sequenced lambda phage DNA in our own laboratory using the ONT Rapid Library Preparation Kit [66] following the Lambda Control protocol with a MinION R9.4.1 flow cell.

We performed basecaller profiling measurements using a Titan XP GPU (server class) and Jetson Xavier GPU (edge class). Their specifications are provided in Table 2.3. We evaluated both Guppy (`dna_r9.4.1_450bps_hac.cfg`) and Guppy-lite (`dna_r9.4.1_450bps_fast.cfg`) without modification using Guppy version 4.2.2 [45]. MiniMap2 version 2.17-r954-dirty [46] aligned basecalled reads.

First, we measured the basecalling throughput of Guppy and Guppy-lite on a dataset of 33,004 full-length reads. Next, we used the proprietary Python libraries `ont-fast5-api` version 3.1.6 [67] and `ont-pyguppy-client-lib` 4.2.2 [68] to basecall the same reads in chunks of 2000 signals, thereby simulating Read Until on the same dataset. The Python code was instrumented to record latency information, and we tuned the number of reads simultaneously in-flight to optimize performance. This online Read Until processing (due to smaller batch size) resulted in 4.05× lower throughput for Guppy-lite and 2.85× lower throughput for Guppy on the Titan XP. Using these measurements and the relative peak throughputs of the Jetson and Titan, the Read Until performance of the Jetson Xavier was estimated (necessitated by the unavailability of `ont-pyguppy-client` ARM binaries for fine-grained Read Until control on the Jetson).

	Edge GPU	**Edge CPU**	**GPU**	**CPU**
Model	Jetson AGX Xavier	ARMv8.2	Titan XP	2× Intel Xeon E5-2697v3
Cores	512 Volta	8	3840 Pascal	56
Clock	1377MHz	2265MHz	1582MHz	2600MHz

Table 2.3: Architectural specifications of evaluated GPUs.

A memory-efficient multi-threaded implementation of sDTW was written in Python for accuracy analysis, and tested on 1000 reads from each of the datasets mentioned above. In order to determine the relative benefits of Read Until using different classification latencies and accuracies, we developed an analytical model to estimate sequencing runtime. This model accounts for factors such as average read length, desired coverage of the reference genome, average DNA capture time, and the Read Until parameters mentioned previously.

The design was first functionally verified via emulation on Amazon Web Service's EC2 F1

instance, which uses a 16nm Xilinx UltraScale+ VU9P FPGA. Further, SquiggleFilter was synthesized using the Synopsys Design compiler for 28nm TSMC HPC and the design is clocked at 2.5GHz. 32GB 256-Bit LPDDR4x is connected to the System-on-Chip along with an 8-core ARM v8.2 64-bit CPU.

2.7 Results

2.7.1 SquiggleFilter Hardware Synthesis

ASIC Element	Area (mm^2)	Power (W)
Normalizer	0.014	0.045
Processing Element	0.001	0.002
Tile (1×2000 PEs)	2.423	2.780
Query buffer	0.023	0.009
Reference buffer	0.185	0.028
Complete 1-Tile ASIC	2.65	2.86
Complete 5-Tile ASIC	13.25	14.31

Table 2.4: SquiggleFilter ASIC synthesis results.

Table 2.4 shows SquiggleFilter synthesized to a 13.25mm^2 ASIC that consumes 14.21W when performing single-stage filtering and clocks at 2.5GHz. It contains 5 fully-independent tiles (which could be individually power-gated to improve energy efficiency). The latency for classifying a 2000-sample read from SARS-CoV-2 is 0.027ms, and for lambda phage is 0.043ms, due to its longer reference genome. This adds insignificant latency to each Read Until decision's critical path, since it takes around 500ms to sequence a sufficient number of bases to make an accurate decision. The single-tile classification throughputs for SARS-CoV-2 and lambda phage are 74.63M samples/s and 46.73M samples/s respectively, which are both considerably higher than MinION's current maximum output of 2.05M samples/sec). Additionally, if each tile is configured to perform multi-stage filtering, it will write intermediate results to DRAM, consuming only 10 GB/s main memory bandwidth per tile. Since Jetson Xavier's main memory supports 137 GB/s, our 5 tile

design is feasible.

2.7.2 Performance Analysis

Latency: Figure 2.15a compares GPU-based basecalling latency to our SquiggleFilter accelerator's latency. Note that we show only basecalling latency as it is the most time consuming step (96% of compute time) of the virus classification pipeline. The measurements demonstrate that it would be impractical to use the high-accuracy Guppy basecaller as its latency is greater than one second, in which time more than 400 bases would have been unnecessarily sequenced for non-target reads. We found that Guppy-lite provides sufficient accuracy for Read Until classification as downstream aligner MiniMap2 is able to account for incorrect basecalls when aligning reads. However, a 149ms basecalling latency for Guppy-lite translates to an additional 60 bases sequenced for each read during classification. Since most non-target reads can be discarded after around 200 bases, this overhead is significant. In comparison, the common-case 0.04ms decision latency of SquiggleFilter ensures that not even a single base pair is unnecessarily sequenced.

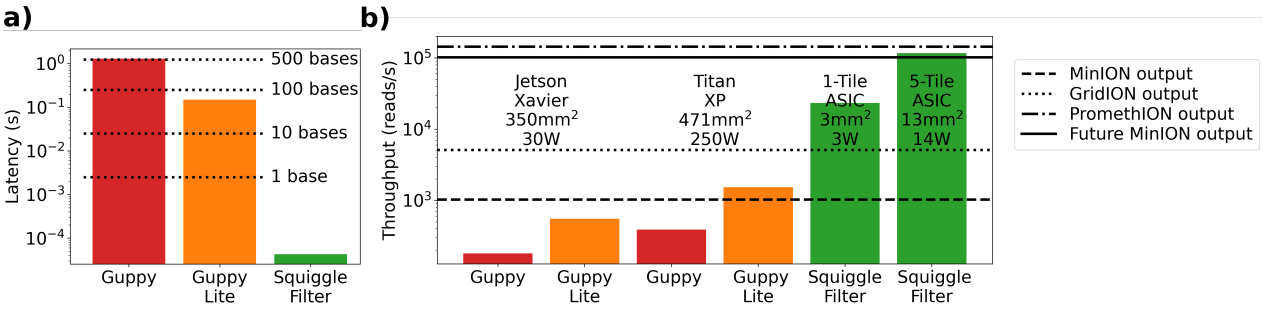

Figure 2.15: **a)** Latency, and **b)** throughput of Guppy, Guppy-lite and SquiggleFilter during Read Until.

Throughput: Figure 2.15b compares the basecalling throughput of Guppy-lite measured over GPU configurations to SquiggleFilter accelerator's classification throughput. An edge GPU such as the Jetson does not have sufficient compute power to basecall data from all pores in real-time and keep up with the maximum sequencing throughput of the MinION. We calculated that the Jetson's throughput would be approximately 95,700 bases per second, which is only 41.5% of the

MinION's maximum output of 230,400 bases per second. In the worst case, Read Until can only be performed using 41.5% of the MinION's pores when basecalling using Guppy-lite on the Jetson. The remaining 59.5% of pores are unable to use Read Until, and will sequence full-length human reads. In contrast, SquiggleFilter's throughput far exceeds MinION's and GridION's sequencing throughputs.

2.7.3 sDTW Algorithm Accuracy

Figure 2.16a compares sDTW accuracy to basecalling and alignment on a dataset of 1000 lambda phage and 1000 human reads, with a line plotted for each prefix length. The MiniMap2 alignment quality and sDTW alignment cost thresholds (for determining which reads to sequence and which to reverse) are swept through the range of possible values to show threshold-dependent accuracies. Although the Read Until accuracy obtained by basecalling and aligning slightly outperforms sDTW, this is to be expected since alignment algorithms such as MiniMap2 use numerous scoring heuristics and have matured significantly over the past two decades [46].

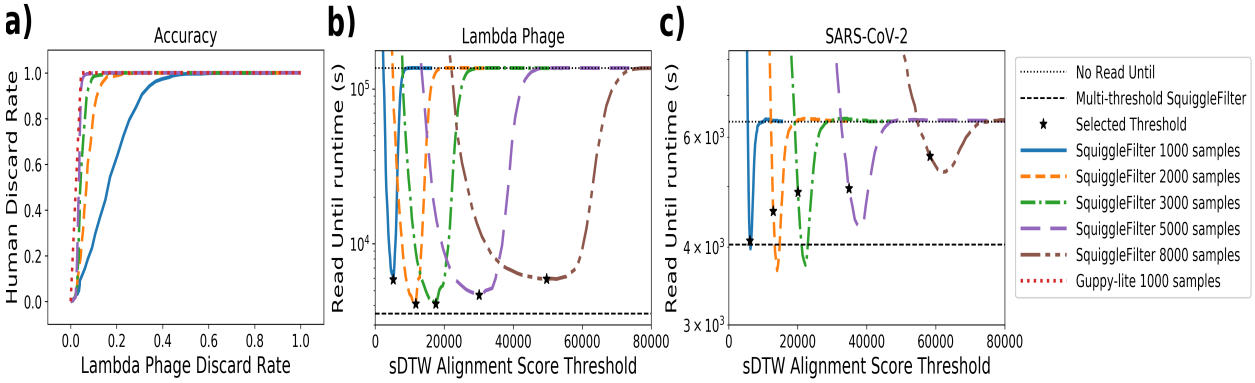

Figure 2.16: SquiggleFilter Read Until **a)** accuracy, and performance on **b)** lambda phage and **c)** SARS-CoV-2 datasets.

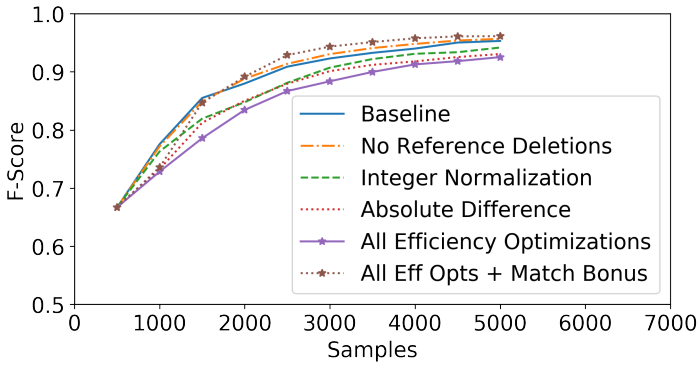

Figure 2.17: Accuracy results for modifications to the standard sDTW algorithm.

Figure 2.17 shows the maximal F-score for all of our algorithm modifications and standard sDTW on the same dataset. As expected, accuracy generally increases along with sample prefix length. We found that using both integer normalization and absolute difference for our distance metric reduce filtering accuracy slightly, a compromise which was expected. Eliminating reference deletions results in a slight accuracy improvement. Combining all three of these optimizations results in the lowest accuracy (but most efficient) of all configurations tested. We find that by including our "match bonus", we can recover lost accuracy and outperform the baseline, with a minor performance penalty. Figure 2.18 furthermore demonstrates that there is no a significant loss in filter accuracy until there is more than a 1,000 base difference between the reference genome and viral strain sequenced.

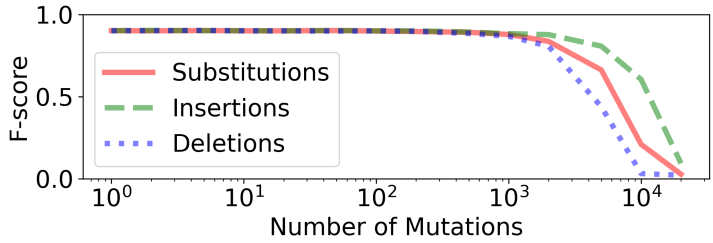

Figure 2.18: SquiggleFilter accuracy is robust against random (lambda phage) reference mutations.

2.7.4 Benefits of Read Until

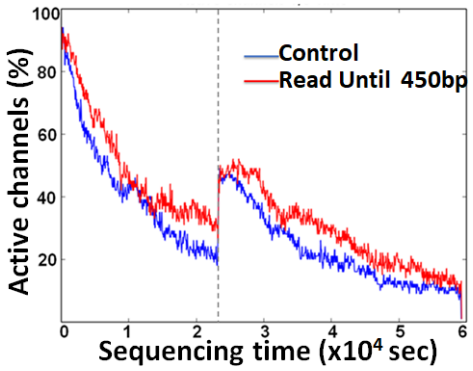

Figure 2.19: Time saved is cost saved for sequencing.

Read Until not only saves sequencing time, but also cost. Figure 2.19 shows our wet-lab experiment. After sequencing for a while, washing the flow cell with nuclease and re-multiplexing (rapid alternations of pore voltage bias direction, shown with dotted black line) leads to control and Read Until pores having the same number of active channels. This means that Read Until does not damage the flow cell any more than normal sequencing, but enables more experiments to be run over the lifetime of any flow cell.

The single-threshold Read Until design space was first explored for our lambda phage dataset. Figure 2.16a shows the accuracy of SquiggleFilter for a variety of Read Until prefix lengths (each line), and for all reasonable sDTW alignment cost thresholds (points on each line). Given this experimentally measured accuracy, the total expected sequencing time to perform Read Until for lambda phage was calculated using our analytical model, and is shown in Figure 2.16b. We found that the best single-threshold configuration for SquiggleFilter outperforms Guppy-lite on this dataset by 12.9% in terms of Read Until runtime. By using multiple thresholds, we can reduce runtime by a further 13.3%.

A similar analysis was then performed for the SARS-CoV-2 dataset, and the results are shown in Figure 2.16c. Optimal sDTW alignment cost thresholds were taken from the Read Until runtime minima from Figure 2.16b, and the corresponding Read Until runtimes using those thresholds are marked for the SARS-CoV-2 dataset.

2.7.5 Looking Forward: Scalability

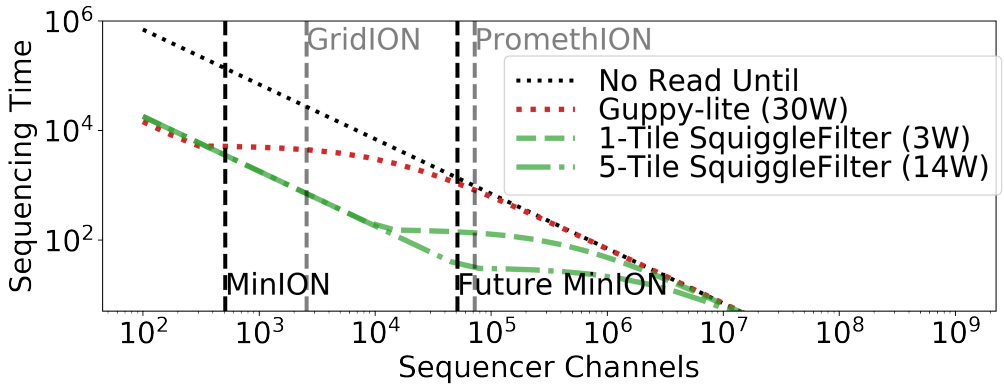

Figure 2.20: Future SquiggleFilter Read Until benefits.

Sequencing throughput is expected to increase by $10-100\times$ within the next few years, due to new nanopore chemistry enabling a denser configuration with many more channels per flow cell [49]. Figure 2.20 shows that without further improvements to basecalling throughput, current GPUs will be unable to keep pace with new sequencing technology. As a result, the time and cost savings gained through Read Until will be largely lost. We can see that Guppy-lite's slight edge over SquiggleFilter in terms of accuracy has already been lost due to its inability to perform Read Until on 512 pores. In contrast, our SquiggleFilter accelerator can tolerate a $114\times$ increase in sequencing throughput.

2.8 Related Work

The MinION was released in 2014 as the first commercially available nanopore-based DNA/RNA sequencing device [69]. The first Read Until software pipeline was developed two years later, in 2016 [30]. In this seminal work, raw nanopore signal was first segmented into events, and then events were aligned to a lambda phage reference using subsequence Dynamic Time Warping (described in Section 2.4.3). Event segmentation is used to detect the most likely positions in the raw signal where a new base has entered the pore, and could be considered a rudimentary form of basecalling. In fact, it has been used as an essential preprocessing step in several older

basecallers [45]. Unfortunately, the throughput measured by this original work on an 8-core ARM processor is 40× lower than the current maximum MinION output.

As basecalling throughput and accuracy has gradually increased over the last few years, the standard approach for Read Until pipelines has been to basecall the signal and use an aligner to determine if each read aligns to the target genome [52, 53, 70, 51]. This method achieves the highest accuracy, but is not scalable. When pairing a server-class GPU with a handheld MinION device, it is just able to perform Read Until with the required throughput, albeit with significant latency (as shown in Section 2.7.2).

UNCALLED, a more recent work, skips basecalling by doing approximate alignments in 3 steps: event segmentation, FM-index look-ups, and seed clustering [54]. However, we evaluated UNCALLED and observed that it requires longer prefix lengths for accurate alignment. 23.63% of 2000-sample long chunks from our lambda phage dataset were not alignable. After segmentation, UNCALLED uses an FM-index to filter reads. UNCALLED aligns only ∼76% of the lambda reads of 2000 samples on a modern Intel i7-7700 desktop processor taking 16ms per read. Moreover, ∼14% of reads take 353ms per read to be aligned as more samples are required for a decision. ∼10% of the reads, however, are left unaligned. On an edge device with an ARM core and lower memory bandwidth, performance would be worse. No existing software-only solution has adequate throughput and low enough latency to effectively perform Read Until on an edge device.

In contrast, our approach shifts to a minimalistic sDTW alignment algorithm, and by designing hardware to accelerate the simple and regular sDTW computation, we can easily meet the desired throughput and latency requirements on an edge device. General purpose DTW accelerators have already been designed to solve alignment problems in other domains such as audio signal processing [71] and astronomy [62], but nanopore viral DNA/RNA filtering required several application-specific optimizations to meet the desired latency, throughput and accuracy requirements. Our design involves several algorithmic modifications to vanilla sDTW (described in Section 2.4.7), uses an on-chip buffer for efficient repeated alignments to the same reference, replaces all floating-

point computation with integer arithmetic for increased efficiency, uses multi-stage filtering for optimal Read Until results, and has been evaluated on a novel virus (SARS-CoV-2).

There has recently been significant work on designing hard-ware accelerators for genomics applications [19, 20, 21, 22, 23, 24, 25, 26], but these accelerators focus on human genome sequencing. As a result, they efficiently align many (usually short) basecalled reads to a long reference genome with high throughput and accuracy. As noted previously in Section 2.3.2, our problem has very different computational needs. We must selectively filter short noisy raw signals (squiggles) with sufficiently high throughput and low latency to effectively exploit Read Until. We achieve this by replacing the basecaller and aligner with SquiggleFilter.

2.9 Availability

The software and RTL design of SquiggleFilter are publicly available at: `https://github.com/TimD1/SquiggleFilter`

CHAPTER 3

DTWax: Accelerated Dynamic Time Warping on GPU for Selective Nanopore Sequencing

3.1 Introduction: SquiggleFilter's limitations

With SARS-CoV-2 evolving and adapting to its new environment[72] and becoming immune-evasive[73], there is a possible threat from a variant that can evade our current gold standard tests and fuel a surge in cases. Reverse Transcription Polymerase Chain reaction (RT-PCR) is the current gold standard[74] for SARS-CoV-2 diagnostic testing. Prior works have shown that RT-PCR requires the design and manufacture of custom PCR primers which is a complex, time-consuming, and error-prone process[75, 13, 12]. This limits the utility of RT-PCR in the early stages of a pandemic. Dunn and Sadasivan et al.[75] developed SquiggleFilter, a portable virus detector that could be re-programmed to speed up the sequencing of reads from a viral target of interest. SquiggleFilter is an ASIC, envisioned to work alongside Oxford Nanopore Technology's (ONT) MinION MK1B (or simply the MinION), a recent-to-market portable DNA sequencer that does not have any compute built into it. However, SquiggleFilter can only be programmed with references of size less than 100Kb and it being an ASIC, is not easily scalable.

Additionally, GPUs are becoming a more common choice for accelerated computing on sequencers– ONT sequencers GridION, PromethION, and MinION MK1C have GPUs built into them[76]. GPUs are also widely available in workplaces and on cloud platforms. While SquiggleFiter's subsequence Dynamic Time Warping (sDTW) algorithm was optimized to work on an

ASIC, we adapt and optimize it to work on the more common GPUs.

3.2 Background: Compute and Accuracy challenges in Read Until

3.2.1 Guppy is inaccurate and slow for Read Until

The state-of-the-art selective sequencing pipeline[77, 53] uses Guppy-fast for basecalling and Minimap2 for classifying the reads[77] as shown in Fig. 4.3. Guppy is a deep neural network-based software that converts the raw signal output of the MinION (noisy squiggles) to bases. However, Guppy has a two-fold performance problem. Prior works have demonstrated how Guppy is slow and does not have the required throughput to handle the throughput of the MinION[75, 78, 79, 80] and SquiggleFilter[75] pointed out how the increasing throughput of the MinION amplifies this problem. We demonstrate the same problem in Fig. 3.2b. Secondly, Guppy is unable to accurately basecall small chunks of data. ∼40% of the bases sequenced from a sample of average read length 2 Kbases is unclassified as shown in Fig. 3.2a because Guppy could not basecall these very short fragments accurately[80].

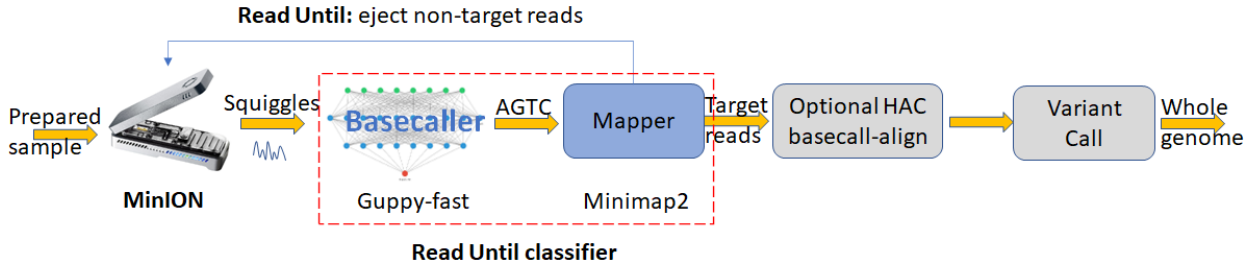

Figure 3.1: State-of-the-art selective sequencing pipeline uses Guppy-fast for basecalling and Minimap2 for classifying the reads

Hardware-accelerated SquiggleFilter[75] was proposed as a replacement for the Read Until classification using Guppy followed by Minimap2.

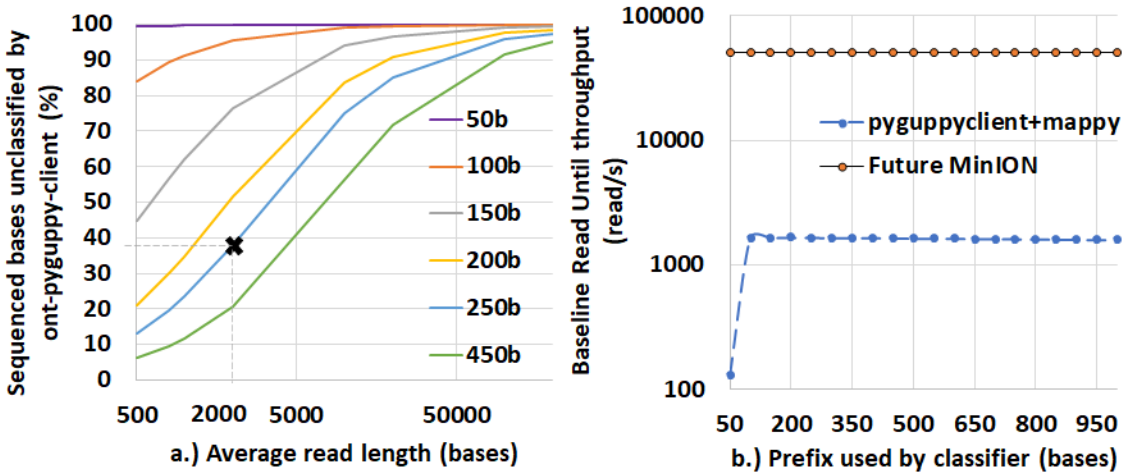

Figure 3.2: Guppy cannot classify a high percent of sequenced bases and also has a throughput problem with Read Until. (a) ∼40% of the bases sequenced are non-target in a 99.9: 0.1 non-target: target mix with an average read length of 2 Kbases. (b) Guppy followed by Minimap2 cannot match the throughput of a future MinION even on a high-end cloud instance that uses an A100 GPU for basecalling.

3.2.2 SquiggleFilter

While traditional RT-PCR tests rely on complex custom primer design and time-consuming wet-lab processes for target enrichment, MinION can be controlled to selectively sequence only the target virus of interest using the Read Until feature. Utilizing the Read Until feature requires making real-time classifications during sequencing but the current MinION does not have any compute power. Dunn and Sadasivan et al.[75] demonstrated how basecalling is the bottleneck and constitutes ∼88-96% of Read Until assembly and how this problem was amplified with the projected 100X increase in ONT's sequencing throughput. Their solution, SquiggleFilter[75], uses hardware accelerated subsequence Dynamic Time Warping (sDTW) to perform Read Until.

SquiggleFilter addresses the compute bottlenecks in portable virus detection and is designed to even handle the higher throughput of a future MinION. SquiggleFilter is programmable and offers better pandemic preparedness apart from saving time and cost of sequencing and compute. However, SquiggleFilter's limited on-chip memory buffer only lets it test for viral genomes smaller than 100Kb. Additionally, SquiggleFilter uses a modified version of sDTW algorithm where the accuracy dip from various hardware-efficiency focussed optimizations are overcome with the match

bonus[75]. The match bonus is a solution to a problem on the ASIC and performing this on the GPU can introduce branch divergence. We eliminate the match bonus, retain the assumption of reference deletions and optimize sDTW to run on GPUs.

3.2.3 subsequence Dynamic Time Warping

sDTW is a two-dimensional dynamic programming algorithm tasked with finding the best map of the whole of the input query squiggle in the longer target reference. In sDTW's output matrix computation, parallelism exists along the off-diagonal of the matrix and therefore, the computation happens in a wavefront parallel manner along this off-diagonal. Diagonals are processed one after the other. If the query is assumed to be along the vertical dimension of the matrix and the target reference along the horizontal dimension, the minimum score on the last row of the matrix will point to the best possible map of the query to the reference. This score may be compared to a threshold to figure out if the query is a target or not. The sDTW cost function is defined as follows:

Algorithm 1 sDTW algorithm

Input: Query (Q) of length N, reference (R) of length M and score threshold (T) for classification.

Output: Target or non-target

```
S ← zeros(N,M)
S[0,0]← (Q[0]-R[0])^2
for i ← 1 to N step: 1 do
    S[i,0]← S[i-1,0] + (Q[i]-R[0])^2
for i ← 1 to N step: 1 do
    for j ← 1 to M step: 1 do
        S[i,j]← (Q[i]-R[j])^2 + min(S[i-1,j-1],S[i,j-1],S[i-1,j])
if min(S[N,:]) < T then
    return "Q is target"
else
    return "Q is non-target"
```

3.2.4 Prior Work

Since the MinION's release in 2014, there has been a few attempts in improving the benefits of Read Until[75, 78, 79, 77, 53, 81, 80], of which only SquiggleFilter[75] has the optimal combination of accuracy and throughput to keep up with a future MinION. The softwares performing Read Until can be broadly classified into two categories based on the inputs they operate on- signal space and basecalled space. Softwares operating in the signal space attempt to classify the input squiggle as a target or non-target while softwares operating in the basecalled space rely on the basecaller to transform raw squiggles to bases in real-time which is computationally expensive. We observe that the basecaller also basecalls smaller signal chunks poorly. The basecalled read prefixes may then be classified as a target or non-target.

Readfish[77] and RUBRIC[53] classify basecalled reads to detect targets using mapping tools like Minimap2. We show that ONT's proprietary basecaller Guppy suffers from not being able to basecall $\sim$10% of the reads confidently leading to them being unclassified by Minimap2. Additionally, Guppy, a deep neural network, also suffers from low throughput on high-end GPUs and cannot meet the real-time compute requirements of a future MinION.

Three of the signal space-based methods rely on event segmentation- a pre-processing step where raw squiggles are segmented into events to detect positions in the signal where we are more likely to see a new base. The very first attempt at Read Until[30], UNCALLED[78], and Sigmap[79] use event segmentation as a pre-processing step. Loose et al.[30] uses sDTW on python to perform Read Until from events on a CPU. This yields sub-optimal performance. UNCALLED follows up with FM-index look-ups and seed clustering to find a target map. Although UNCALLED has a good mapping accuracy for smaller genomes, We observe that UNCALLED does not have the necessary throughput to match the compute requirements of a future MinION. Sigmap does seeding followed by Minimap2-style chaining on the seeds to identify a target map. But we observe that Sigmap needs a relatively long read prefix to identify a sufficient number of seeds and this turns out to be 4000 raw samples. Additionally, we observe that Sigmap has lower mapping accuracy than UNCALLED.

SquiggleNet[82] is a convolutional neural network-based software for classifying squiggles into target or non-target. However, SquiggleNet[82] is slower than guppy followed by Minimap2 and only achieves similar mapping accuracy to Guppy followed by Minimap2. SquiggleFilter[75] is a programmable ASIC that can match the throughput of a future MinION and yield optimal Read Until benefits. However, the initial cost of adoption is high as ASIC needs to be economically manufactured at scale and shipped in order to be deployed worldwide. GPUs on the other hand are already widely available at workplaces, shipped along with some of the sequencers, and also available on the cloud.

DTW has been parallelized in the past for various different applications on architectures including FPGAs[83, 81], Intel Xeon Phis[84], big data clusters[85], customized fabrics[86] and even GPUs[87, 88]. HARU[81] is a recent work that implements SquiggleFilter's algorithm on a budget-constrained FPGA. HARU cannot match the maximum throughput of the current MinION and is not a solution that can match the planned 100X throughput of the MinION. Crescent[89] is a recent closed-source implementation of SquiggleFilter's algorithm directly on the GPU but ends up being 29.5X lower in throughput than DTWax possibly because of several reasons including not utilizing warp synchronized register shuffles for data sharing between threads and fewer number of cells computed per thread. cuDTW++[90] is the best-performing prior work on GPU which accelerates DTW. However, cuDTW++ is $\sim$2.6X slower than DTWax and is built for database querying of very small queries and not for subsequence Dynamic Time Warping that is required to perform Read Until. Additionally, the normalization step is performed very inefficiently.

3.2.5 Our contributions

In this work. we present DTWax, a GPU-accelerated sDTW software for nanopore Read Until to save time and cost of nanopore sequencing and compute. We adapt SquiggleFilter ASIC[75]'s underlying sDTW algorithm to suit a GPU in order to overcome the limitation with reference lengths SquiggleFilter had. While sDTW was optimized for integer compute on the ASIC, we fine-tune sDTW for high throughput on the GPU. While SquiggleFilter uses integer arithmetic

and Manhattan distances on the ASIC, we use floating point operations and Fused-Multiply-Add operations on the GPU. We also demonstrate how to utilize some of the GPU's high throughput tensor core's compute power for non-ML workloads.

As a first step, we speed up the online pre-processing step (normalization) on FP32 tensor cores using the batch normalization functionality from the CUDNN library traditionally used for machine learning workloads. DTWax is optimized to make use of the high throughput Fused-Multiply-Add instructions on the GPU. Further, we use FP16 and FP16 tensor core's Matrix-Multiply-Accumulate (MMA) pipe for higher throughput for sDTW calculation. Using FP16 helps us process the forward and reverse strands, thereby extracting more parallelism to help improve the latency and throughput of classification. We also make use of offline pre-processing of reference squiggle index for coalesced loads, cudastreams for better GPU occupancy, intra-, and inter-read parallelism, register shuffles, and shared memory for low-overhead communication while processing the same query.

DTWax achieves $\sim$1.92X sequencing speedup and $\sim$3.64X compute speedup: costup from using nanopore Read Until for a future MinION (with 100X the current throughput) on an A100 compared to a workflow that does not use Read Until.

3.3 Methods

3.3.1 Offline pre-processing

ONT has published a k-mer current model[91] which provides a reference to map a 6-mer to an expected value of the current output from the MinION. We use this k-mer model to map the reference genome of the target virus to a noise-free FP16 squiggle reference. The squiggle reference will be of length (target_length - 6 + 1). We also pack two FP16 values (one from the forward strand and another from the reverse strand) into a __half2 reference word (built-in CUDA datatype of two FP16 half-words). Further, we make use of the prior knowledge of the target reference to ensure coalesced global memory reads by re-ordering the target reference offline.

3.3.2 Online pre-processing: Normalization

The output squiggle of the MinION (query) is read from ONT's proprietary FAST5 file format. The raw integer data is then scaled to pico-amperes (float32). The first few samples (1000) are trimmed to cut adapters and barcodes off. We re-purpose the CUDNN-Batchnorm to z-score normalize the 1-dimensional FP32 query current signal. CUDNN utilizes tensor cores and performs normalization at a very high throughout (∼6X the throughput of sDTW). The signal is then rounded off to FP16 and copied into a __half2.

3.3.3 DTWax: architecture

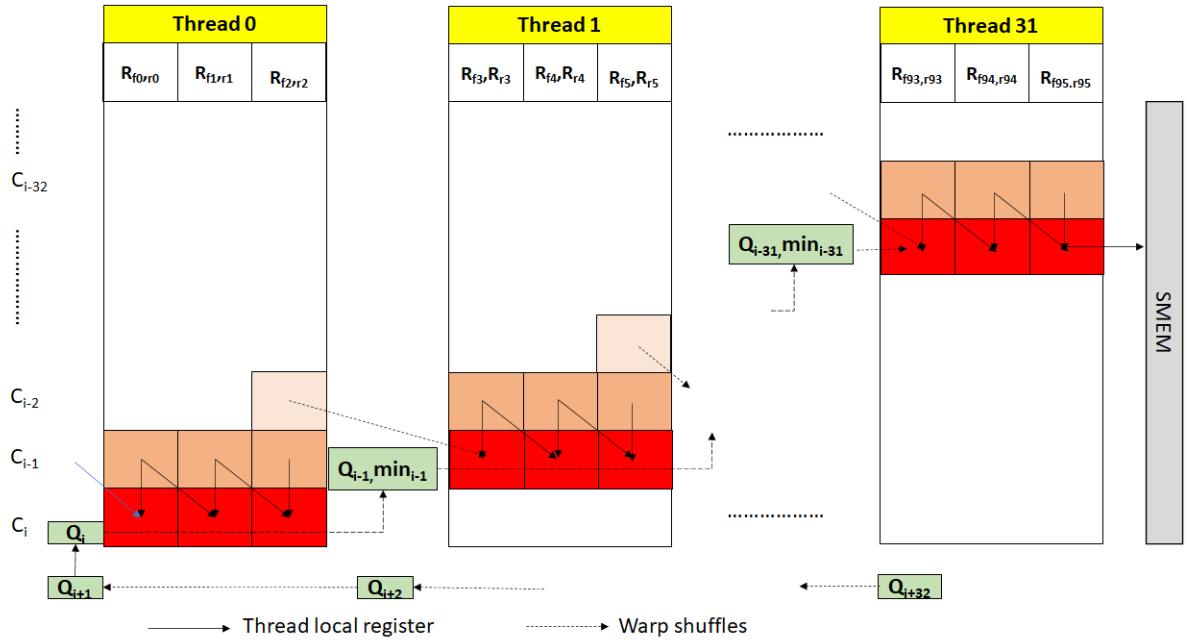

Figure 3.3: Efficient intra- and inter-matrix communication in DTWax

We adopt the segmented-sDTW architecture introduced in prior works[90] where each segment is a fixed number of cells in a row whose scores are computed by a thread. DTWax breaks down the processing of longer target references into multiple sub-matrices, each processing a fixed number of target bases. The reference length processed per sub-matrix is configurable and is set to 832 bases for optimal performance on an A100. Within a sub-matrix, each thread is responsible for pro-

cessing a configurable but equal number of cells (cells per thread is called a segment). Wavefront parallelism exists along the off-diagonal segments in the sub-matrix as shown in Fig.3.3. Thread 0 is the first to finish its computation inside the sub-matrix while thread 31 is the last to finish. Target reference is loaded into registers (one FP16 reference sample each for forward and reverse strands into a single __half2 datatype) from global memory using coalesced loads.

For intra-sub-matrix communication, we exploit warp shuffles for efficient register-to-register transfers within the same warp. This is an idea demonstrated by Schmidt et al[90] but not completely explored. Threads in a warp use warp shuffles to transfer the query sample, the minimum score of the segment, and the score of the last cell in the segment to the thread on its right. Instead of using a global reduction to find the final minimum score for DTWax, we use the efficient warp-shuffles to pass the minimum scores of the segments between threads. Inter-sub-matrix communication happens via shared memory transfers instead of relying on global memory. A thread block processing a read writes the last column of the sub-matrix into the shared memory and reads it back while calculating the consecutive sub-matrix for the same read.

3.3.4 Intra- and inter-read parallelism

Using all the warps on an SM to process a single query would mean that the last warp remains idle and is ineligible for compute for an initial period of time. Therefore, we choose to process one read with a thread block of only 32 threads. We have intra- and inter-read parallelism. Every query is processed by a thread block of 32 threads. Within a thread block, we have intra-read parallelism from 32 parallel threads each computing a segment of the sub-matrix. Across the GPU, we have inter-read parallelism as there as multiple concurrent blocks operating on different reads on any given Streaming Multi-processor (SM).

3.3.5 Coalesced global memory access

The offline re-ordering of the target reference enables us to perform coalesced reads from global memory (as many loads as the length of one segment in a sub-matrix) before computation starts in

the sub-matrix. The normalized target reference is an array of __half2 datatype. This enables the vectorized processing of the input query signals on the high throughput FP16 pipe. Additionally, after the normalized query is read from the global memory in chunks of 32 __half2 query samples using a coalesced load of 128B, it is then efficiently transferred between threads of a warp using warp shuffles.

3.3.6 FP16 for 2X throughput

SquiggleFilter[75] has demonstrated that the information from the ONT sequencer may be captured using 8 bits. While the ASIC was custom-designed for integer arithmetic, GPUs are designed for high throughput floating point arithmetic. Among the floating point pipes available, we use the high throughput FP16 pipe on A100 (2X throughput compared to FP32) for DTWax. Computation with respect to the forward strand of the target reference happens on the first FP16 lane while the second FP16 lane computes with respect to the reverse strand. Utilizing __half2 FP16 pipes (FP16 vectorization) not only helps us to increase throughput but also improves the latency by 2X because we concurrently process both the forward and the reverse strand of the target with respect to the query in every cell of the sub-matrix.

3.3.7 Utilizing tensor core pipe

HFMA2.MMA pipe on the tensor core has one of the highest throughputs on A100. We reformulate the addition in the cost function of DTWax to a Fused Multiply-Add operation in order to utilize the otherwise under-utilized HFMA2.MMA pipe. We are then able to better throttle the compute instructions between HFMA2.MMA and the remaining FP16 pipes instead of increasing the traffic on the FP16 pipe.

3.3.8 Assuming no reference deletion

Using the same assumption from SquiggeFilter[75] that viral strains have minimal reference deletions, we observe our accuracy of mapping using DTWax improves and our new cost function becomes simpler as we now only have to find a single minimum per cell instead of two minimums. Line 7 from Algorithm 1 is simplified to:

$$S[i,j] \leftarrow (Q[i]\text{-}R[j])^2 + \min(S[i\text{-}1,j\text{-}1],S[i\text{-}1,j])$$

3.3.9 Optimizing occupancy and branch divergence

We ensured high SM utilization by finding the right balance between the number of resident warps on the SM and shared memory utilization. Further, we keep the GPU occupancy high by issuing concurrent asynchronous workloads to the GPU using cudastreams. Memory transfers to and from the GPU are overlapped with compute on the GPU.

We reduce the branch divergence via partial loop unrolling. The first sub-matrix does not read from shared memory and the last sub-matrix does not write into shared memory. Unrolling the first and last sub-matrix computations of the query-target matrix helps improve performance.

3.3.10 Configurability and scalability

DTWax can be reprogrammed to test for any target reference of interest. Unlike some of the prior works[90, 75], DTWax can be reporgrammed to test for longer target references. Further, one may easily try and scale DTWax across multiple GPUs for higher throughput on longer or multiple target references.

3.4 Implementation

3.4.1 Experimental setup

For all our GPU evaluations we use an NVIDIA A100 GPU on Google Cloud Platform's (GCP) a2-highgpu-1g instance with 40GB of memory and 85GB of host memory. Our CPU baselines are evaluated with hyper-threading enabled on GCP c2-standard-60 (30 Intel Cascade Lake cores and 240 GB of memory). pyguppy-client and DTWax use GPU while other software use CPU.

NVIDIA Nsight Systems [92] is used to visualize concurrent CUDA events, and NVIDIA Nsight Compute[93] is used to profile GPU events.

We use public datasets used by SquiggleFilter[75]. We sequenced lambda phage DNA on a MinION R9.4.1 flow cell following the Lambda Control protocol at the University of Michigan's laboratory using the ONT Rapid Library Preparation Kit [66]. Human datasets (sequenced with MinION R9.4 and R9.4.1 flow cells) are obtained from ONT Open datasets [64] and the Nanopore Whole-Genome Sequencing Consortium [63].

We use UNCALLED v2.2, Sigmap v0.1, Minimap2 v2.17, pyguppy-client v0.1.0 and Centrifuge v1.0.4-beta. We optimally configure the software for better mapping accuracy. We configure Sigmap for better event detection by setting "–min-num-anchors-output 2 –step-size 1". We turn off minimizers in Minimap2 for better mapping accuracy by setting "-w=1 -k=15".

3.4.2 Optimal GPU configurations

In order to optimize DTWax's throughput on A100, we introduce more inter-read parallelism by fitting 32 blocks on each of the 108 Streaming Multiprocessors on the GPU. We process one read per thread block of size 32 threads. We observe that having multiple concurrent warps per read resident on the SM may not be beneficial as there may be a long latency in the last warp of a read getting valid inputs from the previous warp and starting to calculate useful values. Using Nsight Compute, we maximize the number of reference bases processed per thread (segment size) to 26 amounting to a total of 32x26 reference bases processed per thread block. In order to reduce

global memory transactions, we use shared memory for inter-sub-matrix communication. A shared memory of size 500B is allocated per block to store the output of the last column of a sub-matrix calculated by the last thread in a thread block. This is read by the thread block when it processes the next set of 32x26 reference bases.

3.4.3 Incremental Optimizations

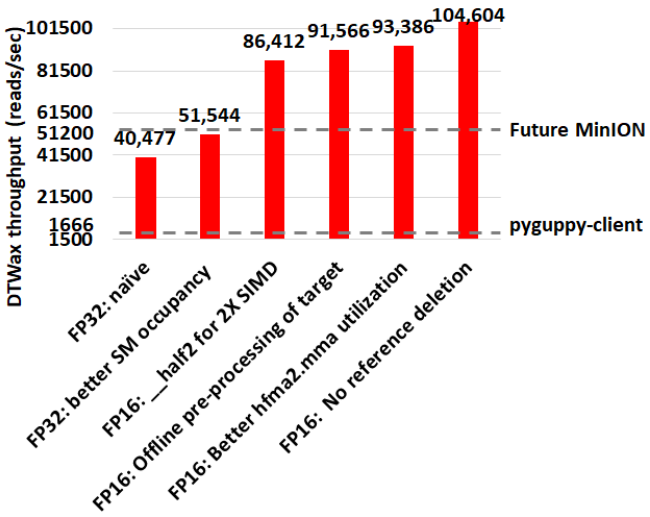

Figure 3.4: A series of performance optimizations enable DTWax to handle more than 2X the projected future sequencing throughput

DTWax is a compute-bound kernel. Fig.3.4 gives the reader a better understanding of the relative benefits of some of the main optimizations that go into DTWax.

3.5 Results

DTWax is the most accurate Read Until classifier as shown in Fig. 3.7. The sequencing speedup and compute time savings from DTWax are higher for longer reads as shown in Fig. 3.8.

We use two metrics to evaluate the benefits of using DTWax– sequencing speedup and compute speedup: costup. Sequencing speedup is defined as the speedup in the end-to-end sequencing time from using DTWax for Read Until over a conventional nanopore sequencing workflow that does

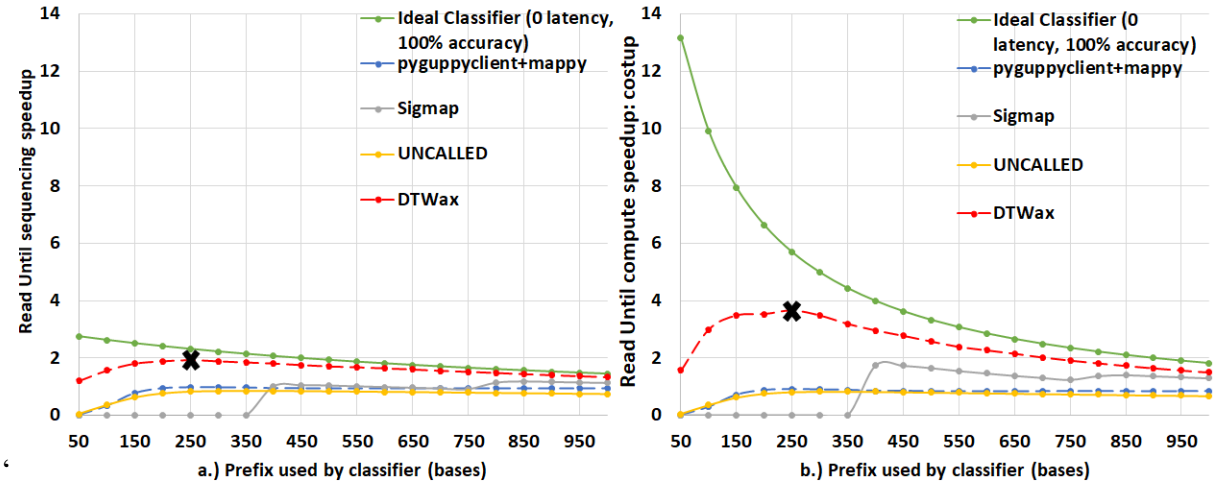

Figure 3.5: DTWax yields ∼1.92X sequencing speedup and ∼3.64X compute speedup: costup . (a) DTWax yields the best sequencing speedup of ∼1.92X over a conventional pipeline that does not use Read Until on reads of length 2000 bases. (b)DTWax yields the best compute speedup: costup of ∼3.64X. The prefix length used for classification is 250 bases for the best performance in both cases.

not use Read Until. Accessing an NVIDIA A100 GPU instance on the cloud is priced ∼10% higher than the CPU instance we use for benchmarking. Hence, we normalize the compute time savings from using DTWax for Read Until to the cost of the cloud GPU instance to estimate compute speedup: costup. Further, we also compare the F1-score of DTWax in making Read Until classifications.

DTWax yields up to ∼1.92X sequencing speedup and ∼3.64X compute speedup: costup with a future MinION of 100X throughput when compared to a sequencing workflow that does not use Read Until as shown in Fig. 3.5. Additionally, we observe that using a prefix length of 250 bases yields the best benefits from using Read Until on a dataset of average read length 2 Kbases. One may also observe that savings from ont-pyguppy client degrades with increasing read-prefix lengths used for classification because ont-pyguppy cannot process streaming inputs and concatenate to outputs. Therefore, ont-pyguppy-client has to basecall the entire prefix again. Sigmap is unable to extract events from signals less than 400 bases. Please note that DTWax and pyguppy-client are GPU solutions and we label them using dotted lines on all the plots.

Fig. 3.6 shows that DTWax can handle more than 2X the throughput of a future MinION and

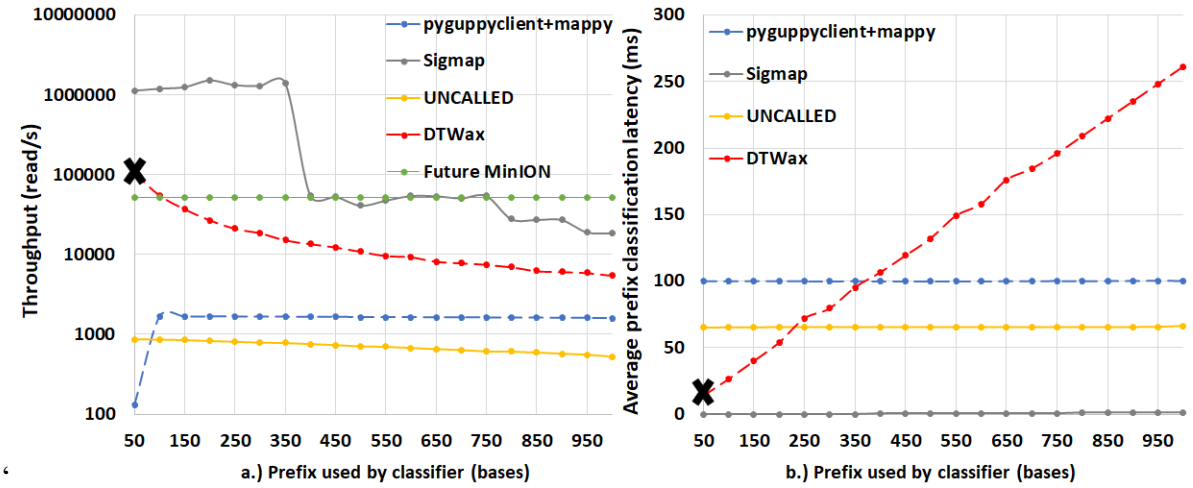

Figure 3.6: DTWax can handle more than 2X the throughput of a future MinION and has ∼7.18X lower latency than pyguppy-client followed by mappy. (a)Unlike pyguppy-client followed by mappy, DTWax operating at a granularity of 50 bases can handle twice the throughput of a future MinION (b)DTWax takes only ∼14 milliseconds to classify 50 bases and is ∼7.18X faster than pyguppy-client followed by mappy .

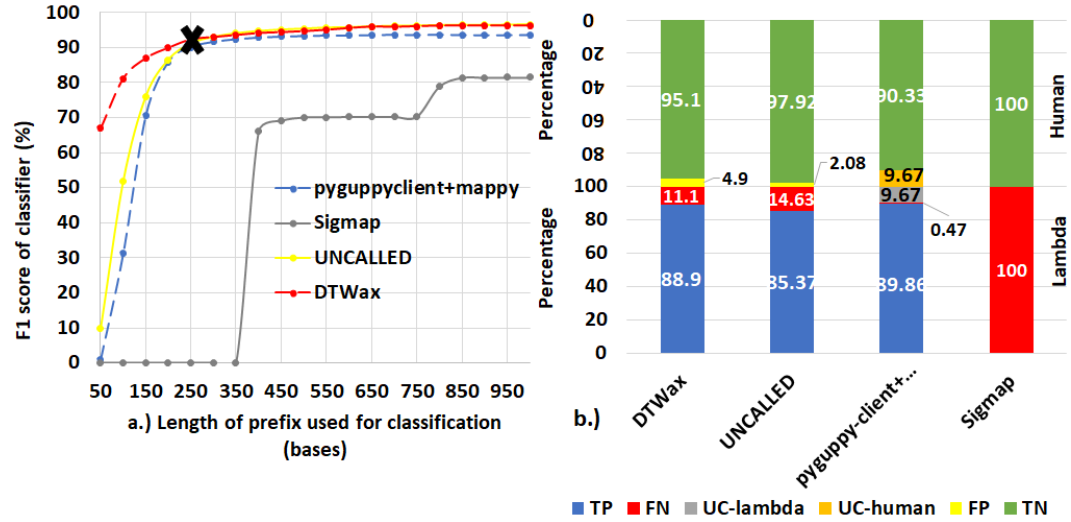

Figure 3.7: DTWax is the most accurate Read Until classifier. (a) DTWax has an F1-score of ∼92.24% and is the most accurate Read Until classifier using a prefix length of 250 bases. (b) DTWax is better at filtering non-target (human) reads out than pyguppy-client followed by mappy while being almost comparable in successfully retaining target reads

.

has ∼7.18X lower latency than pyguppy-client followed by mappy using a prefix length of 50 bases. Because of this, we operate DTWax at a processing granularity of 50 bases over a prefix length of 250 bases to make a Read Until classification decision. To explain the performance of

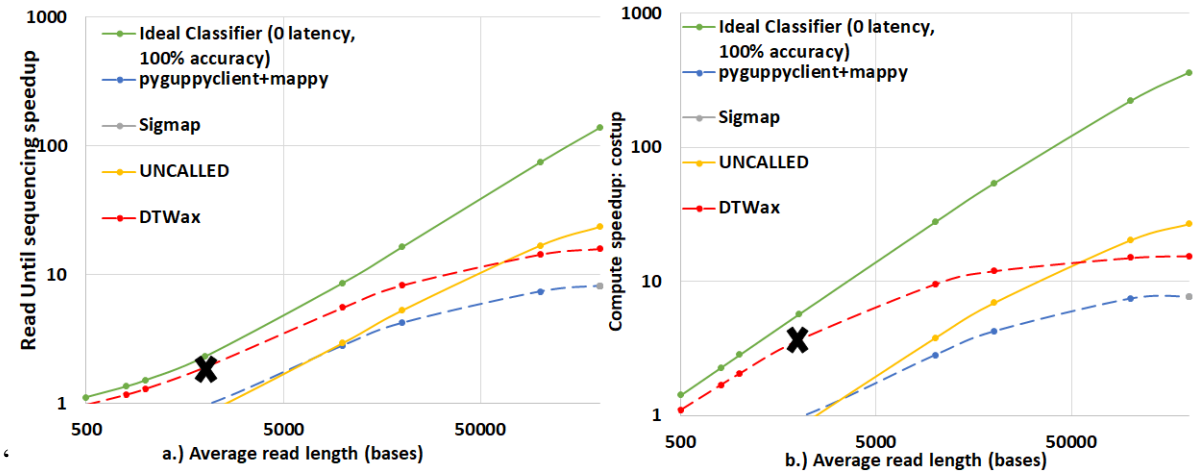

Figure 3.8: DTWax yields higher benefits on longer read lengths. In (a) and (b), we see increasing benefits from using Read Until on higher read lengths and DTWax is the best solution for read lengths shorter than ∼50Kbases.

pyguppy-client on longer prefixes, we profile pyguppy-client using NVIDIA nsight-compute and observe that the GPU occupancy is higher with longer prefix lengths resulting in better benefits from longer prefix lengths. UNCALLED seems to have a very high one-time fixed cost for path buffer management for storing output forest of trees and there is negligible cost with added prefix lengths. Although Sigmap is the best in terms of throughput and latency, it cannot extract useful information from 250 bases and has lower F1 scores in classifying prefix lengths longer than 400 bases as shown in Fig. 3.7.

3.6 Availability

The GPU software of DTWax is publicly available at: `https://github.com/harisankarsadasivan/DTWax/tree/FAST5`.

CHAPTER 4

RawMap: Rapid Real-time Squiggle Classifier for Read Until

4.1 Introduction: Read Until for Microbiome Estimation

Novel infections and pandemics are on the rise [94]. In the context of the COVID-19 pandemic, changes to the human microbiome are increasingly understood as a biomarker [95, 96, 97] which can help in patient risk stratification and mitigate disease severity [98]. Understanding the human microbiome can also provide additional benefits such as providing prophylactic and therapeutic tools to improve human health [99] and thereby, increasing colonization resistance against infections [100]. Along with microbiome identification and quantification, viral load is another metric linked to COVID-19 disease severity and mortality and helps in risk stratification [101].

As a means to estimate the microbiome, DNA sequencing has immense potential to transform personalized healthcare through the early discovery and detection of diseases. Metagenomic abundance estimation (relative quantification of taxa) from long DNA reads is a less explored domain as we learn in Section 4.2. Moreover, efficient enrichment and sequencing of microbial DNA from non-target-rich metagenomic samples with unknown microbial constituents is an unsolved problem.

Oxford Nanopore Technology's (ONT's) portable long-read DNA sequencer, MinION, has a minimal operational and logistical footprint, and real-time capabilities, making it a unique candidate for this purpose [53]. Assays of SARS-CoV-2, Ebola, Zika, tuberculosis, and various other

pathogens have been successfully conducted using MinION [102]. Nanopore sequencers monitor the electrical signal fluctuations from a strand passing through a nanopore channel and decode the specific DNA/RNA sequence. Sequencing costs us both time and money. Flowcell washes and longer sequencing times degrade the quality of a flowcell over time. Replacing degraded flowcells and wet-lab reagents adds to sequencing cost. In order to reduce the sequencing and compute footprint, it is essential to ensure only target microbial reads are completely sequenced and non-target human reads are ejected. Finally, we show that the time saved in nanopore sequencing is cost saved in Section 4.6.8.

Human samples can have a significant fraction of non-target reads – greater than 99% of total reads are non-target human reads in most clinical respiratory samples [103]. Sequencing these non-target reads would otherwise be a waste of sequencing time and cost. Moreover, it is important to not miss target reads in order to accurately characterize the microbiome. Undetected target reads especially from a low-abundant species can offset the relative abundance. Hence, prior works choose to sequence all of the unclassified reads [52]. We follow the same practice when performing Read Until.

The state-of-the-art (baseline) Read Until pipeline, Readfish [52], sequences unclassified reads and iteratively updates the alignment index for read Until target classification. For microbiome abundance estimation, Readfish uses a software pipeline consisting of a basecaller (Guppy [104] is a deep neural network that decodes raw squiggle output from the sequencer to bases), an aligner (Minimap2 [105] maps DNA read to the target genome using approximate string matching), and a metagenome classifier (Centrifuge [106] classifies the read into a taxonomic rank).

However, Readfish has a two-fold performance problem – low throughput and accuracy (on small signal chunks) of the basecaller. Prior works have shown that real-time basecalling using deep neural networks cannot keep up with the throughput of the sequencer and this problem is amplified by the projected growth in future sequencing throughput [75]. Additionally, prior works have reported the inaccuracy of Guppy in basecalling small signal chunks [78, 79]. We observe that this translates to 59.5% of the sequenced bases being unclassified (as shown in Figure 4.2) in a

99:1 host: target sample with an average read length of 8.29 Kbases. We analyze and observe that these unclassified bases are basecalled with low Phred quality scores as shown in Figure 4.1 and hence, aligners like Minimap2 and BLAST are unable to align them.

We envision a portable, fast, and inexpensive diagnostic solution for the digital enrichment of the human microbiome and downstream applications including microbiome abundance estimation and viral load quantification. Our fast and accurate diagnostic solution can decentralize sequencing and democratize personalized healthcare. We propose an efficient CPU-only software solution to classify these low-quality read prefixes using RawMap, a squiggle space Read Until classifier. Our feature engineering enables RawMap to identify non-linear and non-stationary characteristics of a raw read prefix and distinguish the host from the target in a simple 3-D feature space with very low computational overhead. RawMap is also capable of identifying previously undetected microbial species and is offered as a "plug-and-play" solution without disrupting the baseline Read Until pipeline. Our proposed RawMap augmented pipeline 1 saves $\sim$24% sequencing time and cost whereas pipeline 2 saves $\sim$22% compute time. Our evaluations are performed with respect to a baseline Read Until pipeline on a 99:1 host: target sample with an average read length of 8.29 Kbases.

Additionally, we demonstrate how RawMap may be utilized to skip the expensive basecalling step and perform viral load quantification in a sample mix of human and SARS-CoV-2. In some settings, viral load is linked to disease severity and mortality and helps in risk stratification [101]. Colorimetry-based Reverse Transcription-Polymerase Chain Reaction (RT-PCR) test is the most commonly used method for viral load quantification [107]. However, RT-PCR-based tests often have complex primer design, manufacture, and distribution steps [75] and may have a significant number of false positives from various sources of contamination if the assay is not well-validated [108, 109]. In the end, we present a case of using RawMap to skip the expensive basecalling step for sequencing-based viral load quantification.

4.2 Related Work

Metagenomic data analysis and abundance estimation are well-studied for short reads [110, 111, 112, 113, 114, 115] but that is not the case for long reads. MetaMaps [116] is a long-read abundance estimator it is resource hungry. Centrifuge [106] is resource efficient and works for both short and long reads. But, we observe that using Centrifuge alone does not produce sufficiently accurate results as discussed in Section 4.5.1.

Read Until is an emerging domain of research. The first attempt at Read Until sequencing used a signal space technique called subsequence Dynamic Time Warping where a raw nanopore query signal is aligned to an in silico signal representation of a reference sequence [117]. However, this method is not scalable to reference sequences larger than tens of kilobases as the runtime is quadratic in the reference length. SquiggleFilter [75] (ASIC) and HARU [81] (FPGA) are hardware-accelerated Read Until solutions but their performance is optimal only for a small reference target (order of Kilobases) as they are constrained by the hardware's small buffer size and hence, are not suited for abundance estimation of multiple target microbial references (typically millions of bases).

Readfish, the most widely adopted ReadUntil pipeline today [52], uses a basecaller followed by an aligner, Minimap2 to make a decision on a small chunk of data in the basecalled space. However, this method fails to bring out optimum savings as we find that a very high fraction of sequenced bases are from non-target reads as they get basecalled with low-quality scores and are hence, unintentionally sequenced. Additionally, the basecaller Guppy cannot meet the increasing throughput of the sequencer [75].

UNCALLED [78] uses a probabilistic k-mer approximation from the signal space followed by alignment using BWA-MEM to identify target reads and perform metagenomic classification. However, we do not compare directly with UNCALLED because of two reasons. UNCALLED needs to know the constituents of the sample apriori to form the reference index for classification. This is not possible in situations where the target microbiome/infectious agent is unknown. UNCALLED cannot use a non-target (human genome in this case) reference because UNCALLED is

shown to not scale to references above 100Mb and performs poorly on highly repetitive references. Secondly, UNCALLED's k-mer approximation is directly dependent on a k-mer reference current model which is retired by ONT for all newer and future nanopore chemistries. Sigmap [79] is another Read Until classifier in the signal space that detects events and does Minimap2-style [105] chaining to attempt to map the read to a target. However, Sigmap also needs to know the constituents of the input sample apriori and creates an index which is $\sim$20 X the index size of UNCALLED for a single target.

SquiggleNet [82] is the only prior work that can classify long-reads of unseen microbial species. SquiggleNet uses a deep learning model. However, SquiggleNet is shown to be only as accurate as Guppy followed by Minimap2, and is slower [82]. This does not solve our problem of accuracy and throughput.

The viral load has commonly been estimated from time-consuming wet-lab enriched tests [101, 118]. Recent efforts have focussed on direct sequencing from high throughput short read sequencers [119]. However, there exists no prior work discussing viral load quantification from direct nanopore long-read sequencing, to the best of our knowledge.

Although the trends in pore occupancy with Read Until have been previously studied [52, 78], Read Until resulting in reduced sequencing cost has not been quantitatively discussed before.

4.3 Background and Motivation

4.3.1 Microbiome Abundance Estimation

Human lung microbiome is the aggregate of all microbiota that reside on or within lung tissue and biofluids. Characterizing the abundance of the human microbiome helps researchers to understand the health status of the human lung [120]. Lung microbiota composition can be a biomarker of existing health conditions. Hence, the accuracy of microbiome abundance estimation is important. Species abundance as defined by Centrifuge [106] does not incorporate the variability of nanopore read lengths and ploidy (number of sets of chromosomes in a cell) of species. To fix this, we define

cell number microbiome abundance of species j, A_j as follows:

$$A_j = \frac{\frac{B_j}{l_j p_j}}{\sum_{k=1}^{S} \frac{B_k}{l_k p_k}}$$

where B_k is the total number of bases that correspond to species k, l_k is the genomic reference length of species k, p_k is the ploidy of species k and S is the total number of species discovered, excluding the host.

For evaluation, in our case where the ground-truth abundance is known, the error in estimated abundance is quantified using two metrics: mean deviation and maximum deviation.

$$Mean\ \ deviation = \frac{1}{S} \sum_{k=1}^{S} |e_k - g_k|$$

$$Max.\ \ deviation = max\left\{e_k - g_k\right\} \forall\ \ kS$$

where e_k is the estimated abundance and g_k is the groundtruth abundance of species k.

As discussed, we do not need to sequence the human DNA to calculate microbiome abundance. It has been discovered that non-target human reads in most clinical respiratory samples can be greater than 99% [103]. Hence, it would be ideal to discard the non-target reads and sequence the target microbial reads alone if it would help save sequencing time and cost.

4.3.2 Cost of nanopore sequencing

There are two components to the cost involved- sequencing and compute. The cost of sequencing includes flowcell cost and reagent cost. A MinION flowcell costs $475 and can sequence up to 50 Gigabases(Gb) on average during its lifetime. We estimate a fixed reagent cost of ∼$21 (with QIGEN's QIAamp DNA Mini Kit for extraction, ONT's SQK-RAD004 for library prep, and SQK-RBK004 for a 12-way barcoded run) per experiment, and a variable flowcell cost of ∼$6 per every hour of sequencing (based on estimated flowcell throughput of 0.59Gb/hr). Reduced time to answer means the flowcell may be used for other applications within its lifetime. ONT has defined

a protocol for real-time selective sequencing, which we leverage to save pore-use time during a single run. Reduced pore-use time can lower flowcell costs incurred from a single run. We show that sequencing time saved is flowcell cost saved in Section 4.6. For the sake of simplicity, we will refer to the variable flowcell-related cost as sequencing cost. Further, compute-related costs can stem from the costs of using cloud or shared computing resources.

4.3.3 Read Until Pipeline

Our baseline, a two-stage pipeline for microbiome abundance estimation is derived from Readfish [52] but with minor modifications. Guppy is used for basecalling the squiggles. We customize Minimap2 for better accuracy as discussed in Section 4.5. Centrifuge, a metagenomic classifier with a microbial and human index constructed from NCBI non-redundant nucleotide database operates on full-length basecalled reads. Centrifuge does the species-level classification and identifies those taxonomy IDs which get more than 0.005% reads assigned. Reference genomes corresponding to those species' identified by Centrifuge are downloaded in real-time from the online RefSeq database to keep our memory footprint small. Subsequent read prefixes are mapped by Minimap2 on an expanding set of references generated from Centrifuge operating on full-length reads. If Minimap2 detects a non-target, the read is reversed. All unclassified reads from Minimap2 are sequenced so that Centrifuge can iteratively build the 'refined index'- an alignment index for Minimap2 constructed on-the-fly from a small set of target species detected by Centrifuge. In summary, Centrifuge, a low memory footprint, less accurate classifier is used to build a 'refined index' for the highly accurate mapper, Minimap2 to make Read Until decisions.

4.3.4 Inefficiency of the baseline pipeline

The baseline Read Until pipeline cannot classify and detect 59.5% bases sequenced because they are basecalled with lower Phred quality scores as shown in Fig. 4.1 for a 99: 1 host: target sample with an average read length of 8.29Kb. Further, it is observed that low-quality reads translocate slowly at a median rate of 270 bases/s. We cannot eject the unclassified reads because Centrifuge

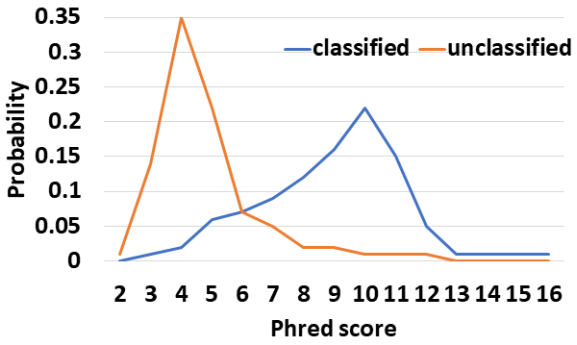

Figure 4.1: Unclassified reads from Guppy followed by Minimap2 have a lower mean Phred quality score compared to classified reads as seen in this probability density function of Phred quality scores.

needs them for building the refined index. The percentage of unclassified non-target bases (99% of 59.5%) sequenced is an even bigger problem for long reads as shown in Fig. 4.2. Reducing these unclassified non-target bases would help reduce irrelevant data footprint, and improve time and cost savings. We realize that classification is a simpler and different task from basecalling. We engineer features from the squiggle space and classify the unmapped read prefixes.

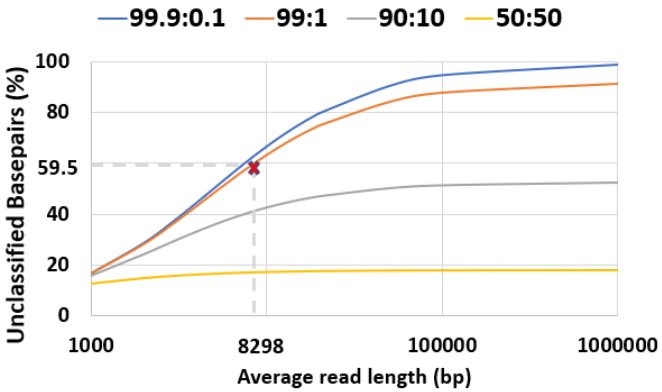

Figure 4.2: 59.5% of the total sequenced bases are unclassified in a 99:1 host: target sample with an average read length of 8.298Kb.

With RawMap, we propose an efficient CPU-only solution to identify non-target reads missed by Guppy. RawMap does not alter the standard Read Until pipeline much, it is a "plug-and-play" solution which grabs information from the squiggle domain to classify a read prefix using a very efficient algorithm in a 3-D feature space. RawMap learns from the non-linear non-stationery characteristics of squiggles to identify microbes from host. Additionally, RawMap is microbial

species-agnostic – it can classify microbial species it is not trained on.

4.4 Methods

Our Read Until pipeline for abundance estimation is a modified version of the metagenomic enrichment pipeline [52] which uses Guppy for basecalling, Minimap2 for Read Until decisions, and Centrifuge for generating 'refined index', a set of species detected in the sample. Minimap2 classifies using the refined index of detected organisms' genomes and instructs MinION to eject host reads. Centrifuge classifies Minimap2's unclassified reads and detects organisms absent in the refined index. Additionally, we use Minimap2's results on the read-prefixes for accurate abundance estimation (we can see that Minimap2 produces consistent accurate mappings above 450 bases in Fig. 4.6).

However, there exists a problem of unclassified reads with Minimap2 because Guppy basecalled these reads poorly as indicated by their poor base quality scores in Fig. 4.1. Basecalling is the complex process of translating raw nanopore signal to a base sequence and Guppy's network is designed for this particular task. Classification is, however, a much simpler problem and utilizes global signal-level information which Guppy may not be focusing on. Therefore, we explore the raw nanopore data space for additional signal characteristics and engineer features out of it for the task of read classification.

Nanopore squiggles (raw data) are also very similar to EEG as they both are non-linear and non-stationary. Prior works have used Hjorth parameters to extract the time domain properties of non-stationary signals like brain EEG [121]. It is known in the past that genomic sequences can be transformed into a phase signal representation to extract Hjorth parameters in order to classify metagenomic data [122, 123]. This is based on the idea that the characteristic changes in the phase signal can identify one species from another. We extend this idea by modifying the Hjorth parameters to work on noisy squiggle space for Read Until to find characteristic current transitions in the read prefix to distinguish microbes from host.

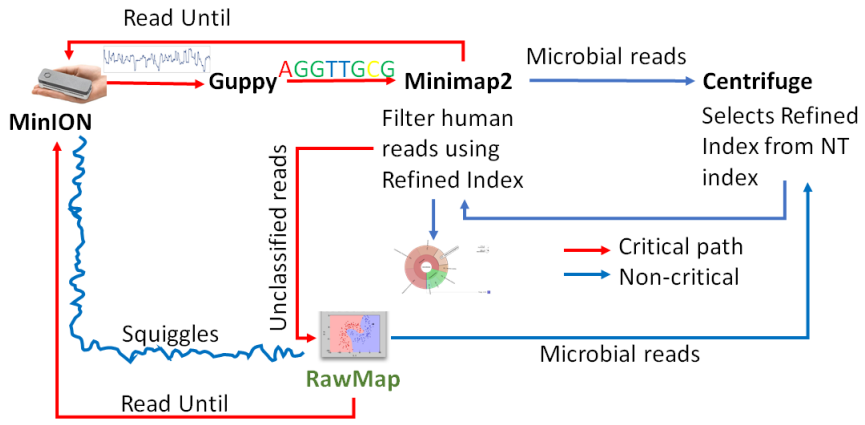

Figure 4.3: Proposed pipeline 1 with RawMap as a secondary filter classifying Minimap2-unclassified-reads, plugged-in after Guppy and Minimap2, yields best savings in sequencing time and cost for 99:1 sample with an average read length of 8.298 Kbases.

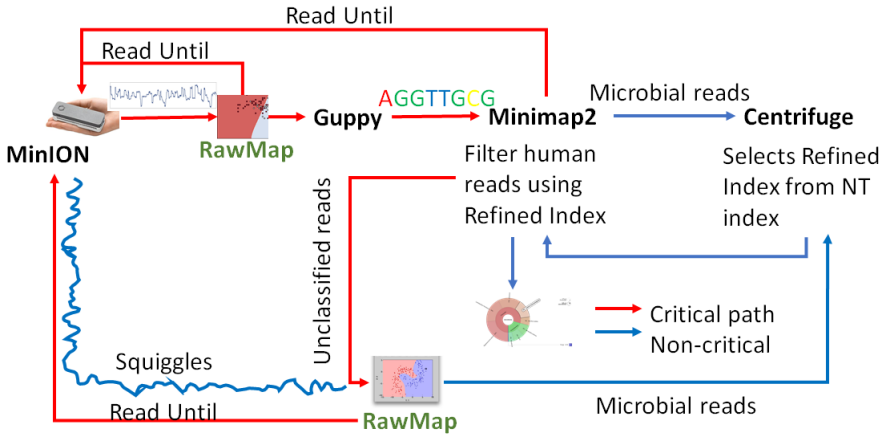

Figure 4.4: Proposed pipeline 2 with RawMap as the primary filter helps skip Guppy and Minimap2 for most of the non-target reads and offers best compute time savings.

4.4.1 RawMap augmented Read Until pipelines

We present RawMap, a direct squiggle-space microbial species-agnostic Read Until classifier for identifying target microbial reads. RawMap is a "plug-and-play" solution which may be plugged into the baseline Read Until pipeline as shown in Fig. 4.3. In the proposed pipeline 1 with Read Until, RawMap is combined with Minimap2 for rapid microbiome abundance estimation. Here, Minimap2 acts as the primary classifier for target versus non-target while RawMap acts as a secondary classifier which identifies the non-target read-prefixes Minimap2 could not and instructs the MinION to eject them. We also demonstrate a very efficient solution with pipeline 2 in Fig. 4.4

where RawMap acts as a fast primary classifier that tries to reduce the workload for the compute-intensive Guppy. Guppy and Minimap2 are acting only on reads that RawMap classifies as target and lets through. There is also a secondary instance of RawMap fine-tuned to classify reads unclassified by Minimap2. We compare the benefits of each of these pipelines in Section 4.6.

RawMap's algorithm has three main components: signal pre-processing, feature extraction, and Support Vector Machine (SVM) based classification. Signal pre-processing normalizes the noisy nanopore signal, feature extraction calculates the modified Hjorth parameters and the SVM classifier does the target vs. host classification.

4.5 Implementation

4.5.1 Read Until baseline for abundance estimation

Similar to the recent work on adaptive sampling for metagenomic enrichment [52], we have a Read Until pipeline with Guppy for basecalling, Centrifuge for iteratively building the 'refined index' which consists of species detected and Minimap2 trying to map every read-prefix to this 'refined index'. Unmapped reads are sequenced in full for Centrifuge to build the 'refined index'. However, our baseline pipeline does not use Centrifuge for abundance estimation but has an additional final stage to do this because we find that a customized version of Minimap2 is better at abundance estimation than Centrifuge as shown in Fig. 4.5. It is observed that the minimizer-based seeding in Minimap2 helps only with the speed of alignment and turning it off can improve the number of reads mapped without affecting the accuracy of mapping as in Fig. 4.6. We turn off the minimizer-based seeding in Minimap2 by using the command line parameters '-w 1 -k 15' during the construction of the 'refined index'. This is referred to as customized Minimap2 or minimap2_custom.

In the last stage of the pipeline, we take batches of 1000 microbial read classifications from Minimap2 to calculate the cell number abundance. We sequence until the 'refined index' does not change and the estimated microbiome abundance does not deviate more than 5% on an average

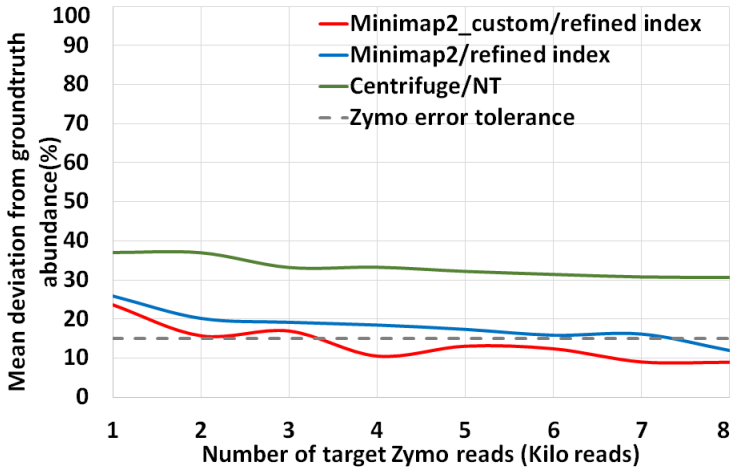

Figure 4.5: Customized Minimap2 with refined index is more accurate than Centrifuge for abundance estimation.

from the previous estimation for two successive iterations. For our experiments with Zymo microbial community standard, it is observed that we need to sequence until approximately 8000 target Zymo High Molecular Weight (HMW) reads (∼1X Zymo HMW coverage). For our evaluations, we stop pipelines 1 and 2 as soon as 8000 target Zymo reads were sequenced and identified.

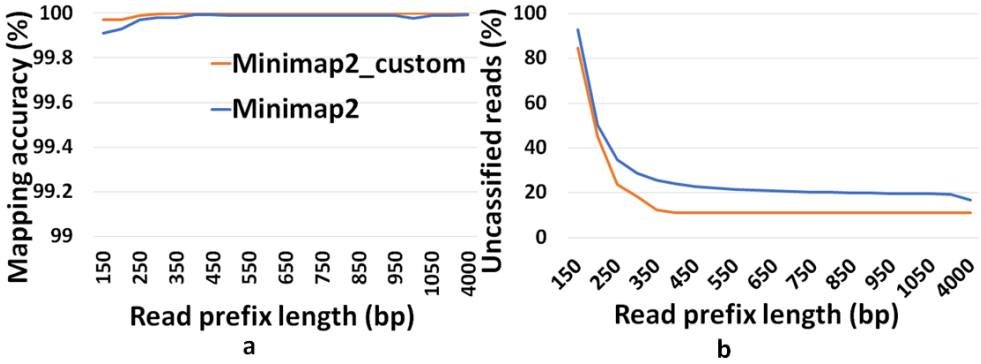

Figure 4.6: (a) Customized Minimap2 with refined index has better accuracy of mapping. (b) Customization helps us classify more reads compared to Minimap2.

4.5.2 Implementing RawMap

We trim the first 2000 raw data samples to eliminate adapter stalls and non-informative adapter-barcode regions and then process the next 450 bases equivalent (∼6667 samples) of raw data. The median and Median Absolute Deviation (MAD) of this raw data are then calculated. Outliers are

filtered out as follows: only raw data within a range of 5 MAD deviations from the median are considered for further processing. The filtered channel-scaled current values are then Median-Median Absolute Deviation (MED-MAD) normalized by the signal pre-processor. This squiggle segment y corresponding to 450 bases of a read is then mapped to a 3-D feature space using a modified version of Hjorth parameters by the feature extractor. It is observed that MED and median are robust to the outliers and hence, yield cleaner nanopore signals. The Hjorth parameters are modified by calculating the variance from MED and MAD instead of the originally used mean and standard deviation. We define modified Hjorth parameters as follows:

$$Activity = var(y)$$

$$Mobility = \sqrt{\frac{var(y')}{var(y)}}$$

$$Complexity = \frac{mob(y')}{mob(y)}$$

where y is the normalized raw data segment corresponding to 450 bases, y' is the first-order difference of the signal and var is the modified variance.

Activity captures the signal power, mobility is the mean frequency and complexity is the change in frequency. The modified Hjorth parameters help us find a localized region where the microbial signals map to, as shown in Fig. 4.7.

RawMap uses a Support Vector Machine (SVM) with a Radial Basis Function kernel. For pipeline 1, the SVM is trained on 6000 squiggles each of human and Zymo from a 50:50 barcoded run with 10-fold cross-validation to capture the non-linear and non-stationary characteristics of the nanopore squiggles. For RawMap to be tuned as a primary classifier for pipeline 2, 100K squiggles of each Zymo and human were used to capture the high variation in current characteristics. AUC was used as a scoring metric for model validation instead of accuracy and hyper-parameters were tuned using grid search.

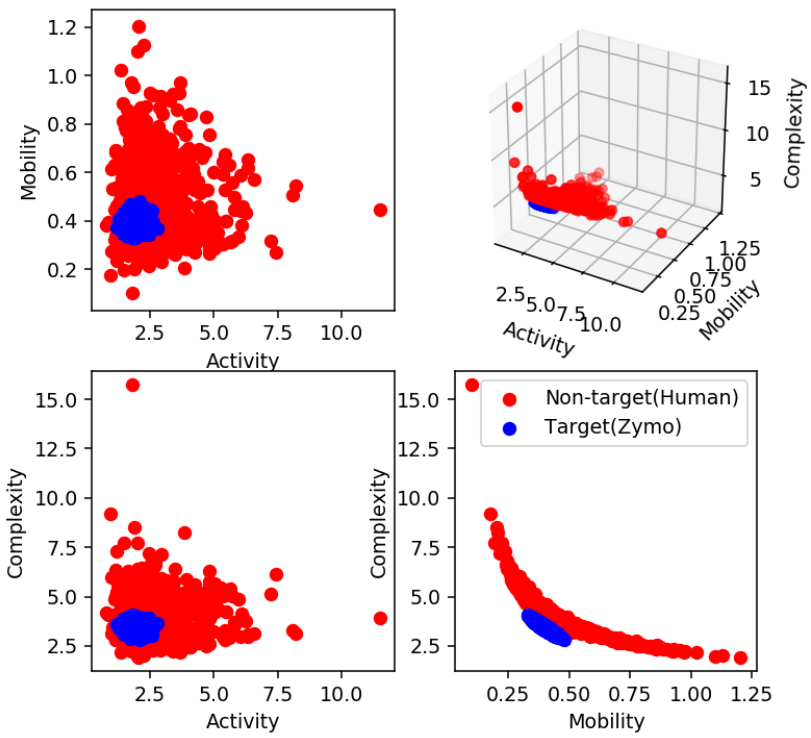

Figure 4.7: Target(Zymo) squiggles are highly localized in the modified Hjorth space as shown in blue. For illustration, only 1000 feature vectors each of target and non-target are shown here.

4.5.3 Configurations

For our Read Until baseline, we use the ONT recommended version of Minimap2, v2.17 for ONT long reads and we turn minimizers off for better classification accuracy. Guppy v4.0.11 in high-accuracy mode is used for basecalling and is invoked using ONT's pyguppyclient server for Read Until. Centrifuge v1.0.4 with a human and microbial nucleotide (NT) index is used as explained under Section 4.3.3. We further add the capability to calculate cell number abundance to Minimap2. RawMap is evaluated using a single-threaded execution on an Intel Xeon E5-2697 x86 processor. Guppy runs on NVIDIA GeForce GTX-1080.

4.5.4 Wet-lab

All sequencing libraries were prepared using ONT's Rapid Sequencing Kit (SQK-RAD004). We conduct two types of experiments: barcoded and unbarcoded. In our barcoded experiments, ONT's

Rapid Barcoding Kit (SQK-RBK004) is used for barcoding quantified host and target separately for ground truth abundance generation. The barcoded experiments are run with extracted human DNA from Coriell's NA12878 and ZymoBIOMICS High Molecular Weight DNA Mock Microbial community ('Zymo HMW', cat #D6322). The non-barcoded experiment is run with HeLa human extracted genomic DNA (New England BioLabs, cat# N4006) and ZymoBiomics Microbial Community DNA Standard ('Zymo', cat# D6306).

Finally, we conduct an experiment to understand how Read Until damages the flowcell. For this, we divided a new high-quality flowcell into two equal sequencing regions: half the active pores sequencing full-length reads and the remaining half rejecting every read at 450bp. The pore status (control vs Read Until) is chosen in a checkerboard fashion and is fixed. The number of active pores is normalized to the number we started with for both sets. After 6.5 hours of sequencing, the flowcell is then washed and MUX-ed to the same set of sequencing pores as before. Comparing the percentage of active pores that recovered in both regions would tell us the damage caused by Read Until.

Additionally, the wet-lab protocol suggested for viral load quantification is ONT's proposed Sequence Independent Single Primer Amplification (SISPA) pipeline [124] for metagenomic sequencing. Here, full-genome amplified RNA is reverse-transcribed and sequenced as cDNA. This protocol is independent of the viral species present and hence, universal.

4.5.5 Definitions & datasets

"Premix" refers to biologically mixing the prepared libraries of NA12878 and Zymo HMW prior to sequencing. "Premix"-ed sequencing runs have unique barcodes for Zymo HMW and HeLa. "Post-mix" refers to datasets sequenced independently from HeLa and Zymo and mixed digitally. "Zymo HMW-subset" is created by using only four out of eight different Zymo HMW species for training and the remaining four for testing.

We have 4 pre-mixed datasets (50:50: for training, 99:1:Run 1, 99:1: Run 2, and 99:1: Run 3). The zymo-HMW subset is from Run 1. We also have 2 post-mixed datasets (50:50: for training,

and 99:1: for testing). We have 200K-1.4M reads in each of the datasets. Training and testing are always performed on different datasets, training on 50:50 and testing on 99:1.

Zymo and Hela datasets sequenced in our lab are available at DOI:https://doi.org/10.5281/zenodo.7349378. The viral load quantification study is performed on 7K SARS-CoV-2 [125] and 105K human cDNA reads [126] which are already publicly available.

4.6 Results

4.6.1 Read Until Benefits

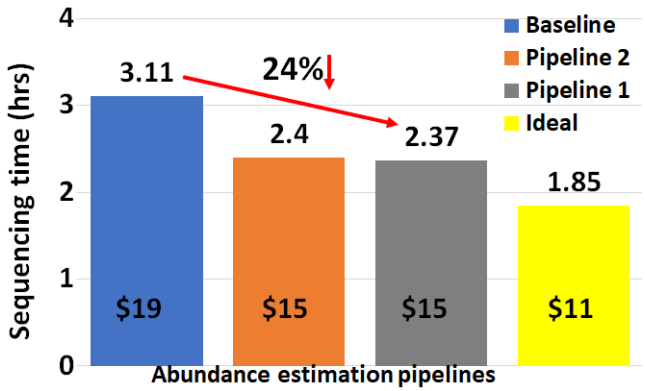

Figure 4.8: Pipeline 1 saves ∼24% sequencing time and cost compared to the baseline Read Until pipeline.

The two proposed pipelines offer savings in terms of sequencing and compute time with respect to baseline Read Until pipeline. In Section 4.6.8, we show that sequencing time saved directly translates to sequencing cost saved. Our proposed pipeline 1 yields the best sequencing time and cost savings with respect to baseline Read Until pipeline (∼24% of savings) as shown in Fig. 4.8. Pipeline 2 performs slightly worse than pipeline 1 because of RawMap's lower classification accuracy compared to Guppy followed by Minimap2.

Pipeline 2 is beneficial if compute time and cost is a concern (Cloud GPU instances are ∼10% costlier than their CPU counterparts). Pipeline 2 yields a 22% compute time savings compared to

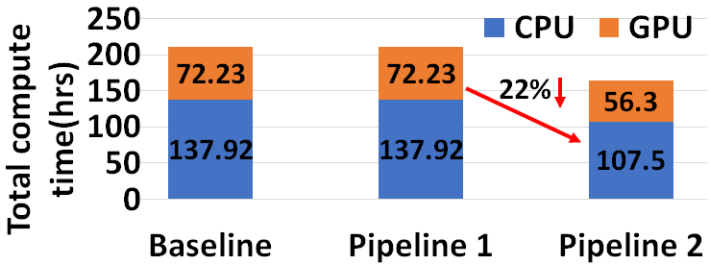

Figure 4.9: Pipeline 2 yields 22% compute time savings compared to the baseline and pipeline 1 because we skip the expensive basecalling step for primary filtering.

baseline and pipeline 1 (Fig. 4.9). This is because non-target human reads are filtered out from the basecalling-aligning path by RawMap.

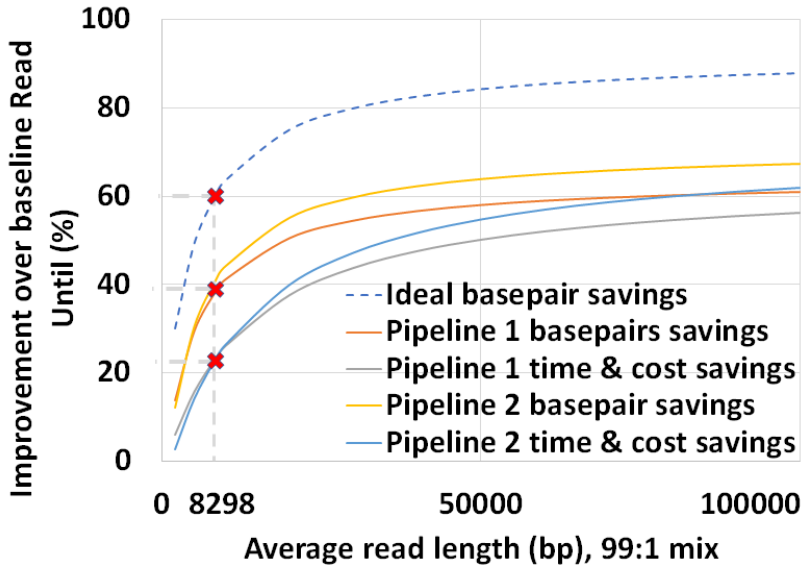

Figure 4.10: Higher read lengths give better sequencing time and cost savings in a 99:1 host: microbial mix.

It is observed that we get higher Read Until benefits from longer average read lengths (Fig. 4.10). The blue-dashed line depicts the maximum benefits attainable with a 100% accurate classifier with zero latency of compute. Additionally, Read Until also yields higher benefits when the host contamination is high i.e, when we are looking for a needle in a haystack (Fig. 4.11).

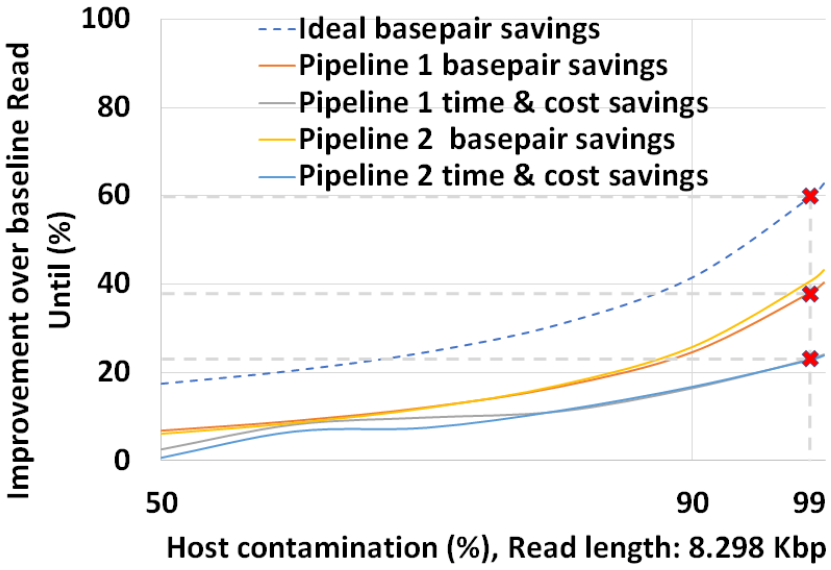

Figure 4.11: Higher host: target ratio yields better sequencing time and cost savings for an average read length of 8.29Kbases.

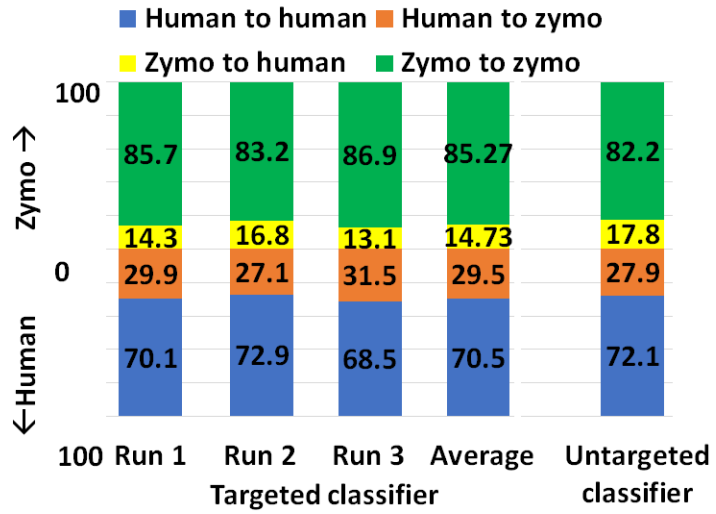

Figure 4.12: RawMap does untargeted classification as well as targeted classification.

4.6.2 Untargeted classifier

In pipeline 1, RawMap complements Minimap2 and improves overall pipeline accuracy and sequencing time while it improves compute time in pipeline 2. Additionally, we also evaluated RawMap as an untargeted filter (capable of correctly detecting new microbial species RawMap is untrained on) as shown in Fig. 4.12. Here, RawMap is freshly trained on a total of 12000 reads from four Zymo HMW species (Pseudomonas aeruginosa, Salmonella enterica, Enterococcus fae-

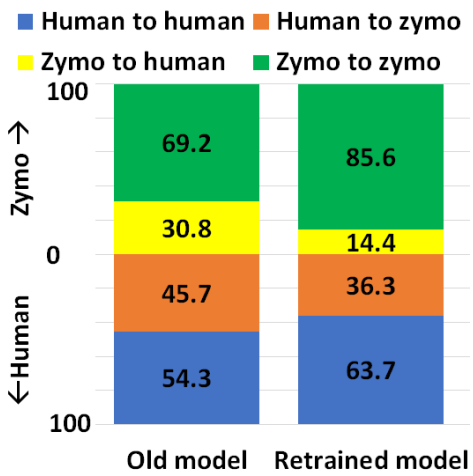

Figure 4.13: RawMap can be customized for different wet-lab protocols by retraining.

calis, and Listeria monocytogenes) and human from a 50:50 premixed barcoded sample. RawMap is then tested on 800K reads of both human and four other Zymo HMW species (Saccharomyces cerevisiae, Escherichia coli, Staphylococcus aureus, and Bacillus subtilis) from a new sequencing run of 99:1 premixed barcoded sample. The confusion matrix values obtained in this case are very close to when RawMap was trained on all 8 species (targeted classifier) of Zymo HMW and human as shown in Fig. 4.12. Hence, RawMap can function both as a targeted and an untargeted (microbial species-agnostic) classifier. This is particularly advantageous for cases where we do not know the input constitution mix.

4.6.3 Sensitivity to Wet-lab

However, RawMap seems to be sensitive to the wet-lab protocols followed. RawMap is retrained for a different set of wet-lab protocols (new extraction techniques and no barcodes). Without re-training, RawMap did not perform as expected on 100K reads from 99:1 post-mixed run of extracted HeLa and ZymoBiomics Microbial Community DNA Standard sequenced (purchased from differently extracted sources) as shown in Fig. 4.13. This is because RawMap is trained to capture nuances in electrical signals of the host and target and the signal: noise ratio is a function of wet-lab protocols followed. However, as shown in Fig. 4.13, RawMap produced good results when retrained on 12000 reads from 50:50 HeLa: Zymo non-barcoded post mixed run which used

the same set of wet-lab protocols (differently extracted DNA and no barcodes) and tested on 200K reads as shown in Fig. 4.13.

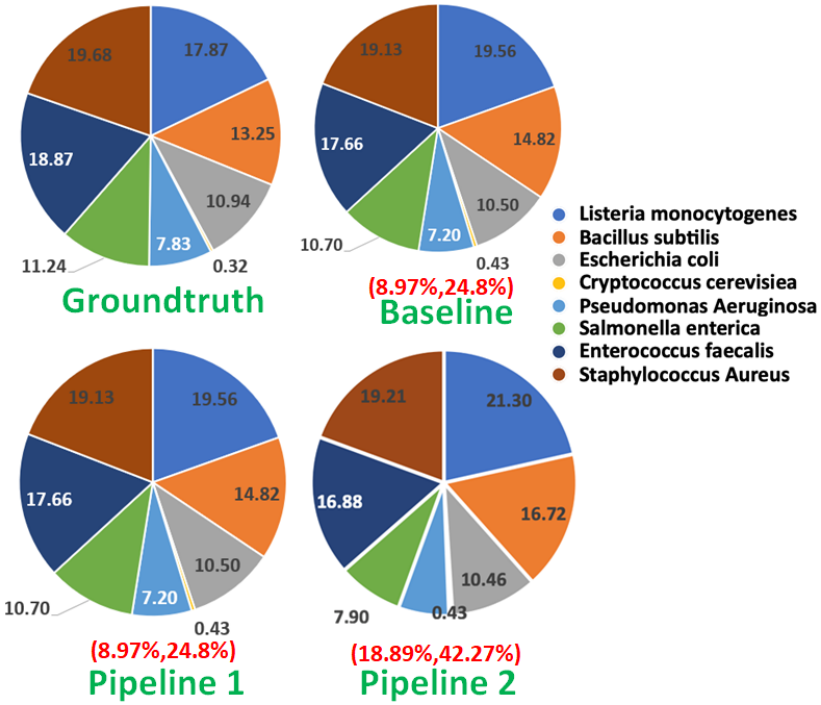

Figure 4.14: Pipeline 1 and baseline produce the best abundance estimate well within the tolerance limit.

4.6.4 Abundance estimation

We also observe that pipeline 1 produces more accurate abundance estimates than pipeline 2 while being faster than the baseline. We observe that pipeline 1 has an estimated cell number abundance with an average deviation of 8.97% and a maximum deviation of 24.8% from Zymo HMW's ground truth. This is in line with what the baseline pipeline identified as shown in Fig. 4.14. The difference is that pipeline 1 is faster than the baseline. Pipeline 2's result is comparatively more erroneous because RawMap as the primary classifier consistently misses a certain fraction of reads which are viable for abundance.

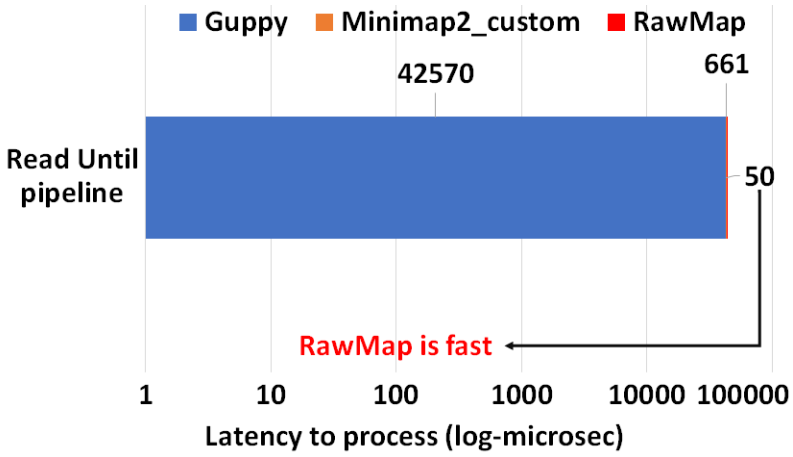

Figure 4.15: RawMap introduces negligible overhead compared to other components in the Read Until decision-making path for 450bp.

4.6.5 Compute efficiency

RawMap on a CPU is 1327X faster than Guppy-followed-by-Minimap2 which uses a GPU. This stems from the fact that RawMap is a highly efficient C++ program that performs a simple set of linear and statistical operations and hence, requires less compute compared to the deep neural network used in Guppy. Fig. 5.7 shows the insignificant compute burden introduced by RawMap on the baseline Read Until pipeline. This motivates the use of RawMap as the primary filter in our proposed pipeline 2 instead of Guppy followed by Minimap2.

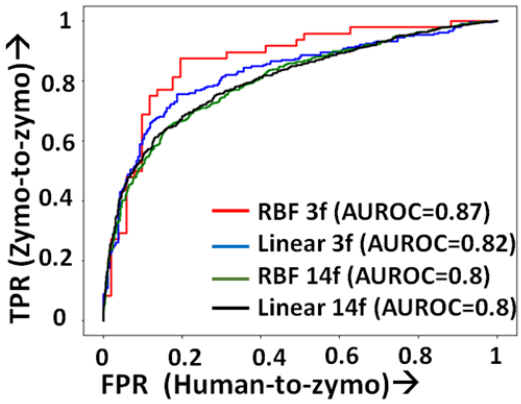

Figure 4.16: Pipeline 2 ROC: SVM with Radial Basis function kernel and three features yields the best base savings compared to a linear kernel and additional features.

4.6.6 Training

RawMap uses a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel as it gives the best AUC (as shown in Fig. 4.16). AUC was used as a scoring metric for model validation and hyper-parameters were tuned using grid search. Additionally, we also tried both linear and RBF kernels, and 11 additional features from the EEG space including Petrosian and Higuchi fractal dimensions, Fischer information, Hurst exponent, third and fourth moments, windowed maximum, minimum, and median (size=10), absolute change and correlation as shown in Fig. 4.16.

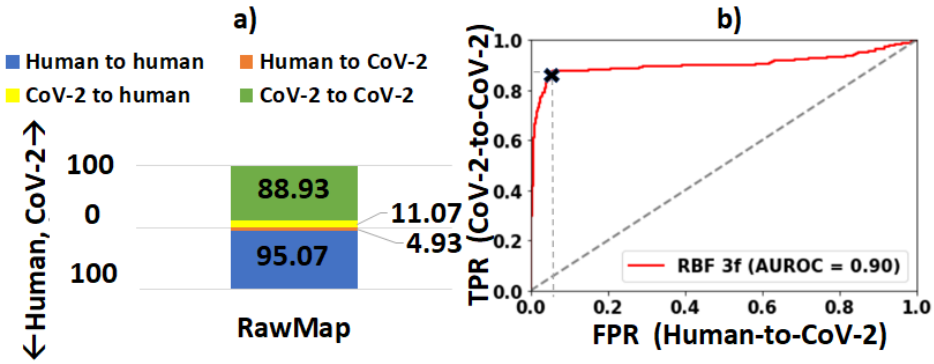

Figure 4.17: (a)RawMap is good at classifying SARS-CoV-2 from human. (b) ROC for human vs SARS-CoV-2 classification.

4.6.7 Viral load quantification

We also demonstrate how pipeline 2 may be used for efficient viral (SARS-COV-2) detection and load quantification. Since the Zymo community standard is not representative of the virus, we re-train RawMap. RawMap is re-trained on a 50: 50 digital mix of target SARS-CoV-2 and human host cDNA (12K reads) and then tested on a digitally mixed 99:1 host: target mix (100K reads). RawMap correctly retained ∼89% of all SARS-CoV-2 reads while filtering out ∼95% of the human reads (Fig. 4.17). Therefore, only a small fraction of human reads are sent to the compute-intensive step of Guppy basecalling. Minimap2 filters out the small number of additional human reads that RawMap missed. This translates to ∼14 viral copies in the test dataset if SARS-

CoV-2 RNA is 30Kbp long and the average read length is 475 bases. If the volume of the wet-lab sample is available, the number of viral copies per μl can then be estimated. This demonstrates a pipeline for viral load quantification that is computationally less expensive, as we skip basecalling and alignment. If one prefers more accurate viral load estimation, pipeline 1 may be adopted. In the future, when a viral community standard is available, RawMap may also be re-trained to function as an untargeted classifier for virus versus host cDNA.

4.6.8 Read Until's effect on pore-life

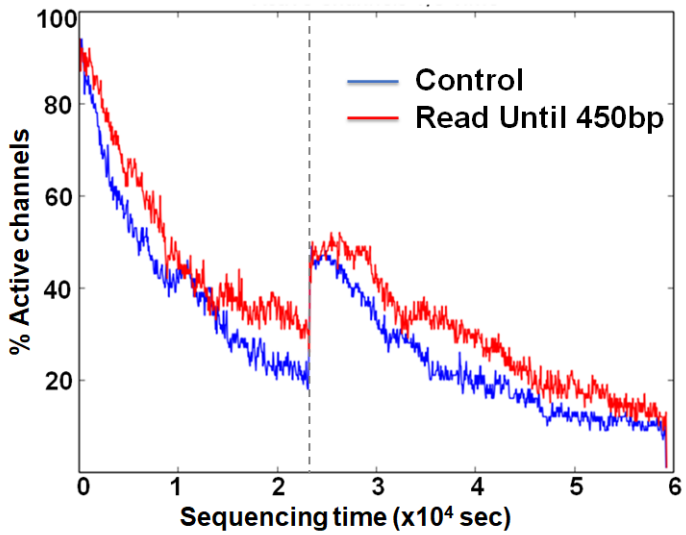

Figure 4.18: Flowcell wear-out characteristics: Read Until does not clog pores any more than normal sequencing as evident from the equal percentage of active channels recovered after wash.

We demonstrate that Read Until does not hurt the pore any more than normal sequencing does. There is no significant difference between the active number of channels between the control and the Read Until regions of the flowcell after washing followed by MUX-ing (at time marked with vertical dotted black line) as shown in Fig. 4.18. The slightly higher active channels with Read Until pores is because of some channels getting temporarily unclogged from using Read Until as noted in prior works [78]. Therefore, using Read Until (with reduced time to answer) will let us pack more useful work into the lifetime of a flowcell.

It should be noted that the Read Until benefits from experiments may depend on many factors

including sample mix constitution, average read length, and capture time of the experiment. We provide an analytical model (explained in the Supplementary section) to help estimate the savings.

4.7 Availability

The RawMap software is publicly available at: `https://github.com/harisankarsadasivan/RawMap`.

4.8 Supplementary Material

4.8.1 Modeling Read Until Benefits with RawMap

Benefits from Read Until can vary based on the sample constitution, wet-lab protocols, and also on the quality of the flowcell. However, we try to present a simplified analytical model to formulate the benefits of Read Until. We try to understand the model parameters that matter the most. We formulate the fraction of bases saved from sequencing with Read Until and analyze the trends in a simple case where we only have one Read Until classifier in the pipeline – Guppy followed by Minimap2 or RawMap. This model helps the reader to interpret the results section. Analyzing sequencing time savings can be done in a similar fashion by including average capture times in the formula for base savings.

4.8.2 Variables and assumptions

Z_t is the number of target microbial reads to be sequenced for accurate abundance estimation. For our model, we assume it to be 8000 based on our experimental results. f is the fraction of target microbes in the input mix. X is the fraction of reads alignable using modified (high accuracy) Minimap2 and is measured to be 90%. The average capture time for a strand is measured to be two seconds. The translocation rate via a nanopore is 450 bases/s. The first 200 bases of a read-prefix are trimmed and not used as it is non-informative. The subsequent 450 bases are used

for Read Until classification. Factoring in the latencies for basecalling and aligning on an Intel i7-7700K CPU, 571 bases would have translocated in the forward direction by the time a classification decision is ready for a read. RawMap's classification latency is negligible as shown in the results section and adds up to just one extra base sequenced. TP is the true positive rate of RawMap - the rate of classifying a target as a target. TN is the true negative rate of RawMap - the rate of classifying a non-target as a non-target.

4.8.3 Modelling the baseline

The total number of reads (target and non-target) to be sequenced for accurate abundance estimation can be written as:

$$N = \frac{Z_t}{X.f}$$

The total number of full-length reads sequenced by the baseline, F_b comes from target microbial reads identified by Minimap2 and also the unclassified reads:

$$F_b = (1 - X).N + X.(f.N)$$

The total number of partial length reads sequenced in the baseline, P_b is the number of human reads identified by Minimap2:

$$P_b = X.(1 - f).N$$

The total number of bases sequenced in the baseline, BP_b is the number of bases contained in full-length and partial-length reads :

$$BP_b = F_b.R_l + 571.P_b$$

4.8.4 Modeling pipeline 1

The total number of full-length reads sequenced with RawMap, F_r comes from target microbial reads identified by Minimap2 and RawMap:

$$F_r = X.(f.N) + (1 - X).\{TP.(f.N) + (1 - TN).(1 - f).N\}$$

The total number of partial length reads sequenced identified as non-target by Minimap2 is the same as the baseline, P_b.

The total number of partial length reads sequenced and missed by Minimap2 but identified as non-target by RawMap, P_r is as follows:

$$P_r = (1 - X).\{TN.(1 - f).N + (1 - TP).(f.N)\}$$

The total number of bases sequenced with RawMap added, BP_r is bases contained in full-length and partial-length reads :

$$BP_r = F_r.R_l + 571.P_b + 572.P_r$$

4.8.5 Bases saved from sequencing

The percentage of bases saved from sequencing with our proposed pipeline with RawMap compared to the baseline is calculated as follows:

$$\%BPsaved = 100\left(1 - \frac{BP_r}{BP_b}\right)$$

where BP_b is the number of bases sequenced in the baseline pipeline and BP_r is the number of bases sequenced in the proposed pipeline.

CHAPTER 5

Mm2-ax: Accelerating Minimap2 for Accurate Long Read Alignment on GPUs

5.1 Introduction: Minimap2 is slow

Amongst the many post-sequencing steps in long read processing workflows, sequence mapping and alignment is one of the first and amongst the most time and cost consuming steps. Sequence alignment [127] in bioinformatics is a way of arranging the primary sequences of DNA, RNA or protein to identify regions of similarity while sequence mapping is a subset of alignment and only finds the approximate origin of query sequence in the target. We observe that sequence mapping and alignment is slow and users often spend costly cloud instance hours to keep up with high throughput sequencers [8]. This problem can worsen as the focus shifts to longer reads.

Additionally, we find that General Purpose Graphics Processing Units (GPGPUs or simply GPUs) are becoming increasingly popular for genomics processing. Several high throughput sequencers from Oxford Nanopore [128] (GridION and PromethION series), Thermofisher's Ion Proton 48 [129] and MGI's DNBSEQ-T7 [130] have in-built GPUs. Many popular genome sequencing workflows also utilize GPUs for computation [131, 48, 132].

In this work, we present minimap2-accelerated (mm2-ax) which speeds up minimap2 (mm2) on the GPU without loosing mapping accuracy and demonstrate its time and cost benefits.

5.2 Background

5.2.1 Minimap2: A brief overview

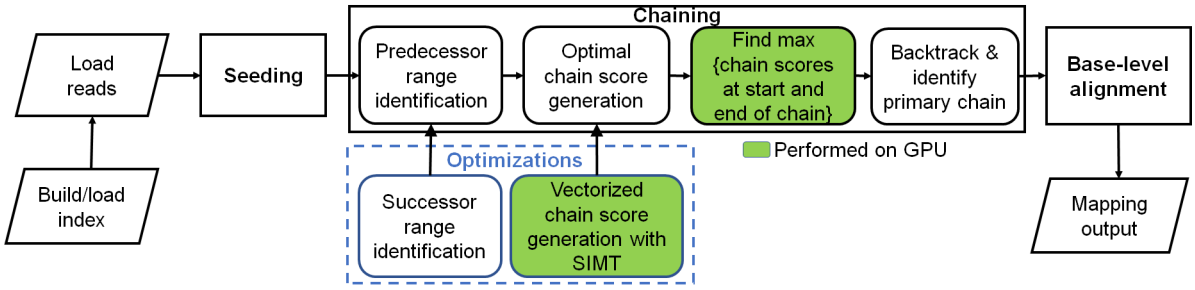

Figure 5.1: Minimap2 operates in 3 main steps: seeding, chaining and base-level alignment. Our optimizations to chaining are shown in blue box. Boxes with green fill show chaining sub-tasks which we perform on the GPU instead of CPU.

Minimap2 (mm2) [133] is the state-of-the-art DNA/mRNA sequence mapper and aligner for the most popular long read sequencing platforms like Oxford Nanopore Technologies (ONT) and Pacific Biosciences (PacBio) [134]. While BLAST[127] (using seed-extend paradigm) remains a powerful tool for full genome alignment, it is very slow especially on very long reads. For faster alignments, more recent aligners[135, 136, 137, 138, 139] including mm2 filter seeds prior to the final step of base-level alignment. mm2's algorithm is based on the seed-chain-align paradigm (detailed in Fig. 5.1) and has an offline pre-processing step to build index from target reference. In the offline pre-processing step, the reference genome is indexed to a multimap using a hash table with the popular time and space-saving k-mer samples called minimizers [140] as the key and minimizer locations on the reference as the values.

Seeding is fast and identifies short fixed-length exact matches (minimizer seeds) between a read and a reference sequence. When mm2 processes a sequenced read, minimizers from the read are used to query the reference index for exact matches (anchors). These anchors are then sorted based on position in the reference and then passed onto the next step, chaining.

Chaining takes anchors sorted based on position in the reference as the input and identifies collinear ordered sub-sets of anchors called chains such that no anchor is used in more than one

chain. mm2 implements chaining via 1-dimensional dynamic programming [141] where a complex problem is recursively broken down into simpler sub-problems. In summary, chaining sub-selects a few regions (chains) on the target reference and reduces the work for the next step of base-level alignment.

Further, if base-level alignment is requested, a 2-dimensional dynamic programming (Needleman-Wunsch[142] with Suzuki-Kazahara formulation [143]) is applied to extend from the ends of chains in order to close the gaps between adjacent anchors in the chains.

mm2 is considered accurate and has multiple use cases [133]. It may be used to map long noisy DNA/cDNA/mRNA reads, short accurate genomics reads, to find overlaps between long reads and for aligning with respect to a full reference genome or genome assembly. It is only for full genome or assembly alignment that mm2 proceeds from chaining to the last step of base level alignment.

For a more detailed understanding of how seeding and base-level alignment operates, one may refer to prior literature [144, 133]. In the context of this work, we discuss chaining in-depth as it is the bottleneck stage in mm2 we optimize.

5.2.2 Minimap2: Sequential chaining

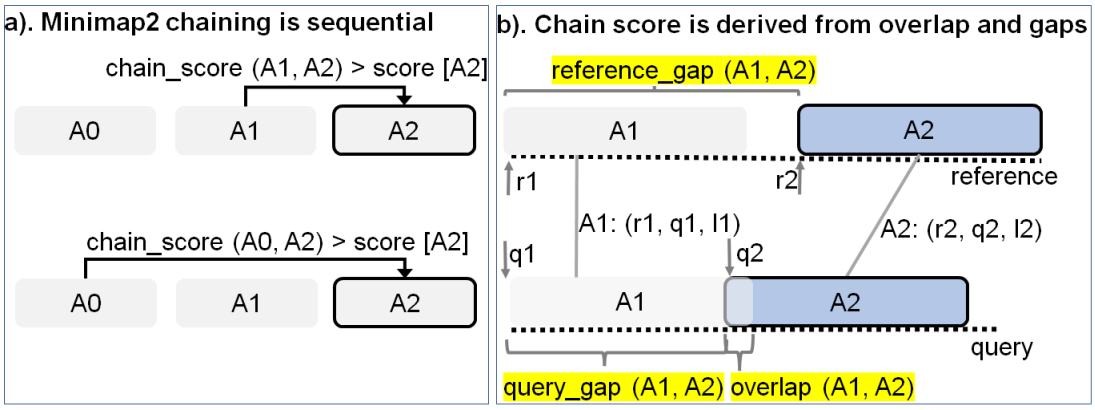

Figure 5.2: Chaining explained. (a) In Minimap2, every current anchor (A2 (r2, q2, l2) in this case) attempts to sequentially chain its predecessors within a pre-calculated predecessor range. If the chaining score with a predecessor is greater than the score value stored at current anchor A2, the new chain score and index of the predecessor is updated at A2 (in the direction of the arrow). (b) The chain score with a predecessor is computed from anchor gap cost (evaluated as a function of reference_gap, query_gap and average length of all anchors) and overlap cost.

Chaining is the second step in mm2 and sub-selects regions on the target reference where the last step of base-level alignment may be performed. An anchor is a short exact-match on the reference and is a 3-tuple (coordinate on the reference sequence, coordinate on the query sequence, length of the anchor). Chaining performs 1-dimensional dynamic programming on the input sorted anchors (from the seeding step) to identify collinear ordered sub-sets of anchors called chains such that no anchor is used in more than one chain. The chaining task can further be sub-divided into 4 sub-tasks: predecessor range selection, optimal chain score generation, finding maximum score from start and end of chains, and backtracking and primary chain identification.

Predecessor range selection is performed for every anchor in the output sorted list of anchors from the seeding step in order to dynamically calculate the number of preceding anchors (0-5000) to which chaining is attempted. While Guo et al. [145] chose a static predecessor range of 64 for every anchor, mm2 does a dynamic calculation of the predecessor range by finding all predecessors within a distance threshold.

Optimal chain score generation finds the preceding anchor within the predecessor range which yields the maximum chain score, if it exists, for every anchor. Chain score for every pair of anchors are derived from gap between anchors on the reference, gap between anchors on the query, overlap between anchors and average length of anchors as shown in Fig. 5.2b (adopted from Kalikar et al. [144] and shown here for clarity). Optimal chain score generation is the most time consuming sub-task in chaining and is sequential within a read. For every anchor in a read, mm2 proceeds sequentially through all the predecessors to generate chain scores and to find the optimal chain score as shown in Fig. 5.2a. However, mm2 has a speed heuristic based on MAX_SKIP parameter which breaks out of the sequential predecessor check if a better scoring predecessor is not found beyond a certain number of total attempts (MAX_SKIP number of attempts) for any anchor. Prior works [145, 144] have shown that removing this speed heuristic (by setting MAX_SKIP to infinity or INF) enables intra-read or more specifically intra-range parallelism (parallelizing the chain score generation with respect to all predecessors for any given current anchor) in chaining and also improves the mapping accuracy.

The third sub-task identifies the maximum of scores at start and end of every chain per anchor and is sequential for every read. Predecessor range selection, chain score generation and finding maximum of scores at start and end of chain takes most of the time (97.42%) in chaining.

Backtracking and primary chain identification together takes only 2.58% of chaining time. Backtracking extends every anchor repeatedly to its best predecessor and ensures no anchor is used in more than one chain. Primary chain identification finds primary and secondary chains based on overlaps and estimates a mapping quality for each primary chain based on an empirical formula.

5.2.3 Minimap2 profile

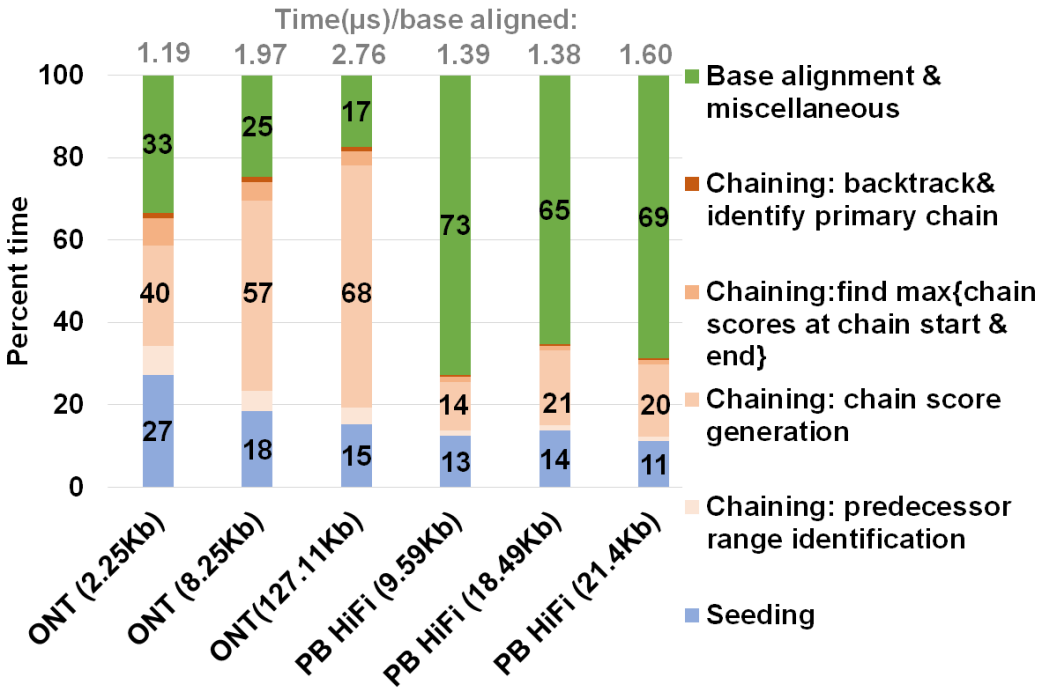

Figure 5.3: Summary of approximate time spent in seed-chain-align. mm2 takes longer to map long noisy ONT reads and spends a greater percent of total mapping time in chaining. X-axis shows the sequencing technology with mean read length of each sets of 100K randomly sub-sampled reads.

We profiled a single threaded CPU execution of mm2 on randomly sub-sampled 100K reads of ONT and PacBio HiFi on an Intel Cascade Lake core and observed different profiles as previously

noted [144]. Fig. 5.3 shows that for ONT, chaining is the bottleneck while alignment is the bottleneck for PacBio. Further, the percentage of time spent in chaining for ONT reads longer than 100Kb is as high as $\sim$68%. When the workload is normalized for the number of bases aligned, we also see that the long noisy ONT reads takes longer than PacBio HiFi on an average to align a base.

Let us consider the randomly sub-sampled ONT dataset with 100K reads of mean read length 8.25Kb (second bar from the left in Fig. 5.3). This sub-set dataset is representative of the 60X HG002 dataset with N50 as 44Kb [146] (N50 is an average read length metric used in genome assembly). Optimal chain score generation and finding the maximum of scores at start and end of a chain contribute to a significant part of the time spent in ONT chaining (90.9%). The other contributors to chaining are relatively smaller: predecessor range identification (6.6% of chaining), and backtracking and primary chain identification (2.5% of chaining).

Irregularity of workload (ONT reads vary in read lengths — a few hundred to a million bases), memory accesses, computation, and control flow associated with mm2 makes accelerating it a difficult task. Further, mm2 does not have any intra-read parallelism in chaining. Optimal chain score generation and finding the maximum of scores at start and end of a chain (which contribute to a total of 90.9% of the time in chaining) are implemented sequentially in mm2.

5.2.4 Prior Work

There have only been a few prior works [144, 145, 147, 148] which try to improve the performance of (accelerate) mm2. Zeni et al. [147] and Feng et al. [148] accelerate the base-level alignment step which is no longer the dominant bottleneck as reads have grown longer in length. Guo et al. [145] and Kalikar et al. [144] remove the MAX_SKIP heuristic for speed in mm2 in order to extract intra-range parallelism and parallelizes chain score generation for each anchor (MAX_SKIP is set to INF). While Guo et al. [145] correctly identifies chaining as the bottleneck for longer reads, introduces the concept of forward transforming the chaining algorithm and accelerates it on GPU and Field Programmable Gate Array (FPGA), this work fails to guarantee output equivalency

to mm2 with MAX_SKIP set to INF. We find that it misaligns (produces mismatched primary alignments) ∼7% of the reads with lengths above N50 while also failing to align ∼2% of those reads from our ONT 60X HG002 dataset. This decrease in mapping accuracy is mainly because Guo et al. follows a static predecessor range selection unlike the dynamic selection in mm2 and also because the chaining score update rules are not modified accordingly with the transform.

mm2-fast [144] is the most recent prior work in accelerating mm2 and accelerates all three steps in mm2 utilizing Single Instruction Multiple Data (SIMD processes multiple data with a single instruction) CPUs. While mm2-fast parallelizes chain score generation, we identify certain sections of chaining which are not parallelized. We profiled mm2-fast on the 100K sub-sampled reads from ONT and find that 34.08% of the total time spent in doing chain score generation and finding the maximum of scores at start and end of chains is in sequential code and not parallelized. mm2-fast does not use SIMD lanes when predecessor range is less than or equal to 5, for finding maximum predecessor score index and finding maximum of scores at start and end of chains for every anchor. This motivates the need for a better parallelization scheme.

5.2.5 Our contributions

In this work, we optimized the dominant bottleneck of mm2 in processing long noisy reads, chaining, on the GPU without compromising accuracy. We show mm2-ax has better speedup and speedup : costup compared to mm2-fast, a SIMD-vectorized version of Minimap2 on 30 Intel Cascade Lake cores. As discussed, mm2 presents a difficult task to parallelize with sequential chaining step and irregular workloads, memory accesses, computation, and control flow. Prior efforts at accelerating chaining either produces alignments significantly deviant from mm2 [145] or still does some amount of sequential execution within chaining and can benefit from a better parallelization scheme [144]. Hence, we attempted to better utilize the inherent parallelism in chaining without compromising accuracy on GPUs which are becoming increasingly popular for genomics workflows. To this end, we forward transform the predecessor range calculation to successor range calculation so as not to lose mapping accuracy and also forward transform the optimal chain score

generation to introduce intra-range parallelism. Forward transformed chaining eliminates the need to sequentially find the maximum of all chain scores from all the SIMD lanes and instead enables better utilization of Single Instruction Multi-Threaded (SIMT is similar to SIMD but on a GPU) parallelization scheme on a GPU. Additionally, we also benefit from inter-read parallelism by concurrently processing multiple reads on the large number of Streaming Multiprocessors (SMs) on the GPU.

We designed a heterogeneous system where the bottleneck step, chaining, is sped up on the GPU while seeding and base-level alignment happens on the CPU. We exploit the low memory footprint of mm2 and trade-off memory for performance via better occupancy of the GPU resources by the highly irregular workload in mm2 chaining. Minimal branch divergence, coalesced global memory accesses and better spatial data locality are some of the optimizations.

We compare our accelerated minimap2 (mm2-ax) on GPU to SIMD-vectorized mm2-fast on CPU. Our evaluation metrics include accuracy, speedup, and speedup : costup. We show that mm2-ax produces 100% identical alignments to mm2-fast (same accuracy as mm2 with MAX_SKIP set to INF) and delivers 5.41 - 2.57X speedup and 4.07 - 1.93X speedup : costup with respect to mm2-fast on ONT 60X HG002 dataset.

5.3 Methods

5.3.1 Parallelizing chaining: forward loop transformation

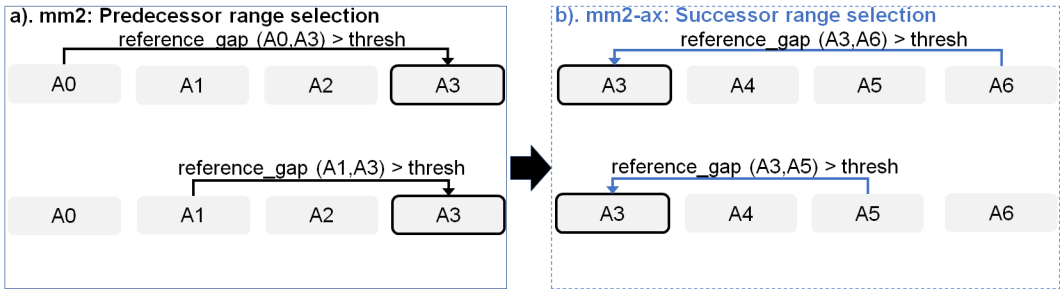

Figure 5.4: Forward transforming predecessor range selection to successor range selection: The cell with solid black outline represents the current anchor for which predecessor/successor range calculation is performed. The arrow starts from the predecessor/successor and points to the current anchor A3 whose range is updated sequentially.

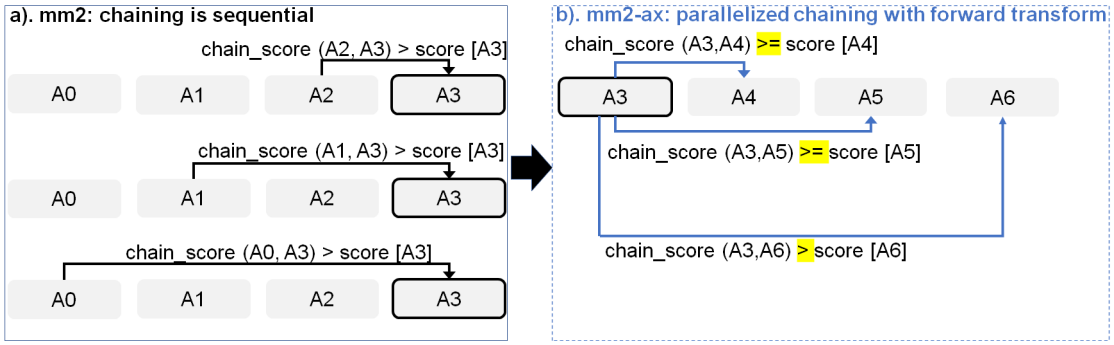

Figure 5.5: Parallelizing Minimap2's chain score generation (shown in a) by forward transformation (shown in b). Additionally, we retain mapping accuracy by modifying the score comparison check ($>$ to $>=$) with all anchors except the immediate neighbor to enable farther anchors to take precedence over neighboring anchors to be forward chained.

Chaining in mm2 identifies optimal collinear ordered subsets of anchors from the input sorted list of anchors. mm2 does a sequential pass over all the predecessors and does sequential score comparisons to identify the best scoring predecessor for every anchor. The exact chaining algorithm used in mm2 is not parallelizable and hence, mm2 is only able to utilize inter-read parallelism. Prior works [144, 145] have shown that removing the speed heuristic in chaining by setting MAX_SKIP to INF enables intra-range parallelism (parallel chain score generation for all predecessors for any

given anchor, i.e, parallelizing the inner for loop in Algorithm 2) and improves mapping accuracy. However, the total amount of work to be performed per anchor increases. We apply the same configuration in mm2-ax.

We find that $\sim$34% of the run time in mm2-fast's optimal chain score generation and finding maximum of scores at start and end of all chains is spent sequentially. Chain score generation when the predecessor range is lesser than or equal to 5, finding maximum chaining score from among the 16 vector lanes and finding maximum of scores at start and end of all chains are all performed sequentially. In order to make better use of intra-range parallelism in chaining, we forward transform predecessor range selection (Fig. 5.4) and optimal chain score generation (Fig. 5.5 and Algorithm 2). This saves us the sequential passes which mm2-fast does to find the maximum chaining score.

In this context, forward transformation refers to changing the order of computation to parallely evaluate successor anchors instead of iterating through predecessor anchors. This enables us to perform chain score generation and update in parallel as shown in Fig. 5.5. Although the forward transformation of optimal chain score generation is first introduced by Guo et al. [145], in order to retain mapping accuracy, we implement two novel modifications. First, we calculate dynamic successor range instead of a static range of 64 for every anchor prior to chaining. We efficiently implement the successor range calculation with few iterations based on insights from cumulative distribution function of predecessor ranges for all anchors (discussed later in Fig. 5.6b). Secondly, the chain score update policy is modified from $>$ to $\geq$ (except for the immediately neighboring anchor) for the forward traversal as shown in Fig. 5.5b. This ensures that farther anchors get precedence over nearer ones for forward chaining.

Algorithm 2 Forward transformed chaining in mm2-ax

Input: $L : \{a1, a2, ..., an\}$ - List of anchors sorted according to the reference positions

Output: S, P: Maximal chaining score for each anchor in L and their optimal predecessor's indices; V: maximum of scores at start and end of chain for every anchor.

for i $\leftarrow$ 1 **to** n step: 1 **do**
    end $\leftarrow$ i + 5000
    while delta **in** {0, 16, 512, 1024, 2048, 3072, 4096, 5000} **do** ▷ Speed heuristic to select range
        if reference_gap (a_i, $a_{i+delta}$) is too large **then**
            end $\leftarrow$ i + delta
            break
    while reference_gap (a_i, a_{end}) is too large **do** ▷ Dynamic successor range selection
        end $\leftarrow$ end - 1
    for j $\leftarrow$ i+1 **to** end step: 1 **do**
        if query_gap (a_i, a_{end}) is too large or negative **then**
            continue
        $s_j \leftarrow$ S[i] + a_i.l - (gap_cost(i,j)+overlap(i,j))
        if $s_j >=$ S[j] and ! ($s_j ==$ S[j] and S[j] == a_j.l) **then** ▷ Find better successor to chain
            S[j] $\leftarrow s_j$
            P[j] $\leftarrow$ i
    if P[i] >= 0 and V[P[i]] > S[i] **then** ▷ Find maximum of scores at start and end of chain
        V[i] $\leftarrow$ V[P[i]]
    else
        V[i] $\leftarrow$ S[i]

5.3.2 Heterogeneous system design

mm2-ax is a heterogeneous design (uses specialized compute cores, GPUs in this case) which performs seeding and successor range identification on the CPU and efficiently implements optimal chain score generation and finding maximum of scores at start and end of chain on the GPU. The output scores and optimal successor index arrays from chaining are returned to the host CPU for backtracking. From mm2's profile in Fig. 5.3, optimal chain score generation and finding maximum of scores at start and end of chain contribute 90.6% of chaining time and is accelerated on the GPU. Seeding, successor range identification or forward transformed predecessor range identification (6.6% of chaining), backtracking and primary chain identification (2.5% of chaining) and base-level alignment are performed on the CPU.

Further, the heterogeneous design also helps us better balance the workload and reduce resource idling on the GPU, as discussed in below sections.

5.3.3 GPU occupancy: Condensed workload vector and workload balancing

We find that $\sim$67% of the input anchors do not start a chain and this contributes to the sparsity of the successor range vector which is to be input to optimal chain score generation. In order to better occupy the GPU resources with the irregular workload, we perform successor range identification (steps 3-8 in Algorithm 2) on the CPU to convert this sparse input vector of successor ranges which defines the workload in chain core generation, into a condensed one with non-zero successor ranges. This incurs a GPU and host memory trade-off for better performance by ensuring GPU threads do not idle on anchors with a successor range of zero. Further, the compute overhead on the host CPU from successor range identification is minimized by implementing a speed heuristic (steps 3-6 in Algorithm 2) to reduce the number of iterations in identifying the successor range for every anchor. This is based on the observation that $\sim$67% of the anchors on an average have a predecessor/successor range of zero and $\sim$93% have a range lesser than or equal to 16.

Further, we also implement a series of additional measures to ensure better GPU occupancy, as we realize that this is one of the most important problems [56] while dealing with ONT reads

of variable lengths and predecessor ranges. To ensure GPU occupancy from workload balancing, we bin and batch reads of similar lengths together onto the GPU. For example, reads of length 2Kb-3Kb, 3Kb-4Kb and 4Kb-5Kb are binned together. For smaller read lengths (<= 10Kbp), we define each concurrently launched workload at a coarser grain, i.e, as many reads as it takes to concurrently occupy all the SMs on the GPU. For longer reads, we observe that this does not yield the optimal performance because long reads present a case of highly imbalanced workloads as reads are more variable in length and any SM which may finish early remains unused. For example, in 50-150Kb range, reads are highly varying in read lengths, and it is difficult to find multiple reads within 1Kb variance in lengths. Hence, for longer reads we keep bin ranges wider: 45-50Kb, 50-100Kb and 100-150Kb. For longer reads, we follow a two-fold strategy for higher GPU occupancy. First, we define fine-grained workloads, i.e, with only as many reads as it takes to occupy an entire SM. Second, we always follow up very-long read bin workloads with fine-grained workloads of shorter read lengths (2Kb). This twofold strategy helps better balance highly imbalanced workloads of very long reads.

For better GPU occupancy, we also launch multiple concurrent GPU kernels (functions) using CUDA streams (GPU work queues). As soon as a hardware resource gets free on the GPU, the scheduler executes the next kernel. Additionally, each Streaming Multiprocessor (SM) concurrently processes multiple reads.

Data transfer between the CPU and GPU are overlapped with compute on the GPU by issuing asynchronous memory copies on CUDA streams. We also benefit from the higher bandwidth of HBM2 and the eight copy engines on A100.

5.3.4 Inter-read and intra-range parallelism

A server-class GPU like NVIDIA A100 has 108 SMs. The key to high performance on the GPU is to ensure that all the SMs always have useful work to do and there are sufficient Single Instruction Multi-Threaded (SIMT) warps/sub-warps (groups of threads) concurrently on the GPU to hide the relatively higher global memory access latencies (i.e, ensure higher warp occupancy). While we

utilize only inter-read parallelism (concurrently processing multiple reads per SM) for finding the maximum of scores at start and end of chain, we utilize both inter-read and intra-range parallelism (via forward transformation) for the optimal chain score generation. Intra-range parallelism comes from concurrent warps (sets of 32 parallel threads) performing chain score generation in parallel for all successors within the successor range of a given anchor.

The next anchor attempts to chain only after it's previous anchor's optimal chain score generation step is completed. To this end, we have a thread synchronization barrier (__syncthread()) waiting on all the threads to finish chain score generation and update for all the successors of a given anchor. Please note that Guo et al.[145] uses more synchronization barriers (six of them) in the chain score generation kernel. However, we only need one as we reduce the number of points of branch divergence by combining multiple condition checks together.

5.3.5 Data locality

We observe optimal benefits from optimizing for better spatial data locality rather than temporal locality. Temporal cache locality refers to re-use of data in cache, while spatial cache locality refers to use of data from adjacent storage locations. For example, we pre-fetch data for a group of successors per current anchor in each concurrently processed read into L1 cache for better performance from improved spatial data locality. We use the PTX instruction __prefetch_global_l1 to prefetch the successor anchor's inputs (query and reference coordinates) and chaining output (score and parent values) from global memory to L1 cache in a coalesced fashion for every set of 32 successors per anchor.

While Guo et al. [145] attempted to exploit temporal locality from using shared memory (memory shared between parallel threads of a read) with a static successor range, this approach does not prove beneficial with a dynamic successor range because of limited scope for any benefits from temporal data locality. Frequent cache misses due to different successor ranges lead to data transfer latency from shared memory to registers, adding up to outweighing any benefit from using shared memory at all. We therefore use more registers per GPU thread instead of utilizing shared

memory.

We also coalesce global memory reads and writes for successor anchors to reduce the total number of transactions to high-latency global memory.

5.3.6 Minimal branch divergence

Conditional branches are kept to a minimum in our implementation by combining conditions when successors are not updated after score generation. This helps reduce branch divergence, which affects performance on the GPU. There are only two conditional blocks for every read that is processed within the chain score generation kernel (one for score generation and the other for update, as seen in Algorithm 2). On the other hand, Guo et al.[145] has nine conditional blocks evaluated per read.

Further, we utilize CUDA's warp-synchronized integer intrinsics to efficiently perform operations like logarithm and absolute differences. __clz() lets us efficiently calculate logarithm during the chain score generation step from counting the leading zeros and subtracting this count from the number of bits in int32 datatype (32). __sad() enables us to efficiently compute the overlap cost from the absolute difference of query_gap and reference_gap (shown in Fig. 5.2b).

5.4 Implementation

5.4.1 Experimental Setup

Minimap2 (mm2) is a fast evolving software with 7 new releases on the master branch and 2 new branches incorporating mm2-fast in the year 2021 alone. We decided to accelerate Minimap2 v2.17 which is used in Oxford Nanopore's variant calling pipeline with Medaka [48]. Kalikar et al. [144] has accelerated Minimap2 versions v2.18 and v2.22. mm2 v2.18 and v2.17 produce equivalent results in chaining and are, hence, comparable.

We demonstrate the benefits of our chaining optimizations on a server class NVIDIA A100

GPU. Our evaluations are performed on Google Cloud Platform (GCP). Speedup is normalized to a costup factor to evaluate a speedup : costup metric in order to take into account the usually higher GPU costs on the cloud. Our costup factor on GCP is 1.33X, but this would be lower if one were to use Amazon Web Services (AWS). mm2-ax is evaluated on a single GCP instance of a2- highgpu-1g with 85GB of host memory and one NVIDIA A100 GPU of 40GB memory. We compare mm2-ax to the SIMD accelerated mm2-fast (fast-contrib branch of mm2) on a single GCP c2-standard-60 instance (30 AVX-512 vectorized Intel Cascade Lake cores and 240GB memory).

We use NVIDIA Nsight Compute [93] for profiling GPU events and Nsight Systems [92] for visualization of concurrent GPU events. Seqtk [149] is used for random sub-sampling of DNA sequences. We used Perf [150] for profiling mm2 on the CPU.

To ensure better GPU resource utilization with nanopore reads of varying length (few hundred to a million bases), we bin reads based on read lengths before batching their sorted anchors onto the GPU for chaining. For example, reads of length 1-2 Kilobases (Kb) go to the same bin, reads of length 2-3Kb go to the same bin etc. However, for longer read lengths, we bin 50K-100K, 100K-150K etc. because it is relatively harder to find reads closer in read lengths.

The reads within a bin could still present an unbalanced workload as the predecessor ranges of every anchor is different. This binning may be done very efficiently during basecalling as the basecaller has access to read lengths, and hence the overhead introduced is negligible. We also try to fit in as many reads as possible on to the GPU's DRAM for every read bin. We measure the compute time for optimal chain score generation and sub-task to find maximum of scores at start and end of chain on the GPU and compare it to that of the SIMD baseline to evaluate SpeedUp metric (compute time taken by CPU baseline mm2-fast divided by time taken by mm2-ax on GPU). We then divide this with 1.33X to normalize for cost and calculate the speedup : costup metric. The overhead presented by successor range selection over predecessor range selection on the host CPU is very negligible ($< 2.8\%$ of total CPU time) and is, hence, not considered for our analysis. Further, it is worthwhile to note that successor range identification can outperform predecessor range identification using SIMD vectorization on the host CPU as our forward transform essentially

makes successor range identification parallelizable.

Further, we also evaluate mapping accuracy of mm2-ax vs mm2-fast (or mm2 with MAX_SKIP set to INF). Mapping accuracy is defined as the number of reads from mm2-ax producing bit-exact chains to mm2-fast. If any of the 12 fields in mm2-ax's Pairwise Alignment Format (PAF) formatted output differs from that of mm2's in the primary alignments, we treat the read as misaligned. The datasets we use are publicly available [151, 152, 153, 154] — HG002 genome sequenced by ONT PromethION with 60X coverage and 15Kb and 20Kb PacBio HiFi reads with 34X coverage.

5.4.2 Optimal GPU configurations

Of the two chaining sub-tasks offloaded to the GPU, chain score generation takes approximately greater than 95% of the time on the GPU. Hence, we discuss how we performed design-space exploration to identify the optimal GPU kernel launch parameters for this sub-task. Kernel launch parameters refer to a predefined configuration with which a kernel or function may be executed on the GPU. In this context, we can define the chain score generation kernel launch parameters as a 3-tuple (thread blocks per SM, number of concurrent reads processed per block in an SM , number of parallel threads per read). In this context, thread blocks are groups of parallel threads within an SM which may or may not be processing the same read.

We find the register requirement per thread on an NVIDIA A100 GPU to figure out the achievable upper bound of GPU kernel launch parameters on the A100 GPU. Using NVIDIA's Nsight Compute Profiler, we profiled mm2-ax and observed that we require 53 registers per thread for the optimal chain score generation kernel, and this observation helps provide an upper bound on the maximum number of parallel threads that can be launched on the SM in our case. From Fig. 5.6b, one may try to fit more concurrent reads with 16 or 32 threads per read, but it is observed that this configuration hurts spatial cache locality across reads and is hence, not beneficial. The optimal configuration is observed to be in the direction of higher concurrent reads per SM instead of per thread bock and towards more threads allocated per read for chaining. This is because having more threads per read enables better spatial data locality in L1 cache through larger coalesced

global memory accesses. We find that (9 thread blocks per SM, 1 concurrent read per thread block, 128 parallel threads per read) is the best performing kernel configuration. This is followed by (3, 3, 128) and (1, 4, 256).

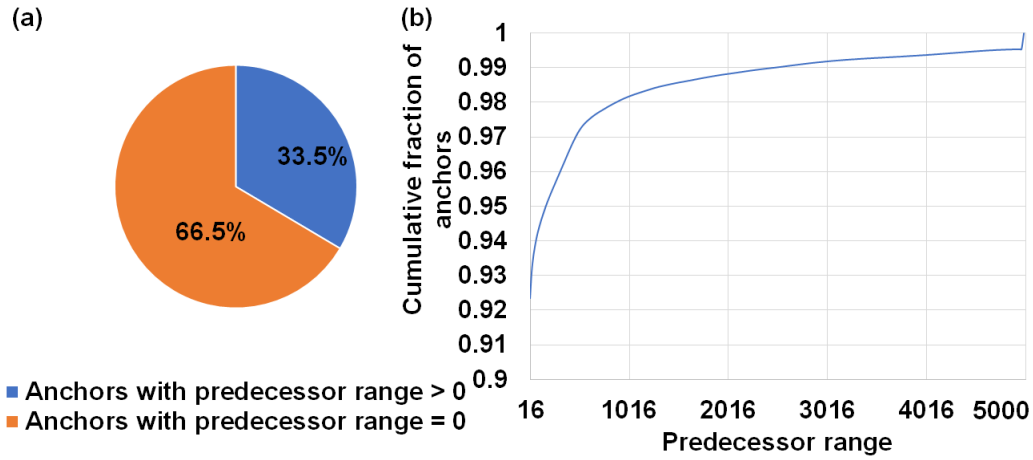

Figure 5.6: Workload is sparse and irregular. (a) $\sim$67% of anchors fed to the chaining step do not start a chain. (b) Predecessor range is less than or equal to 16 for $\sim$92% of all anchors and goes as high as 5000 only for a small fraction of total anchors.

From Fig. 5.6a, we observe that $\sim$67% of anchors fed to the chaining step do not start a chain. This observation helps us to ensure better arithmetic intensity (more computations per byte of data fetched from high latency global memory). In this regard, we perform successor range identification on the host CPU and condense the sparse vector of successor ranges to a dense one with non-zero successor range before offloading the chain score generation sub-task to the GPU. Further, Fig. 5.6b informed us to efficiently implement successor range identification. 67% of the anchors have predecessor ranges equal to zero, and greater than 92% have predecessor ranges less than or equal to 16. We use this information to efficiently implement successor range selection by reducing the number of total iterations.

We did a design space exploration to identify the granularity of dispatching work to the GPU for optimal performance. While concurrently dispatching coarse grained workloads (each workload is defined with as many reads as it takes to fill all the 108 SMs on A100) yielded the best results for reads smaller than or equal in length to 10Kb, coarse grained workloads do not work well for longer reads as any SM that finishes early may idle. Concurrently dispatching fine-grained workloads

(each workload is sized with as many as reads as it takes to fill only a single SM) yielded the best performance for reads longer than 10Kb. Concurrent fine-grained dispatches with each workload sized to fill 2 and 4 SMs closely competes but gives slightly lower benefits. Additionally, a fine-grained workload for longer reads is always followed by fine-grained workload of smaller reads (we choose 2Kb) to yield a better performance. This is because multiple fine-grained workloads of smaller reads can better load balance without adding a significant tail to the critical path.

5.5 Results

mm2-ax demonstrates a 5.41 - 2.57X speedup and 4.07 - 1.93X speedup : costup over SIMD-vectorized mm2-fast baseline as shown in Fig. 5.7a. It is observed that coarse-grained load dispatching to the GPU is better for read lengths smaller than 10Kb while fine-grained load dispatching to each SM is better for longer reads. In Fig.5.7b we show the chaining performance gain factor without including the data transfer related costs (memory allocation, asynchronous memory copies on CUDA streams and data serialization).

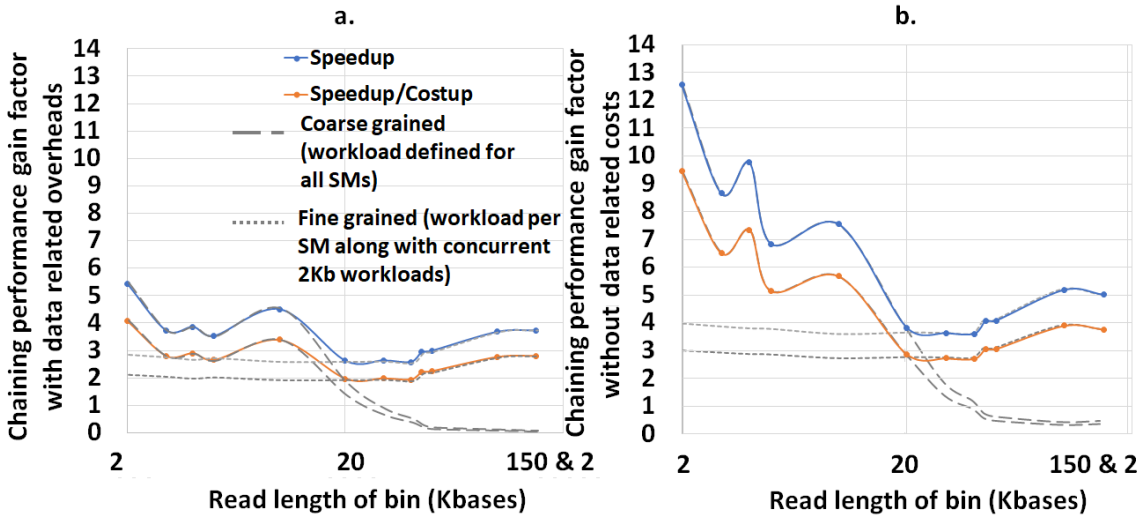

Figure 5.7: (a) mm2-ax yields 5.41 - 2.57X speedup and 4.07 - 1.93X speedup : costup over SIMD-vectorized mm2-fast baseline. (b) The chaining performance across various read lengths may be further improved by ∼1.3-2.3X if we can engineer to hide the data transfer related costs.

Additionally, we also evaluate the mapping accuracy of mm2-ax with respect to mm2-fast on

the complete 60X HG002 ONT dataset and we observe that all the output chains match in all the first 12 fields of the output Paiwise Mapping Format (PAF) file. As mm2-ax is limited by the DRAM capacity of the GPU, we use minimap2 modified with our forward transformed chaining logic on the CPU to evaluate accuracy on the very large 60X HG002 dataset. We then compare and validate the output PAF file to the outputs of both mm2-fast and mm2 modified with MAX_SKIP set to INFINITY.

5.6 Discussion

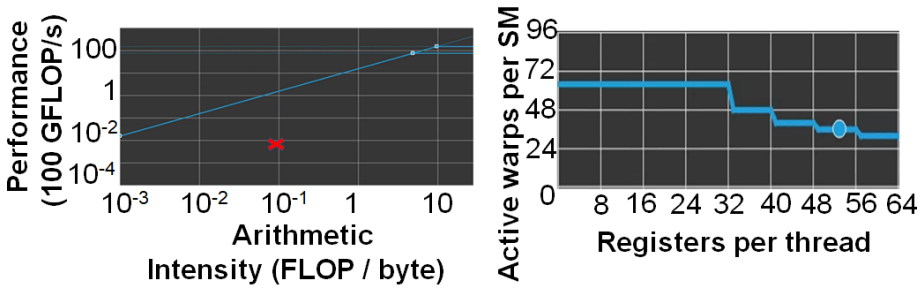

Figure 5.8: mm2-ax is memory bound. (a)Roofline Plot: Chain score generation is memory bound. Operating point is shown in a red cross mark. (b) Theoretical warp occupancy on the GPU is bounded by the number of registers used by each thread.

We only discuss the profile of the most time-consuming kernel of mm2-ax on the GPU, optimal chain score generation. Here we present the profile of optimal chain score generation kernel on the 2Kb bin of reads. From Fig. 5.8a, we see that the chain score generation kernel on the GPU is memory bound. Unless we increase the arithmetic intensity of the kernel, we cannot transform it to a compute bound kernel. From Fig. 5.8b, we observe that high register usage per thread limits the theoretical number of active warps per SM on the GPU. The higher the warp occupancy, the better the kernel is in hiding the relatively longer global memory access latency on the GPU. The achieved number of active warps per SM is 12 (33% of the theoretical maximum). One way to ensure there are enough warps on the SM is by integrating asynchronous FIFOs at the input and output of chaining to better manage input and output out of the GPU. One may also try to better hide the data transfer related costs to gain further improve in performance.

Further, it may be noted that one may further improve performance by making the range of read lengths that go into a bin even smaller. In our case, the HG002 dataset presented reads in 50-150Kb range to be highly varying in read lengths and hence, we defined long read bins with higher variance in read lengths. It is also worthwhile to consider porting the entire mm2 software to the GPU as most long read sequencing workflows are now shifting to GPUs.

5.7 Availability

mm2-ax is currently closed-source. However, a docker image is publicly available to test the software: `https://github.com/hsadasivan/mm2-ax`.

CHAPTER 6

Portable Microbial Diagnostics

We envision a future where clinics and homes are equipped with MinIONs that may be connected to a laptop for portable microbial diagnostics. With the portable VolTRAX, the wetlab preparation is soon to be accessible to the common man. On the laptop, we hope to see DTWax being used for targeted pathogen detection, and RawMap for untargeted classification and microbiome abundance estimation. mm2-ax will perform mapping and alignment.

On the laptop, the CPU cores can be utilized for untargeted microbial classification using RaMap. The GPU can be utilized for real-time Read Until using DTWax and offline steps like basecalling and variant calling. It is possible that mm2-ax may be developed further to run all the stages of Minimap2 on the GPU.

We strongly believe a laptop is the ideal portable client device for a future MinION due to several reasons. Firstly, it might be challenging to have a custom SquiggleFilter chip on every MinION because of its limited on-chip memory and the rapid improvements in ONT's sequencing technology and algorithms. Further, the high amount of data parallelism in the sequencing output and algorithms suggests a strong case for using GPUs. However, we estimate that not even a superior edge platform like the NVIDIA Jetson Orin AGX has the necessary compute power to keep up with a future MinION sequencer (Orin AGX's GPU offers $\sim$7X lower throughput than A100) to perform Read Until. From our measurements, we note that an NVIDIA RTX 3080 Ampere GPU is well-equipped to handle the throughput of a future MinION. RTX 3080 is only $\sim$1.65X slower than an A100 for mm2-ax and DTWax. This will only improve as NVIDIA ushers

in the latest generation of GPUs with the DPX instruction set for dynamic programming. RawMap, on the other hand, is a lightweight software and can utilize the CPU cores to match the future MinION's sequencing throughput.

Additionally, it is also possible that the MinKNOW software can upload the results to a prompt-based AI software that can keep track of patient history, make diagnostics and answer any questions the patient may have.

CHAPTER 7

Conclusion

The MinION is a handheld DNA sequencer increasingly used in Precision Medicine applications but lacks onboard computing, limiting its portability. This thesis introduces two clinical applications of the MinION and identifies and solves performance bottlenecks through hardware-software solutions to enable portable microbial diagnostics. Finally, we discuss how our accelerated solutions will fit on a laptop with a GPU.

More than 99% of DNA reads in a typical human sample are non-target (human), which may be skipped in real-time using MinION's Read Until feature. This thesis analyzes the performance of the Read Until pipeline in detecting target microbial species for two different applications– viral pathogen detection and human microbiome abundance estimation. We find new sources of performance bottlenecks that are not addressed by past genomics accelerators.

SquiggleFilter (ASIC-based) and DTWax (GPU based) are our programmable solutions to targeted pathogen detection. RawMap is a smart and efficient species-agnostic (untargeted) classifier on the CPU for human microbiome abundance estimation. We also discuss mm2-ax which accelerates the bottleneck stage in Minimap2 software used in all MinION sequencing workflows.

In designing a universal virus detector, we identify the basecaller as a significant bottleneck in filtering non-target reads. This compute problem is only going to get worse, as the throughput of nanopore sequencers is expected to increase by 10-100$\times$ in the near future. We address this problem using hardware-accelerated SquiggleFilter for filtering non-target reads without basecalling them. SquiggleFilter is programmable and portable. We show that our 14.3W 13.25mm^2

accelerator has 274× greater throughput and 3481× lower latency than existing approaches while consuming half the power, enabling Read Until for the next generation of nanopore sequencers.

SquiggleFilter's programmability is limited by the size of the on-chip memory. We adapt SquiggleFilter's underlying subsequence Dynamic Time Warping (sDTW) algorithm to the more easily programmable and scalable GPUs. To make sDTW performant on the GPU, we do offline pre-processing of target reference to ensure coalesced loads from global memory, reduce branch divergence, and utilize FP16 vectorization and tensor core pipes. Further, we use warp-shuffles for efficient intra-sub-matrix communication and shared memory for low-latency inter-sub-matrix communication. Further, we assume no reference deletions to improve both the throughput and F1-score. We show that DTWax on an NVIDIA A100 GPU achieves ∼1.92X sequencing speedup and ∼3.64X compute speedup: costup over a sequencing workflow that does not use Read Until.

sDTW-based techniques are not fast enough to classify multiple large target bacterial species. RawMap is a smarter and more efficient CPU-only microbial species-agnostic squiggle-space Read Until classifier that is developed as a "plug-and-play" solution to complement the baseline Read Until pipeline. We perform feature engineering to extract non-linear non-stationery characteristics out of ONT squiggles and learn the differences between human and microbe using a Support Vector Machine.

RawMap is 1327X faster than the state-of-the-art solution and improves the sequencing time and cost, and compute time savings from using Read Until. We demonstrate two different pipelines to optimally utilize RawMap. RawMap as a secondary filter (pipeline 1) yields ∼24% sequencing time and cost savings whereas RawMap as a primary filter (pipeline 2) yields ∼22% compute time savings compared to the baseline Read Until pipeline. We also show that RawMap can serve as an untargeted filter (classify unseen species) with nearly the same accuracy. Additionally, we also present how RawMap may be utilized instead of RT-PCR tests for viral load quantification of SARS-CoV-2.

Minimap2 is the state-of-the-art aligner used in all MinION workflows. We identify chaining as the bottleneck step that constitutes up to ∼67% of the total alignment time. We address

this problem with mm2-ax (minimap2-accelerated), a heterogeneous design for accelerating the chaining step of minimap2 with bit exact output. We implement various optimizations to ensure better occupancy and workload balancing on the GPU. Some key optimizations include forward transformed chaining for better intra-read parallelism, workload condensing, trading-off host and GPU memory for better performance on the GPU, better spatial data locality, and minimal branch divergence. We show mm2-ax on an NVIDIA A100 GPU improves the chaining step with 5.41 - 2.57X speedup and 4.07 – 1.93X speedup: costup over the fastest version of minimap2, mm2-fast, benchmarked on a single Google Cloud Platform instance of 30 AVX-512 vectorized cores (Intel Cascade Lake).

Finally, we discuss how all these solutions can fit onto a laptop with a GPU such as an RTX 3080. The GPU can be utilized for real-time Read Until using DTWax and also for offline steps like basecalling and variant calling. It is possible that mm2-ax may be developed further to run all the stages of Minimap2 on the GPU. The CPU cores can be utilized for untargeted microbial classification using RaMap.

BIBLIOGRAPHY

[1] J. Hasell, E. Mathieu, D. Beltekian, B. Macdonald, C. Giattino, E. Ortiz-Ospina, M. Roser, and H. Ritchie, "A cross-country database of covid-19 testing," *Scientific data*, vol. 7, no. 1, pp. 1–7, 2020.

[2] T. Sauvage, W. E. Schmidt, H. S. Yoon, V. J. Paul, and S. Fredericq, "Promising prospects of nanopore sequencing for algal hologenomics and structural variation discovery," *BMC genomics*, vol. 20, no. 1, pp. 1–17, 2019.

[3] E. C. Hayden, "Genome researchers raise alarm over big data," *Nature*, vol. 7, 2015.

[4] M. J. Khoury and K. E. Holt, "The impact of genomics on precision public health: beyond the pandemic," 2021.

[5] J. E. Lunshof, J. Bobe, J. Aach, M. Angrist, J. V. Thakuria, D. B. Vorhaus, M. R. Hoehe, and G. M. Church, "Personal genomes in progress: from the human genome project to the personal genome project," *Dialogues in clinical neuroscience*, 2022.

[6] E. Pennisi, "A $100 genome new dna sequencers could be a "game changer" for biology, medicine," Jun 2022.

[7] T. Mantere, S. Kersten, and A. Hoischen, "Long-read sequencing emerging in medical genetics," *Frontiers in genetics*, vol. 10, p. 426, 2019.

[8] J. E. Gorzynski, S. D. Goenka, K. Shafin, T. D. Jensen, D. G. Fisk, M. E. Grove, E. Spiteri, T. Pesout, J. Monlong, G. Baid, *et al.*, "Ultrarapid nanopore genome sequencing in a critical care setting," *The New England journal of medicine*, 2022.

[9] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, and Q. Gouil, "Opportunities and challenges in long-read sequencing data analysis," *Genome biology*, vol. 21, no. 1, pp. 1–16, 2020.

[10] D. Kilburn, J. Burke, R. Fedak, H. Olsen, M. Jain, K. Miga, S. Mayes, and K. Liu, "High Data Throughput and Low Cost Ultra Long Nanopore Sequencing."

[11] H. Sadasivan, D. Stiffler, A. Tirumala, J. Israeli, and S. Narayanasamy, "Accelerated dynamic time warping on gpu for selective nanopore sequencing," *bioRxiv*, pp. 2023–03, 2023.

[12] N. V. Patel, "Why the CDC Botched Its Coronavirus Testing." MIT Technology Review.

[13] M. Park, J. Won, B. Y. Choi, and C. J. Lee, "Optimization of primer sets and detection protocols for sars-cov-2 of coronavirus disease 2019 (covid-19) using pcr and real-time pcr," *Experimental & molecular medicine*, vol. 52, no. 6, pp. 963–977, 2020.

[14] J. Quick and N. Loman, "ARTIC V3 Update Notes."

[15] A. J. McMichael, "Environmental and social influences on emerging infectious diseases: past, present and future," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 359, no. 1447, pp. 1049–1058, 2004.

[16] L. Zhang, X. Cui, K. Schmitt, R. Hubert, W. Navidi, and N. Arnheim, "Whole genome amplification from a single cell: implications for genetic analysis," *Proceedings of the National Academy of Sciences*, vol. 89, no. 13, pp. 5847–5851, 1992.

[17] A. L. Greninger, S. N. Naccache, S. Federman, G. Yu, P. Mbala, V. Bres, D. Stryke, J. Bouquet, S. Somasekar, J. M. Linnen, *et al.*, "Rapid metagenomic identification of viral pathogens in clinical samples by real-time nanopore sequencing analysis," *Genome medicine*, vol. 7, no. 1, p. 99, 2015.

[18] H. S. Edwards, R. Krishnakumar, A. Sinha, S. W. Bird, K. D. Patel, and M. S. Bartsch, "Real-time selective sequencing with rubric: Read until with basecall and reference-informed criteria," *Scientific Reports*, vol. 9, no. 1, pp. 1–11, 2019.

[19] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw, and S. Narayanasamy, "Genax: a genome sequencing accelerator," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 69–82, IEEE, 2018.

[20] T. J. Ham, D. Bruns-Smith, B. Sweeney, Y. Lee, S. H. Seo, U. G. Song, Y. H. Oh, K. Asanovic, J. W. Lee, and L. W. Wills, "Genesis: a hardware acceleration framework for genomic data analysis," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 254–267, IEEE, 2020.

[21] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 199–213, 2018.

[22] D. Fujiki, S. Wu, N. Ozog, K. Goliya, D. Blaauw, S. Narayanasamy, and R. Das, "Seedex: A genome sequencing accelerator for optimal alignments in subminimal space," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 937–950, IEEE, 2020.

[23] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Gencache: Leveraging in-cache operators for efficient sequence alignment," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 334–346, 2019.

[24] S. K. Khatamifard, Z. Chowdhury, N. Pande, M. Razaviyayn, C. Kim, and U. R. Karpuzcu, "A non-volatile near-memory read mapping accelerator," *arXiv preprint arXiv:1709.02381*, 2017.

[25] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand, *et al.*, "Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 951–966, IEEE, 2020.

[26] L. Wu, D. Bruns-Smith, F. A. Nothaft, Q. Huang, S. Karandikar, J. Le, A. Lin, H. Mao, B. Sweeney, K. Asanović, *et al.*, "Fpga accelerated indel realignment in the cloud," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 277–290, IEEE, 2019.

[27] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, vol. 10, pp. 359–370, Seattle, WA, USA:, 1994.

[28] P. Senin, "Dynamic time warping algorithm review," *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, vol. 855, no. 1-23, p. 40, 2008.

[29] G. Mahmoudabadi and R. Phillips, "A comprehensive and quantitative exploration of thousands of viral genomes," *Elife*, vol. 7, p. e31955, 2018.

[30] M. Loose, S. Malla, and M. Stout, "Real-time selective sequencing using nanopore technology," *Nature methods*, vol. 13, no. 9, p. 751, 2016.

[31] NVIDIA, "Jetson agx xavier developer kit."

[32] S. Wei, Z. R. Weiss, and Z. Williams, "Rapid multiplex small dna sequencing on the minion nanopore sequencing platform," *G3: Genes, Genomes, Genetics*, vol. 8, no. 5, pp. 1649–1657, 2018.

[33] Abbott, "Navica App and BinaxNOW COVID-19 Ag Test Card," 2021. Abbott Point of Care Testing.

[34] M. A. GOUILH, R. CASSIER, E. MAILLE, C. Schanen, L.-M. ROCQUE, and A. VABRET, "An easy, reliable and rapid sars-cov2 rt-lamp based test for point-of-care and diagnostic lab," *medRxiv*, 2020.

[35] NEB, "SARS-CoV-2 Rapid Colorimetric LAMP Assay Kit," 2021. New England Biolabs.

[36] LGC, "2019-nCoV CDC-qualified Probe and Primer Kits for SARS-CoV-2," 2021. LGC Biosearch Technologies.

[37] ONT, "cDNA PCR Sequencing Kit," 2021. Oxford Nanopore Technologies.

[38] P. James, D. Stoddart, E. D. Harrington, J. Beaulaurier, L. Ly, S. Reid, D. J. Turner, and S. Juul, "Lampore: rapid, accurate and highly scalable molecular screening for sars-cov-2 infection, based on nanopore sequencing," *medRxiv*, 2020.

[39] ONT, "Direct RNA Sequencing Kit," 2021. Oxford Nanopore Technologies.

[40] ONT, "Direct cDNA Sequencing Kit," 2021. Oxford Nanopore Technologies.

[41] M. Nagura-Ikeda, K. Imai, S. Tabata, K. Miyoshi, N. Murahara, T. Mizuno, M. Horiuchi, K. Kato, Y. Imoto, M. Iwata, *et al.*, "Clinical evaluation of self-collected saliva by rt-qpcr, direct rt-qpcr, rt-lamp, and a rapid antigen test to diagnose covid-19," *Journal of Clinical Microbiology*, 2020.

[42] J. R. Tyson, P. James, D. Stoddart, N. Sparks, A. Wickenhagen, G. Hall, J. H. Choi, H. Lapointe, K. Kamelian, A. D. Smith, *et al.*, "Improvements to the artic multiplex pcr method for sars-cov-2 genome sequencing using nanopore," *bioRxiv*.

[43] G. Moreno and D. O'Connor, "Sequence-Independent, Single-Primer Amplification of RNA viruses V.3," 2020. University of Wisconsin-Madison.

[44] ONT, "Metagenomic analysis of SARS-CoV-2 respiratory samples via Sequence-Independent Single Primer Amplification (SISPA) and nanopore sequencing," 2020. Oxford Nanopore Technologies.

[45] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of neural network basecalling tools for oxford nanopore sequencing," *Genome biology*, vol. 20, no. 1, p. 129, 2019.

[46] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[47] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome research*, vol. 27, no. 5, pp. 737–746, 2017.

[48] C. Wright, "Medaka," 2020. Medaka - Medaka 1.2.0 documentation.

[49] C. Brown, "Technology update." Nanopore Community Meeting, 2019.

[50] R. Han, Y. Li, X. Gao, and S. Wang, "An accurate and rapid continuous wavelet dynamic time warping algorithm for end-to-end mapping in ultra-long nanopore sequencing," *Bioinformatics*, vol. 34, no. 17, pp. i722–i731, 2018.

[51] R. Ronan, "Read until adaptive sampling." Oxford Nanopore Technologies.

[52] A. Payne, N. Holmes, T. Clarke, R. Munro, B. J. Debebe, and M. Loose, "Readfish enables targeted nanopore sequencing of gigabase-sized genomes," *Nature Biotechnology*, pp. 1–9, 2020.

[53] H. S. Edwards, R. Krishnakumar, A. Sinha, S. W. Bird, K. D. Patel, and M. S. Bartsch, "Real-time selective sequencing with rubric: Read until with basecall and reference-informed criteria," *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019.

[54] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with uncalled," *BioRxiv*, 2020.

[55] M. Stoiber, J. Quick, R. Egan, J. Eun Lee, S. Celniker, R. K. Neely, N. Loman, L. A. Pennacchio, and J. Brown, "De novo identification of dna modifications enabled by genome-guided nanopore signal processing," *bioRxiv*, 2017.

[56] N. J. Loman, J. Quick, and J. T. Simpson, "A complete bacterial genome assembled de novo using only nanopore sequencing data," *Nature methods*, vol. 12, no. 8, pp. 733–735, 2015.

[57] O. N. Technologies, "kmer_models," *GitHub repository*, 2017.

[58] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: a survey and empirical demonstration," vol. 7, pp. 349–371, Springer, 2003.

[59] P. Zhou, X.-L. Yang, X.-G. Wang, B. Hu, L. Zhang, W. Zhang, H.-R. Si, Y. Zhu, B. Li, C.-L. Huang, *et al.*, "A pneumonia outbreak associated with a new coronavirus of probable bat origin," *nature*, vol. 579, no. 7798, pp. 270–273, 2020.

[60] J. Hadfield, C. Megill, S. M. Bell, J. Huddleston, B. Potter, C. Callender, P. Sagulenko, T. Bedford, and R. A. Neher, "Nextstrain: real-time tracking of pathogen evolution," *Bioinformatics*, vol. 34, no. 23, pp. 4121–4123, 2018.

[61] Y. Shu and J. McCauley, "Gisaid: Global initiative on sharing all influenza data–from vision to reality," *Eurosurveillance*, vol. 22, no. 13, p. 30494, 2017.

[62] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with gpus and fpgas," in *2010 IEEE International Conference on Data Mining*, pp. 1001–1006, IEEE, 2010.

[63] R. E. Workman, A. D. Tang, P. S. Tang, M. Jain, J. R. Tyson, P. C. Zuzarte, T. Gilpatrick, R. Razaghi, J. Quick, N. Sadowski, *et al.*, "Nanopore native rna sequencing of a human poly (a) transcriptome," *BioRxiv*, p. 459529, 2018.

[64] O. N. Technologies, "Ont open datasets: Gm24385 dataset release," 2020.

[65] CADDE, "Brazil-uk centre for arbovirus discovery, diagnosis, genomics and epidemiology," 2020.

[66] ONT, "Rapid Library Preparation Kit (SQK-RAD004)," 2021. Oxford Nanopore Technologies.

[67] ONT, "ont-fast5-api," 2020. FAST5 API: a simple interface to HDF5 files of the Oxford Nanopore .fast5 file format.

[68] ONT, "ont-pyguppy-client-lib," 2020. PyGuppy: Python bindings for the GuppyClient library.

[69] ONT, "MinION DNA Sequencer," 2021. Oxford Nanopore Technologies.

[70] ONT, "Read until api," 2021. Oxford Nanopore Technologies.

[71] V. Sundaresan, S. Nichani, N. Ranganathan, and R. Sankar, "A vlsi hardware accelerator for dynamic time warping," in *11th IAPR International Conference on Pattern Recognition. Vol. IV. Conference D: Architectures for Vision and Pattern Recognition,*, vol. 1, pp. 27–30, IEEE Computer Society, 1992.

[72] M. Amicone, V. Borges, M. J. Alves, J. Isidro, L. Zé-Zé, S. Duarte, L. Vieira, R. Guiomar, J. P. Gomes, and I. Gordo, "Mutation rate of sars-cov-2 and emergence of mutators during experimental evolution," *Evolution, medicine, and public health*, vol. 10, no. 1, pp. 142–155, 2022.

[73] B. J. Willett, J. Grove, O. A. MacLean, C. Wilkie, G. De Lorenzo, W. Furnon, D. Cantoni, S. Scott, N. Logan, S. Ashraf, *et al.*, "Sars-cov-2 omicron is an immune escape variant with an altered cell entry pathway," *Nature microbiology*, vol. 7, no. 8, pp. 1161–1179, 2022.

[74] M. Dramé, M. T. Teguo, E. Proye, F. Hequet, M. Hentzien, L. Kanagaratnam, and L. Godaert, "Should rt-pcr be considered a gold standard in the diagnosis of covid-19?," *Journal of medical virology*, vol. 92, no. 11, p. 2312, 2020.

[75] T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K.-Y. Chen, D. Blaauw, R. Das, and S. Narayanasamy, "Squigglefilter: An accelerator for portable virus detection," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 535–549, 2021.

[76] H. Sadasivan, M. Maric, E. Dawson, V. Iyer, J. Israeli, and S. Narayanasamy, "Accelerating minimap2 for accurate long read alignment on gpus," *Journal of Biotechnology and Biomedicine*, vol. 6, no. 1, pp. 13–23, 2023.

[77] A. Payne, N. Holmes, T. Clarke, R. Munro, B. J. Debebe, and M. Loose, "Readfish enables targeted nanopore sequencing of gigabase-sized genomes," *Nature biotechnology*, vol. 39, no. 4, pp. 442–450, 2021.

[78] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with uncalled," *BioRxiv*, 2020.

[79] H. Zhang, H. Li, C. Jain, H. Cheng, K. F. Au, H. Li, and S. Aluru, "Real-time mapping of nanopore raw signals," *Bioinformatics*, vol. 37, no. Supplement_1, pp. i477–i483, 2021.

[80] H. Sadasivan, J. Wadden, K. Goliya, P. Ranjan, R. P. Dickson, D. Blaauw, R. Das, and S. Narayanasamy, "Rapid real-time squiggle classification for read until using rawmap," *Archives of Clinical and Biomedical Research*, vol. 7, no. 1, pp. 45–47, 2023.

[81] P. J. Shih, H. Saadat, S. Parameswaran, and H. Gamaarachchi, "Efficient real-time selective genome sequencing on resource-constrained devices," *arXiv preprint arXiv:2211.07340*, 2022.

[82] Y. Bao, J. Wadden, J. R. Erb-Downward, P. Ranjan, W. Zhou, T. L. McDonald, R. E. Mills, A. P. Boyle, R. P. Dickson, D. Blaauw, *et al.*, "Squigglenet: real-time, direct classification of nanopore signals," *Genome biology*, vol. 22, no. 1, pp. 1–16, 2021.

[83] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with gpus and fpgas," in *2010 IEEE International Conference on Data Mining*, pp. 1001–1006, IEEE, 2010.

[84] Y. Kraeva and M. Zymbler, "Scalable algorithm for subsequence similarity search in very large time series data on cluster of phi knl," in *International Conference on Data Analytics and Management in Data Intensive Domains*, pp. 149–164, Springer, 2018.

[85] A. Ziehn, M. Charfuelan, H. Hemsen, and V. Markl, "Time series similarity search for streaming data in distributed systems.," in *EDBT/ICDT Workshops*, Lisbon, 2019.

[86] X. Xu, F. Lin, A. Wang, X. Yao, Q. Lu, W. Xu, Y. Shi, and Y. Hu, "Accelerating dynamic time warping with memristor-based customized fabrics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 4, pp. 729–741, 2017.

[87] C. Hundt, B. Schmidt, and E. Schömer, "Cuda-accelerated alignment of subsequences in streamed time series data," in *2014 43rd International Conference on Parallel Processing*, pp. 10–19, IEEE, 2014.

[88] L. Xiao, Y. Zheng, W. Tang, G. Yao, and L. Ruan, "Parallelizing dynamic time warping algorithm using prefix computations on gpu," in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 294–299, IEEE, 2013.

[89] T. Li, X. Li, Y. Li, R. Song, and X. Wang, "Crescent: A gpu-based targeted nanopore sequence selector," in *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2357–2365, IEEE, 2022.

[90] B. Schmidt and C. Hundt, "cudtw++: Ultra-fast dynamic time warping on cuda-enabled gpus," in *European Conference on Parallel Processing*, pp. 597–612, Springer, 2020.

[91] O. N. Technologies, "6-mer model for r9.4 chemistry," 2016. Oxford Nanopore Technologies.

[92] "Nsight systems." *NVIDIA* `https://developer.nvidia.com/nsight-systems`.

[93] "Nsight compute." *NVIDIA* `https://docs.nvidia.com/nsight-compute/NsightCompute/index.htmla`.

[94] M. Marani, G. G. Katul, W. K. Pan, and A. J. Parolari, "Intensity and frequency of extreme novel epidemics," *Proceedings of the National Academy of Sciences*, vol. 118, no. 35, p. e2105482118, 2021.

[95] K. Kalantar-Zadeh, S. A. Ward, K. Kalantar-Zadeh, and E. M. El-Omar, "Considering the effects of microbiome and diet on sars-cov-2 infection: nanotechnology roles," *ACS nano*, vol. 14, no. 5, pp. 5179–5182, 2020.

[96] S. Villapol, "Gastrointestinal symptoms associated with covid-19: impact on the gut microbiome," *Translational Research*, 2020.

[97] T. Zuo, F. Zhang, G. C. Lui, Y. K. Yeoh, A. Y. Li, H. Zhan, Y. Wan, A. C. Chung, C. P. Cheung, N. Chen, *et al.*, "Alterations in gut microbiota of patients with covid-19 during time of hospitalization," *Gastroenterology*, vol. 159, no. 3, pp. 944–955, 2020.

[98] D. V. Ward, S. Bhattarai, M. Rojas-Correa, A. Purkayastha, D. Holler, M. Da Qu, W. G. Mitchell, J. Yang, S. Fountain, A. Zeamer, *et al.*, "The intestinal and oral microbiomes are robust predictors of covid-19 severity the main predictor of covid-19-related fatality," *medRxiv*, 2021.

[99] I. Cho and M. J. Blaser, "The human microbiome: at the interface of health and disease," *Nature Reviews Genetics*, vol. 13, no. 4, pp. 260–270, 2012.

[100] C. G. Buffie, V. Bucci, R. R. Stein, P. T. McKenney, L. Ling, A. Gobourne, D. No, H. Liu, M. Kinnebrew, A. Viale, *et al.*, "Precision microbiome reconstitution restores bile acid mediated resistance to clostridium difficile," *Nature*, vol. 517, no. 7533, pp. 205–208, 2015.

[101] J. Fajnzylber, J. Regan, K. Coxen, H. Corry, C. Wong, A. Rosenthal, D. Worrall, F. Giguel, A. Piechocka-Trocha, C. Atyeo, *et al.*, "Sars-cov-2 viral load is associated with increased disease severity and mortality," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[102] M. Wang, A. Fu, B. Hu, Y. Tong, R. Liu, Z. Liu, J. Gu, B. Xiang, J. Liu, W. Jiang, *et al.*, "Nanopore targeted sequencing for the accurate and comprehensive detection of sars-cov-2 and other respiratory viruses," *Small*, vol. 16, no. 32, p. 2002169, 2020.

[103] L. Yang, G. Haidar, H. Zia, R. Nettles, S. Qin, X. Wang, F. Shah, S. F. Rapport, T. Charalampous, B. Methé, *et al.*, "Metagenomic identification of severe pneumonia pathogens in mechanically-ventilated patients: a feasibility and clinical validity study," *Respiratory research*, vol. 20, no. 1, p. 265, 2019.

[104] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of neural network basecalling tools for oxford nanopore sequencing," *Genome biology*, vol. 20, no. 1, p. 129, 2019.

[105] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[106] D. Kim, L. Song, F. P. Breitwieser, and S. L. Salzberg, "Centrifuge: rapid and sensitive classification of metagenomic sequences," *Genome research*, vol. 26, no. 12, pp. 1721–1729, 2016.

[107] D. Jacot, G. Greub, K. Jaton, and O. Opota, "Viral load of sars-cov-2 across patients and compared to other respiratory viruses," *Microbes and infection*, 2020.

[108] A. N. Cohen and B. Kessel, "False positives in reverse transcription pcr testing for sars-cov-2," *medRxiv*, 2020.

[109] E. Surkova, V. Nikolayevskyy, and F. Drobniewski, "False-positive covid-19 results: hidden problems and costs," *The Lancet Respiratory Medicine*, vol. 8, no. 12, pp. 1167–1168, 2020.

[110] D. E. Wood and S. L. Salzberg, "Kraken: ultrafast metagenomic sequence classification using exact alignments," *Genome biology*, vol. 15, no. 3, pp. 1–12, 2014.

[111] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with kraken 2," *Genome biology*, vol. 20, no. 1, p. 257, 2019.

[112] J. Lu, F. P. Breitwieser, P. Thielen, and S. L. Salzberg, "Bracken: estimating species abundance in metagenomics data," *PeerJ Computer Science*, vol. 3, p. e104, 2017.

[113] D. H. Huson, A. F. Auch, J. Qi, and S. C. Schuster, "Megan analysis of metagenomic data," *Genome research*, vol. 17, no. 3, pp. 377–386, 2007.

[114] L. Schaeffer, H. Pimentel, N. Bray, P. Melsted, and L. Pachter, "Pseudoalignment for metagenomic read assignment," *Bioinformatics*, vol. 33, no. 14, pp. 2082–2088, 2017.

[115] D. T. Truong, E. A. Franzosa, T. L. Tickle, M. Scholz, G. Weingart, E. Pasolli, A. Tett, C. Huttenhower, and N. Segata, "Metaphlan2 for enhanced metagenomic taxonomic profiling," *Nature methods*, vol. 12, no. 10, pp. 902–903, 2015.

[116] A. T. Dilthey, C. Jain, S. Koren, and A. M. Phillippy, "Strain-level metagenomic assignment and compositional estimation for long reads with metamaps," *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

[117] M. Loose, S. Malla, and M. Stout, "Real-time selective sequencing using nanopore technology," *Nature methods*, vol. 13, no. 9, p. 751, 2016.

[118] Y. Pan, D. Zhang, P. Yang, L. L. Poon, and Q. Wang, "Viral load of sars-cov-2 in clinical samples," *The Lancet infectious diseases*, vol. 20, no. 4, pp. 411–412, 2020.

[119] B. Huang, A. Jennison, D. Whiley, J. McMahon, G. Hewitson, R. Graham, A. De Jong, and D. Warrilow, "Illumina sequencing of clinical samples for virus detection in a public health laboratory," *Scientific reports*, vol. 9, no. 1, pp. 1–8, 2019.

[120] M. Sommariva, V. Le Noci, F. Bianchi, S. Camelliti, A. Balsari, E. Tagliabue, and L. Sfondrini, "The lung microbiota: role in maintaining pulmonary immune homeostasis and its implications in cancer development and therapy," *Cellular and Molecular Life Sciences*, pp. 1–11, 2020.

[121] B. Hjorth, "Eeg analysis based on time domain properties," *Electroencephalography and clinical neurophysiology*, vol. 29, no. 3, pp. 306–310, 1970.

[122] K. Kupková, "Alignment-free visualization of metagenomic data by genomic signal processing," *Complexity*, vol. 100, p. 3, 2014.

[123] K. Kupkova, K. Sedlar, and I. Provaznik, "Reference-free identification of phage dna using signal processing on nanopore data," in *2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE)*, pp. 101–105, IEEE, 2017.

[124] ONT, "Metagenomic analysis of SARS-CoV-2 respiratory samples via Sequence-Independent Single Primer Amplification (SISPA) and nanopore sequencing," 2020.

[125] N. R. Faria, "First cases of coronavirus disease (COVID-19) in Brazil, South America," 2020.

[126] D. Kim, J.-Y. Lee, J.-S. Yang, J. W. Kim, V. N. Kim, and H. Chang, "The architecture of sars-cov-2 transcriptome," *Cell*, 2020.

[127] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[128] "ONT products." *Oxford Nanopore Technologies* https://nanoporetech.com/products, Mar 2022.

[129] "Ion proton™ sequencer specifications: Thermo fisher scientific - US." *Ion Proton* https://www.thermofisher.com/us/en/home/life-science/sequencing/next-generation-sequencing/ion-torrent-next-generation-sequencing-workflow/ion-torrent-next-generation-sequencing-run-sequence/ion-proton-system-for-next-generation-sequencing/ion-proton-system-specifications.html.

[130] "DNBSEQ-T7: High-speed,high flexibility and ultra-high throughput sequencer-mgi-leading life science innovation." *MGI* https://en.mgi-tech.com/products/instruments_info/5/s.

[131] K. Shafin, T. Pesout, P.-C. Chang, M. Nattestad, A. Kolesnikov, S. Goel, G. Baid, M. Kolmogorov, J. M. Eizenga, K. H. Miga, *et al.*, "Haplotype-aware variant calling with pepper-margin-deepvariant enables high accuracy in nanopore long-reads," *Nature methods*, vol. 18, no. 11, pp. 1322–1332, 2021.

[132] "GPU applications: High performance computing." *NVIDIA* https://www.nvidia.com/en-us/gpu-accelerated-applications/.

[133] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[134] "Sequence with confidence." *PacBio* https://www.pacb.com/, Mar 2022.

[135] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *Arxiv*, 2013. Preprint at https://doi.org/10.48550/arXiv.1303.3997.

[136] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[137] S. M. Kiełbasa, R. Wan, K. Sato, P. Horton, and M. C. Frith, "Adaptive seeds tame genomic sequence comparison," *Genome research*, vol. 21, no. 3, pp. 487–493, 2011.

[138] I. Sović, M. Šikić, A. Wilm, S. N. Fenlon, S. Chen, and N. Nagarajan, "Fast and sensitive mapping of nanopore sequencing reads with graphmap," *Nature communications*, vol. 7, no. 1, pp. 1–11, 2016.

[139] J. Ren and M. J. Chaisson, "lra: A long read aligner for sequences and contigs," *PLOS Computational Biology*, vol. 17, no. 6, p. e1009078, 2021.

[140] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.

[141] R. E. Bellman, R. E. Kalaba, and T. Teichmann, "Dynamic programing," *Physics Today*, vol. 19, no. 4, p. 98, 1966.

[142] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[143] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC bioinformatics*, vol. 19, no. 1, pp. 33–47, 2018.

[144] S. Kalikar, C. Jain, M. Vasimuddin, and S. Misra, "Accelerating minimap2 for long-read sequencing applications on modern cpus," *Nature Computational Science*, vol. 2, no. 2, pp. 78–83, 2022.

[145] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "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)*, pp. 127–135, IEEE, 2019.

[146] H. Alhakami, H. Mirebrahim, and S. Lonardi, "A comparative evaluation of genome assembly reconciliation tools," *Genome biology*, vol. 18, no. 1, pp. 1–14, 2017.

[147] A. Zeni, G. Guidi, M. Ellis, N. Ding, M. D. Santambrogio, S. Hofmeyr, A. Buluç, L. Oliker, and K. Yelick, "Logan: High-performance gpu-based x-drop long-read alignment," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 462–471, IEEE, 2020.

[148] Z. Feng, S. Qiu, L. Wang, and Q. Luo, "Accelerating long read alignment on three processors," in *Proceedings of the 48th International Conference on Parallel Processing*, pp. 1–10, 2019.

[149] H. Li, "seqtk toolkit for processing sequences in fasta/q formats," *GitHub*, vol. 767, p. 69, 2012.

[150] Torvalds, "Linux/tools/perf at master · torvalds/linux." *Github* `https://github.com/torvalds/linux/tree/master/tools/perf`.

[151] H. P. R. Consortium, "HG002 data freeze (v1.0) ONT 60x coverage reads." *AWS* `https://s3-us-west-2.amazonaws.com/human-pangenomics/index.html?prefix=NHGRI_UCSC_panel/HG002/hpp_HG002_NA24385_son_v1/nanopore/downsampled/standard_unsheared/`.

[152] H. P. R. Consortium, "HG002 data freeze (v1.0) PacBio HiFi 15Kb reads." *AWS* `https://s3-us-west-2.amazonaws.com/human-pangenomics/NHGRI_UCSC_panel/HG002/hpp_HG002_NA24385_son_v1/PacBio_HiFi/15kb/m64012_190920_173625.Q20.fastq`.

[153] H. P. R. Consortium, "HG002 data freeze (v1.0) PacBio HiFi 20Kb reads." *AWS* https://s3-us-west-2.amazonaws.com/human-pangenomics/NHGRI_UCSC_panel/HG002/hpp_HG002_NA24385_son_v1/PacBio_HiFi/20kb/m64011_190830_220126.Q20.fastq.

[154] K. Shafin, T. Pesout, R. Lorig-Roach, M. Haukness, H. E. Olsen, C. Bosworth, J. Armstrong, K. Tigyi, N. Maurer, S. Koren, *et al.*, "Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes," *Nature biotechnology*, vol. 38, no. 9, pp. 1044–1053, 2020.