GPU acceleration of DNA alignment algorithms

Tong Dong Qiu

CE-MS-2018-XX

Delft University of Technology

Abstract

TU Delft Delft University of Technology

Quantum & Computer Engineering

TITLE

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

AUTHOR
born in PLACE, COUNTRY

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

TITLE

by AUTHOR

Abstract

Laboratory	:	Computer Engineering
Codenumber	:	CE-MS-2018-number
Committee Members	:	
Advisor:		, CE, TU Delft
Chairperson:		, CE, TU Delft
Member:		, CE, TU Delft
Member:		, CE, TU Delft

Dedicated to my family and friends

Contents

List of Figures

List of Tables

Acknowledgements

The author acknowledges the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu

AUTHOR
Delft, The Netherlands
November 27, 2018

1 Introduction

The processes in biology are extremely complex, especially those in multicellular organisms. Pretty much all of these processes are governed by DNA, the building blocks of life. Our kind has spent ages trying to unravel its secrets. Literally, because we found out that DNA consists of a double helix, which must be unwound to be used.

In 2012, we discovered that the human genome consists of about three billion basepairs, wrapped up in 23 chromosomepairs [38]. And that it shares 60% with the genome of a banana [39]. However, DNA still has many unexposed secrets. For example, some diseases are partially caused by genes, like breast cancer or sickle cell disease [40].

To gain more knowledge about the influence of DNA and genes in general, they must be identified. This is where DNA sequencing comes in. With a blood sample, or some saliva, the DNA of an individual can be mapped. Combining the DNA of many people can teach us what the effect is of some genes. However, DNA sequencing machines are not perfect, they only provide small parts of the DNA, and they contain errors too. These small parts must be assembled into a whole genome. One phase of this assembly is the alignment part, where overlaps between those small parts are found. Many algorithms have been developed to perform this alignment, for different lengths and error rates. Daligner and Darwin are two algorithms that find overlaps for so-called 'long reads'. These are produced by the third-generation of DNA sequencing machines, and can be tens of thousands of bases long, but have errors rates of 15-30%.

Performing the normal, exact way of alignment using Smith-Waterman is not feasible, especially for these long reads. Daligner and Darwin are heuristics, that reduce the amount of computation needed, without compromising the output much. Still, they do a lot of computation.

Since DNA alignment is inherently parallel (comparing reads A and B does not depend on the result of comparing reads C and D, or even A and C), speedup can be gained by parallelising the workload. GPUs are built for graphics applications, but modern, programmable GPUs allow for GPGPU (General-Purpose computing on GPUs). This means that any algorithm can be run on a GPU.

In this work, Daligner and Darwin are implemented to run on a GPU, in particular a Tesla K40.

Outline Chapter 2 Background contains details about the working of DNA in biology, as well as DNA sequencing and assembly techniques. Chapter 3 Concept further explains the algorithms of Daligner and Darwin. Chapter 4 Specification contains measured statistics that help understand the working of the algorithm, as well as all the implementations/optimizations done on both algorithms. The evaluation of these implementations is presented in Chapter 5 Results. In Chapter 6 Conclusion, conclusions and recommendations for future work are presented.

2 Background

2.1 Biology

The most basic unit for a living organism is a *cell*, often called the 'building blocks of life'. Each living thing consists of one or more cells. Cells can have different shapes, sizes and functions, but they all have some things in common. Each cells consists of cytoplasm surrounded by a membrane. This membrane is the boundary between the exterior and the inside of the cell and acts as a filter to allow certain molecules to enter or exit the cell. A cell can grow by taking nutrients from the environment, and using them to create other molecules, or letting them interact with existing molecules [41]. Cells can also reproduce when there are enough components in the original call to produce a duplicate cell. Many processes in the cell are driven by *proteins*. Proteins are long chains of *amino acids*. These amino acids are joined by peptide bonds to form polypeptides. When these polypeptides are folded by the forces between the atoms, it is called a protein. The exact form of the protein is crucial to its function [42]. Examples of protein functions are breaking down molecules like other proteins, fat or carbohydrates, fighting off foreign particles like viruses or bacteria, and assisting in a chemical reaction, in which case the protein is called an *enzyme* [43]. Enzymes bind to the reagents of the reaction to lower the activation energy and increase the speed of the reaction, but they are not consumed during the reaction and can be reused. Enzymes are usually highly specific, meaning that they will only catalyse certain reactions [44][45].

It is clear that proteins play a huge role in sustaining a cell. The information needed to create proteins is stored in *DNA* or deoxyribonucleic acid. DNA can be *translated* to create new proteins (this process is called *genetic expression*), or *replicated* to allow for reproduction. A section of DNA that codes a certain protein is called a *gene*. A DNA molecule consists of two long chains of nucleotides (also called strands), which are intertwined with each other in a double helix. Each nucleotide is composed of a nucleobase, a sugar called deoxyribose and a phosphate group [46]. The four different nucleobases are: adenine (A), thymine (T), cytosine (C) and guanine (G). The bases of one strand bond with bases in the opposite strand, but adenine can only bond with thymine, and cytosine only with guanine. Figure 2.1 shows a schematic overview of the components of DNA.

Each strand has two ends: the *5'-end* (pronounced five-prime) and the *3'-end*. The 5'-positions can bind a phosphate group, the 3'-positions can bind a sugar. This leads to a *backbone* of alternating sugar and phosphate groups. The two strands are *antiparallel*, this means one strands 5'-end is matched to the others 3'-end. The orientation of the strand is significant: replication and translation can only be done in 5'→3' direction. Figure 2.2 shows how DNA is replicated.

During DNA replication, the two strands are split, so the nucleotides can be read.

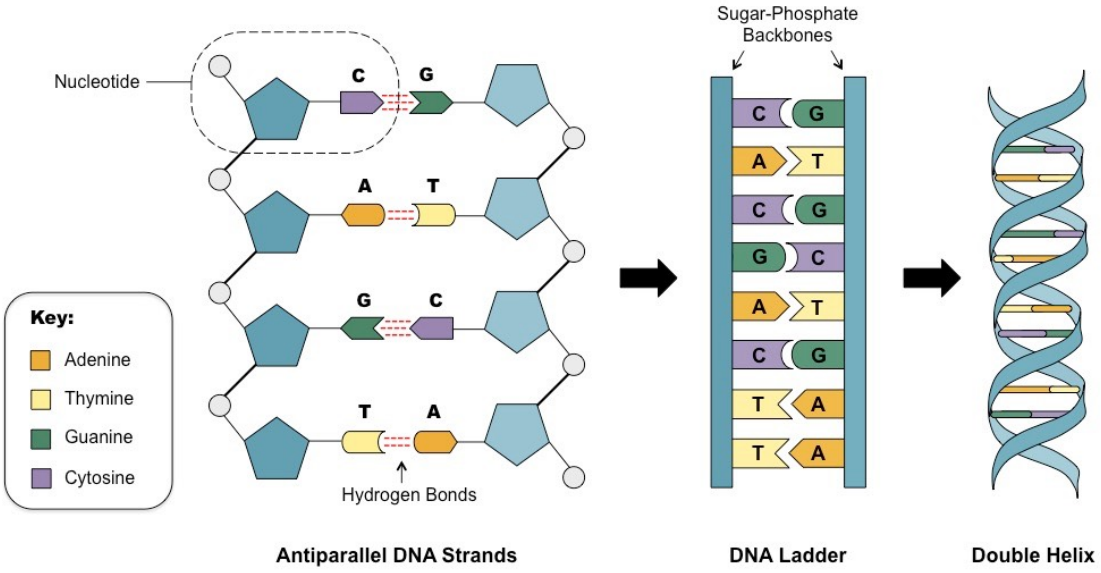

Figure 2.1: Structure of DNA, source: [1]

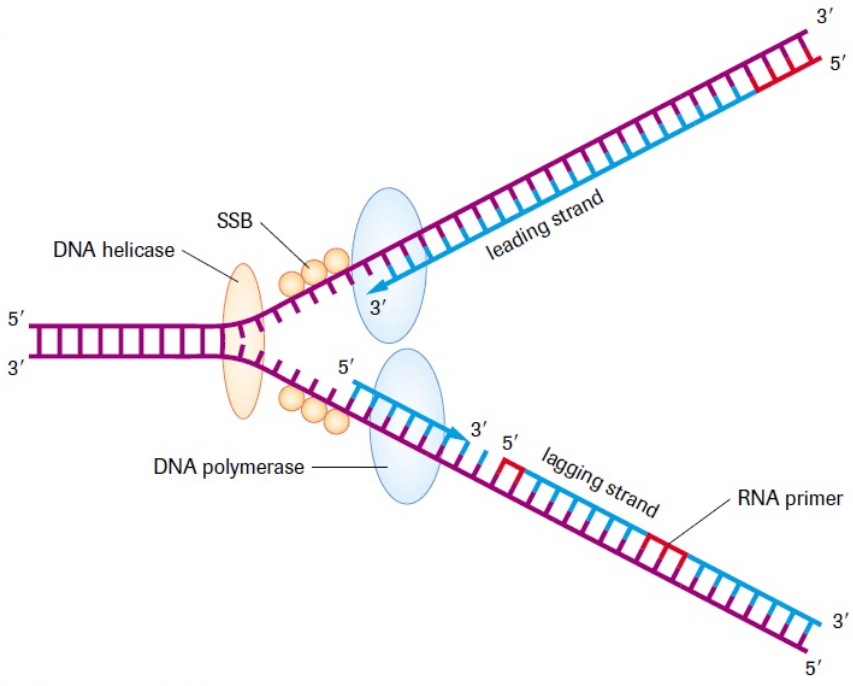

Figure 2.2: DNA replication, note that the lagging strand is copied in multiple pieces, source [2]

Single-stranded binding (SSB) proteins prevent the DNA strands from bonding together (annealing) again. RNA primers are added to the strands, one for the leading strand,

and several for the lagging strand. DNA polymerase starts building from those primers. The leading strand is built in one go, but since there are several primers in the lagging strand, it consists of multiple pieces, including the primers. These pieces are called the Okazaki fragments [47]. RNase removes the primers, and ligase stitches the DNA fragments together.

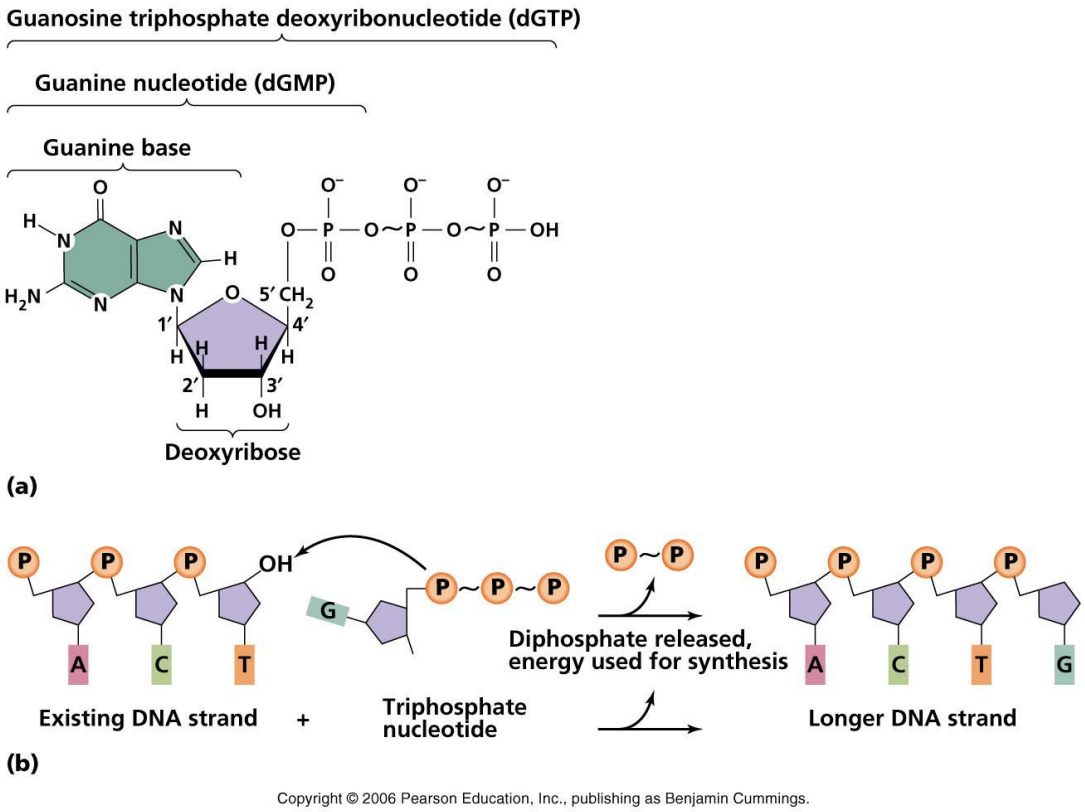

Figure 2.3: Molecules involved in DNA replication, source [3]

Creating new proteins from DNA is done via messenger RNA or *mRNA*, which looks like DNA, but has only one strand. It has the same structure as DNA, but thymine (T) is replaced by uracil (U), and the sugar is *ribose* instead of deoxyribose. When reading the DNA, three bases are considered together, and are called a *codon*. Each codon describes an amino acid, but since there are only 20 different amino acids, and $4^3 = 64$ different codons, multiple codons map to the same amino acid, and some codons have special functions, like start (ATG) and stop (TAA, TAG, TGA) [48].

The process starts with the enzyme *RNA polymerase* binding to the correct place on the DNA with the help of a *promotor*, which indicates the start of a gene. The mRNA strand will look like the *coding (or sense) strand* of the DNA, the other DNA strand is called the *template (or antisense) strand*. The RNA polymerase create a so-called *transcription bubble*, which is a short section of separated DNA, where the RNA polymerase can access the bases of the coding strand. The polymerase now adds RNA nucleotides to the template strand, creating an mRNA strand. This strand is separated

from the DNA template strand, allowing the DNA strands to join back together. The mRNA strand now looks like the DNA coding strand, with the exception that thymine is replaced by uracil.

The mRNA strand is now ready to be translated into a protein, this is done by a *ribosome*, a very complex molecule consisting of several ribosomal RNA molecules and dozens of proteins [49]. The ribosome binds to the start of the mRNA strand. The codons of the mRNA strand are interpreted by *transfer RNA* (tRNA). The first tRNA molecule binds to the start codon (AUG), and always carries the amino acid methionine. The ribosome then keeps finding tRNA molecules that fit the mRNA codons, and adds their amino acids to the growing chain, creating the protein. The tRNA molecules are not consumed during the process and can, after picking up a new amino acid, be reused. There are no tRNA molecules that can fit a stop codon (UAA, UAG or UGA). So instead of a tRNA molecule, a protein from a group called 'release factors' binds to the mRNA, and the ribosome releases the protein [50]. The used mRNA molecules degrade after they are used [51].

The protein is now a chain of amino acids, linked together by peptide bonds. This is called its *primary structure* [52]. Most amino acids are nonpolar, this means their electrical charge is zero and balanced. Others have positive or negative charges, or have no charge, but do have a dipole, these are called polar. Polar amino acids can form hydrogen bonds [53], charged amino acids can form ionic bonds [54]. Hydrophobic amino acids can form weaker van der Waals bonds. Most of these bonds noncovalent, which means they do not share electrons [54]. Cysteine is the only amino acid that can form covalent bonds [52].

Bonds between parts of the protein can cause folding patterns to appear. The most occurring types are *alpha helices* and *beta sheets*, shown in Figure 2.4. These patterns form the *secondary structure* of a protein. When these patterns interact with eachother, for example due to van der Waals bonds, the *tertiary structure* forms. Finally, a protein consisting of multiple chains, or subunits, is called the *quaternary structure* [52].

A partially folded protein can interact with different molecules in the cell, causing improper folding. Misfolded proteins can also clot together with other molecules, causing large aggregates. To prevent this, *chaperone proteins* surround a protein during the folding process. Many chaperone proteins are *heat shock* proteins, because heat makes proteins less stable. The cells produces more heat shock proteins when it is exposed to heat [55].

2.2 DNA sequencing

DNA sequencing is the process of finding out which nucleotides make up a DNA molecule, or all DNA molecules of an organism. Separate DNA molecules are called 'chromosomes'. Humans have 46 chromosomes, containing about 3.5 billion basepairs [56] and about 20000 genes [57].

Knowing the exact DNA sequence is useful for a number of applications: analyzing bacteria/virusses, analyzing genetic diseases, forensic analysis, ethnicity and ancestry analysis.

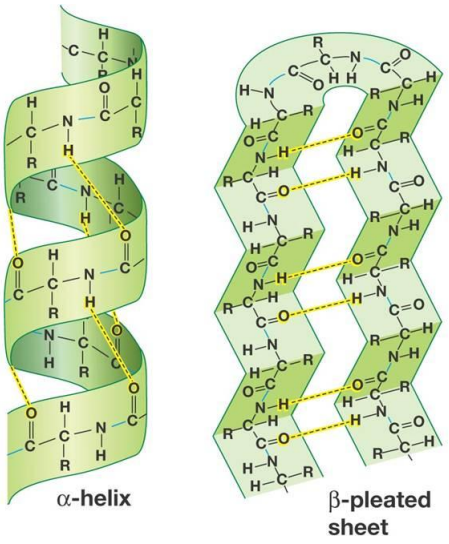

Figure 2.4: Alpha helix and beta sheet visualized, source: [4]

Another application is genetic modification of bacteria or plants. If their DNA can be altered, they could produce useful substances at a very low marginal cost.

DNA sequencing has evolved over the years, becoming orders of magnitude cheaper and faster.

2.2.1 Sanger sequencing

Sanger sequencing, also called chain-termination sequencing, is the first major DNA sequencing technique. It is published in 1977 [58], and later improvements led to the development of commercial DNA sequencing machines in 1991 [8]. It is also the technique used by the Human Genome Project [5], a project to sequence the whole genome of a human. The project start around 1990, and finished in 2003 [38]. Since Sanger sequencing has a maximum chain length of about 900 basepairs [5], the resulting pieces of DNA had to be assembled after sequencing.

To sequence a DNA sample, it is mixed in a tube with: a *primer*, DNA polymerase, normal DNA nucleotides (dATP, dTTP, dGTP and dCTP), and dye-labeled, chain-terminating dideoxynucleotides. The primer is a piece of DNA that can fit onto a specific spot of a DNA strand. Primers usually contains 18 to 24 bases [59]. The chain-terminating nucleotides are ddATP, ddTTP, ddGTP and ddCTP. They consist of a normal nucleotide, but with an oxygen atom removed, as shown in Figure 2.5. This small difference means that no new nucleotide can be attached to it, thus ending the chain. The mixture is heated to split the two DNA strands. Once the primer is bound, the DNA polymerase can start adding nucleotides to form a chain. At one point, a dideoxynucleotide is added, and the chain cannot grow anymore. The distribution of chain lengths is determined by the ratio of normal and dideoxynucleotides [60].

After a certain period, most of the dideoxynucleotides have terminated a chain, and the next phase can begin. The chains are sorted by length by means of *cappilary elec-*

trophoresis (CE). During CE, the chains are run through a long glass capillary filled with a gel polymer. An electrical field is applied and the DNA fragments move through the capillary. The speed of the fragment is inversely proportional to its weight, so the shortest chains arrive earliest at the end. The resolution is high enough to separate chains that differ in length by one base. A laser excites the dye-labeled dideoxynucleotides, which emits a light at a characteristic wavelength. These lights are detected and interpreted to get the actual DNA sequence. Figure 2.6 shows the process of Sanger of sequencing.

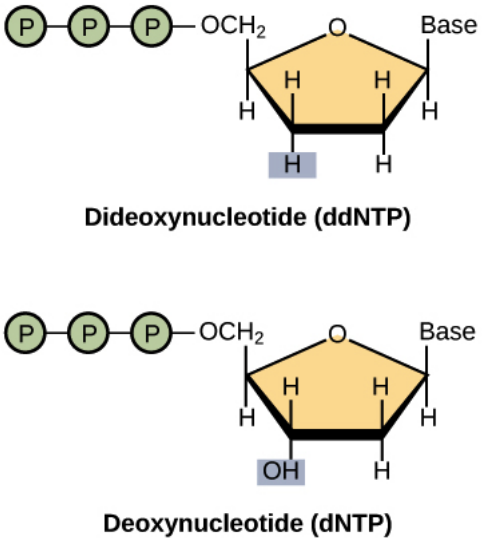

Figure 2.5: Normal nucleotides and dideoxynucleotides, source: [5]

2.2.2 Next Generation Sequencing

Next Generation Sequencing (NGS) involves a number of different techniques. They are different from the previous Sanger sequencing in that they are massively parallel, have high throughput at a much lower cost [61].

The first NGS method is *pyrosequencing*, developed in 2005 [8]. Like Sanger sequencing, it is a *sequence-by-synthesis* method, because it relies on DNA polymerase to recreate the DNA. The DNA strand is split, and fragmented to pieces, which are attached to microscopic beads. The strands are cloned using Polymerase Chain Reaction (PCR) [62][7], such that each bead has about 10 million identical copies of its fragment [63]. Each bead is place into a separate well, and each well is given a mixture that contains DNA polymerase, adenosine phosphosulfate (APS), ATP sulfurylase, luciferin, and luciferase. ATP sulfurylase is an enzyme that combines APS and pyrophosphate (PPi) into ATP, an energy carrier. For one cycle, one type of nucleotide is added to each of the wells. When a nucleotide is added to the chain by DNA polymerase, it releases PPi. This is converted to ATP by ATP sulfurylase, this energy is used by luciferase to oxidize luciferin and create light. The amount of light is proportional to the amount of nucleotides added, and since it is known which type of nucleotide is added, the bases

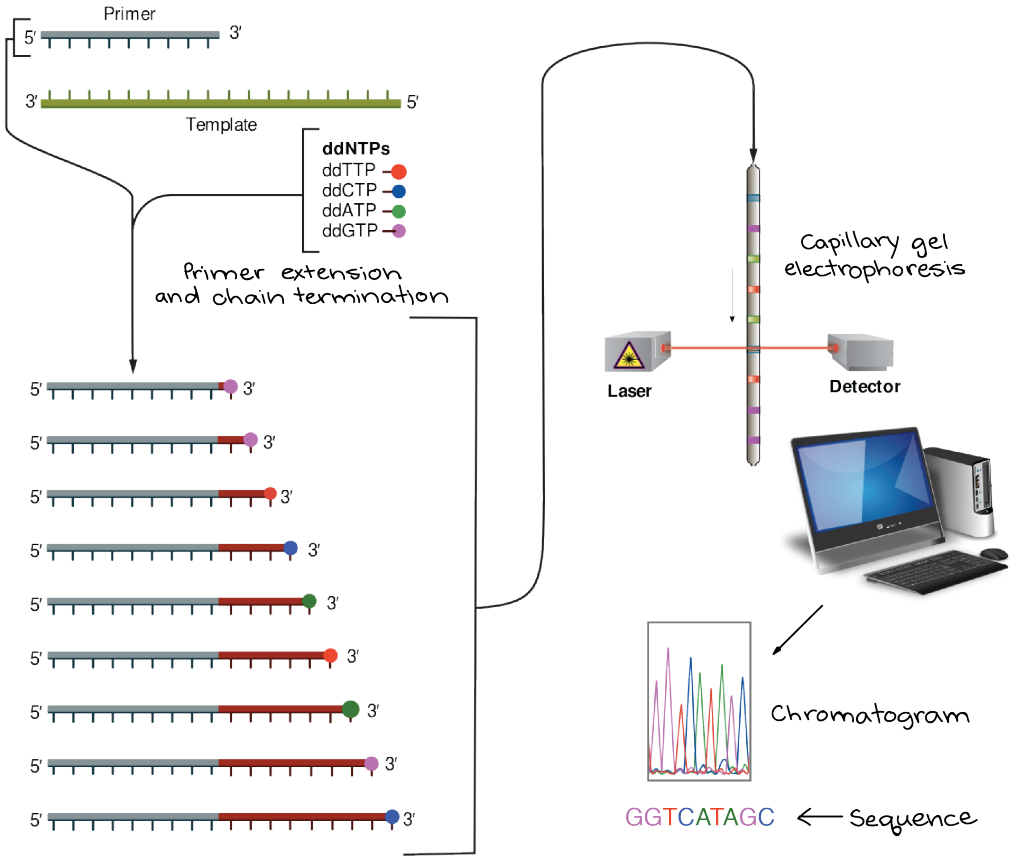

Figure 2.6: Workflow of Sanger sequencing, source: [5]

can be read. The wells are cleaned by the enzyme apyrase, which degrades ATP and the unused nucleotides. Now, the next type of nucleotide can be added. The whole sequencing process is shown in Figure 2.7.

The newest pyrosequencing machines can produce readlengths up to 700 basepairs.

The most popular NGS method is created by Illumina [64], released in 2007. It is similar to Sanger sequencing, but uses *reversible terminators*. Another major difference is that instead of emulsion PCR, bridge PCR is used, which is more efficient [7]. Bridge PCR is further explained in Figures 2.8 and 2.9. At the end of the PCR step, the reverse strands are washed away [65]. The reversible terminators are regular nucleotides, but fluorescent dye occupies the 3' end. First, the reversible terminators are added, and bound by the DNA polymerase. The unused nucleotides are washed away, and the dye of the bound nucleotides is released by an enzyme, allowing the bases to be read, and the chain to be extended during the next cycle.

Reads produced by this method are shorter (about 100 basepairs [61][66]).

Illumina also developed paired-end (PE) sequencing. Both ends of a DNA strand are sequenced, with an unsequenced gap in the middle, this creates more data per DNA strand, but also valuable information, since the length of the strand is known. These paired-end reads can be used during assembly, namely during the scaffolding (see Section

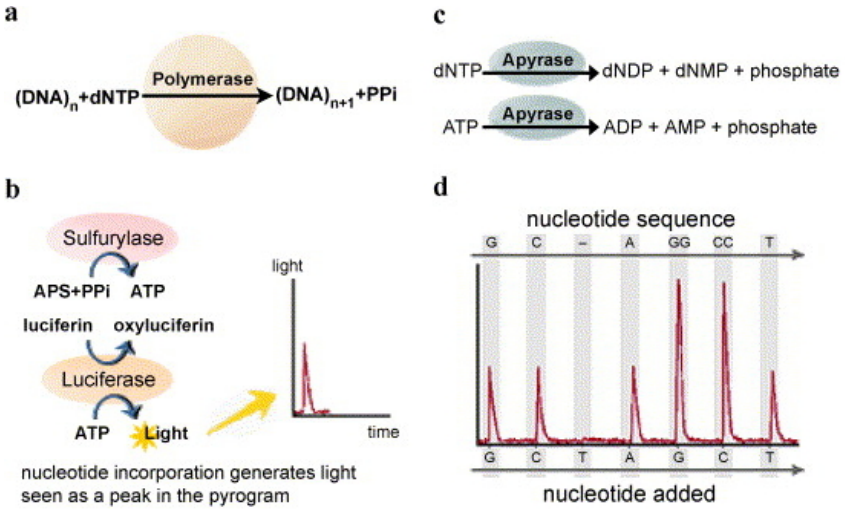

Figure 2.7: Workflow of pyrosequencing, adapted from: [6]

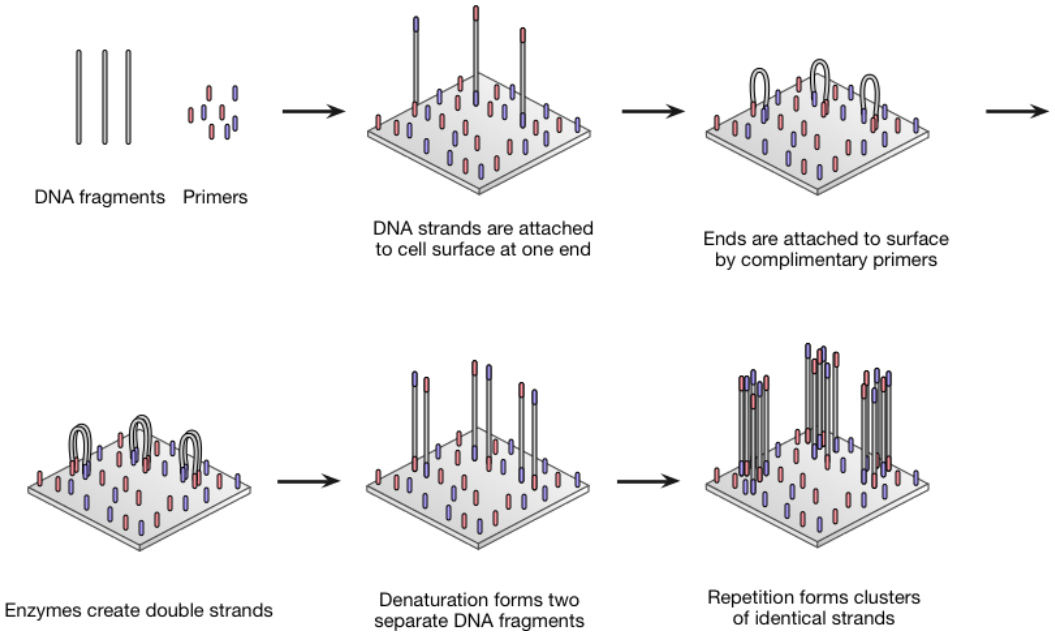

Figure 2.8: Process of bridge PCR, source: [7]

find paired-end lengths?

2.4). Figure 2.10 shows how they can be used to resolve repetitive regions.

The third major NGS method is Sequencing by Oligonucleotide Ligation and Detection (SOLiD), released by Applied Biosystems Instruments (ABI), which later became Life Technologies [8]. SOLiD relies on ligation to chain nucleotides. DNA ligase is able to ligate double-stranded DNA [7], whereas DNA polymerase only adds one base to a growing chain, complementary to a template strand [67].

A single strand DNA piece, known as an adapter, is merged with the template strand

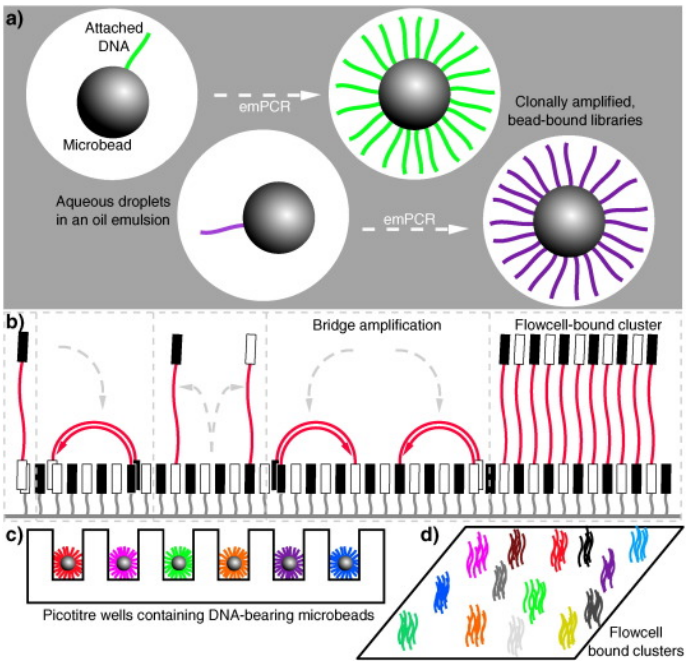

Figure 2.9: (a) emulsion PCR, (b) bridge PCR, (c) emulsion PCR results in beads with cloned DNA strands, each well can fit one bead, (d) bridge PCR results in clusters of cloned DNA strands, source: [8]

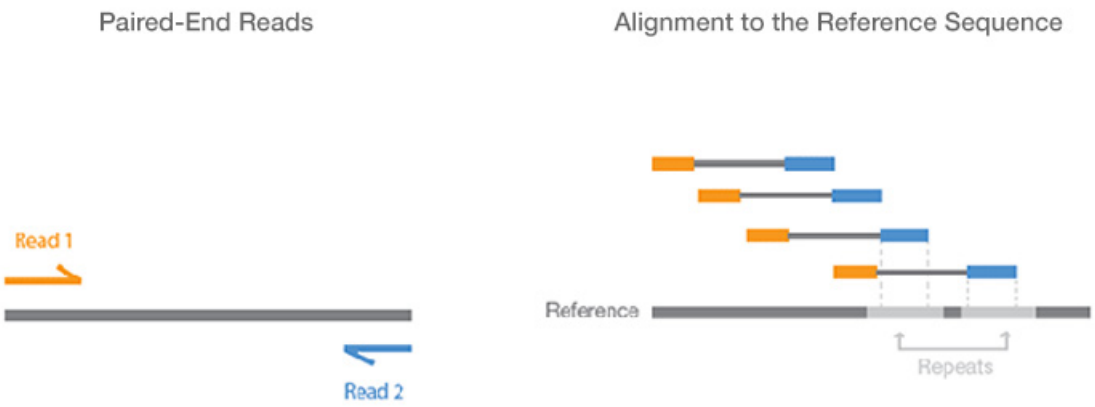

Figure 2.10: Paired-end sequenced reads can be used to resolve repetitive regions due to their length. Source: [9]

on one side, and attached to a bead on the other. A primer of length N is connected to the adapter. Now the DNA ligase can add oligonucleotides, these are strands of eight nucleotides, with fluorescent dye at the 5' end, shown in Figure 2.11. The first two bases have to be complementary to the bases in the template strand. The degenerate and

universal bases are not used. Silver ions are used to cleave the link between bases five and six, allowing the dye to be observed. The cleave leaves a normal 5' end, which can be used to bind new oligonucleotides. After the first round of sequencing, the new chain is removed, and a second round is started, but with a primer of length N-1. Multiple rounds with primers with different lengths ensure that each bases is measured twice. After the rounds, the colors can be decoded to get the actual bases, like in Figure 2.12. The total workflow in shown in Figure 2.13.

Because each base is measured twice, the accuracy can go up to 99.999% with six primers, which is the highest of the NGS methods [7]. A downside is that the read lengths are usually short, 50 to 75 basepairs [61].

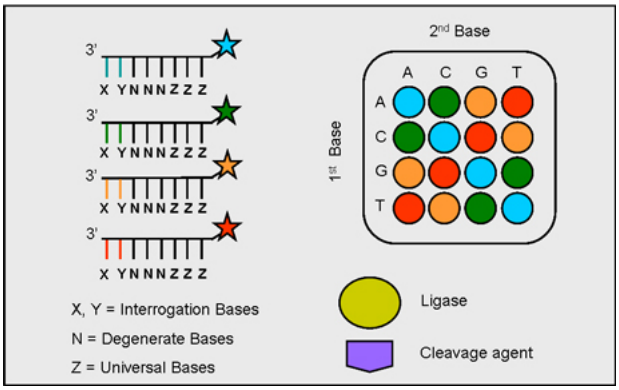

Figure 2.11: Color coding of SOLiD sequencing results, source: [10]

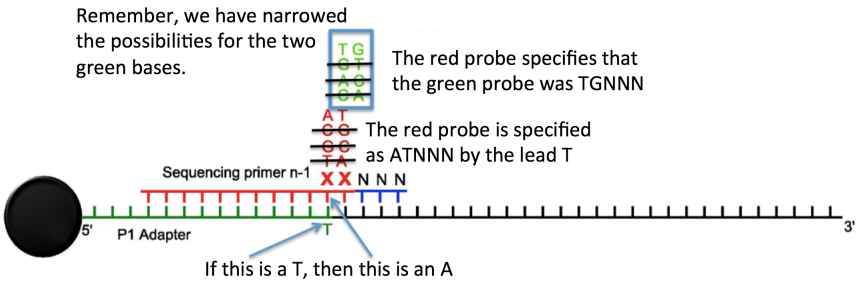

Figure 2.12: Color decoding of SOLiD sequencing results, source: [11]

There are two other popular NGS techniques: DNA Nanoball Sequencing (DNBS) and Ion Torrent. DNBS relies on Rolling Circle Amplification to clone the DNA strands that need to be sequenced. The cloned strands are sequenced like in SOLiD sequencing [61]. Ion Torrent is very similar to pyrosequencing, but uses hydrogen ions instead of fluorescent dye to detect bases [7]. The number of ions is proportional to the number bases that was attached, however, at large repeating sections the accuracy decreases. Ion Torrent produces reads of up to 400 basepairs, but the error rate is relatively high at 1.5%.

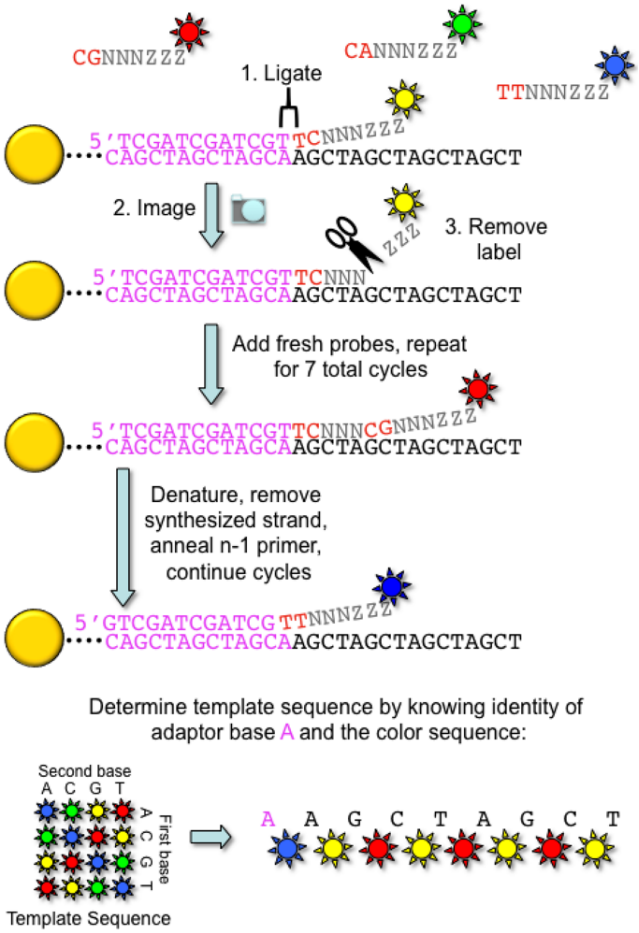

Figure 2.13: Workflow of SOLiD sequencing, source: [12]

2.2.3 Third Generation Sequencing

There is no clear division between Next Generation Sequencing and Third Generation Sequencing (TGS). The technological improvements follow eachother quickly, and do not usually fit into welldefined timescales [68]. However, an often used definition is that TGS approaches are able to sequence a single DNA molecule, thus eliminating the need for amplification [8][7].

The first TGS method is created by Helicos Biosciences, a company that went backrupt in 2012 [8]. DNA template strings are attached to a surface. Fluorescent nucleotides called Virtual Terminators are added, the dye is then cleaved off and detected. The samples are washed, and a new type of nucleotide can be added. The sequencing time for this method is relatively high, because only one type of nucleotide can be read at once, like with most NGS methods. The read lengths are also quite short: about 32 nucleotides [68].

The other TGS are roughly placed into three categories: (a) sequencing by synthesis, (b) sequencing by nanopore, (c) direct imaging [68].

One of the most used TGS platforms is the Single Molecule RealTime (SMRT) sequencing platform from Pacific Biosciences. It is based on sequencing by synthesis. A very important part of SMRT sequencing is the Zero-Mode Waveguide (ZMW) technology. A ZMW is a very tiny hole, with a diameter of 60-100 nm, and a height of 100 nm [69]. The metal that the holes are created in is placed on a glass substrate, giving the holes a glass bottom. At the bottom of each well is a DNA polymerase molecule attached. The wells are illuminated from beneath, by a laser with a wavelength of approximately 600 nm [8]. The diameter of the well is important, because it is smaller than the wavelength of the used light, the light penetrating the bottom of the well decays exponentionally, such that only the bottom of the well is illuminated. The well is filled with fluorescent nucleotides, and as one of these is held by the polymerase, a light pulse is produced and recorded. There are other nucleotides floating in the well, but the nucleotides held by the polymerase is illuminated approximately three orders of magnitude longer, this creates a high signal-to-noise ratio [68]. As the polymerase does not have to wait for the nucleotides to be washed away, the process is realtime, which greatly increases the sequencing speed. Another advantage is that it produces some of the longest read lengths in the industry, with an average of 10000 basepairs, and longest reads of 60000 basepairs [15], Figure 2.16 shows a distribution of the readlengths. The fluorescent dyes are attached to the phosphate chain, which is cleaved off naturally when incorporated into the DNA strand, as shown in Figures 2.3, 2.14 and 2.15.

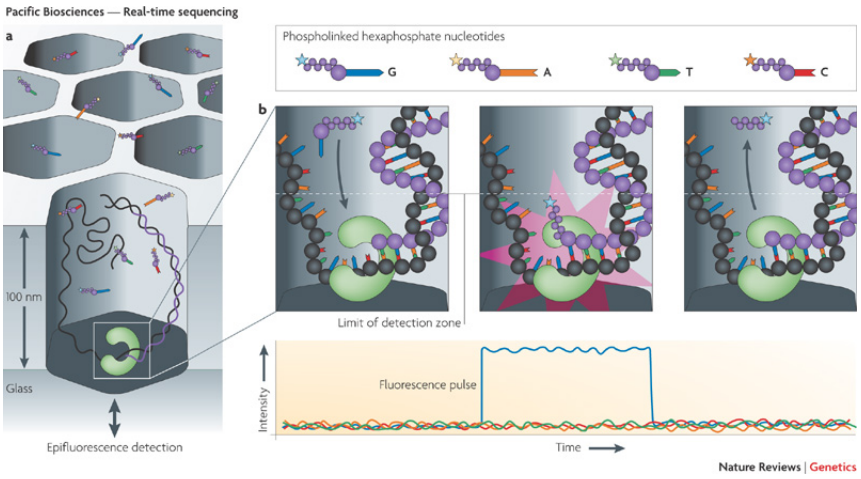

Figure 2.14: (a) A zero-mode waveguide with a DNA polymerase molecule at the bottom (b) a dyed base is incorporated into the strand, and the phosphate groups with tye dye are cleaved off, source: [13]

A new application of SMRT sequencing includes adding single-stranded loops to a double-stranded template, creating a circular template called a SMRTbell [16], shown in Figure 2.17. The double-stranded DNA region and single-stranded loops are called insert and hairpins respectively. The insert can be of any length, where lengths from 40 bp to 25000 bp have been tested [16]. The hairpins contain a sequence to which a primer can bind to. When the whole SMRTbell is sequenced, the DNA polymerase arrives at the 5' end of the primer. The primer is separated from the SMRTbell, and the sequencing

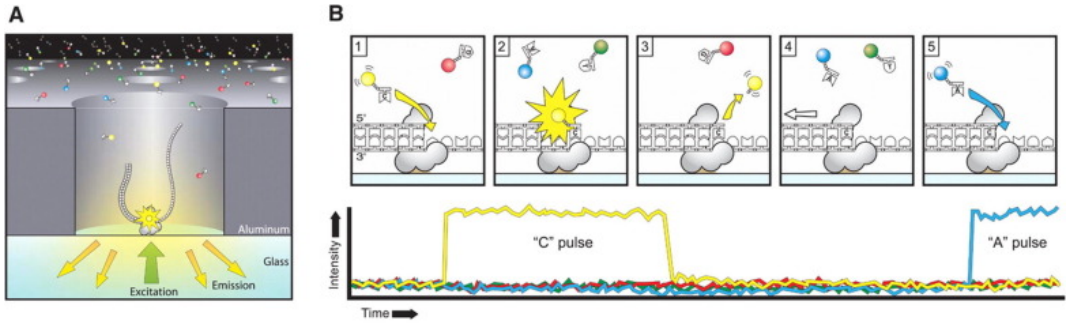

Figure 2.15: A: a ZMW with DNA polymerase and a DNA template strand, B: a C nucleotide is added to the chain, the next one is an A, source: [14]

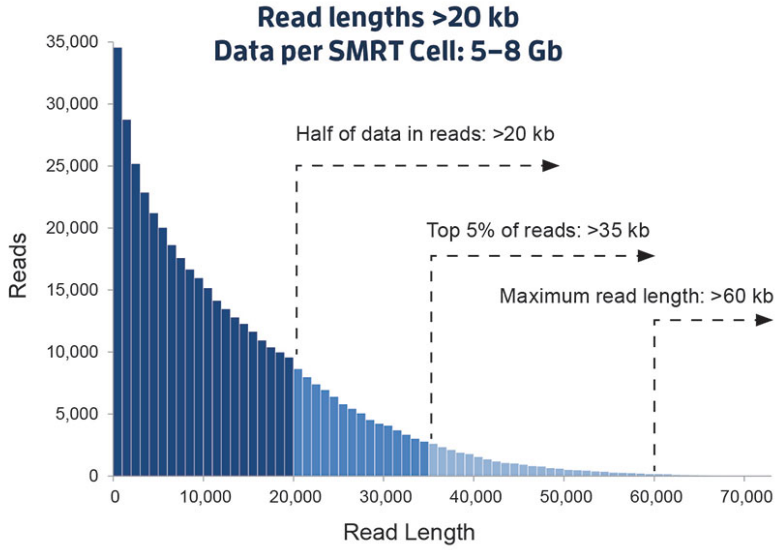

Figure 2.16: A distribution of read lengths, from a human library, source: [15] Courtesy of Pacific Biosciences of California, Inc., Menlo Park, CA, USA

can continue. This means that length of the sequence can become much larger than the length of the insert, depending on the lifetime of the polymerase [14]. After sequencing, the reads can be split up by removing the hairpin regions, to obtain subreads. Since each base can be sequenced multiple times, a consensus sequence can be constructed from all subreads, with a quality score for each base. This process is shown in Figure 2.18.

PacBio's SMRT sequencing also has some disadvantages: throughput and error rate. Only 23-46% of the ZMWs produce successful reads [8]. This can be caused by failing to attach a polymerase molecule to the bottom of a ZMW, or by loading more than one DNA strand in a ZMW. The throughput of the older RS II system is about 2 billion bases per day, but the newer Sequel System can produce 20 billion bases per day [70]. Still, this is much lower than the high throughput offered by Illumina HiSeq 2500, with HiSeq SBS v4 reagent kits, it can produce about 167 billion bases per day in High Output Run

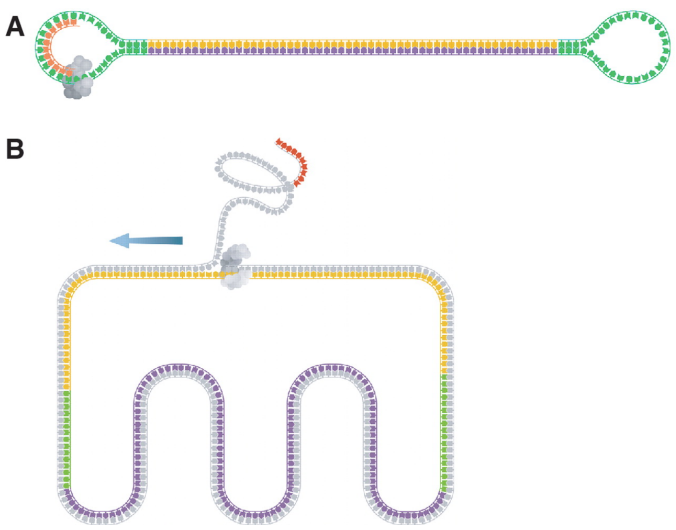

Figure 2.17: (a) A SMRTbell template has a double-stranded region (insert) and two hairpin loops on the side. The orange part is a primer. (b) As the regions split, a circular template is formed, which is sequenced from both template and coding strand. The separated part contains the orange primer, source: [16]

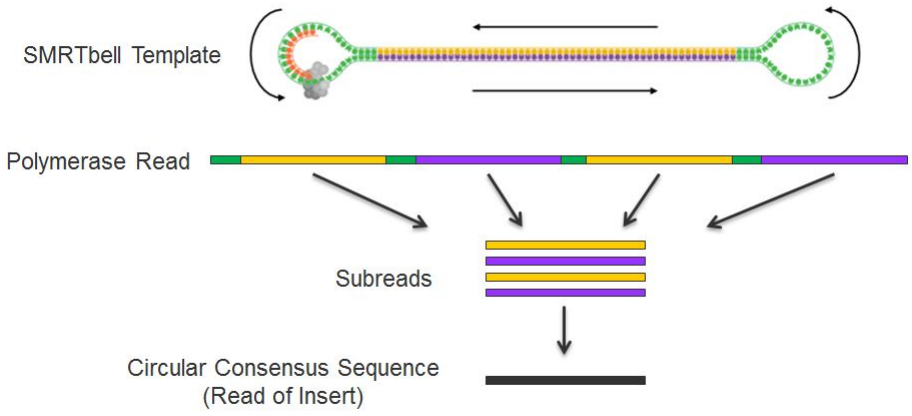

Figure 2.18: Since the SMRTbell is circular, the insert can be sequenced multiple times, the resulting read is split, and a consensus sequence is created, source: [17] Courtesy of Pacific Biosciences of California, Inc., Menlo Park, CA, USA

Mode [14].

A nanopore is a small opening through which a DNA molecule can move, one base at a time. While doing this, these bases can be detected by an electrical or optical signal. The nanopore are usually made of biological proteins, or synthetic structures like carbon nanotubes [71]. Oxford Nanopore Technologies (ONT) has build a system for DNA sequencing where three natural biological molecules form a nanopore. This system is called MinION and is a tiny portable DNA sequencer, shown in Figure 2.19. The nanopore is

put in a synthetic lipid bilayer, which consists of two sheets of lipid molecules forming a thin polar membrane. A voltage is applied across the bilayer, causing an ionic current through the nanopore. The double-stranded DNA is extended with a lead adaptor with a molecular motor, a hairpin adaptor, and a trailing adaptor. These adaptors enable the nanopore and DNA molecule to bond. The motor includes a processive enzyme, which denatures the DNA strands, and moves one strand through the nanopore at sufficient speed. The adaptors also concentrate the DNA around the nanopore, increasing the DNA capture rate by several thousand-fold [19]. As the bases move through the nanopore, they disrupt the ionic current flowing through the pore. Since the nucleotides have different shapes, the disruption they cause is also different. The exact change on the ionic current must be measured to determine the nucleotide.

Figures 2.20 and 2.21 show the working of the nanopore.

Figure 2.19: A MinION device connected to a USB 3.0 cable, source: [18]

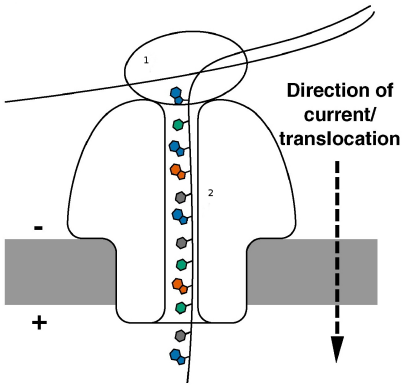

Figure 2.20: Double-stranded DNA gets denatured by an enzyme (1), one of the strands moves through the nanopore (2), adapted from: [8]

Oxford Nanopore boasts very long reads, over 200 kilobases, but a disadvantage of using this system is the error rate is very high: 12-38% [72][73].

The third category is direct imaging, usually with some sort of microscope.

A Transmission Electron Microscope (TEM) can be used to detect atoms attached to nucleotides. The atoms that normally make up nucleotides are too small and light to be detected, so heavy elements elements are added, like Mercury [20]. Figures 2.22 and

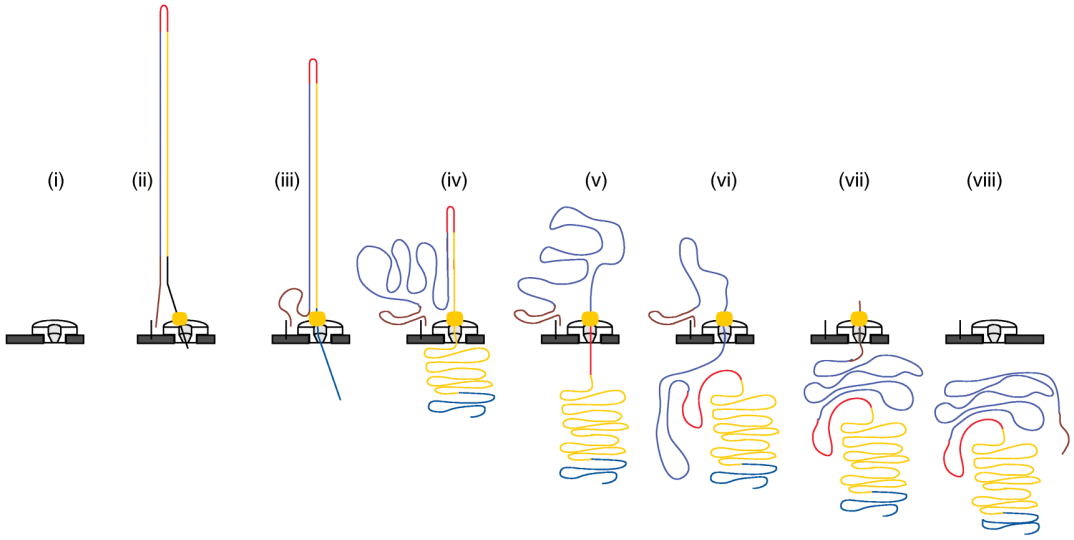

Figure 2.21: (i) nanopore is empty, (ii) dsDNA with lead adaptor (blue), molecular motor (orange) and hairpin adaptor (red), (iii) lead adaptor, (iv) template strand (gold), (v) hairpin adaptor, (vi) complement strand (dark blue), (vii) trailing adaptor (brown), (viii) nanopore is empty again, adapted from: [19]

2.23 show a DNA strand with labelled nucleotides, and a resulting image. To be able to read the bases, the DNA's natural state, a double helix, must be flattened and placed on a substrate. Figure 2.24 shows the different forms of DNA. Some ways to prepare the DNA molecules include combing [74][75], molecular threading [76] and transfer printing [77].

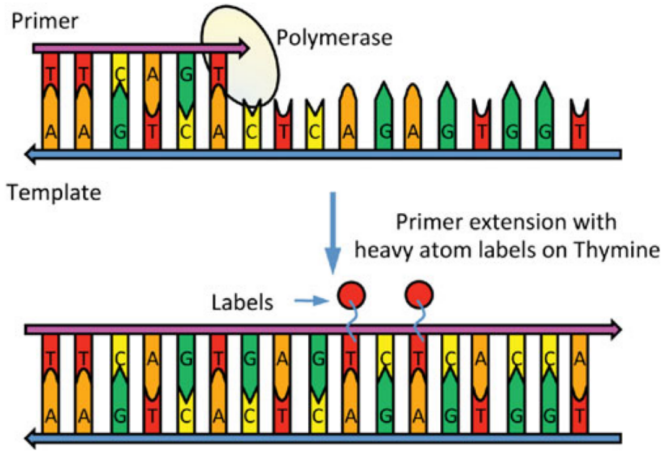

Figure 2.22: Single-strand DNA is primed and a complementary strand is synthesized. Thymine nucleotides have a heavy atom attached, source: [20]

ZS Genetics uses annular dark-field (ADF) imaging [78]. Dark-field imaging measures the photons/electrons that have come into contact with the object, leaving out the

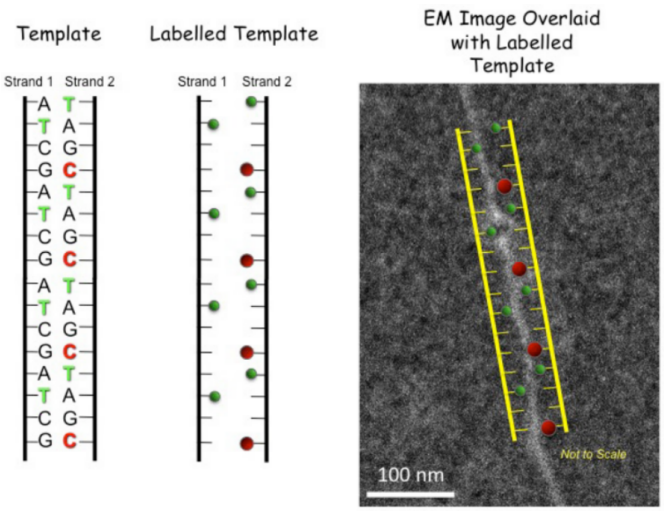

Figure 2.23: A labeled DNA molecule, ready to be read. Image on the right shows an image from an ADF STEM, source: [21]

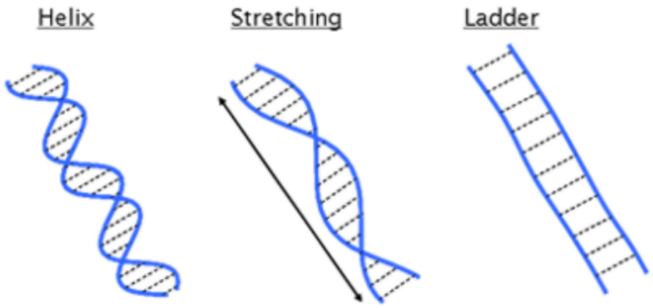

Figure 2.24: Different forms of double-stranded DNA, source: [21]

uninterrupted beam. Light-field imaging is the opposite: it measures the amount that did not get interrupted. Annular dark-field imaging uses an annular (ring) to capture the photons/electrons, as shown in Figure 2.25.

Another microscope capable of sequencing DNA is the Scanning Tunnelling Microscope (STM) [79]. The machine is invented in 1982 by Binnig and Rohrer, for which they won the 1986 Nobel Prize in Physics [80]. The conductive tip of the machine is located very close to the sample, which is deposited on a conductive surface. The tip is extremely thin, to allow for very precise spatial measurements. Electrons can move from the sample to the tip, by tunnelling [81], a voltage difference between the sample and the tip is applied to allow for this phenomenon. The tunnelled electrons form a current, which can be measured. This current is very sensitive to the distance between the tip and the sample, a change of 10^{-10} m can cause the current to change an order of magnitude [82]. The width of a DNA molecule is about $2 \cdot 10^{-9}$. The current can also depends on the applied voltage. This means the current can be used to measure the exact distance between the tip and the sample. At least, that is the theory. The reality

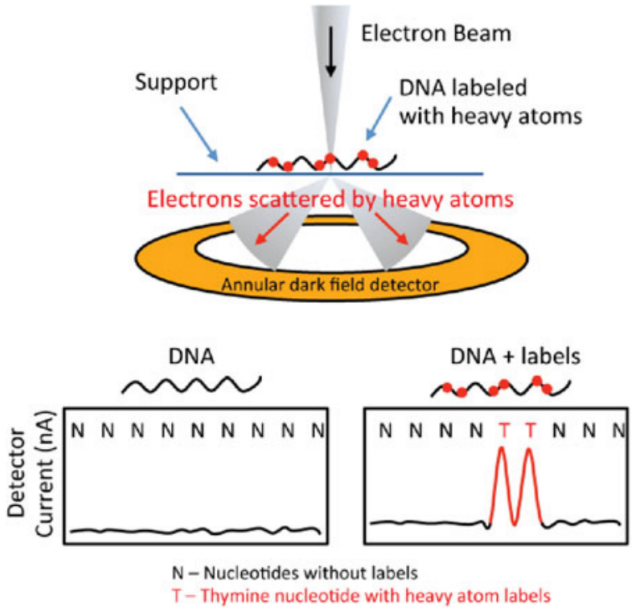

Figure 2.25: Top: electrons are scattered by heavy atoms and detected. Bottom: unlabeled DNA bases scatter fewer electrons and are harder to detect, source: [20]

is that DNA has a low conductivity when attached to a hard surface [83]. As the tip moves closer to the sample to try and get a current, it touches the DNA, deforming it [82]. Other challenges include difficulties in preparing clean samples and the fact that measurements can only be done at cryogenic temperatures [83]. These, and the relatively high costs of an STM, are probably the reasons why there is currently no practical case for STMs in DNA sequencing.

A different approach combines nanopores and tunnel current. Traditional nanopores, like from Oxford Nanopore Technologies, measure ionic current to determine the base that occupies the nanopore. A drawback of this method is throughput: a base has to block the pore long enough for the current to change and be measured. This is done by stopping, slowing down, or cleaving bases off. In either case, the read out of the current is in the order of tens or hundreds of ms [84]. The new approach measures the tunnelling current between two electrodes that make up a nanopore. Using tunnelling currents can potentially be orders of magnitude faster than using ionic currents, because electrical currents can easily be detected in ranges of over MHz and possibly tens of MHz for the current amplitudes (tens of pA to nA) [84]. By applying a trans-membrane voltage, the movement of the DNA strands can be controlled, and using tunnelling current gives a fast, accurate readout. Figure 2.26 shows the two different methods. In 2005, it was shown that each nucleotide has a unique electrical signature, which is independent of its nearest-neighbours [85]. In 2013, Quantum Biosystems was founded [86]. They have two generations of devices, and use nanopores with a diameter of 0.8 nm [87].

mention protein align-ment?

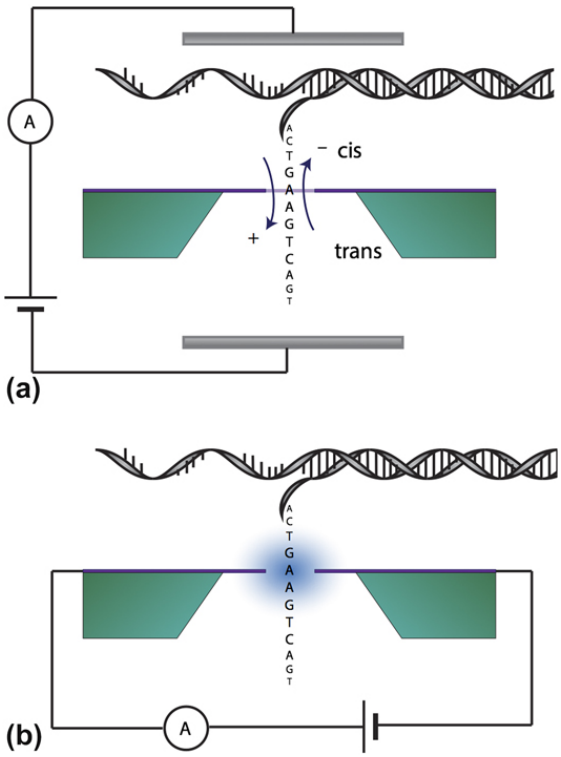

Figure 2.26: (a) a traditional nanopore measuring the ionic current in the direction of the ssDNA strand, (b) a tunnelling nanopore measuring the tunnelling current perpendicular to the ssDNA strand, source: [22]

2.3 DNA Alignment

DNA alignment involves determining the similarity between two DNA reads, a high similarity indicates the two reads originated from the same part of the genome. This information can be used to reconstruct the whole genome, during the assembly phase, which is discussed in Section 2.4 An example of an alignment is shown in Figure 2.27. Alignment algorithms are also useful in applications like text comparison [88].

```
tcctctgcctctgccatcat---caaccccaaagt
|||| ||| ||||| |||||   ||||||||||||
tcctgtgcatctgcaatcatgggcaaccccaaagt
```

Figure 2.27: An example alignment, vertical stripes indicate matching nucleotides, horizontal stripes indicate gaps in one of the reads, source: [23]

If the two reads originate from the same part of the genome, the mismatches can be explained by sequencing errors. But copying the genome during cell division can also cause errors [89]. Since DNA sequencing is usually performed on more than one cell, the variations in these cells are all sequenced. citation needed?

The mismatches between reads can be distinguished between substitutions, insertions and deletions. A substitution is shown between the fifth bases in the figure. An insertion is where a read has a base that does not occur in the other read, a deletion means the opposite. Note that usually, it cannot be determined if read A has an insertion, or that read B has a deletion, unless they can be compared to other reads. Due to the similarity between insertions and deletions, they are often referred to as 'indels'.

A match increases the score, a mismatch (substitution) or indel decreases the score. The alignment with the highest score is the most likely alignment. Exact or heuristic algorithms can be used to determine the best alignment between two reads. Exact algorithms are guaranteed to find the best alignment, heuristic algorithms reduce the search space, which (dramatically) improves runtime, but the results might be worse than exact algorithms.

A simple scoring scheme has a static score/penalty for an indel or mismatch, but biologically speaking, one larger gap is more likely to happen than two smaller gaps, since it is more likely to add arbitrary bases to an existing gap than to create such a gap. This is why an affine gap penalty in the form of $y = a * N + b$ is often used, where b is the gap open penalty, a the gap extend penalty, and N the number of bases in the gap - 1. Other variations in gap penalty include constant ($y = a$), double affine (), convex ($y = a + b \cdot ln(N)$).

citation needed?

double affine description?

Global alignment aligns the full length of both sequences. Every base will either have a match/mismatch with a base from the other read, or a gap. Local alignment will look for the two subsequences from the reads that have the highest score. This means that low scoring regions could be ignored. Local alignment is especially useful when the reads are very different in length. If one read represents a whole genome of 10000 bases, and the other represents a small sequenced part of 100 bases, global alignment will contain at least 9900 gaps, and at most 100 matches. Local alignment will ignore most of those gaps, and only match the small read to a small part of the larger read. Figure 2.28 shows another example of a case where local alignment is better.

Figure 2.28: Local alignment can find small, overlapping regions very well, source: [24]

should dynamic programming be a separate subsection? Could be an appendix as well

2.3.1 Dynamic programming

Dynamic programming is a programming technique that solves problems by dividing them recursively into smaller, simpler problems [90]. An important aspect is that the subproblems overlap, saving the results of one subproblem to use in another subproblem. This makes it different than a divide-and-conquer approach, where the problem is divided into non-overlapping subproblems [91]. There are two ways to solve these problems: top-

down, and bottom-up. Top-down is a direct result of the recursive formulation. It starts by asking the end result, if this is not available, follow the recursion, and check for the answer(s) of the subproblem(s). If these are available, use them, if not, calculate them and store them. This technique is called memoization. Bottom-up firsts solves the subproblems, and uses these results to build the solution for the original problem.

2.3.2 Global alignment

For global alignment, the Needleman-Wunsch (NW) algorithm [92] is widely used. It was published in 1970, and designed to compare sequences of amino acids, but as both are represented by letters, comparing nucleotides works in the same way. Needleman-Wunsch is an exact algorithm, producing the optimal global alignment.

cite/prove that NW is proven optimal?

In Needleman-Wunsch, the smallest subproblems are trivial alignments, such as aligning two bases, or aligning zero bases in one read with N bases in the other. The first can result in two cases: a match or a mismatch, the latter results in N gaps. The alignments are built incrementally, a non-trivial alignment is always a combination of a smaller (known) alignment, plus a match, mismatch or indel. The scores in this thesis used are $match = 1$, $mismatch = gap = -1$ unless stated otherwise. Of these cases, the maximum resulting score is considered as the score for that alignment. This recursion is shown in Equation 2.1, where $S(i, j)$ is the maximum score when aligning subreads $a[0 : i]$ and $b[0 : j]$, $s(a_i, b_j)$ is the (mis)match score of those two bases, and gap is the gap penalty. $a[i]$ indicates a single base, and is shortened to a_i.

$$S(i,j) = max \begin{cases} S(i-1,j) + gap \\ S(i,j-1) + gap \\ S(i-1,j-1) + s(a_i, b_j) \end{cases} \tag{2.1}$$

The score of every calculated alignment is stored in a matrix, where the score in the bottom-right corner indicates the score of the total alignment. To find out the actual alignment, like in Figure 2.27, traceback must be performed. Traceback starts at the bottom-right corner, and for each matrix element, the source of the maximum score is found, either because it was stored in a different matrix, or because it is recalculated. The traceback of one element is shown in Figure 2.29.

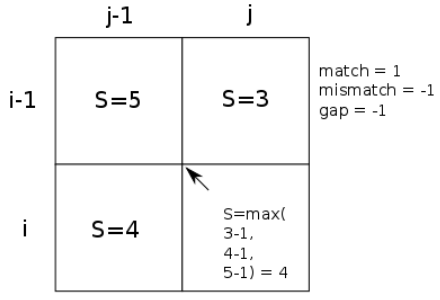

Figure 2.29: Calculating a single element, the source of its score is either element on the left, top, or top-left, this determines the direction of the traceback pointer.

The algorithm consists of three steps: initialization, filling the matrix, and performing traceback. The initialization writes scores to the first row and column, as these alignments are trivial. Figure 2.30 shows an initialized matrix.

D		C_1	C_2	G_3	G_4	T_5	C_6	T_7
	0	-1	-2	-3	-4	-5	-6	-7
A_1	-1							
C_2	-2							
G_3	-3							
T_4	-4							
T_5	-5							
C_6	-6							
A_7	-7							
Score:								

Figure 2.30: The NW matrix, initialised, source: [25]

The matrix is filled according to the recurrence relation, where the elements have to be calculating in a particular order, because certain previous values need to be known. A filled matrix is shown in Figure 2.31.

D		C_1	C_2	G_3	G_4	T_5	C_6	T_7
	0	-1	-2	-3	-4	-5	-6	-7
A_1	-1	-1	-2	-3	-4	-5	-6	-7
C_2	-2	0	0	-1	-2	-3	-4	-5
G_3	-3	-1	-1	1	0	-1	-2	-3
T_4	-4	-2	-2	0	0	1	0	-1
T_5	-5	-3	-3	-1	-1	1	0	1
C_6	-6	-4	-2	-2	-2	0	2	1
A_7	-7	-5	-3	-3	-3	-1	1	1
Score: 1								

Figure 2.31: The NW matrix, filled, source: [25]

The traceback starts at the bottom-right element. This $S(N, M)$ element indicates the score of aligning the total reads in an optimal way. The traceback path either follows the traceback pointers, denoted by arrows in Figure 2.32, or recalculates the possible scores, and chooses the direction based on the maximum. The traceback path always ends at element $S(0, 0)$.

D		C_1	C_2	G_3	G_4	T_5	C_6	T_7
	0	-1	-2	-3	-4	-5	-6	-7
A_1	-1	-1	-2	-3	-4	-5	-6	-7
C_2	-2	0	0	-1	-2	-3	-4	-5
G_3	-3	-1	-1	1	0	-1	-2	-3
T_4	-4	-2	-2	0	0	1	0	-1
T_5	-5	-3	-3	-1	-1	1	0	1
C_6	-6	-4	-2	-2	-2	0	2	1
A_7	-7	-5	-3	-3	-3	-1	1	1
Score: 1								

Figure 2.32: The NW matrix, with the traceback path highlighted, source: [25]

The affine gap penalty is implemented via 3 matrices, one for a match/mismatch, one for a gap in read A, one for a read in B. Multiple recurrences are used to fill these matrices:

$$M(i,j) = max \begin{cases} A(i-1,j-1) + s(a_i, b_j) \\ B(i-1,j-1) + s(a_i, b_j) \\ M(i-1,j-1) + s(a_i, b_j) \end{cases} \tag{2.2}$$

$$A(i,j) = max \begin{cases} M(i-1,j) + gap\_open \\ A(i-1,j) + gap\_extend \end{cases} \tag{2.3}$$

$$B(i,j) = max \begin{cases} M(i,j-1) + gap\_open \\ B(i,j-1) + gap\_extend \end{cases} \tag{2.4}$$

$A(i, j)$ indicates the score between subreads $a[0 : i]$ and $b[0 : j]$, where a_i is aligned with a gap. This could be a newly opened gap, in which case the score is based on $M(i-1, j)$, the optimal score for aligning $a[0 : i-1]$ and $b[0 : j]$, and the *gap_open* penalty. Or it could be an extension of an existing gap, in which case the score if based on $A(i-1, j)$, where base a_{i-1} is also aligned with a gap, and the *gap_extend* penalty.

2.3.3 Local alignment

Smith-Waterman (SW)[93] is a variation on Needleman-Wunsch, it finds local alignments instead of global alignments. It is also based on dynamic programming, but has small changes which allow the algorithm to start and stop an alignment at any point in the matrix. Cells with a negative score are set to zero. Traceback starts at the highest scoring cell, instead of in the bottom-right corner. NW traceback always ends in the top-left corner, but SW traceback can also end in a cell with score zero. Smith-Waterman is also exact, producing the optimal local alignment.

During initialization, the first row and column are set to zero. The recurrence relation is slightly changed:

$$S(i,j) = max \begin{cases} S(i-1,j) + gap \\ S(i,j-1) + gap \\ S(i-1,j-1) + s(a_i, b_j) \\ 0 \end{cases} \tag{2.5}$$

These changes allow stretches with a negative alignment score to be ignored. Note that the scores are still based on previous scores, one cannot take a filled NW matrix and set the negative scores to zero to perform the SW algorithm.

Figure 2.33 shows the filled SW matrix, some positive values are different from the NW matrix. Figure 2.34 shows the traceback path, note that it does not span the whole sequences.

S		C_1	C_2	G_3	G_4	T_5	C_6	T_7
	0	0	0	0	0	0	0	0
A_1	0	0	0	0	0	0	0	0
C_2	0	1	1	0	0	0	1	0
G_3	0	0	0	2	1	0	0	0
T_4	0	0	0	1	1	2	1	1
T_5	0	0	0	0	0	2	1	2
C_6	0	1	1	0	0	1	3	2
A_7	0	0	0	0	0	0	2	2
Score: 3								

Figure 2.33: The SW matrix, filled, source: [26]

S		C_1	C_2	G_3	G_4	T_5	C_6	T_7
	0	0	0	0	0	0	0	0
A_1	0	0	0	0	0	0	0	0
C_2	0	1	1	0	0	0	1	0
G_3	0	0	0	2	1	0	0	0
T_4	0	0	0	1	1	2	1	1
T_5	0	0	0	0	0	2	1	2
C_6	0	1	1	0	0	1	3	2
A_7	0	0	0	0	0	0	2	2

Score: 3

Figure 2.34: The SW matrix, with the traceback path highlighted, source: [26]

2.3.4 Semi-global methods

Global alignment aligns every base in both sequences, local alignment allows for gaps at the start and end of both sequences. By applying a subset of the differences between NW and SW, hybrid or semi-global methods can be created [94][95]. It might be useful to penalize gaps at the start, while allowing gaps at the end. Or to only allow gaps for one sequence.

2.3.5 Heuristic algorithms

A heuristic algorithm does not guarantee to find an optimal solution, but gives an approximate solution to a particular problem [96]. Heuristics are often used to reduce the runtime, and a well designed heuristic will prevent large drops in the quality of the solution.

In DNA alignment, a heuristic will often reduce the number of elements in a NW or SW matrix that are calculated.

Many heuristic aligners implement a seed-and-extend technique [97]. A seed is a small piece that contains k consecutive nucleotides, also known as Kmer, where k is usually 3-25. Exact Kmer matches are found, and the overlap is extended from there.

A small k will result in lots of hits, many of which will not result in a convincing overlap. A large k will limit the number of hits, but also reduces the sensitivity, since actual overlaps could have zero large consecutive matching nucleotides. To increase the sensitivity when using a large k, one could allow one or more mismatches in the seed, to create so-called spaced seeds [98]. To reduce the number of hits, multiple nearby Kmer matches can be required to start extension, since an actual overlap is likely to have lots of matching nucleotides.

For the extension phase, a variant of dynamic programming can be used, which forces the traceback to start at the end of the seed. The dynamic programming can include an area that is smaller than the matrix that would be calculated by regular Smith-Waterman. Variants include Banded Smith-Waterman [99] and Xdrop [100].

2.3.6 Multiple Sequence Alignment

Multiple Sequence Alignment algorithms [101] try to align three or more sequences, usually of similar length. The results can be used to create phylogenetic trees or to study evolutionary relationships [102].

expand this section?

2.4 DNA Assembly

Every DNA sequencing technique produces reads, which are pieces of data that represent a part of the sampled DNA, possibly containing some errors. These length and error rate of these reads depend of the sequencing technique used, but they all have one thing in common: they have to be assembled to be useful. DNA assembly tries to combine the reads into large, high quality parts.

The process of DNA assembly is often compared to taking a copy of a book, shredding it, and trying to piece is back together. However, when taking a read and splitting it into two parts, there is no information indicating that the two parts were originally together, as it would with shredded paper. This is why DNA is always oversampled, this means that each base is sequenced multiple times. Figure 2.35 shows a simplified assembly process.

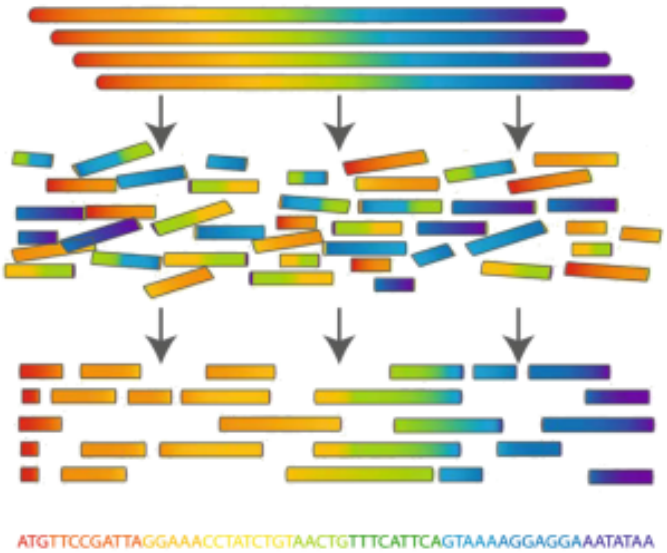

Figure 2.35: A DNA strand is cloned and sequenced, the reads are assembled and the original DNA strand is recovered, source: [27]

There are two types of assembly: denovo and reference based. Denovo assembly works

as described above, but reference based assembly first maps the reads to a reference. If multiple reads map to the same place in the reference, they are likely to be overlapping. Denovo assembly compares each read with every other read, resulting in a naive time complexity of $O(N^2)$, where N is the number of reads. Denovo assembly is an NP-hard problem [103]. Mapping each read to a reference takes $O(N)$, and then a much smaller number of reads has to be compared with each other.

The drawback of reference based assembly is that a read must first be mapped to a reference, this means that reads that contain new or very different stretches of DNA are filtered away, this results in a biased assembly. Reference based assembly is good at finding small variations between the sampled DNA and the reference [104].

2.4.1 Denovo assembly

There are two types of denovo assemblers: string-based and graph-based. String-based assemblers are greedy [105]. Examples of string-based assemblers are SSAKE [106], SHARCGS [107] and QSRA [108]. The greedy approach works like this: from the set of reads, pick the two strings that have the 'best' overlap and merge them, repeat until there is one string left or there are no suitable overlaps anymore. The 'best' overlap can differ: a perfect overlap between two strings s and t is a string y, such that s = xy and t = yz for some non-empty strings x and z [109], but different scoring schemes allowing a few mismatches here and there are possible. A suitable overlap will have a minimum length: since there are only four different bases, a set will likely have two reads with an overlap of one base, one can imagine merging these is not likely to be correct. The greedy approach requires all reads to be compared with all reads, which is not suited for the high throughput of NGS systems. Another problem are repeat regions, which are very common in eukaryotic genomes, the human genome can contain more than 50% repeats or repetitive areas [110][111].

Figure 2.36 shows how a greedy algorithm can misassemble reads containing a repeat region.

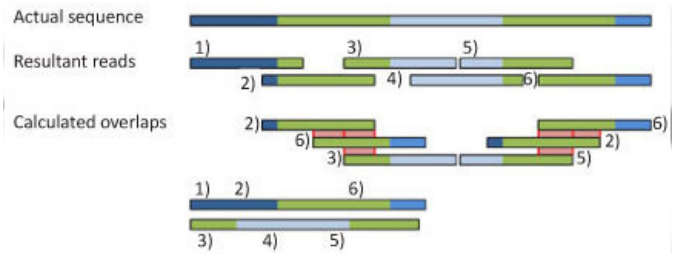

Figure 2.36: Pieces of a repeat region (green) occur in multiple reads, fragments 2 and 6 have a bigger overlap than 2 and 3 or 5 and 6, causing a misassembly, adapted from: [28]

For NGS reads, graph-based assemblers are used, which come in two flavours: overlap-layout-consensus (OLC) and de Bruijn Graph (dBG) [112].

The OLC assemblers first find overlaps and build an overlap graph. Each node

represents a read, and each edge represents an overlap between two reads. During the layout phase, the graph is analysed to find paths, corresponding to segments of the original genome. The perfect graph contains one path that visits each node exactly once. This problem can be described as finding a Hamiltonian Path [113], which is also an NP-hard problem [114]. A Hamiltonian Path includes all vertices of a graph exactly once. Assemblers that use the OLC approach are Newbler [115], SGA [116] and Edena [117]. Figure 2.37 shows the general workflow for an OLC assembler.

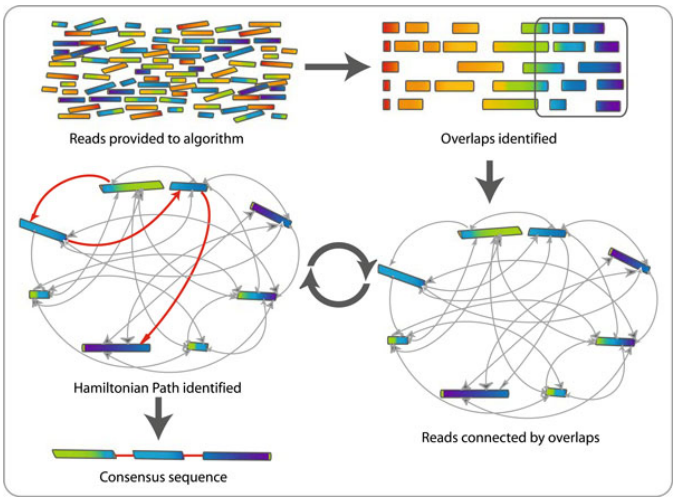

Figure 2.37: Overlaps are found and put in a graph, each arrow represents an overlap, a Hamiltonian path is found to create the contig, source: [29]

To build a de Bruijn Graph, each read is divided into Kmers. Each Kmer represents a directed edge between two vertices, where the source vertex represents the first k-1 bases of the Kmer, and the destination vertex the last k-1 bases of the Kmer. When a particular k-1 vertex does not exist, it is created, otherwise the existing one is reused. This means there are at most 4^{k-1} vertices in the graph. The weights of the edges indicate how many times a particular Kmer is encountered. The next step is to find an Eulerian Path [113], which is a path that includes all edges of a graph exactly once. A small value of k means the number of vertices is limited, but it also creates more edges, and loses more information. Examples of dBG assemblers are Velvet [118], ABySS [119] and SOAPdenovo2 [120].

Constructing a dBG is easier than constructing an OLC graph, since no overlaps are computed. It is also easier to find an Eulerian Path than to find a Hamiltonian Path. However, there can be an exponential number of Eulerian Paths in a graph, where only one is needed. De Bruijn Graphs are quite susceptible to sequencing errors, one substituted base causes k incorrect Kmers. Pair this with a often-used value of k between 19 and 27 and a PacBio error rate of about 15%, and it is clear that the graph will contain a lot of incorrect edges. Another drawback of using this strategy is that information is lost when dividing the reads into Kmers. This information could overcome short repeats in the genome. That is the reason why a variant of the Eulerian Path problem was formulated: the Eulerian Superpath problem [121]. It tries to find an Eulerian path,

that contains a set of subpaths, given a graph and the set of subpaths.

The previously described methods cannot reconstruct the whole genome on their own, their output consists of contigs: sequences of bases which are known. The next step is building scaffolds [122][123]. A scaffold consists of contigs and gaps. The gaps contain unknown bases, denoted by 'N', but their lengths is roughly known. To create scaffolds, paired-end reads can be used. When two ends of the paired-end read are found in different contigs, these contigs are likely to belong next to each other, with a gap between them.

consider talking about Optical Mapping to aid scaffolding

2.4.2 Reference based assembly

The biggest part in reference based assembly is finding the location in the reference that the read came from, this process is called 'mapping'. Mapping is usually done by a 'seed-and-extend' method. This means that first a position is found that is promising (seed), and then the algorithm looks at both ends of the seed to see if those parts also match with the reference.

talk about error correction before/after assembly?

The most used method of finding seeds is to divide the reads into Kmers. A Kmer is a piece of DNA with exactly k bases [124]. The reference is also divided into Kmers, and exact Kmer matches are found using various methods. A popular and basic method is hashing, used by MAQ [125], Seqmap [126], Mosaik [127], Snap [128], Shrimp2 [129], Stampy [130] and SeqAlto [131]. A different approach uses BWT [132] or suffix arrays [133] to find matches, used by BWA [134], Soap2 [135], Bowtie2 [136], BWA-SW [137], Aryana [138] and BFAST [139]. An uncommon approach is to use sorting to find matching Kers, like in merge-sort [140][141]. This technique is used by Slider [142] and Syzygy [143].

The value of k is very important: too low and there will be too many hits, too high and there will not be enough hits. The value is also dependent on the length of the reads, and the type of reference (a different k might be used for humans or bacteria). Values between 12 and 32 are common.

When a seed is found, the rest of the read must also be checked against the reference. This is the 'extend' part. Not all mappers have an extend part, like SeqMap. Most use Smith-Waterman for this stage. Snap uses an $O(ND)$ algorithm from Ukkonen [144]. Stampy uses a probabilistic aligner to extend their seeds. SeqAlto and Aryana use a banded Needleman-Wunsch.

When all the reads are mapped to the reference, they have to be combined. This can be done by local denovo assembly, or a consensus algorithm [145].

2.5 GPU processing

A GPU is a Graphics Processing Unit, it is a processor that is mainly used to perform video processing. This type of processing often includes rotation and translation of objects in a space, calculating shadows and rendering images to display on a monitor. They contain many cores that allow it to perform parallelizable tasks very quickly. A GPGPU, or General Purpose GPU can be programmed to perform tasks that are different from

video processing. An algorithm like matrix addition is easily parallelized by assigning a matrix cell to each thread. Each thread can perform the addition in parallel, let this take one cycle. A sequential implementation would have taken $N \times M$ cycles, where N and M are the dimensions of the matrices.

CPUs usually have large caches and a complex instruction set and execution that includes out-of-order execution and branch prediction [146][147].

GPUs cannot operate on their own, they must be guided by a CPU. The functions that run on a GPU are called *kernels*, and are usually launched by a CPU. Kernels can also be called from other kernels. A general workflow using a GPU is shown in Figure 2.38.

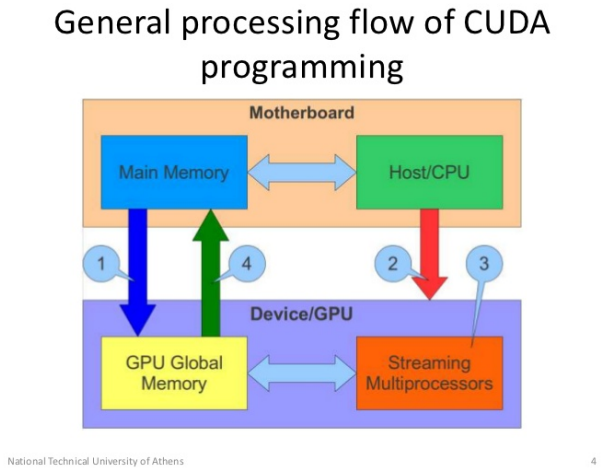

Figure 2.38: 1: data is copied from main memory to the GPU memory, 2: the CPU launches the GPU kernel, 3: the GPU executes the kernel, using the GPU memory, 4: results are copied from the GPU to the main memory, source: [30]

maybe divide into sub-sections

2.6 CUDA

CUDA is a parallel computing platform that allows people to use Nvidia GPUs for their own applications. Developers can write functions that will execute on the GPU called *kernels*, these are launched from a CPU function. The GPU is referred to as *device* and the CPU as *host*. Kernels for CUDA are written in C++.

Kernels can be launched from the CPU in a *grid* with a certain number of thread blocks or *blocks* and *threads*. A grid is one-/two- or three-dimensional, and has an array of blocks in each direction. Each block itself is also one-/two- or three-dimensional, and has an array of threads in each direction. Figure 2.39 shows a full grid.

Each thread executes the kernel code, although they usually operate on different data. Threads in a block can communicate via shared memory.

On a hardware level, an NVIDIA GPU is divided into Streaming Multiprocessors (SMX). Each SMX contains a number of cores, or Streaming Processors (SP), these are

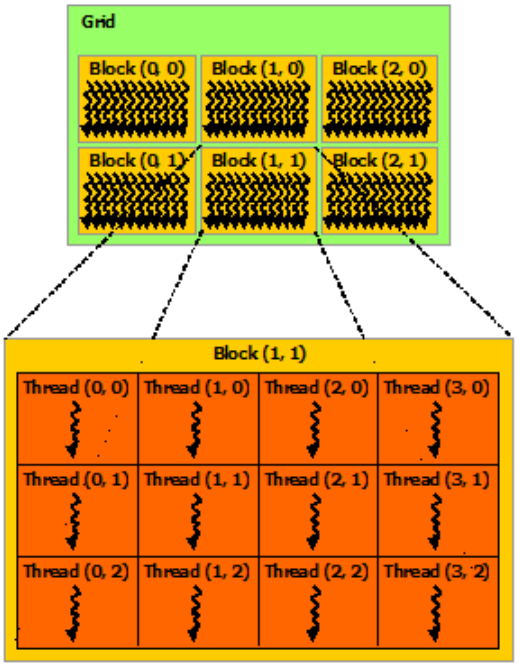

Figure 2.39: A CPU thread can launch a grid, which contains one or more blocks, each block contains one or more threads. Both grids and blocks can be organised in up to three dimensions, source: [31]

the basic building blocks and perform the actual calculations. Each block is assigned to at most one SMX. This block's threads are then executed as warps, with 32 threads per warp. Each SMX has multiple warp schedulers, so multiple warps can run in parallel on an SMX. All threads in a warp must execute the same instruction, if a thread is the only to take a branch, the other threads must wait until the branch is completed, this is called divergence.

Memory operations are also executed in parallel, this means that all threads try to read/write to the memory in parallel. If the addresses are next to each other, only one memory transaction is needed, since a transaction processes a whole memory line. This is known as coalescing. Uncoalesced memory accesses cause multiple memory transactions.

2.6.1 Memory hierarchy

GPUs have several different memory types and levels. Not all of these are accessible from the host or from other components of the memory hierarchy [31]. Figure 2.41 shows three types of memory.

- Global memory: The largest and slowest memory, located outside the chip. Can be read and written from the host. Best used for coalesced operations. Figure 2.40 shows the difference between coalesced and uncoalesced reading.
- Local memory: Each thread has private memory for when registers are not enough. It is cached in L1 cache, or stored in global memory when too large.

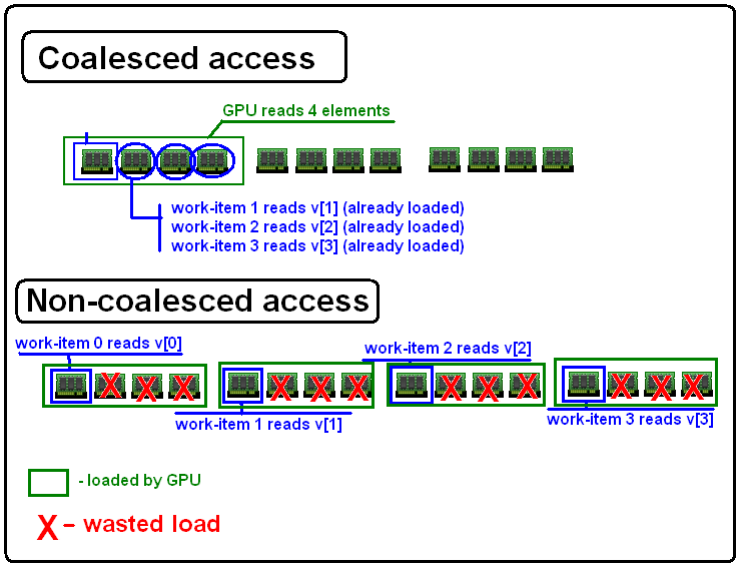

Figure 2.40: Uncoalesced accesses are inefficient, source: [32]

- Registers: The fastest memory available, located on-chip. A thread has a maximum number of registers available depending on Compute Capability.
- Shared memory: Each SMX has shared memory, it can be accesses by every thread in every block on the SMX. This can be used by threads in a block to communicate. It is located on-chip, so very fast.
- Constant memory: The host can initialize this memory, the kernel can only read it. Reading is as fast as reading a register, but only when all addresses of a half-warp are the same, otherwise reading is serialized. It resides off-chip, but is cached on-chip.
- Texture memory: Can only be written from the host. Resides off-chip, but is cached on-chip, like constant memory. The main feature about the texture memory is that it is cached for 2D spatial locality. Figure 2.42 shows an access pattern that would not be cached with a typical scheme.
- L2 cache: Behaves as cache for device memory and is shared among all SMXs. The cache line size is 32B.
- L1 cache: Cache line size is 128B. Its default behaviour is to only cache local memory, not global memory. However, for certain architectures applications can opt-in to cache both global and local loads in L1 [148].

2.6.2 Streams

When copying data to or from the GPU, there are usually no kernels running in a naive implementation. This means that if the copy time is large, the overall efficiency is quite

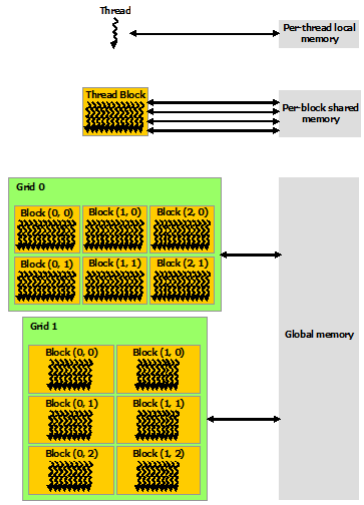

Figure 2.41: Memory hierarchy [31]

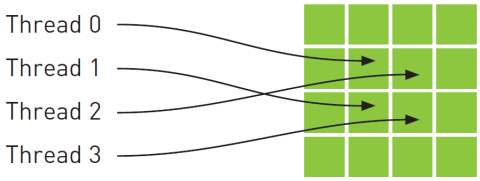

Figure 2.42: This access pattern has spatial locality, and could be cached in the texture cache [33]

low. These operations can be overlapped or pipelined by using streams. Figure 2.43 shows how the execution time can decrease by overlapping copying and computing.

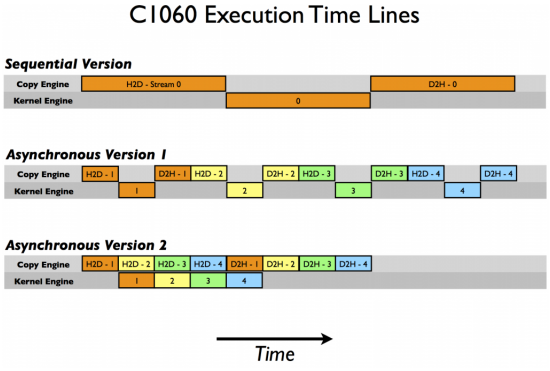

Figure 2.43: Different versions of the same computation [34]

add section on OpenCL?

2.6.3 Bioinformatics on GPU

Multiple efforts to accelerate DNA alignment algorithm on GPU have been made. MUMmerGPU [149][150] is one of the first GPU accelerated algorithms, it stores a suffix tree of the reference sequence on the GPU, and aligns it with queries. Its newest GPU implementation shows a 13x speedup over the CPU implementation. CUDAlign [151] accelerates the exact Smith-Waterman algorithm, and allows an affine gap. The input sequence length is only restricted by the available global memory. CUDAlign 2.0 [152] uses linear space and boasts a 702x and 19.5x speedup compared to 1 core and 64 cores respectively.

SeqNFind [35] is a hardware/software solution based on NVIDIA Tesla GPUs. It can be used to speed up numerous algorithms.

Figure 2.44: An image of the SeqNFind hardware, source [35]

CUSHAW [153] is an accelerated short read aligner that uses BWT. It uses CUDA and is optimized for the Fermi architecture. Each GPU thread is assigned a short read and its complement to align to the reference.

mention other/-more GPU algo-rithms?

Concept 3

3.1 Pacbio reads

Daligner finds alignments between long, noisy reads. Pacific Biosciences has commercially launched its first sequencer in 2011. It is able to output reads with an average of 1000 bases, which is significantly longer than Next-Generation Sequencing (NGS) reads [154]. In 2014, a new polymerase-chemistry combination was released, called P6-C4. This version can output average read lengths of 10000-15000 bases, and its longest reads can exceed 40000 bases [155]. While the drawback is that these reads have an error rate of 12-15%, this can be compensated by the distribution of these errors [156]. First, the set of reads is a nearly Poisson sampling of the sampled genome. This implies that there exists a coverage c for every target coverage k, such that every region of the genome is covered k times [157]. Secondly, the work of Churchill and Waterman [158] implies that the accuracy of the consensus sequence of k sequences is $O(\epsilon^k)$, which goes to 0 as k increases. This means that if the reads are long enough to handle repetitive regions, in principle a near perfect denovo assembly of the genome is possible, given enough coverage [156].

Important points for denovo DNA sequencing are: what level of coverage is needed for high quality assembly? And how to build an assembler that is able to deal with high error rates and long reads? Most previous assemblers work with NGS reads, which are much shorter and have much lower error rates. Some algorithms used in these assemblers, such as De Bruijn Graph (DBG) [159] would grow too large for high error rates and long reads. Since Daligner was build, new methods of using DBG with long reads have been developed, but they rely on a short read based DBG to correct errors in long reads [160][161].

3.2 Daligner

The first step in an Overlap-Layout-Consensus (OLC) assembler is usually finding overlaps between reads [162]. BLASR [163] was the only long read aligner at the time, and inspired Daligner. It uses the same filtering concept, but with a cache-coherent threaded radix sort to find seeds, instead of a BWT index [132]. The most time-consuming step is extending the seed hit to find an alignment. To do this, Daligner uses a novel method which adaptively computes furthest reaching waves of the older O(nd) algorithm [164], combined with heuristic trimming and a data structure that describes a sparse path from the seed hit to the furthest reaching point.

Daligner performs all-to-all comparison on two input databases $\mathcal{A}$, with M long reads $A_1, A_2, ...A_M$ and $\mathcal{B}$, with N long reads $B_1, B_2, ...B_N$ over alphabet $\Sigma = 4$ It reports alignments $P = (a, i, g)x(b, j, h)$ such that $len(P) = ((g - i) + (h - j))/2 \geq \tau$ and the

next part is copied straight from daligner paper

optimal alignment between $A_a[i+1, g]$ and $B_b[j+1, h]$ has no more than $2\epsilon \cdot len(P)$ differences, where a difference can be either an insertion, a deletion or a substitution. Both τ and ϵ are user settable parameters, where τ is the minimum alignment length and ϵ the average error rate. The correlation, or percent identity of the alignment is defined as $1 - 2\epsilon$.

An edit graph for read $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ is a graph with $(m+1)(n+1)$ vertices $(i, j) \in [0, M] \times [0, N]$. It also has three types of edges:

- deletion edges $(i-1, j) \rightarrow (i, j)$ with label $\begin{bmatrix} a_i \\ - \end{bmatrix}$ if $i > 0$.
- insertion edges $(i, j-1) \rightarrow (i, j)$ with label $\begin{bmatrix} - \\ b_j \end{bmatrix}$ if $j > 0$.
- diagonal edges $(i-1, j-1) \rightarrow (i, j)$ with label $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$ if $i, j > 0$.

Figures 3.1 and 3.2 show images from a tool, created by Nicholas Butler [36], that shows how the edit graph is traversed by the O(nd) algorithm.

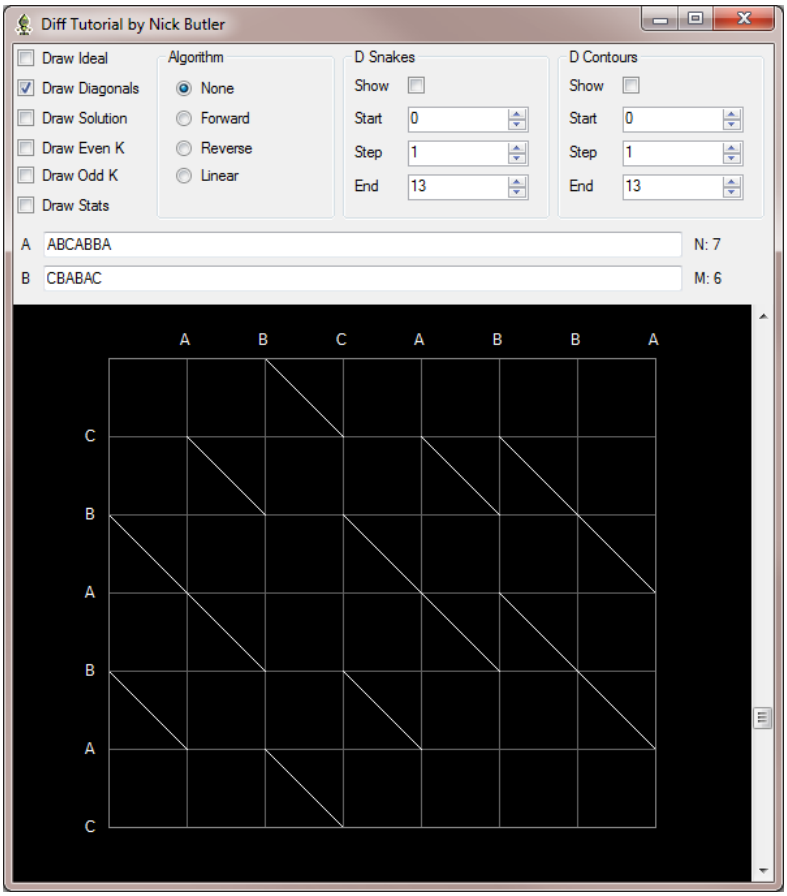

Figure 3.1: An empty edit graph, source: [36]

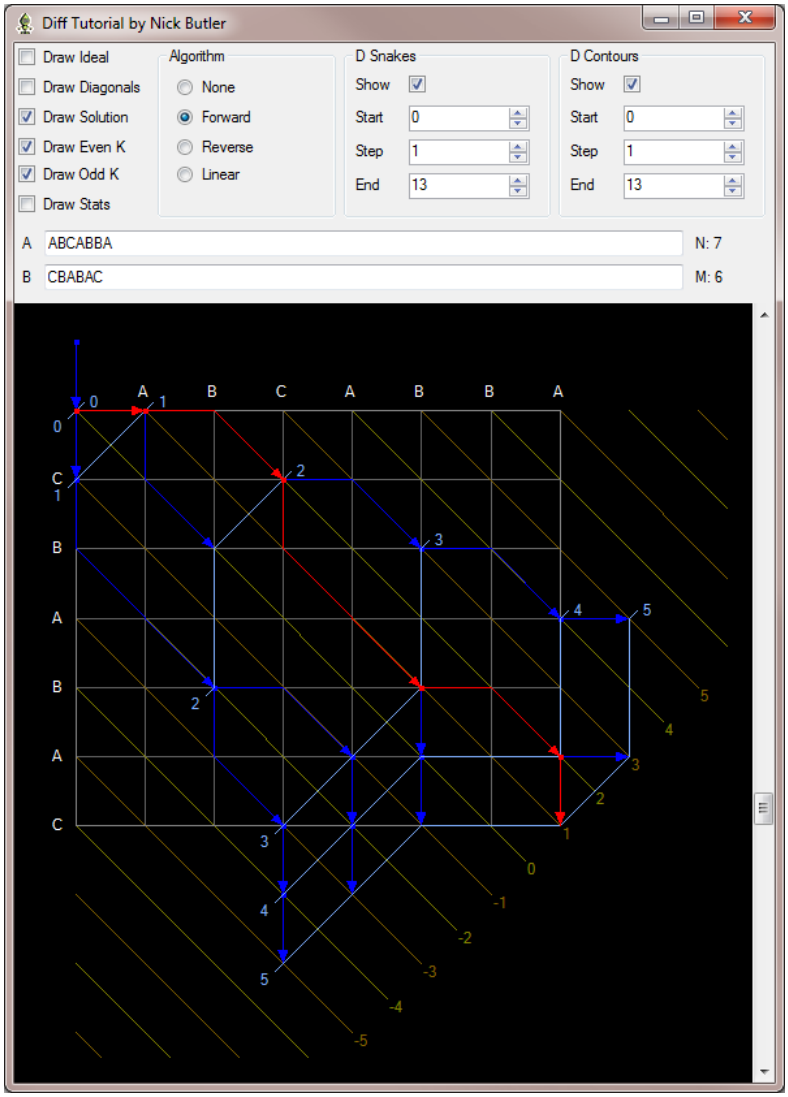

Figure 3.2: A filled edit graph, the dark blue lines represent mismatches and their subsequent snakes, the light blue lines are the d-waves, the brown lines are odd diagonals, yellow lines are even diagonals, the red path is the shortest path, source: [36]

An alignment between $A[i+1, g]$ and $B[j+1, h]$ is described as a sequence of labels from vertex (i, j) to (g, h). A diagonal edge can be either be a match edge, when $a_i = b_j$, or a substitution edge. If a match edge has weight 0, and the other edges have weight 1, the weight of the total path is the number of differences in the alignment it represents. To find suitable alignments, we have to find a read subset pairs P such that $len(P) \geq \tau$ and the weight of the lowest scoring path between (i, j) and (g, h) in the edit graph of A_a and B_b is not more than $2\epsilon \cdot len(P)$.

The O(ND) algorithm tries to find progressive waves of furthest reaching (f.r.) points until the endpoint is reached. The goal is to find longest possible paths starting at a starting point $\rho = (i, j)$ with 0 differences, then 1 difference, then 2 and so on. After

d differences, the possible paths can end in diagonals $\kappa \pm d$, where $\kappa = i - j$ is the diagonal of the starting point. The furthest reaching point on diagonal k that can be reached from ρ with d differences is called $F_\rho(d,k)$. A collection of these points for a particular value of d is called the d-wave emanating from ρ, and defined as $W_\rho(d) = \{F_\rho(d,\kappa-d),...,F_\rho(d,\kappa+d)\}$. $F_\rho(d,k)$ will be referred to as $F(d,k)$, where ρ is implicitly understood from the context.

In the O(ND) paper it is proven that:

$$F(d,k) = Slide(k, max\{F(d-1,k-1)+(1,0), F(d-1,k)+(1,1), F(d-1,k+1)+(0,1)\} \tag{3.1}$$

where $Slide(k,(i,j)) = (i,j) + max\{\Delta : a_{i+1}a_{i+2}...a_{i+\Delta} = b_{j+1}b_{j+2}...b_{j+\Delta}\}$.

A slide is a path of sequential match edges. The f.r. d-point on diagonal k is calculated by finding the furthest of

- the f.r. (d-1)-point on $k-1$ followed by an insertion
- the f.r. (d-1)-point on k followed by a substitution
- the f.r. (d-1)-point on $k+1$ followed by a deletion

and then continuing as far as possible along the slide. A point (i,j) is furthest when its anti-diagonal $i+j$ is greatest. The best alignment between reads A and B is the smallest d such that $(m,n) \in W_{(0,0)}(d)$, where m and n are the length of reads A and B. The O(ND) algorithm computes d-waves from starting point $(0,0)$ until the end point (m,n) is reached. The complexity of this algorithm is $O(n+d^2)$ when A and B are non-repetitive sequences [156]. Because seeds are not always at the beginning, so waves are computed in both forward and reverse direction. The latter is easily done by reversing the direction of edges in the edit graph.

3.2.1 Seeding: concept

To find suitable starting points for the edit graphs, seeding is done. A seed is a section where $A[i,g]$ and $B[j,h]$ have a certain high similarity that indicates that these reads probably originate from the same part of the genome. Finding a seed includes finding matching Kmers for every readpair (a,b) with $a \in \mathcal{A}$ and $b \in \mathcal{B}$. Previous methods to match Kmers include Suffix Arrays [133] and BWT indices [132].

Assuming that the Kmer matches are independent, the probability that a Kmer is conserved while sequencing is $\pi = (1-2\epsilon)^k$. The number of conserved Kmers in an alignment of τ basepairs is a Bernouilli distribution with rate π, so an average of $\tau \cdot \pi$ Kmers are expected in this alignment. An example: $k = 14$, $\epsilon = 15\%$ and $\tau = 1500$, then $\pi = .7^{14} = 0.0067$ and the average number of conserved Kmers is 10. Only .046% of the expected readpairs have 1 or fewer Kmers, and only 0.26% have 2 of fewer. To filter with 99.74% sensitivity, only readpairs with 3 or more Kmer matches need to be examined.

The specificity of the filter is increased in two ways:

- computing the number of Kmer matches in diagonal bands of width 2^s instead of in the whole reads

- thresholding on the number of bases in Kmer matches, instead of the number of Kmers themselves

The first way decreases the false positive rate because it only allows readpairs that have their Kmer matches relatively close, indicating a smaller region with higher similarity. The second way relies on the fact that 3 overlapping Kmers have a higher probability ($\pi^{1+2/k}$) than 3 disjoint Kmers with $3k$ basepairs (π^3).

The actually find the Kmer matches, Daligner uses a sort-merge procedure:

- Build a list $List_X = \{(kmer(X_x, i), x, i)\}_{x,i}$ of all Kmers for database $X \in \{\mathcal{A}, \mathcal{B}\}$, where $kmer(R, i)$ is the Kmer $R[i - k + 1, i]$.
- Sort both lists in order of their Kmers.
- Merge the two lists and build $List_M = \{(a, b, i, j) : kmer(A_a, i) = kmer(B_b, j)\}$ of read and position pairs that have the same Kmer.
- Sort $List_M$ lexicographically on a, b and i where a is most significant.

All entries for a certain read pair (a, b) are in a continuous segment of the list. This makes it easy to determine if that read pair has enough Kmers and in the right places to constitute a seed hit. Given parameters h and s, each entry (a, b, i, j) for the current read pair is placed in diagonal bands $d = \lfloor (i - j)/2^s \rfloor$ and $d + 1$. Now determine the number of bases in the A-read covered by Kmers in each pair of neighbouring diagonal bands. Note that only bases in matching Kmers are counted, not the number of Kmers. When there are k+1 matching consecutive bases, two Kmers are generated. These are less 'valuable' than two non-overlapping Kmers. If $Count(d) \geq h$ for any diagonal band d, there is a seed hit for each position (i, j) in the band d unless position i was already in the range of a previously calculated local alignment. In that case the seed is called 'skippable', Figure 3.3 illustrates this concept.

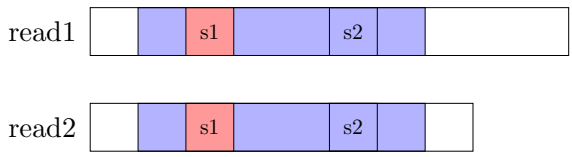

Figure 3.3: s1 (red) and s2 are seeds, s1 is extended first, and its extension (blue) includes s2. If s2 were to be extended, its extension would be the same as that of s1, therefore s2 does not need to be extended

The best values for h and s depend on things like ϵ and the read lengths.

For Daligner, the default k is 14, and assumed error rate is 0.85.

3.2.2 Seeding: implementation

Daligner is designed to use multiple threads and use the cache efficiently. Building and merge the lists in steps 1 and 3 is easy, since only one pass is needed for both actions.

The elements of the lists are compressed into 64-bit integers. Daligner uses a radix sort [141][165] to sort the lists in steps 2 and 4. Each 64-bit integer is a vector of $P = \lceil hbits/B \rceil$, B-bit digits $(x_P, x_{P-1}, x, ..., x_1)$ where B is a parameter. The sort needs P passes, where each pass sorts the elements on a B-bit digit x_i. Each pass is done with a bucket sort [141] with 2^B buckets. Instead of a linked list, the integers in the list *src* are moved in precomputed segments $trg[bucket[b]...bucket[b+1]-1]$ of an array $trg[0...N-1]$ with the same size as *src*. For the p^{th} pass, $bucket[b] = \{i : src[i]_p < b\}$ for each $b \in [0, 2^B - 1]$. The pseudocode is shown is Listing 3.1.

Listing 3.1: Basic radix sort

```
1 for i = 0 to N-1 do
2 {
3     b = src[i]_p
4     trg[bucket[b]] = src[i]
5     bucket[b] += 1
6 }
```

The algorithm takes $O(P(N + 2^B))$ time, but B and P are small fixed numbers so it is effectively $O(N)$. There are a lot of parallel sorting algorithms [166][167], but Daligner uses a new method that needs half the number of passes that traditional methods use. Each thread sorts a contiguous segment of size $part = \lceil N/T \rceil$ of *src* into *trg*, where T is the number of threads. This means each thread $t \in [0, T-1]$ has a bucket array $bucket[t]$ where $bucket[t][b] = \{i : src[i] < b$ or $src[i] = b$ and $i/part < t\}$. To reduce the number of passes, a bucket array for the next pass is filled during the current pass. Each thread counts the number of B-bit digits that will be handled in the next pass by itself and every other thread separately. If the number at index i will be at index j and in bucket b next pass, then the count in the current pass must not be recorded for the thread $i/part$ that currently sorts the number, but for thread $j/part$ that will sort it in the next pass. To do this we need to count the number of these events in $next[j/part][i/part][b]$ where *next* is a $T \times T \times 2^B$ matrix. If $src[i]$ is about to be moved in the p^{th} pass, then $j = bucket[src[i]_p]$ and $b = src[i]_{p+1}$. This algorithm takes $O(N/T + T^2)$ time, assuming B and P are fixed. The new algorithm is shown in Listing 3.2.

Listing 3.2: Radix sort in Daligner

```
1  int64 MASK = 2^B-1
2
3  sort_thread(int t, int bit, int N, int64 *src, int64 *trg, int
       *bucket, int *next)
4  {
5      for i = t*N to (t+1)*N-1 do
6      {
7          c = src[i]
8          b = c >> bit
9          x = bucket[b & MASK] += 1
10         trg[x] = c
11         next[x/N][(b >> B) & MASK] += 1
```

```
12      }
13  }
14
15  int64 *radix_sort(int T, int N, int hbit, int64 src[0..N-1],
        int64 trg[0..N-1])
16  {
17      int bucket[0..T-1][0..2^B-1], next[0..T-1][0..T-1][0..2^B-1]
18      part = (N-1)/T + 1
19      for l = 0 to hbit-1 in steps of B do
20      {
21          if (l != lbit)
22              bucket[t,b] = Sum_t next[u,t,b]
23          else
24              bucket[t,b] = |{i : i/part == t and src[i] & MASK == b
                    }|
25          bucket[t,b] = Sum_u,(c<b) bucket[u,c] + Sum_(u<t) bucket[
                u,b]
26          next[u,t,b] = 0
27          in parallel: sort_thread(t,l,part,src,trg,bucket[t],next[
                t]])
28          (src,trg) = (trg,src)
29      }
30      return src
31  }
```

This algorithm is particularly cache efficient because each bucket sort uses two small arrays *bucket* and *next* that will usually fit in the L1 cache. Each bucket sort makes one sweep through *src* and 2^B sweeps through the bucket segments of *trg*. This totals $2^B + 1$ sweeps during each pass. Each sweep can be prefetched if it is small enough. This means a smaller B is better for caching behaviour, but this increases the number of passes $hbit/B$ that are needed. On most processors, e.g. an Intel i7, $B = 8$ gives the fastest radix sort [156]. The optimal number of threads is more complex, because they usually do not have their own caches.

3.2.3 Local Alignment

Assuming the filter finds a seed-hit $\rho = (i, j)$, the basic idea is compute furthest reaching waves in forward and reverse direction to find the alignment. The problem is that as the wave propagates further from ρ, it spans wider and wider since it occupies $2d + 1$ diagonals. The final alignment will have only one point from each wave so most of the points are wasted, but we only which ones until the whole alignment is done. Several strategies are used to trim the width of the wave by removing f.r. points that are unlikely to be in the final alignment.

One of the trimming strategies is stopping when a segment with very low correlation is found. This referred to as the *regional alignment quality*. F.r. points with less than $\mathcal{M}$ matches in the last C columns are removed. For example, when $\epsilon = .15$ then a segment

with $M[k] < .55C = 33$ if $C = 60$ is probably not desirable.

To keep track of the matching/mismatching bases, we keep a bitvector $B(d, k)$ that represents the last $C = 60$ columns of the best path from ρ to a given f.r. point $F(d, k)$. A 0 will denote a mismatch and a 1 a match. This is easily updated by left-shifting a 0 or 1. The number of matches $M(d, k)$ can be tracked by observing the bit that is shifted out. Listing 3.3 shows pseudo-code that computes $W_\rho(d+1)[low-1, hgh+1]$ from $W_\rho(d)[low, hgh]$ assuming $[low, hgh] \subseteq [\kappa - d, \kappa + d]$ is the trimmed result of the wave $W_\rho(d)$. Note that the array W only holds the B-coordinate j of each f.r. point (i, j), since $i = j + k$.

Listing 3.3: Local Alignment

```
1  MASKC  = 1 << (C-1)
2  W[low-2] = W[hgh+2] = W[hgh+1] = y = yp = -1
3  for k = low-1 to hgh+1 do
4  {
5      (ym,y,yp) = (y,yp+1,W[d+1]+1)
6      if (ym = min(ym,y,yp))
7          (y,m,b) = (ym,M[k-1],B[k-1])
8      else if (yp = min(ym,y,yp)
9          (y,m,b) = (yp,M[k+1],B[k+1])
10     else
11         (y,m,b) = (y,M[k],B[k])
12
13     if (b & MASKC != 0)
14        m -= 1
15     b <<= 1
16     while (B[y] == A[y+k])
17     {
18         y += 1
19         if (b & MASKC == 0)
20            m += 1
21         b = (b << 1) | 1
22     }
23     (W[k],M[k],B[k]) = (y,m,b)
24 }
```

The second trimming principle involves only keeping f.r. points which are within $\mathcal{L}$ anti-diagonals of the maximal anti-diagonal reached by its wave. It makes sense that the f.r. point on diagonal k^* that will be in the final alignment is on a greater anti-diagonal $i + k$ than points that are not. As the other f.r. points in the wave move away from diagonal k^*, their anti-diagonal values decrease rapidly, and the wave gets the appearance of an arrowhead. The higher the correlation of the alignment, the sharper the arrowhead becomes. This means that points far enough behind the tip can be almost certainly removed. A value of $\mathcal{L} = 30$ is a universally good value for trimming [156]. Formally, for each wave computed from the previous trimmed wave, each f.r. point from $[low-1, hgh+1]$ that has either $M[j] < \mathcal{M}$ or $(2W[k^*]+k^*)-(2W[j]+j) > \mathcal{L}$ is removed.

Note that $W[k]$ contains the B-coordinate j, and $k = i - j$, so $(2W[k] + k) = i + j$.

The Daligner paper does not give a formal proof, but an argument why the average width of the wave $hgh - low$ is constant for any fixed value of ϵ. This means that the alignment finding algorithm has linear expected time as a function of alignment length. Consider two f.r. points A and B, where A lies on an alignment path with correlation $1 - 2\epsilon$ or higher, and B does not. For the next wave, point A moves forward with one difference and then slides on average $\alpha = (1-\epsilon)^2/(1-(1-\epsilon)^2)$ bases. Point B has no such correlation, so it jumps one difference and then only slides $\beta = 1/(\Sigma - 1)$ bases, assuming each base is equally likely. So an f.r. point d diagonals away from the final path has involved d jumps off the path, and is on average $d(\alpha - \beta)$ anti-diagonals behind the best f.r. point in the wave. The average width of a wave with an $\mathcal{L}$ lag cutoff is less than $2\mathcal{L}/(\alpha - \beta)$. This last step is incorrect because the statistics of average random path length under this difference model is complexer than assuming all random steps are the same. However, there is a definite expected value of path length with d differences, so the basis of the argument holds, although with a different value for β. When ϵ goes to 0, there is a very long slide from the starting point ρ. Each f.r. point not on the path should lag a lot behind the best f.r. point. As α increases as ϵ goes to 0, this explains further why the wave becomes very pointy and narrow.

The alignment finding algorithm ends because either the boundary of the edit graph is reached, or because all f.r. points have failed the regional alignment criterion, this means that the reads are probably not correlated anymore. In the second case, the best point in the last wave should not be reported as endpoint of the alignment, because the last columns could all be mismatches. Because the overall path should have an average correlation of $1 - 2\epsilon$, only a polished point with greatest anti-diagonal can be the end of a path. A *polished point* is a point for which the last $E \leq C$ columns are such that every suffix of the last E columns have a correlation of $1 - 2\epsilon$ or better, this is called being *suffix positive*. Daligner keep track of the polished f.r. point with the greatest anti-diagonal during the computation of the waves, it does so by testing if each bit-vector of the leading f.r. points is suffix positive. Testing bit-vector e can be done in $O(1)$ time by precomputing a table $SP[e]$ with 2^E elements. Define $Score(\emptyset) = 0$ and recursively $Score(1b) = Score(b) + \alpha$ and $Score(0b) = Score(b) - \beta$ where $\alpha = 2\epsilon$ and $\beta = 1 - 2\epsilon$. Note that if bit-vector b has m matches and d differences, then $Score(b) = \alpha m - \beta d$. If this is non-negative then $m/(m+d) \geq 1 - 2\epsilon$, which means b has a correlation of $1 - 2\epsilon$ or higher. Now let $SP[e] = min\{Score(b) : b \text{ is a suffix of } e\}$. The table SP can be built in linear time by computing $Score$ over the trie [168][169] of all E-bit vectors and taking the minimum of each path of the trie.

which last step of the argument is wrong exactly? Probably the formula of β

However for large values of E, say 30, the table SP gets too big. To solve this, a size D, say 15, is chosen for which the SP table is reasonable. Consider an E-bit vector e that consists of $X = E/D$, D-bit segments $e_X \cdot e_{X-1} \cdot ... \cdot e_1$. Precompute table SP as before, but now only for D bits, and a table SC for D-bit vectors as well, where $SC[b] = Score(b)$. With these two 2^D tables it takes $O(X)$ time to determine if the longer bit-vector e is suffix positive by calculating if $Polish(X)$ is true with the

following recurrences:

$$Score(x) = \begin{cases} Score(x-1) + SC[e_x] & \text{if } x \geq 1 \\ 0 & \text{if } x = 0 \end{cases} \tag{3.2}$$

$$Polish(x) = \begin{cases} Polish(x-1) \text{ and } Score(x-1) + SP[e_x] \geq 0 & \text{if } x \geq 1 \\ true & \text{if } x = 0 \end{cases} \tag{3.3}$$

3.3 Darwin

Darwin is a hardware-accelerated framework for genomic sequence alignment [170][37]. It consists of a filter called D-SOFT (Diagonal-band Seed Overlapping based Filtration Technique), which finds seeds, and GACT (Genome Alignment using Constant memory Traceback), which performs alignment between reads of arbitrary length, using constant traceback. The fact that GACT uses constant traceback makes it very suitable for hardware acceleration. Darwin boasts a 39.000x more energy-efficient approach then software, and a 15.000x speedup for reference-guided alignment for third generation reads using an ASIC. On an FPGA, Darwin has a 183.0x speedup for reference-guided and 19.9x speedup for denovo over state-of-the-art software.

3.3.1 D-SOFT

Like Daligner, D-SOFT filters by counting bases in Kmer matches. For each band of diagonals (bin), a particular threshold must be exceeded to constitute a seed hit, thus invoking the GACT algorithm. When using D-SOFT to do denovo alignment, each read is padded with unknown nucleotides (N), so that each read starts at the start of a bin. Figure 3.4 illustrates the working of D-SOFT for k = 4, N = 10, h = 8, N_B = 6. N is the number of Kmer matches that are considered, h is the minimum number of unique bases that the Kmer matches in a particular bin must have, N_B is the number of bins. The start positions of the matches are marked by red dots, the rest of the seed is a red line. Bin 1 contains six consecutive matching bases, which form three Kmer matches. Bin 3 also contains three Kmer matches, but they contain nine matching bases, for h = 8, only bin 3 produces a seed. As oppose to Daligner, D-SOFT only considers the Kmer matches from one bin, not taking its direct neighbours into account.

First, a lookup table for the Kmer all Kmers and their positions is made. This can be done with any implementation, like seed position tables, suffix arrays [133] or BWT [132] based indices.

D-SOFT uses two arrays, *bp_count* and *last_hit_pos*. Both have N_B values, and store the number of bases in Kmer matches, and the last seed hit position for that bin, respectively.

A newer version of Darwin uses minimizers [171]. This technique reduces the amount of storage needed to store seeds, at the cost of some sensitivity. The window size (w) is the most important parameter of the minimizers, and must be smaller than the Kmer size. Whenever this window size is not mentioned, it is equal to one.

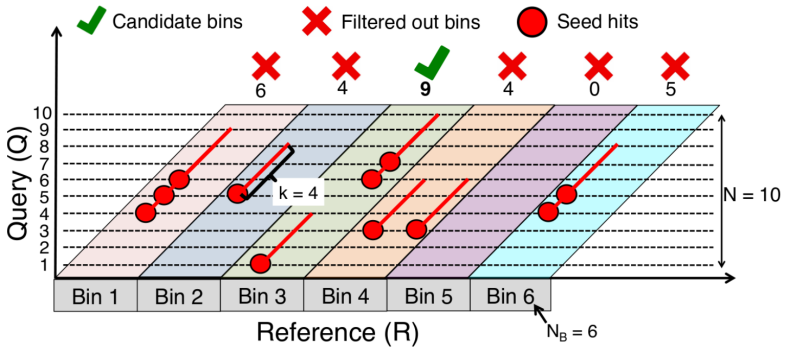

Figure 3.4: Illustration of D-SOFT algorithm, source: [37]

3.3.2 GACT

Normal Smith-Waterman will take too much memory and time for long reads, because the complexity is quadratic for the length of the reads. Numerous efforts to accelerate Smith-Waterman have been made, both by using hardware and by using heuristics. Even when using heuristics, the memory requirements are still linear. GACT finds the alignment between reads of arbitrary length using constant traceback memory. It performs normal Smith-Waterman on a submatrix called a tile of size T, and then moves to the next tile, which overlaps with the previous tile with O bases. For reasonable values for T and O, GACT produces an optimal result.

name accelerated variants here, or in Chapter 2 Background?

The algorithm for left extension is shown in Listing 3.4. Positions i_curr and j_curr are produced by D-SOFT. The start and end position of the current tile are stored in i, j_start and i, j_curr respectively. The traceback path of the whole left extension is kept in tb_left. The function $Align()$ uses modified Smith-Waterman to compute the optimal alignment between subsequences R_tile and Q_tile, with traceback starting from the bottom-right cell, except for the first tile, where traceback starts from the highest-scoring cell. $Align()$ returns the tile's alignment score, the number of bases in R and Q aligned by this tile (i, j_off), the traceback pointers tb and the position of the highest-scoring cell (ignored except for the first cell). $Align()$ also limits i, j_off to at most T - O bases, to ensure the next tile overlaps by at least O bases. The left extension finishes when it hits the end of R or Q, or when traceback cannot add any bases to the existing alignment. The memory needed for the traceback is $O(T^2)$, which is constant since T is chosen up front. The traceback for the whole alignment tb_left is linear with read length, but not performance critical. GACT calculates the full alignment strings and uses these to calculate the score of the whole alignment. The right extension operates on the reverse of R and Q.

Listing 3.4: GACT

```
1  tb_left = []
2  (i_curr, j_curr) = (i_seed, j_seed)
3  t = 1
4  while ((i_curr > 0) and (j_curr > 0)) do
5      (i_start, j_start) = (max(0, i_curr - T), max(0, j_curr - T)
```

```
        )
6       (R_tile, Q_tile) = (R[i_start : i_curr], Q[i_start : i_curr
            ])
7       (TS, i_off, j_off, i_max, j_max, tb) = Align(R_tile, Q_tile,
            t, T-O)
8       tb_left.prepend(tb)
9       if (t == 1) then
10          (i_curr, j_curr) = (i_max, j_max)
11          t = 0
12      if ((i_off == 0) and (j_off == 0)) then
13          break;
14      else
15          (i_curr, j_curr) = (i_curr - i_off, j_curr - j_off)
16  return (i_max, j_max, tb_left)
```

Figure 3.5 shows a left extension from GACT, using T = 4 and O = 1. For the first tile (T1), traceback starts at the highest-scoring cell (green), the other tiles start at the bottom-right cell (yellow), where the traceback from the previous tile ends. The traceback is ended earlier for higher overlaps, so that the tiles will overlap.

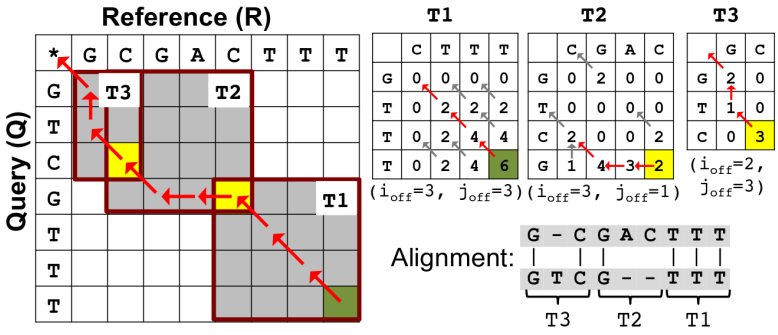

Figure 3.5: Illustration of GACT algorithm, source: [37]

The performance of GACT is linear ($O(max(m,n) \cdot T)$) with respect to the length of the reads. It is more suited to long reads than banded alignment, because banded alignment uses a static band around the main diagonal. GACT allows for flexible bands, since the position of the new tile depends on the traceback path, this is useful for long reads that have high indel rates.

flexible bands are better when indels are present, prove/cite that it does not matter when only having substitutions?

4 Specification

MAIN IS TRUE

4.1 Profiling

4.1.1 Daligner

Gprof [172] was used to profile Daligner, the function Local_Alignment() was found to take between 90% and 95% of the runtime. An arbitrary 130MB dataset from the Human 54x dataset from PacBio [173] has been used, the first 50MB of those files are used as a smaller dataset, this will be referred to as 'the 50MB dataset'. Since Local_Alignment() is the most time-consuming function, that will be the first one the be considered for acceleration. Each Local_Alignment() call originates from a particular readpair A, B, so executing multiple Local_Alignment() instances from different readpairs causes no datahazards. This is why Daligner is able to use multiple threads, each one running the filter-align function report_thread() on its own list of readpairs. The Local_Alignment() function consists mainly of forward_wave() and reverse_wave(), which implement the snakefinding algorithm in both directions respectively. Their structure is very similar, but combining them into one function would be difficult.

4.1.2 Darwin

Darwin uses gettimeofday [174] to measure the runtime of various elements of the algorithm. Three notable parts are building the seedposition table, finding the seeds, and aligning usign GACT. Of those three, aligning takes the most time, namely 99.9% for a small PacBio human denovo alignment, and 99.4% for a small simulated PacBio E.coli reference based alignment. GACT is also the part that has been accelerated on FPGA by the original creators because of the constant memory requirement. This is why GACT will be implemented on GPU.

4.2 Statistics

4.2.1 Daligner

Numerous metrics have been measured to characterize the process performed by Daligner. The 130MB dataset was used. Tables 4.1 to 4.7 show the results. A column with 75% indicates that 75% of the observed values is less than or equal to the listed value.

Benchmarks show that not trimming the width of the wave slows down the runtime by about 8x.

Table 4.1: Width of the d-wave, the last two rows indicate the width if it was not trimmed

	max	median
forward	190	24
reverse	189	28
forward*	13522	428
reverse*	13530	214

Table 4.2: Number of waves

	max	median	75%	90%	99%	99.9%
forward	5887	153	309	573	2693	2761
reverse	5458	103	193	359	1083	2156

Table 4.3: Length of snake

	max	median	75%	90%	99%	99.9%
forward	87	0	0	1	5	9
reverse	78	0	0	1	5	9

Table 4.4: Number of Kmer matches per readpair

max	median	75%	99%
32733	4	6	34

Table 4.5: Max drift (|initial diag - observed diag|)

	max	median
forward	1938	41
reverse	1323	37

Table 4.6: Number of skippable LAcalls per performed LAcall

max	median	90%	99%
18384	2	15	70

Table 4.7: Length of overlap

	max	median	75%	90%	99%	99.9%
a	27240	627	1341	2493	6380	10726
b	26465	629	1345	2502	6394	10755

The total number of LAcalls is 8x more than the number of performed LAcalls. This means the average number of skippable LAcalls per performed LAcalls is about 7.

The number of times each base is read is relatively low, between one and five. This means caching the bases somewhere might not give a whole lot of speedup, but it could help nonetheless. To cache 95% of the accesses, about 40 bases must be cached for both a- and bread. When compressed, this would take three integers per thread, for both

reads.

4.2.2 Darwin

Darwin has a different filtering algorithm, which results in less skippable LAcalls/GACTcalls. The 130MB dataset generates 820499 calls, of which 97% belong to a unique readpair. Skipping GACTcalls would remove at most 3%.

The default tilesize is 320. If some GPU threads have a smaller tilesize, this will cause divergence, because they will have to wait until the threads with larger tilesizes are finished. For denovo alignment, 63% of the tilesizes where 320x320. For reference-based alignment, this percentage is 79%.

The traceback is done inside the alignment function. All used tracebackpointers (max 2*(T-O)) are returned. For denovo, 74% of the traceback results had the maximum of 200 bases in either direction, this is 84% for reference-based alignment.

4.3 Acceleration

4.3.1 Daligner

The individual snakes from a d-wave are independent, they can be calculated in parallel. However, the width of a trimmed d-wave is usually lower than 32, this means a warp cannot be fully utilised if it is assigned a single readpair. An option is to assign two readpairs to a block, each readpair would then have 16 threads. This means that the half-warp is fully utilised more often, but when the width of the d-wave is larger than 16, there well be idle threads.

For this reason, coarse-grained parallelism is chosen to be implemented. The alignment and filter functions are separated, instead, filter() produces a large list of seed hits (LAcalls), which should be send to the local alignment function. An advantage of having alignment and filtering intertwined is that Kmer matches that are included in previous alignments can be skipped, Figure **??** illustrates this. This advantage is lost when generated the large list, but can be reimplemented using a scheduler that keeps track of the furthest position that was reached in an alignment for each readpair. Each Kmer match that has enough neighbouring Kmer matches constitutes its own LAcall in the large list, but these are filtered out later. Each GPU thread is assigned a readpair, and each readpair can have multiple LAcalls. The host side will assign an LAcall to each GPU thread (unless there are not enough readpairs). The GPU performs the local alignment, and returns the results. If a readpair has no unskippable LAcalls left, its GPU thread is assigned a new readpair.

Coalesce main data arrays (WORK) The main data of a wave is stored in a few arrays. These arrays are not interleaved when used by the CPU, since each thread has its own cache. On the GPU, they need to be interleaved to allow for coalesced reads.

Cache variables (CVARS) The alignment kernel of Daligner is quite complex and contains many variables. This results in a large register usage per thread. To try and

Table 4.8: The minimum and maximum values for different variables.

	min	max
V	-1	63225
M	0	61
H	-1	4677
N	-100	34500

combat this, a few often used variables can be placed in the unused shared memory.

Encoding main data arrays (CWORK) Each variable in a main data array is a 32-bit integer. Not all of these values need to use all of those bits. Table 4.8 shows the possible values that the arrays can hold. Figure 4.1 shows the encoding. Care must be taken considering the signs of the values. H only contains non-negative value and -1. When placed in the least-significant (LS) part and read by masking out the most-significant part (MS), the -1 would give a positive value. Since checking each time for -1 is inefficient, it could be put in the MS part. However, N contains negative values as well, it starts at -s (s is 100 by default). Storing N in the LS part would mean reading and writing a negative value that is unknown at compile time. It it easier to give H an offset of 1, so that the stored value is always non-negative. Since N can still be negative, it is put in the MS part, where shifting right is possible for both positive and negative values. M is always non-negative, but V can contain -1, therefore V is put in the MS part, and M in the LS part. T contains a bitvector that represents the history of the last C (default = 60) bases, based on this, the number of matches is updated. T cannot be easily reduced in size without reducing C, although M can be removed by calculating the popcount of T when it is needed. CWORK reduces the memory per diagonal per thread from 8 ints to 5 ints.

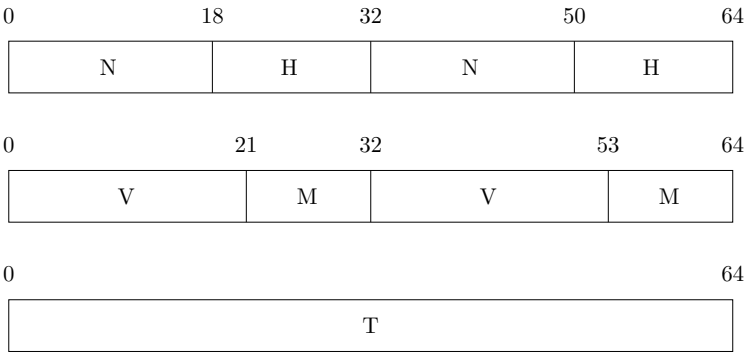

Figure 4.1: Encoding of data.

Reduce number of tracked columns (B30) By default, the last 60 columns are tracked in a long integer. A certain percentage of these columns must have been a

match, according to the regional alignment quality criterion. By reducing the number of columns to 30, the output is slightly changed, but the amount of required memory is reduced by 20% (5B to 4B, when used in combination with CWORK).

Remove M (RM) The M array is stored in global memory, so reading and writing is relatively expensive. Its only purpose is to require a certain number of matches in the last C (60) match/mismatches, but it is only checked if the V value is larger than the highest V value for the current wave. Since it is used infrequently, an option is to calculate the M when it is needed, based on the T vector. This can be done using the popcount instruction, which is available as an intrinsic instruction in CUDA (__popc()) [175].

No pebbles (NP) Pebbles are checkpoints in the alignment to allow for easier recalculation of the full alignment when needed. Shorter alignments do not need these, but long reads create long alignments. However, since Falcon does not use them [176], they could be removed. This means some arrays and loops can be removed.

Cache main diagonals in variables (CAWORK) Inside each wave, multiple diagonals have to be explored. Some data is propagated between diagonals, since the possible previous diagonals they read from, overlaps. This is usually done via the main data arrays in global memory, but since that could be slow, it could be worthwhile to store some data in registers, so that they can be used in the next diagonal. This can be combined wih encoding these data arrays (CWORK).

Cache main diagonals in shared memory (DIAGSX) Since caching in registers is relatively expensive (registers are sparse), caching in shared memory might be a better idea. An SMX of compute capability 3.x has 64KB of on-chip memory that can be divided among L1 cache and shared memory. There are three settings: 48KB shared memory / 16KB L1 cache, 32KB shared memory / 32KB L1 cache, 16KB shared memory / 48KB L1 cache [177]. Shared memory is designed to be shared among all threads in a threadblock, but can be used as local memory, when carefully calculating the addresses. To avoid bank conflicts, the data can be stored as shown in Figure 4.2.

Since shared memory is expensive, this technique can be combined with RM and NP to reduce the memory requirements per diagonal.

Diagonals are cached per $2^n + 1$, because the diagonal in the last cached position must be moved to the first one, and detecting if the current diagonal is a factor 2^n away from the first one in the wave is easy. The diagonals are recached every 2^n diagonals. Possible values include 5, 9 and 17. An example for 5 cached diagonals is listed below:

- wave spans diagonals 20-29
- cache diagonals 20-24 in shared memory
- compute diagonals 20-23

b0		b1		b2		b3		···
0 $B_0[0]$		8 $B_1[0]$		16 $B_2[0]$		24		
256 $B_{32}[0]$		$B_{33}[0]$						
512 $B_{64}[0]$								
768 $B_{96}[0]$								
1024 $B_0[1]$		$B_1[1]$						
1280 $B_{32}[1]$								
$B_{64}[1]$								
$B_{96}[1]$								
⋮								
B_{96}[N-1]								
$V_0[0]$	$M_0[0]$							
⋮	⋮							
V_{96}[N-1]	M_{96}[N-1]							
$HA_0[0]$	$HB_0[0]$							
⋮	⋮							
HA_{96}[N-1]	HB_{96}[N-1]							

Figure 4.2: Encoding of data in shared memory, NA and NB are not shown. A name $X_n[m]$ indicates thread n, array X, index m.

- when current diagonal is 24, writeback cached diagonals to global memory, move diagonal 24 to position 0 in the cache, and cache diagonals 25-28

- compute diagonals 24-27

A diagonal normally takes 8 ints. CWORK reduces that to 5 ints. RM removes the need to store M, which saves one int, or zero when CWORK is active. NP saves 4 ints, or 2 ints when CWORK is active. The minimum space need is 3 ints per diagonal per thread, assuming 60 columns are tracked. This can be further reduced to 2 ints if B30 is used.

XOR to find snakelength (CABSEQ) The CPU version uses one byte per base, and each base is retrieved on its own. Since each base only needs two bits, four bases can be encoded into one byte, or sixteen bases into one integer. The actual comparison is done via the XOR operation, matching bases result in zeros. By counting the leading zeros via the intrinsic __clz() [175] and dividing by two, the snakelength can be determined. If the result contains 32 zeros, the next integers must also be checked. The position that needs to be checked might not be divisible by eight, in this case, the integers that hold the encoded bases must be shifted and masked to prevent comparison between sequences of unequal length. Another option is to have 16 copies of the encoded bases, so that there is always an integer available that starts at the correct position, eliminating the need for shifting and masking, but it costs 16x more memory, and shifts and masks are relatively cheap on a GPU. The T vector is filled with the appropriate amount of ones afterwards.

Cache bases (CAABSEQ) The bases are reused a few times, if the loads are expensive, caching them can save some time. This can also be combined with CABSEQ. Since the bases are constant, they might be put into constant memory. However, this would serialize all accesses since each tread requires a base from its own read. Texture memory is also an option, since the bases can be interleaved to try and get some spatial locality. But shared memory is faster, is large enough and allows for bank-conflict free reading, so this is the memory where the caching is implemented.

Inline PTX The CUDA language provides a parallel thread execution (PTX) instruction set architecture (ISA) [178][179]. This is similar to writing inline assembly. An example of where this could be useful is for conditions. Instructions like 'if' or 'for' can cause divergence, making part of the warp wait for the other, increasing execution time. Sometimes, a loop can be unrolled. According to section 5.4.2 Control Flow Instruction in the programming guide[31], an 'if' can be translated to predicated instructions instead. These are instructions whose execution depends on the value of the predicate, which is either true o false. The compiler decides if the number of instructions controlled by the condition is small enough. In an earlier version, that number used to be 7, if the compiler determines that the condition is likely to produce many divergent warps, and 4 when otherwise [180]. The current programming guide does not state a number.

There are multiple places where this technique was used: - replacing 'if(__clz(a1) == 32)abpos -= max;' in combination with CABSEQ - finding which of three values is the max - loading data based on the max of three values - keeping track of furthest reaching point in the current wave

Maxrregcount = 64 The number of register assigned to a thread can be limited. This can increase the number of active threads on an SMX, since the number of registers on an SMX is also limited. The tradeoff lies with the reduced capabilities of the thread, the data that was stored in those registers now spills to local memory. The optimal registercount was found to be 64.

Streaming CUDA functions (STREAM) Certain CUDA functions, like copying data from or to the device, and kernels, can be executed at the same time if different streams are used. This function only matters if multiple CPU threads are used.

4.3.1.1 Splitting the alignment between GPU and CPU

There are other ways of using the GPU, the implementations GPU_SINGLE, GPU_SNAKE and COMPRESS_ABSEQ try to do that. GPU_SINGLE lets the GPU compute a large bitmatrix, where 0 means a mismatch, and 1 means a match. These bits will be read by the CPU, instead of having the CPU do the comparisons itself. GPU_SNAKE computes the whole snake on the GPU, and stores it at some index that is based on the positions in reads a and b. When the CPU wants to know the length of a snake during alignment, it only has to look up the length in the array. COMPRESS_ABSEQ calculates the XORred bitvector between integers with compressed bases like described above, but the results returned to the CPU. During alignment on the CPU, it only has to read the correct bitvector, and count the leading zeros.

4.3.1.2 CPU acceleration

Using the XOR operation on compressed bases (CABSEQ) can also be implemented on the CPU. For this to work, the 'count leading/trailing zeros (clz/ctz)' and popcount operations have to be implemented. GCC has builtin functions for these operations [181]. The clz and ctz operations return undefined values when the input is zero, so that case must be handled differently.

4.3.2 Darwin

maybe refer to pseudocode?

It is possible to run the whole GACT kernel on the GPU, for both left and right extension. However, since it is not known how long the resulting alignment will be, and all GPU threads have to wait until all threads are done, this will cause lots of idle time. Instead, it is chosen to only have a single tile aligned per GPU-thread per GPU-invocation. The scheduling and preparation of the tiles will be done by the CPU. Tracking the full alignment in two strings is also done on the CPU, they are used to calculate the total alignment score. The CPU will receive a large list with seed hit positions, for each hit, the GACT algorithm is performed. It is not necessary to implement skipping seeds/Kmer matches for Darwin, since it generates less redundant seeds than Daligner.

cite wavefront parallelism?

It is also possible to implement a more fine-grained parallelism, and assign multiple threads to the same tile using wavefront parallelism The tilesize of 320 should be enough to allow for enough utilization. This method uses less memory, since the Smith-Waterman matrices and the sequences are shared. However, this complicates the

implementation, since the number of elements on an anti-diagonal is often not equal to 32. Inter-thread communication or scheduling is needed to efficiently use the hardware. This is not the case when using coarse-grained parallelism. Since the tiles are almost always the same size (it only differs when aligning the end of a sequence), the exact operations that need to be executed are the same for all threads. This requires enough seed hits to keep every processing element busy, but the 130MB dataset produces two lists of 400k seeds using default settings. Besides, the memory used by a single thread is not very large, since the tiles that need to be aligned are usually at most 320 bases.

GACT is designed to allow for an affine gap penalty, so it uses three matrices to keep track of the scores. The whole tile is calculated per column, so only two columns need to be kept from the matrices: the current and the previous. This means the implementation uses $O(T)$ memory for the score, where T is the tilesize. The traceback pointers need to be kept for the whole tile, this takes $O(T^2)$.

Coalesce matrices (CMAT) Since the matrices and traceback pointers are read-/written for every score element, it makes sense to coalesce these, to optimize the memory bandwidth and reduce the number of memory transactions. There are four matrices with score data, which cause 14*T*T transactions per thread. Swapping the current and previous column takes 8*T*T transactions per thread. Initialization takes 8*T transactions per thread. The traceback pointers need T*T transactions for filling and 2*T-1 for initialization per thread. Computing the traceback path takes at most 2*T transactions, but these cannot be coalesced, because each thread can have a different path. After coalescing, a warp of 32 threads will have reduced the needed transactions by 32. Each value in these matrices is an integer.

GASAL (GASAL) GASAL [182] provides an API for GPU accelerated functions that perform different types of genomic alignment, such as local, global and semi-global. It can perform one-to-one, one-to-many and all-to-all alignment. GASAL first packs the bases in a 4-bit format, where 8 bases are packed into a 32-bit integer. The integers are padded with 'N' nucleotides, these do not affect the score when aligned. The packed bases do not have to be copied back to the CPU, because they are used during alignment. The second phase is the actual alignment, since there are 8 bases in one integer, the matrix is divided into 8x8 tiles. Figures 4.3 and 4.4 show the order of the tiles and elements in which they are calculated.

Instead of keeping two whole columns of the matrices in memory, it keeps two rows of 8 elements (the width of a tile) as working memory. This way, a whole column of 8-by-8 tiles can be calculated. Because the elements of the next column need the scores of the last, an array of scores is kept for the vertical column. It does not provide traceback information, when the start position of an alignment is requested, it performs the alignment again, but in reverse.

GASAL needs to be adapted to perform actual traceback, to guarantee that the tiles of GACT have a minimum overlap. It is chosen to perform the actual traceback on the GPU, instead of copying all the pointers to the CPU, and doing the traceback there.

Since the order of accesses is slightly different between the CPU version and GASAL, the reported coordinates of the element with the maximum score could differ.

build 0.18

this choice might seem odd, since traceback on GPU is usually very divergent,

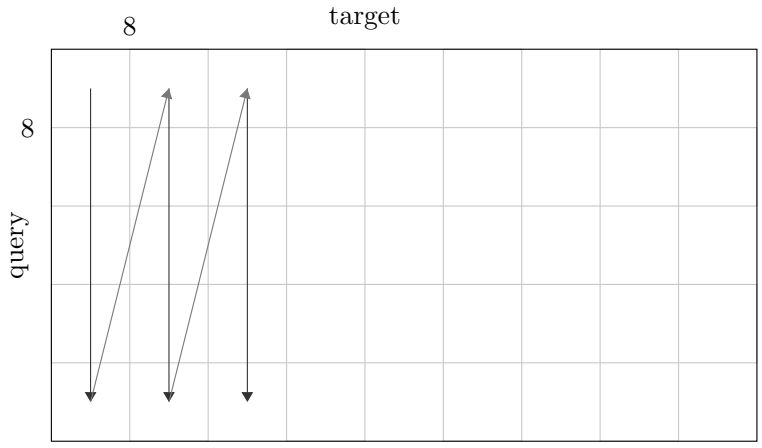

Figure 4.3: Illustration of GASAL algorithm, high level.

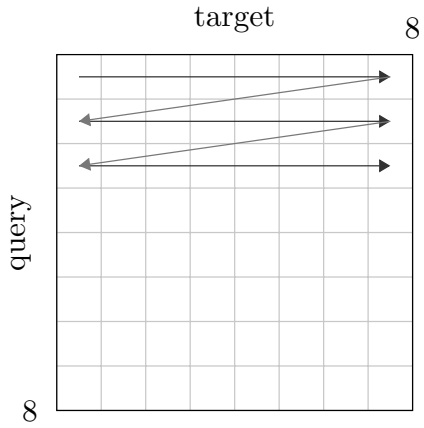

Figure 4.4: Illustration of GASAL algorithm, within an 8-by-8 tile.

Coalescing the returned, used tracebackpointers The CPU needs to know what path the traceback took to compute the full aligned strings. While following the trace, the GPU writes the used pointers to an output buffer uncoalesced. This buffer could be coalesced, but the tracebackprocedure takes only a fraction of the GPU time, so this is not implemented.

No score (NOSCORE) If the score of the alignment is not needed, the used traceback pointers do not have to returned to the CPU. It also saves the time spend on building the aligned strings on the CPU.

Coalesce global matrix (GLOBAL) The score matrices in GASAL are small enough to fit in the registers, and cannot be coalesced. There are two main matri-

ces left: the global array that stores the scores of one column, and the tracebackpointer matrix. The global array actually stores the scores of the three matrices needed to allow for an affine gap penalty, coalescing them means that they have to be separated. The scores are stored via 16-bit shorts, but since the smallest memory transaction is 32 bytes (Compute Capability 3.5) [31], and the uninterleaved elements are far enough, it causes unnecessary transactions too.

Coalesce packed bases (CPBASES) The bases are uninterleaved when they are send to the GPU, and after packing. This means that during alignment, they cause uncoalesced reads. To prevent this, the writing of the packed bases can interleave them, but this means the packing kernel has write in an uncoalesced manner. When preparing the bases on the CPU, they are interleaved, so that the packing kernel can read and write coalesced, and the alignment kernel can read coalesced too.

Inline PTX This technique was used to keep track of the maximum score of the tile, and the coordinates of that element. It was also used to find the maximum of four values in the alignment kernel.

Maxrregcount = 64 As with Daligner, the optimal registercount was found to be 64.

Streaming CUDA functions (STREAM) Same as for Daligner.

5 Results

MAIN IS TRUE

5.1 Hardware

Tests on two systems were done: TACC and ce-cuda.

TACC nodes have an IBM Power8 S824L, 256 GB RAM and a Tesla K40m. The CUDA version for TACC is 7.5.

ce-cuda has an Intel Xeon E5-2620 @ 2.4 GHz, 32 GB RAM and a Tesla K40c. The clocks on the K40 were 745 MHz and 3004 MHz for core and memory respectively. The CUDA version for ce-cuda is 9.2.

The K40c and K40m have the same performance, the only difference lies in their cooling solution [183].

5.2 Daligner

5.2.1 Runtimes

The different runtime configurations are indicated with A B C, where A is the number of CPU threads, B the number of GPU blocks that each CPU thread launches, and C the number of GPU threads in each block. Table 5.1 shows the results. The first column lists the used optimizations.

Replacing 'if(__clz(a1) == 32)abpos -= max;' with inline PTX gave a 4% speedup. The others did not give a consistent speedup.

5.2.2 Profiling

Nvprof [184] was used to profile the experiments.

For 3000 bp reads from a custom PacBio generator:

- CWORK reduces the L2 cache misses by 40%, increases the global load throughput by 35%, and the number of global load transactions by 12%.
- CABSEQ reduces the number of global load transactions by 15%, and increases the global load efficiency from 11.8% to 14.2%.

For 10000 bp uniform reads:

- DIAGS5 reduces global memory traffic by 25-30%, and increases the number of executed instructions by 35-40%.
- B30 reduced global memory traffic by 28%.

Table 5.1: Runtimes for different Daligner optimizations, with 3000 bp custom PacBio reads, run with 8 64 64, note that B30 and NP change the output.

	runtime (s)
base	75.1
CWORK	67.1
CAWORK	75.1
CABSEQ	71.8
CWORK CABSEQ	61.2
CWORK CABSEQ STREAM	46.6
CABSEQ STREAM	53.1
STREAM	54.8
CWORK STREAM	48.7
CABSEQ CAABSEQ CWORK STREAM	58.7
B30 STREAM	55.0
B30 RM STREAM	49.7
RM STREAM	51.0
B30 CWORK CABSEQ STREAM	46.1
RM NP CWORK CABSEQ STREAM WORK	27.8
RM NP CWORK CABSEQ STREAM WORK B30	27.2
RM NP STREAM CABSEQ WORK	27.5
RM NP STREAM CABSEQ WORK B30	26.7
DIAGS5 RM NP STREAM CABSEQ WORK	25.0
DIAGS9 RM NP STREAM CABSEQ WORK	25.4
DIAGS17 RM NP STREAM CABSEQ WORK	33.3
DIAGS5 RM NP STREAM CABSEQ WORK B30	24.0
DIAGS9 RM NP STREAM CABSEQ WORK B30	23.8
DIAGS17 RM NP STREAM CABSEQ WORK B30	28.1

Table 5.2: Divergence counteracts the effect of optimizations

type of reads	implementation	runtime (s)
uniform reads	STREAM	35
	STREAM WORK CWORK CABSEQ RM NP	6
3000 bp custom PacBio reads	STREAM	20
	STREAM WORK CWORK CABSEQ RM NP	9

Table 5.3: Runtimes for different Daligner parameters, one CPU thread, run on ce-cuda, 10x E.coli, denovo. Sensitivity and specificity are measured for A vs A, using a score threshold of 200.

parameters	runtime A vs A (s)	runtime A vs B	sensitivity	specificity
default	32.5	61.0	99.78	86.7
h45	31.5	57.6	99.58	86.8
k15	31.6	56.9	98.9	87.1
k16	29.4	55.2	96.9	87.5

Table 5.4: Runtimes and speedup for different implementations, all variantions have STREAM, run on a 2x E.coli dataset with 8 16 32 on TACC.

implementation	runtime (s)	speedup
baseline	82	-
BATCH	87	0.94
GPU	279	0.29
GPU CBASES	269	0.30
GPU CMAT 83	0.99	
GPU CMAT CBASES	82	1.00
GASAL	22	3.7
GASAL CMAT	10	8.2
GASAL CMAT CPBASES	11	7.5

Table 5.5: Runtimes and speedup for different implementations, N = 800, run on the 50MB dataset with 8 32 64 on TACC.

implementation	runtime	speedup vs CPU 64
CPU 8 threads	64m35	-
CPU 32 threads	29m54	-
CPU 64 threads	27m56	-
BATCH 32 8 64	37m17	0.75
GPU CMAT	14m20	1.95
GPU CMAT CBASES	11m28	2.44
GASAL	11m40	2.39
GASAL CMAT	1m17	21.8
GASAL CMAT CPBASES	1m9	23.0
GASAL CMAT CPBASES NOSCORE	58s0	28.9

- ABSEQ reduced the number of global load transactions by 80%, increased the efficiency of global loads from 41% to 91%, reduced the number of L2 cache reads by 60%, and L1 global loads by 80%.

For uniform reads, the optimizations can reach global read and store efficiencies of 92% and 96% respectively, for fixed length simulated PacBio reads, this effect diminishes greatly. The result of the divergence can also be observed in the runtimes in Table 5.2

5.3 Darwin

5.3.1 Runtimes

Darwin has the same run configuration, A B C, where A is the number of CPU threads, B the number of GPU blocks, and C the number of threads in a block. Table 5.4 shows the runtimes for CPU version with 8 threads, and the fastest GPU configuration.

Table 5.6: Runtimes and speedup for different implementations, N = 800, run on the 50MB dataset with 8 32 64 on ce-cuda.

implementation	runtime	speedup vs CPU 8
CPU 8 threads	138m21	-
GPU CMAT CBASES	11m26	12.1
GASAL	11m46	11.8
GASAL CMAT	1m19	105
GASAL CMAT CPBASES	1m16	109
GASAL CMAT CPBASES NOSCORE	56s1	148

Table 5.7: Runtimes for different Darwin parameters, 1 64 64, 10x E.coli, denovo, A vs A. Sensitivity and specificity are measured using thresholds of 600 and 990 for score and length respectively.

number of seeds (N)	runtime TACC (s)	runtime ce-cuda (s)	sensitivity	specificity
1400	105.6	62.8	99.33	85.0
1100	94.2	56.1	98.9	85.4
800	89.4	48.5	97.8	86.2

5.3.2 Profiling

Coalescing of the GLOBAL array and coalescing of the packed bases did improve the global load efficiency, as shown in Table 5.9.

The alignment kernel consists of two distinct phases: alignment and traceback. The kernel was split into two kernels, and their runtimes measured, the alignment part takes 96% of the time. This makes optimizing the traceback part not efficient.

The optimization CPBASES coalesces the packed GASAL bases. This requires a more complex preparation on the GPU size, which interleaves the unpacked bases. For the 50MB dataset, ran wih 1 256 64, the timing results are listed in Table 5.10. The preparation and packing take more time, but this is compensated by a faster alignment

Table 5.8: Runtimes for different Darwin run configurations, 50MB dataset, denovo, A vs A.

Run configuration	runtime TACC (s)
8 32 64	1m9
16 16 64	1m32
16 8 64	2m6
4 32 64	1m19
4 64 64	1m17
1 64 64	2m25
1 128 64	2m14
1 256 64	2m28
32 8 64	3m8

Table 5.9: Profile data for different optimizations, run on the 50MB dataset, with 1 64 64 threads.

optimizations	global ld efficiency
baseline	12%
GLOBAL	49%
GLOBAL CPBASES	73%
CPBASES	45%

Table 5.10: CPBASES saves some time in alignment, but adds preperation time.

	preperation (s)	packing (ms)	aligning (s)
base	4.2	54	41.1
CPBASES	6.0	96	38.9

phase.

Using inline PTX did not give consistent speedup for either implementation. This should not be a large surprise, modern compilers are quite good at creating efficient low-level code.

During a 240 second run of the 130MB dataset, 232 seconds (97%) was spent on GACT. This is shown in Table **??**.

A common performance metric for Smith-Waterman based solutions is CUPS, which stands for Cell Updates Per Second. For 10x denovo A vs A, the Darwin GPU implementation did 11.1 GCUPS. It did up to 13.3 GCUPS during the 130MB dataset, A vs A, and 13.5 for A vs B.

5.4 Sensitivity and specificity

The algorithms should ultimately be used to find overlaps, the sensitivity and specificity indicate the quality of the report output. Sensitivity and specificity are defined as:

$$\text{Sensivitity} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TP}{TP + FP}$$

where TP, FN and FP are the number of true positives, false negatives and false positives respectively. Sensivitity indicates how many of the overlaps that the aligner was supposed to find, were actually found. Specifitiy indicates how many of the reported overlaps were

Table 5.11: Percentage spent in GACT, run configuration: 1 64 128, N = 800, w = 4.

	total runtime (s)	GACT time (s)	percentage GACT
130MB A vs A	240	232	97
130MB A vs B	237	193	81

actually real overlaps. To boost sensitivity, if an overlap between A and B was found (denovo), the overlap between B and A is also assumed to be found.

PBSIM [185] is used to generate synthetic PacBio datasets. The only parameters that is not default is the accuracy, which is set to 0.85%, to mimic the 15% error rate of PacBio. The distribution is also changed: 1.5% substitution, 9.0% insertion, 4.5% deletion. The reads are generated from an E.coli reference. For reference-based alignment (mapping), a read that is aligned to the reference within 50 bases of the original location is a true positive. For denovo based alignment, a true positive must have an overlap of 1000 bases according to PBSIM, and be found by the aligner. The run configuration of Darwin consists of three numbers, the first denotes the number of CPU threads, the second the number of GPU blocks that each CPU thread launches, the thrid the number of GPU threads in each of those GPU blocks. Since Daligner is faster of CPU than on GPU, the CPU version is used, and the number indicates the number of CPU threads. For Darwin, h (default:21) indicates the number of non-overlapping bases that the Kmers must have to constitute a seed, n (1400) is the number of seeds that are considered for that readpair, and w (4) is the size of the window for the minimizers. The overlap length threshold is set to 990 during denovo alignment. For Daligner, h (35) means the same, l (1000) is the minimum length that an overlap must have to be reported. Both aligners can use a minimum score threshold to further filter the found overlaps. This will be denoted by s. The scoring scheme for Darwin is: match = 1, mismatch = -1, gap_open = -1, gap_extend = -1, unless noted otherwise. Daligner does not actually calculate the score, instead the score is defined here as the length of the overlap in a, minus the reported number of differences.

Ecoli ref was sent by Yatish, mention?

Darwin is not designed to compare two different files against each other, but a particular file against itself. This will be denoted by 'A vs A', if two different files are compared against each other, this is 'A vs B'.

Figure 5.1 shows the clear tradeoff between sensitivity and specificity for reference-based alignment. The exact sensitivities for l200 and l100 are 99.97% and 100.00% respectively. PBSIM generates reads with a minimum size of 100, this is why lowering the default minimum overlap length of 1000bp increases the sensitivity in reference-based alignment. If the input data is longer than that, or not aligning those shorter reads is acceptable, the setting for l can be higher. Figures 5.3 and 5.4 show a similar tradeoff between sensitivity and specificity for Darwin. While the default value for the number of seeds is 1400, this number can be lowered to 400 without noticable impact on the sensitivity. When going lower than 400, the sensitivity is reduced, but the runtime is also lower. The runtime and number of seeds have an affine relation, so trading runtime for sensitivity is easy. The specificity is higher when the number of seeds is low, it makes sense that producing less GACTcalls by reducing the number of seeds in the seedposition table reduces the number of false positives. However, the N = 1400 and 800 options have a higher specificity when also using a high score threshold. A tradeoff between sensitivity and specificity can also be observed from the figures.

Figures 5.5 and 5.6 show the same tradeoff as the others. The score thresholds below 200 give the same result, that is why part of the x-axis is missing. The output reacts quite strongly when changing the score threshold from 220 to 260. Larger Kmers result in lower sensitivity, and higher specificity, as can be expected.

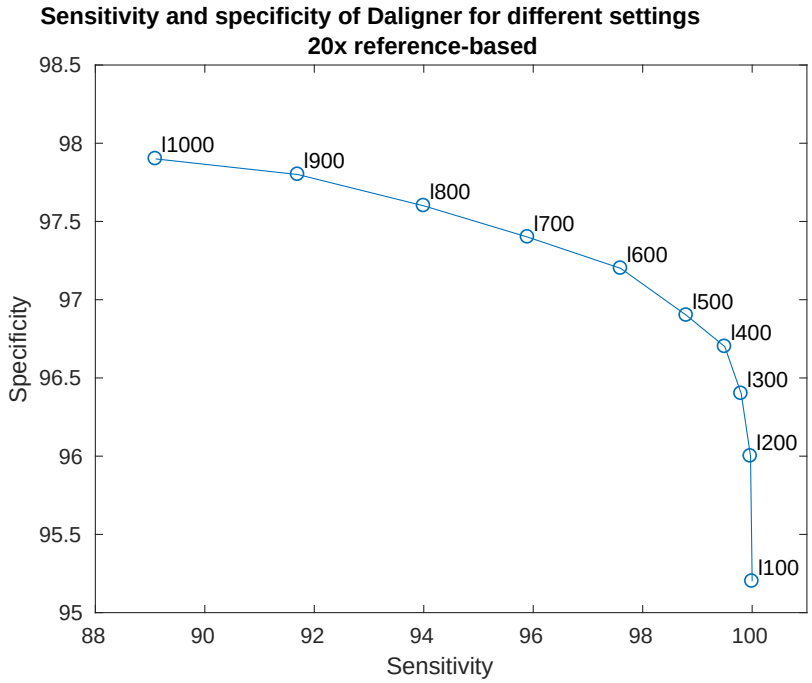

Figure 5.1: Sensitivity vs specificity tradeoff depends on the setting of l.

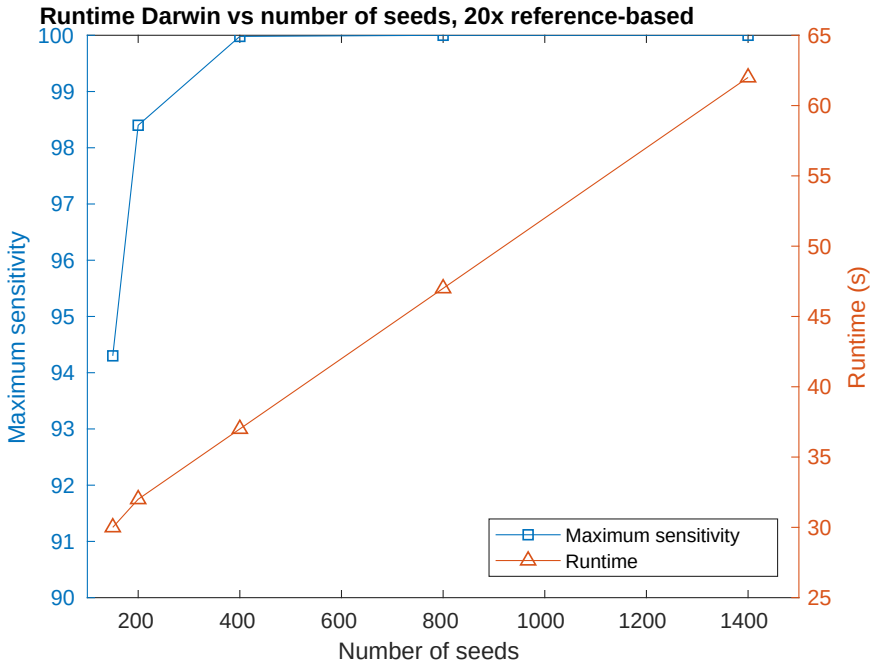

Figure 5.2: Sensitivity and runtime depend on number of seeds (N), w = 1.

Figure 5.7 combines the sensitivity and runtime, for different windows sizes. The sensitivity shows a peak when window size is four, so this is the default size for the

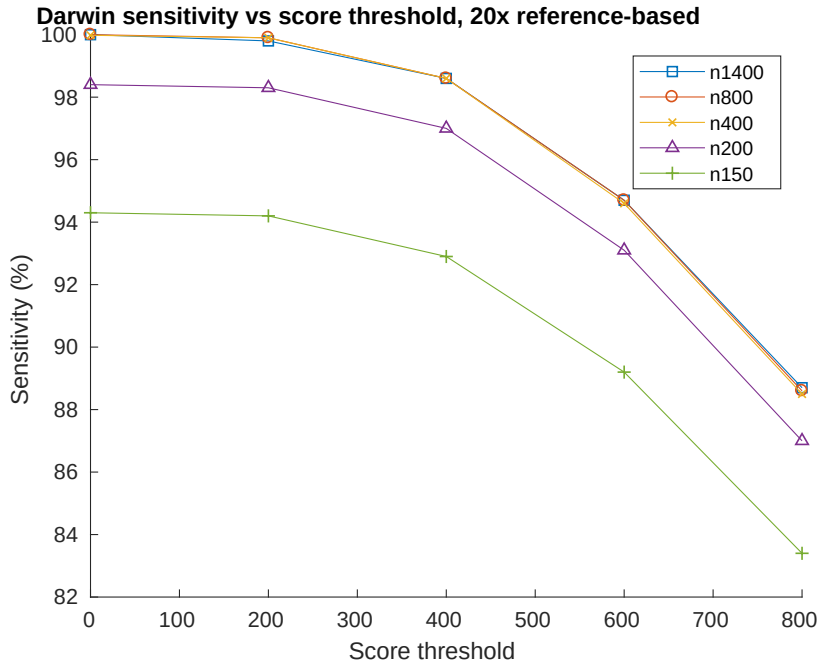

Figure 5.3: Sensitivity as a function of score threshold (s) and number of seeds (n), the runtimes are shown in Table 5.6, w = 1.

denovo experiments. For the last two experiments, the sensitivity remains constant between score thresholds 0 and 600, so part of the x-axis is missing again. The exact threshold where the sensitivity starts dropping will depend on the scoring scheme. It makes sense that there is such a threshold, since the theoretical overlaps have a minimum length of 1000 bp, which should produce a certain score given the scoring scheme and the distribution of errors.

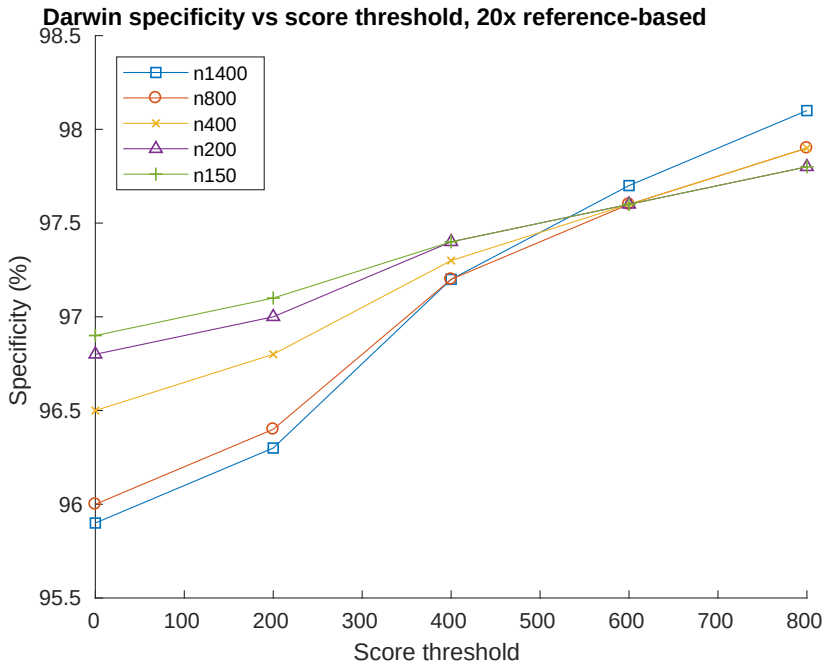

Figure 5.4: Specificity as a function of score threshold (s) and number of seeds (n), the runtimes are shown in Table 5.6, w = 1.

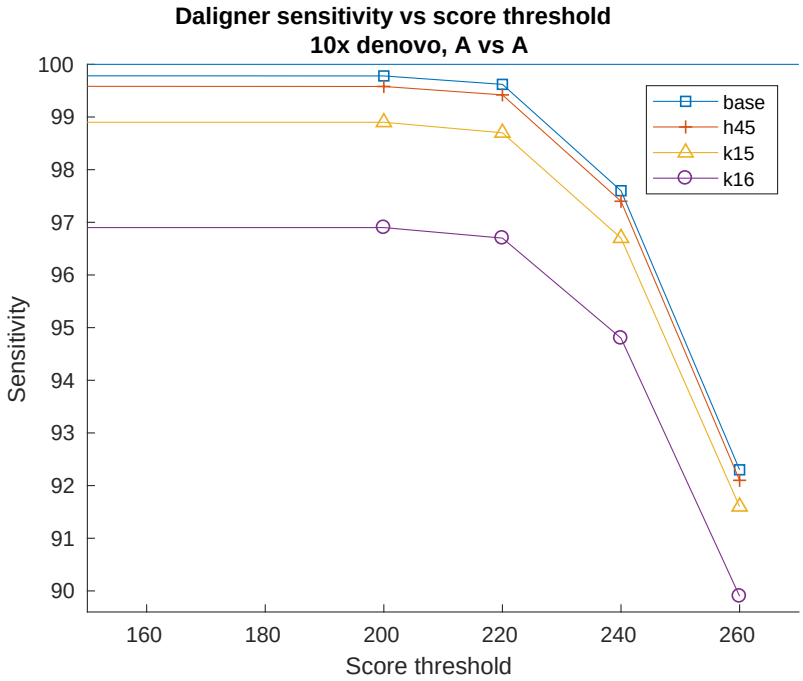

Figure 5.5: Sensitivity for different Daligner options, their runtimes are listed in Table 5.3

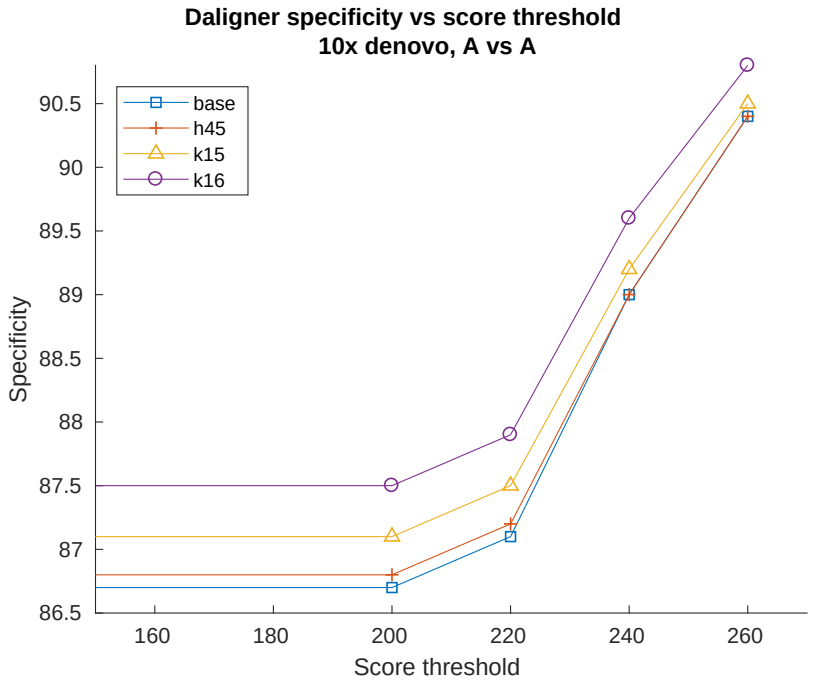

Figure 5.6: Specificity for different Daligner options, their runtimes are listed in Table 5.3

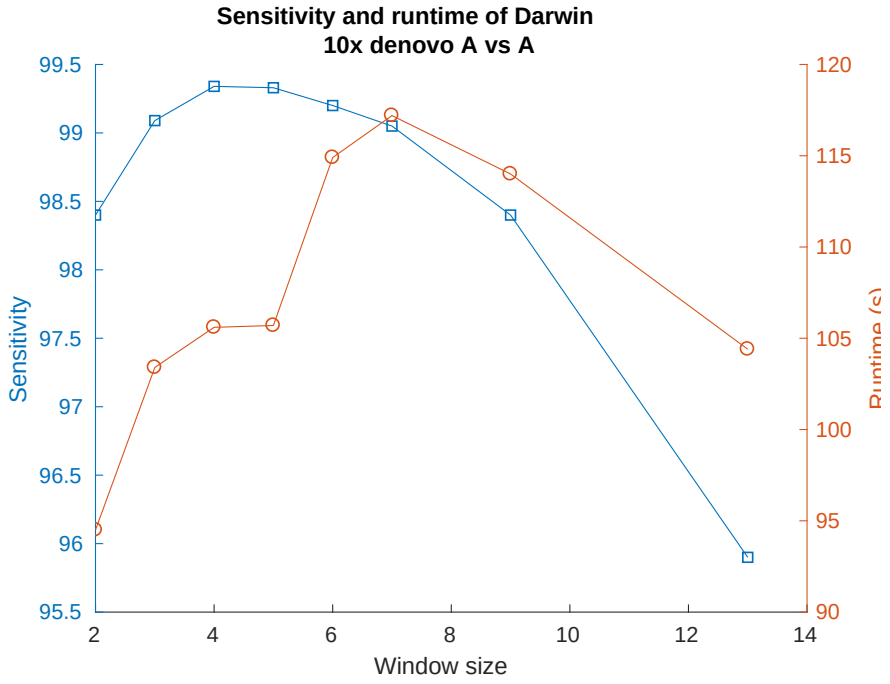

Figure 5.7: Runtime and sensitivity for different window sizes (w).

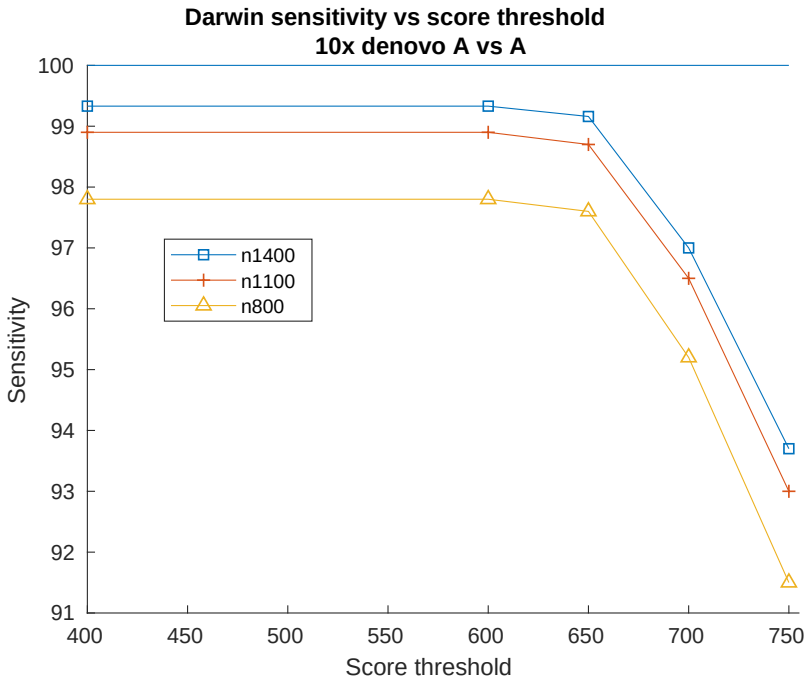

Figure 5.8: Sensitivity as a function of score threshold (s) and number of seeds (n), w = 4.

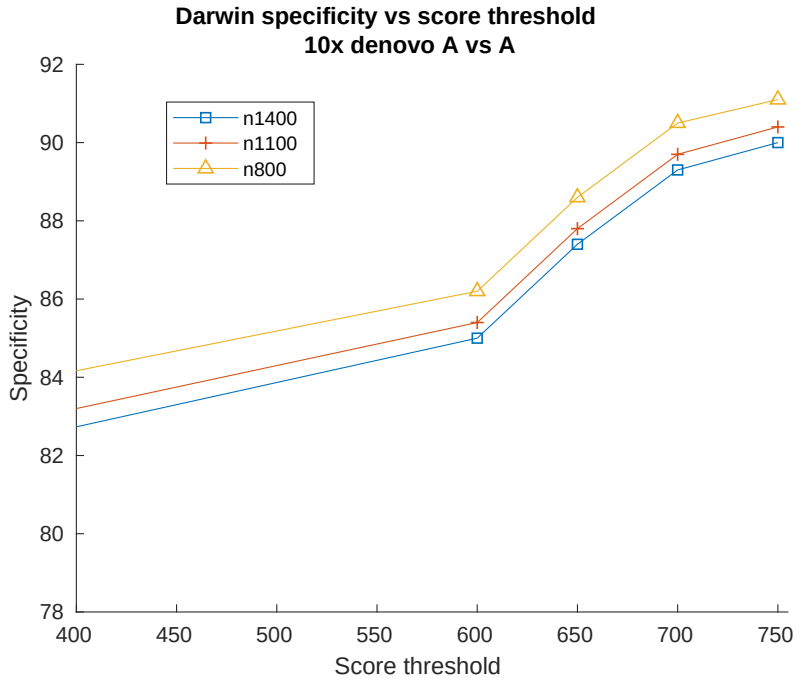

Figure 5.9: Specificity as a function of score threshold (s) and number of seeds (n), w = 4.

6 Conclusion

6.1 Performance

GPU implementations for both Daligner and Darwin were made. For Daligner, the CPU version is clearly the better choice. Considering the nature of this algorithm, this is not very surprising. Darwin showed better results on GPU, with substantial speedups. However, Daligner can produce more sensitive alignments in a shorter time. The main reason is the amount of work that needs to be done by the algorithms, Daligner calculates around 26 diagonals on average, Darwin calculates $T^2 = 102400$ SW elements. The main advantage of Darwin is that is calculates an actual SW score, instead of doing match/mismatches. This means Darwin can be configured to prefer substitutions or gaps, and customize it to suit a particular sequencing technology.

A clear tradeoff between sensitivity and specificity can be observed from the experiments. This shows that both algorithms can be configured to suit the input requirements of later stages in the assembly pipeline.

6.2 Future Work

A way to interpret the produced overlaps more efficiently is to use transitive overlaps. If reads A and B overlap, and B and C too, then A and C could also overlap. However, this should only result in marginal improvements of the sensitivity, since the overlap between A and C should already be found.

The CPU version could be optimized as well. The current multithreaded version has no load balancing. All CPU threads can a roughly equal amount of reads that must be test by D-SOFT. The number of seeds that emerge can vary (observed values are 17000 and 30655), this is partially caused by the different readlengths. Another reason for the imbalance is that the extension phase for each seed can have a different duration. One way to solve this is to create a large list with all the seeds, possibly partioned so that the threads can write to this list in parallel. When starting the GACT phase, each thread gets a batch of reads to extend. When a thread is done with its batch, or has less seeds than GPU threads, it gets another batch from the large list.

Yatish told me this, how to cite?

Bibliography

[1] B. Cornell, "Dna sturcture," http://ib.bioninja.com.au/standard-level/topic-2-molecular-biology/26-structure-of-dna-and-rna/dna-structure.html. vii, 4

[2] R. Kaur, "Dna replication," https://ramneetkaur.com/dna-replication/, August 2016. vii, 4

[3] B. Cummings, "Dna replication," http://academic.pgcc.edu/ kroberts/Lecture/Chapter%207/replication.html. vii, 5

[4] "Levels of protein structure," https://canvas.instructure.com/courses/1021937/pages/levels-of-protein-structure. vii, 7

[5] "Dna sequencing," https://www.khanacademy.org/science/biology/biotech-dna-technology/dna-sequencing-pcr-electrophoresis/a/dna-sequencing. vii, 7, 8, 9

[6] e. a. Zhenzhong Zhang, Wei Liu, "Use of pryosequencing to detect clinically relevant polymorphisms of genes in basal cell carcinoma," *Clinica Chimica Acta*, vol. 342, no. 1-2, pp. 137–143, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0009898103005989 vii, 10

[7] Unknown, "Next generation sequencing," https://www.atdbio.com/content/58/Next-generation-sequencing. vii, 8, 9, 10, 12, 13

[8] B. C. James M. Heather, "The sequence of sequencers: The history of sequencing dna," *Genomics*, vol. 107, no. 1, pp. 1–8, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888754315300410 vii, 7, 8, 10, 11, 13, 14, 15, 17

[9] Unknown, https://www.illumina.com/science/technology/next-generation-sequencing/paired-end-vs-single-read-sequencing.html. vii, 11

[10] A. E. Ozan Gundogdu, "Next generation sequencing," http://grf.lshtm.ac.uk/sequencing.htm. vii, 12

[11] A. Baldor, "Solid - sequencing by ligation," http://www.anthonybaldor.com/solid-sequencing-by-ligation/, November 2015. vii, 12

[12] I. S. Matthew W. Anderson, "Next generation dna sequencing and the future of genomic medicine," *Genes*, vol. 1, no. 1, pp. 38–69, 2010. [Online]. Available: https://www.mdpi.com/2073-4425/1/1 vii, 13

[13] M. L. Metzker, "Sequencing technologies - the next generation," *Nature Reviews Genetics*, vol. 11, pp. 31–46, 2010. [Online]. Available: http://www.nature.com/articles/nrg2626 vii, 14

[14] K. F. A. Anthony Rhoads, "Pacbio sequencing and its applications," *Genomics, Proteomics & Bioinformatics*, vol. 13, no. 5, pp. 278–289, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1672022915001345 vii, 15, 16

[15] Unknown, "Smrt sequencing: Read lengths," http://www.pacb.com/smrt-science/smrt-sequencing/read-lengths/. vii, 14, 15

[16] e. a. Kevin J. Travers, Chen-Shan Chin, "A flexible and efficient template format for circular consensus sequencing and snp detection," *Nucleid Acids Research*, vol. 38, p. e159, 2010. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2926623/ vii, 14, 16

[17] "Smrt sequencing: Single-molecule resolution," https://www.pacb.com/smrt-science/smrt-sequencing/single-molecule-resolution/. vii, 16

[18] "Products," https://nanoporetech.com/products. vii, 17

[19] e. a. Miten Jain, Hugh E. Olsen, "The oxford nanopore minion: delivery of nanopore sequencing to the genomics community," *Genome Biology*, vol. 17, 2016. [Online]. Available: https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-1103-0 vii, 17, 18

[20] e. a. David C. Bell, W. Kelly Thomas, "Dna base identification by electron microscopy," *Microscopy and Microanalysis*, vol. 18, pp. 1049–1053, 2012. [Online]. Available: https://www.cambridge.org/core/journals/microscopy-and-microanalysis/article/dna-base-identification-by-electron-microscopy/CC8E9C69093282A80BF02823F1AEEC0B viii, 17, 18, 20

[21] ——, "Dna sequencing with tem," *Microscopy and Microanalysis*, vol. 16, pp. 1768–1769, 2010. [Online]. Available: https://www.cambridge.org/core/journals/microscopy-and-microanalysis/article/dna-sequencing-with-tem/4CC11BBC721E4CC40982A62F9C8BC7E6 viii, 19

[22] J. A. M. Pawel Puczkarski, Jacob L. Swett, "Graphene nanoelectrodes for biomolecular sensing," *Journal of Materials Research*, vol. 32, pp. 3002–3010, 2017. [Online]. Available: https://www.cambridge.org/core/journals/journal-of-materials-research/article/graphene-nanoelectrodes-for-biomolecular-sensing/E0042E193CBC1CBEF6DB72121FAB429F/core-reader viii, 21

[23] Unknown, "Sequentie-alignering," https://nl.wikipedia.org/wiki/Sequentie-alignering. viii, 21

[24] ——, "Local alignment," http://rosalind.info/glossary/local-alignment/. viii, 22

[25] A. Mattheis, "Teaching - needleman-wunsch," http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Needleman-Wunsch, 2012. viii, 24, 25

[26] ——, "Teaching - smith-waterman," http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Smith-Waterman, 2012. viii, 26, 27

[27] P. Dawyndt, http://www.spoj.com/COGE2013/problems/PROG0385/, 2013. viii, 28

[28] S. Schlebusch and N. Illing, "Next generation shotgun sequencing and the challenges of de novo genome assembly," *South African Journal of Science*, vol. 108, pp. 62 – 70, 01 2012. [Online]. Available: http://www.scielo.org.za/scielo.php?script=sci_arttext&pid=S0038-23532012000600016&nrm=iso viii, 29

[29] e. a. Jennifer Commins, Christina Toft, "Computational biology methods and their application to the comparative genomics of endocellular symbiotic bacteria of insects," *Biological procedures online*, vol. 11, pp. 52–78, 04 2009. viii, 30

[30] A. Karatarakis, "Hybrid cpu/gpu computing with domain decomposition," https://www.slideshare.net/alexkaratarakis/hybrid-cpugpu-computing-with-domain-decomposition, 2011. viii, 32

[31] "Cuda c programming guide," http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. viii, 33, 35, 55, 59

[32] Unknown, "Optimization strategies," http://www.cmsoft.com.br/opencl-tutorial/optimization-strategies/. viii, 34

[33] N. Gupta, "Texture memory in cuda — what is texture memory in cuda programming," http://cuda-programming.blogspot.nl/2013/02/texture-memory-in-cuda-what-is-texture.html, 2013. viii, 35

[34] M. Harris, "How to overlap data transfers in cuda c/c++," https://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/, 2012. viii, 35

[35] Unknown, "Seqnfind - bioinfomatics software tool," https://www.environmental-expert.com/software/seqnfind-bioinformatics-software-tool-505887. viii, 36

[36] N. Butler, "Myers' diff algorithm : The basic greedy algorithm," http://simplygenius.net/Article/DiffTutorial1, September 2009. viii, ix, 38, 39

[37] e. a. Yatish Turakhia, Gill Bejerano, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: ACM, 2018, pp. 199–213. [Online]. Available: http://doi.acm.org/10.1145/3173162.3173193 ix, 46, 47, 48

[38] H. Chial, "Dna sequencing technologies key to the human genome project," *Nature Education*, vol. 1, no. 1, p. 219, 2008. [Online]. Available: https://www.nature.com/scitable/topicpage/dna-sequencing-technologies-key-to-the-human-828 1, 7

[39] Unknown, "How similar is human dna to plant dna?" https://www.genome.gov/dnaday/q.cfm?aid=785&year=2010, 2010. 1

[40] ——, "Specific genetic disorders," https://www.genome.gov/10001204/specific-genetic-disorders/, May 2018. 1

[41] J. Cohen, "Bioinformatics - an introduction for computer scientists," *ACM Computing Surveys*, vol. 36, no. 2, pp. 122–158, 2004. [Online]. Available: https://dl.acm.org/citation.cfm?doid=1031120.1031122 3

[42] P. Enrique Reynaud, "Protein misfolding and degenerative diseases," *Nature Education*, vol. 3, no. 9, p. 28, 2010. [Online]. Available: http://www.nature.com/scitable/topicpage/protein-misfolding-and-degenerative-diseases-14434929 3

[43] "What are proteins and what do they do?" https://ghr.nlm.nih.gov/primer/howgeneswork/protein, October 2017. 3

[44] Unknown, "Specificity of enzymes," http://www.worthington-biochem.com/introbiochem/specificity.html. 3

[45] J. M. B. Lubert Stryer, *Biochemistry 5th edition.* New York: W H Freeman, 2002. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK22380/ 3

[46] A. Purcell, "Dna," https://basicbiology.net/micro/genetics/dna/, February 2016. 3

[47] R. O. Akio Sugino, Susumu Hirose, "Rna-linked nascent dna fragments in escherichia coli," *Proceedings of the National Academy of Sciences*, vol. 69, no. 7, pp. 1863–1867, 1972. [Online]. Available: http://www.pnas.org/content/69/7/1863 5

[48] "The genetic code," http://www.biology-pages.info/C/Codons.html, December 2016. 5

[49] "translation / rna translation," http://www.nature.com/scitable/definition/translation-173, 2014. 6

[50] W. B. Suzanne Clancy, "Translation: Dna to mrna to protein," https://www.nature.com/scitable/topicpage/translation-dna-to-mrna-to-protein-393, p. 101, 2008. 6

[51] M. Vötsch, "mrna degradation," http://www.eb.tuebingen.mpg.de/research/research-groups/remco-sprangers/mrna-degradation.html. 6

[52] Unknown, "Protein structure," http://www.nature.com/scitable/topicpage/protein-structure-14122136, 2014. 6

[53] e. a. Elangannan Arunan, Gautam R. Desiraju, "Definition of the hydrogen bond (iupac recommendations 2011)," *The Scientific Journal of IUPAC*, vol. 83, July 2011. 6

[54] "Ionic and covalent bonds," https://chem.libretexts.org/Core/Organic_Chemistry/Fundamentals 6

[55] D. Goodsell, "Chaperones," https://canvas.instructure.com/courses/1021937/pages/levels-of-protein-structure, August 2002. 6

[56] Unknown, http://www.ensembl.org/Homo_sapiens/Location/Genome, 2017. 6

[57] e. a. Iakes Ezkurdia, David Juan, "Multiple evidence strands suggest that there may be as few as 19 000 human protein-coding genes," *Human Molecular Genetics*, vol. 23, pp. 5866–5878, 2014. [Online]. Available: https://academic.oup.com/hmg/article/23/22/5866/2900773 6

[58] A. C. Frederick Sanger, Steve Nicklen, "Dna sequencing with chain-terminating inhibitors," *Prc Natl Acad Sci USA*, vol. 74, no. 12, pp. 5463–5467, 1977. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/271968 7

[59] "Sanger dna sequencing: Primer design," https://dnacore.mgh.harvard.edu/new-cgi-bin/site/pages/sequencing_pages/primer_design.jsp. 7

[60] "Calculating the optimum ddntp:dntp ratio in sanger sequencing," https://sciencesnail.weebly.com/science/calculating-the-optimum-ddntp-to-dntp-ratio-in-sanger-sequencing, February 2015. 7

[61] J. K. Kuiski, *Next-Generation Sequencing - An Overview of the History, Tools, and 'Omic' Applications.* Biochemistry, Genetics and Molecular Biology, 2016. 8, 9, 12

[62] J. Kimball, "The polymerase chain reaction (pcr): Cloning dna in the test tube," http://www.biology-pages.info/P/PCR.html. 8

[63] ——, "Pyrosequencing," http://www.biology-pages.info/P/Pyrosequencing.html. 8

[64] V. K. Evelina Gasperskaja, "The most common technologies and tools for functional genome analysis," *Acta Med Litu*, vol. 24, no. 1, pp. 1–11, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5467957/ 9

[65] Unknown, "08. bridge pcr," https://binf.snipcademy.com/lessons/ngs-techniques/bridge-pcr. 9

[66] I. N. R. José F. Siqueira Jr., Ashraf F. Fouad, "Pyrosequencing as a tool for better understanding of human microbiomes," *J Oral Microbiol.*, vol. 4, no. 10, 2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3266102/ 9

[67] Unknown, "Molecular mechanism of dna replication," https://www.khanacademy.org/science/biology/dna-as-the-genetic-material/dna-replication/a/molecular-mechanism-of-dna-replication. 10

[68] A. K. Eric E. Schadt, Steve Turner, "A window into third-generation sequencing," *Human Molecular Genetics*, vol. 19, no. R2, pp. 227–240, 2010. [Online]. Available: https://academic.oup.com/hmg/article/19/R2/R227/641295 13, 14

[69] e. a. Mathieu Foquet, Kevan T. Samiee, "Improved fabrication of zero-mode waveguides for single-molecule detection," *Journal of Applied Physics*, vol. 103, no. 3, 2008. [Online]. Available: http://aip.scitation.org/doi/10.1063/1.2831366 14

[70] "Pacific biosciences," http://allseq.com/knowledge-bank/sequencing-platforms/pacific-biosciences/. 15

[71] e. a. Haitao Liu, Jin He, "Translocation of single-stranded dna through single-walled carbon nanotubes," *Science*, vol. 327, pp. 64–67, 2010. [Online]. Available: http://science.sciencemag.org/content/327/5961/64 16

[72] e. a. T. Laver, J. Harrison, "Assessing the performance of the oxford nanopore technologies minion," *Biomolecular Detection and Quantification*, vol. 3, pp. 1–8, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214753515000224 17

[73] e. a. Jason L. Weirather, Mariateresa de Cesare, "Comprehensive comparison of pacific biosciences and oxford nanopore technologies and their applications to transcriptome analysis [version 1; referees: 2 approved with reservations]," *F1000Research*, vol. 6, 2017. [Online]. Available: https://f1000research.com/articles/6-100/v1 17

[74] e. a. A. Bensimon, A. Simon, "Alignment and sensitive detection of dna by a moving interface," *Science*, vol. 265, pp. 2096–2098, 1994. [Online]. Available: http://science.sciencemag.org/content/265/5181/2096 18

[75] e. a. M. Oshige, K. Yamaguchi, "A new dna combing method for biochemical analysis," *Analytical Biochemistry*, vol. 400, pp. 145–147, 2010. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/20085744 18

[76] e. a. A.C. Payne, M. Andregg, "Molecular threading: mechanical extraction, stretching and placement of dna molecules from a liquid-air interface," *PLoS ONE*, vol. 8, p. e69058, 2013. [Online]. Available: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0069058 18

[77] e. a. Aline Cerf, Thomas Alava, "Transfer-printing of single dna molecule arrays on graphene for high resolution electron imaging and analysis," *Nano Letters*, vol. 11, pp. 4232–4238, 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3205448/ 18

[78] e. a. S. Bals, B. Kabius, "Annular dark field imaging in a tem," *Solid State Communications*, vol. 130, pp. 675–680, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038109804002200 18

[79] H. R. G. Binnig, "Scanning tunneling microscopy," *IBM Journal of Research and Development*, vol. 30, pp. 355–369, 1986. [Online]. Available: http://idwebhost-202-73.ethz.ch/praktika/analytisch/stm/Surf%20Sci%20126,%20p236.pdf 19

[80] Unkown, "Press release: The 1986 nobel prize in physics," https://www.nobelprize.org/nobel_prizes/physics/laureates/1986/press.html, October 1986. 19

[81] J. Hoffman, "Scanning tunneling microscopy," http://hoffman.physics.harvard.edu/research/STMintro.php, January 2010. 19

[82] M. P. S.M. Lindsay, "Can the scanning tunneling microscope sequence dna?" *Genetic Analysis: Biomolecular Engineering*, vol. 8, pp. 8–13, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/105038629190003A 19, 20

[83] D. Porath, "Scanning tunnelling microscopy: A dna sequence scanned," *Nature Nanotechnology*, vol. 4, pp. 476–477, 2009. [Online]. Available: http://www.nature.com/articles/nnano.2009.212 20

[84] M. D. Ventra, "Fast dna sequencing by electrical means inches closer," *Nanotechnology*, vol. 24, p. 342501, 2013. [Online]. Available: http://stacks.iop.org/0957-4484/24/i=34/a=342501 20

[85] M. D. V. M. Zwolak, "Electronic signature of dna nucleotides via transverse transport," *Nano Letters*, vol. 5, pp. 421–424, 2005. [Online]. Available: http://mike.zwolak.org/wp-content/uploads/2015/01/2005NL5-2.pdf 20

[86] Unknown, "Establishment of quantum biosystems inc. as a venture company," http://quantumbiosystems.com/news/2013/01/07/quantum-biosystems-inc-established.html, January 2013, press Release. 20

[87] ——, "Technology," http://quantumbiosystems.com/features/tech. 20

[88] R. W. Irving, "Plagiarism and collusion detection using the smith-waterman algorithm," *University of Glasgow*, 2004. [Online]. Available: https://pdfs.semanticscholar.org/cf79/e3882fa3cb985e647c5a38e1428c15d767c9.pdf 21

[89] L. A. Pray, "Dna replication and causes of mutation," *Nature Education*, vol. 1, no. 1, p. 214, 2008. [Online]. Available: https://www.nature.com/scitable/topicpage/dna-replication-and-causes-of-mutation-409 21

[90] Dumitru, "Dynamic programming from novice to advanced," https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/. 22

[91] Unknown, "Tutorial for dynamic programming," https://www.codechef.com/wiki/tutorial-dynamic-programming, 2009. 22

[92] C. D. W. Saul B. Needleman, "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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022283670900574?via%3Dihub 23

[93] M. S. W. Temple F. Smith, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981. 26

[94] D. Durand, "Sequence alignment," http://www.cs.cmu.edu/ durand/03-711/2015/Lectures/PW_sequence_alignment_2015.pdf, Fall 2015. 27

[95] R. Hochberg, "Semi-global alignment," http://dimacs.rutgers.edu/archive/BMC/TeacherMaterials/LocalA 2005. 27

[96] S. S. Vincent Kenny, Matthew Mathal, "Heuristic algorithms," https://optimization.mccormick.northwestern.edu/index.php/Heuristic_algorithms, May 2014. 27

[97] Z. A.-A. Nauman Ahmed, Koen Bertels, "A comparison of seed-and-extend techniques in modern dna read alignment algorithms," *Proc. Workshop on Accelerator-Enabled Algorithms and Applications in Bioinformatics*, pp. 1426–1433, 2016. [Online]. Available: http://ce-publications.et.tudelft.nl/publication/view/id/1597 27

[98] M. L. Bin Ma, John Tromp, "Patternhunter: faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002. [Online]. Available: https://academic.oup.com/bioinformatics/article/18/3/440/236636 27

[99] W. M. Kun-Mao Chao, William R. Pearson, "Aligning two sequences within a specified diagonal band," *Bioinformatics*, vol. 8, no. 5, pp. 481–487, 1992. [Online]. Available: https://academic.oup.com/bioinformatics/article/8/5/481/213891 28

[100] e. a. Stephen F. Altschul, Thomas L. Madden, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997. [Online]. Available: https://academic.oup.com/nar/article/25/17/3389/1061651 28

[101] C. Notredame, "Recent evolutions of multiple sequence alignment algorithms," *PLoS Computational Biology*, vol. 3, no. 8, p. e123, 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1963500/ 28

[102] E. B. Institute, "Multiple sequence alignment," https://www.ebi.ac.uk/Tools/msa/, 2017. 28

[103] M. P. Niranjan Nagarajan, "Parametric complexity of sequence assembly: theory and applications to next generation sequencing," *Journal of Computational Biology*, vol. 16, no. 7, pp. 897–908, 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/19580519 29

[104] Unknown, https://era7bioinformatics.com/en/page.cfm?id=1500. 29

[105] P. Wei-Che, "De novo sequence assembly," http://lsl.sinica.edu.tw/Services/Class/files/20150612 July 2015. 29

[106] e. a. René L. Warren, Granger G. Sutton, "Assembling millions of short dna sequences using ssake," *Bioinformatics*, vol. 23, pp. 500–501, 2007. [Online]. Available: https://academic.oup.com/bioinformatics/article/23/4/500/181258 29

[107] e. a. Juliane C. Dohm, Claudio Lottaz, "Sharcgs, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing," *Genome Research*, vol. 17, no. 11, pp. 1697–1706, 2007. [Online]. Available: https://genome.cshlp.org/content/17/11/1697.long 29

[108] e. a. Douglas W. Bryant Jr., Weng-Keen Wong, "Qsra a quality-value guided de novo short read assembler," *BMC Bioinformatics*, vol. 10, no. 1, p. 69, 2009. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-69 29

[109] B. Sinaimeri, "Introduction to (de novo) assembly," http://pbil.univ-lyon1.fr/members/sagot/htdocs/coursesENS/Lecture5.pdf, Nov 2016. 29

[110] e. a. Guy-Franck Richard, Alix Kerrest, "Comparative genomics and molecular dynamics of dna repeats in eukaryotes," *Microbiology and Molecular Biology Reviews*, vol. 72, no. 4, pp. 686–727, 2008. [Online]. Available: http://mmbr.asm.org/content/72/4/686 29

[111] W. G. A. P. Jason de Koning, "Repetitive elements may comprise over two-thirds of the human genome," *PLoS Genetics*, vol. 7, no. 12, p. e1002384, 2011. [Online]. Available: http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1002384 29

[112] e. a. Zhenyu Li, Yanxiang Chen, "Comparison of the two major classes of assembly algorithms: overlaplayoutconsensus and de-bruijn-graph," *Briefings in Functional Genomics*, vol. 11, no. 1, pp. 25–37, 2012. [Online]. Available: http://dx.doi.org/10.1093/bfgp/elr035 29

[113] G. Baker, "Euler and hamiltonian paths," http://www.cs.sfu.ca/ ggbaker/zju/math/euler-ham.html, 2013. 30

[114] M. Pop, "Genome assembly reborn: recent computational challenges," *Briefings in Bioinformatics*, vol. 10, no. 4, pp. 354–366, 2009. [Online]. Available: https://academic.oup.com/bib/article/10/4/354/299108 30

[115] Unknown, "Research consortium announces significant progress to close gaps and uncover novel genes in the human reference genome sequence," https://www.roche.com/media/store/releases/med-cor-2013-02-22t.htm, February 2013, media Release. 30

[116] R. D. Jared T. Simpson, "Efficient de novo assembly of large genomes using compressed data structures," *Genome Research*, vol. 22, no. 3, pp. 549–556, 2012. [Online]. Available: https://genome.cshlp.org/content/22/3/549.long 30

[117] D. Hernandez, P. François, L. Farinelli, M. Østerås, and J. Schrenzel, "De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer," *Genome research*, 2008. 30

[118] D. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de bruijn graphs," *Genome research*, pp. gr–074 492, 2008. 30

[119] e. a. Jared T. Simpson, Kim Wong, "Abyss: a parallel assembler for short read sequence data," *Genome research*, pp. gr–089 532, 2009. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2694472/ 30

[120] e. a. Ruibang Luo, Binghang Liu, "Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler," *Gigascience*, vol. 1, no. 18, pp. 1–6, 2012. [Online]. Available: https://academic.oup.com/gigascience/article/1/1/1/2656146 30

[121] e. a. Pavel A. Pevzner, Haixu Tang, "An eulerian path approach to dna fragment assembly," *PNAS*, vol. 98, no. 17, pp. 9748–9753, 2001. [Online]. Available: http://www.pnas.org/content/98/17/9748/tab-article-info 30

[122] Unknown, "Genomes vs. gennnnes: The difference between contigs and scaffolds in genome assemblies," https://www.pacb.com/blog/genomes-vs-gennnnes-difference-contigs-scaffolds-genome-assemblies/, September 2016. 31

[123] e. a. Henrik Nordberg, Michael Cantor, "The genome portal of the department of energy joint genome institute: 2014 updates," *Nucleic Acids Research*, vol. 42, no. D1, pp. D26–D31, 2014. [Online]. Available: https://academic.oup.com/nar/article/42/D1/D26/1047465 31

[124] Unknown, "K-mers, k-mer specificity, and comparing samples with k-mer jaccard distance," https://angus.readthedocs.io/en/2017/kmers-and-sourmash.html. 31

[125] e. a. Heng Li, Jue Ruan, "Mapping short dna sequencing reads and calling variants using mapping quality scores," *Genomic Research*, vol. 18, pp. 1851–1858, 2008. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2577856/ 31

[126] W. H. W. Hui Jiang, "Seqmap: mapping massive amount of oligonucleotides to the genome," *Bioinformatics*, vol. 24, pp. 2395–2396, 2008. [Online]. Available: https://academic.oup.com/bioinformatics/article/24/20/2395/258850 31

[127] e. a. Wan-Ping Lee, Michael P. Stromberg, "Mosaik: A hash-based algorithm for accurate next-generation sequencing short-read mapping," *Public Library of Science One*, vol. 9, p. e90581, 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3944147/ 31

[128] e. a. Matei Zaharia, William J. Blosky, "Faster and more accurate sequence alignment with snap," *CoRR*, 2011. [Online]. Available: https://arxiv.org/abs/1111.5572 31

[129] e. a. Matei David, Misko Dzamba, "Shrimp2: Sensitive yet practical short read mapping," *Bioinformatics*, vol. 27, pp. 1011–1012, 2011. [Online]. Available: https://academic.oup.com/bioinformatics/article/27/7/1011/231070 31

[130] M. G. Gerton Lunger, "Stampy: A statistical algorithm for sensitive and fast mapping of illumina sequence reads," *Genomic Research*, vol. 21, pp. 936–939, 2011. [Online]. Available: https://genome.cshlp.org/content/21/6/936.full 31

[131] e. a. John C. Mu, Hui Jiang, "Fast and accurate read alignment for resequencing," *Bioinformatics*, vol. 28, pp. 2366–2373, 2012. [Online]. Available: https://academic.oup.com/bioinformatics/article/28/18/2366/253622 31

[132] D. W. Michael Burrows, "A block sorting lossless data compression algorithm," Digital Equipment Corporation, Tech. Rep., 1994, technical Report 124. 31, 37, 40, 46

[133] G. M. Udi Manber, "Arrays: A new method for on-line string searches," *Proceedings of the first annual ACM-SIAM symposium on Descrete algorithms*, pp. 319–327, 1990. [Online]. Available: http://dl.acm.org/citation.cfm?id=320176.320218 31, 40, 46

[134] R. D. Heng Li, "Fast and accurate short read alignment with burrowswheeler transform," *Bioinformatics*, vol. 25, pp. 1754–1760, 2009. [Online]. Available: https://academic.oup.com/bioinformatics/article/25/14/1754/225615 31

[135] Y. C. Li R., "Soap2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, pp. 1966–1967, 2009. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/19497933 31

[136] S. L. S. Ben Langmead, "Fast gapped-read alignment with bowtie 2," *Nature Methods*, vol. 9, pp. 357–359, 2012. [Online]. Available: https://www.nature.com/articles/nmeth.1923 31

[137] R. D. Heng Li, "Fast and accurate long-read alignment with burrowswheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp698 31

[138] A. A. Milad Gholami, "Aryana: Aligning reads by yet another approach," *BMC Bioinformatics*, vol. 15, no. 9, p. S12, Sep 2014. [Online]. Available: https://doi.org/10.1186/1471-2105-15-S9-S12 31

[139] e. a. Nils Homer, Barry Merriman, "Bfast: An alignment tool for large scale genome resequencing," *Public Library of Science One*, vol. 4, p. e7767, 2009. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2770639/ 31

[140] Unknown, "Data structures - merge sort algorithm," https://www.tutorialspoint.com/data_structures_algorithms/merge_sort_algorithm.htm. 31

[141] e. a. Thomas H. Cormen, Charles E. Leiserson, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. 31, 42

[142] e. a. Nawar Malhis, Yaron S.N. Butterfield, "Slider-maximum use of probability information for alignment of short sequence reads and snp detection," *Bioinformatics*, vol. 25, pp. 6–13, 2009. [Online]. Available: https://academic.oup.com/bioinformatics/article/25/1/6/301571 31

[143] e. a. Arun S. Konagurthu, Lloyd Allison, "Design of an efficient out-of-core read alignment algorithm," *WABI 2010: Algorithms in Bioinformatics*, vol. 6293, pp. 189–201, 2010. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-15294-8_16#citeas 31

[144] E. Ukkonen, "Algorithms for approximate string matching," *Information and Control*, vol. 64, no. 1, pp. 100–118, 1985, international Conference on Foundations of Computation Theory. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019995885800462 31

[145] e. a. Robert Vaser, Ivan Sović, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome Research*, vol. 27, no. 5, pp. 737–746, 2017. [Online]. Available: https://genome.cshlp.org/content/27/5/737.full 31

[146] S. Collange, "Architecture and micro-architecture of gpus," https://www.irisa.fr/alf/downloads/collange/talks/ufmg_scollange.pdf, May 2011, presentation. 32

[147] P. Richmond, "Gpu computing: Gpu architectures," http://paulrichmond.shef.ac.uk/teaching/NVIDIA/rabat/02presentation. 32

[148] "Kepler tuning guide," http://docs.nvidia.com/cuda/kepler-tuning-guide/index.html, 1.4.4.2 L1 Cache. 34

[149] e. a. Michael C. Schatz, Cole Trapnell, "High-throughput sequence alignment using graphics processing units," *BMC Bioinformatics*, vol. 8, no. 1, p. 474, 2007. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-474 36

[150] M. C. S. Cole Trapnell, "Optimizing data intensive gpgpu computations for dna sequence alignment," *Parallel Computing*, vol. 35, no. 8, pp. 429–440, 2009. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2749273/ 36

[151] A. C. M. A. d. M. Edans F. de O. Sandes, "Cudalign: Using gpu to accelerate the comparison of megabase genomic sequences," *SIGPLAN Not.*, vol. 45, no. 5, pp. 137–146, 2010. [Online]. Available: https://dl.acm.org/citation.cfm?id=1693473 36

[152] ——, "Smith-waterman alignment of huge sequences with gpu in linear space," in *2011 IEEE International Parallel Distributed Processing Symposium*, 2011, pp. 1199–1211. [Online]. Available: https://ieeexplore.ieee.org/document/6012857/ 36

[153] D. L. M. Yongchao Liu, Bertil Schmidt, "Cushaw: a cuda compatible short read aligner to large genomes based on the burrowswheeler transform," *Bioinformatics*, vol. 28, no. 14, pp. 1830–1837, 2012. [Online]. Available: https://academic.oup.com/bioinformatics/article/28/14/1830/218764 36

[154] K. Davies, "Get smrt: Pacific biosciences unveils software suite with commercial launch," April 2011. [Online]. Available: http://www.bio-itworld.com/news/04/29/2011/Pacific-Biosciences-software-commercial-launch.html 37

[155] P. Biosciences, "Pacific biosciences releases new dna sequencing chemistry to enhance read length and accuracy for the study of human and other complex genomes," http://investor.pacificbiosciences.com/releasedetail.cfm?releaseid=876252, October 2014, press release. 37

[156] G. Myers, "Efficient local alignment discovery amongst noisy long reads," *Algorithms in Bioinformatics, WABI 2014*, pp. 52–67, 2014. 37, 40, 43, 44

[157] M. S. W. Eric S. Lander, "Genomic mapping by fingerprinting random clones: A mathematical analysis," *Genomics*, vol. 2, no. 3, pp. 231–239, 1988. 37

[158] M. S. W. Gary A. Churchill, "The accuracy of dna sequences: Estimating sequence quality," *Genomics*, vol. 14, no. 1, pp. 89–98, 1992. 37

[159] M. S. W. Pavel A. Pevzner, Haixu Tang, "An eulerian path approach to dna fragment assembly," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 17, pp. 9748–9753, 2001. 37

[160] e. a. Leena Salmela, Riku Walve, "Accurate self-correction of errors in long reads using de bruin graphs," *Bioinformatics*, vol. 33, no. 6, pp. 799–806, 2017. 37

[161] e. a. Giles Miclotte, Mahdi Heydari, "Jabba: hybrid error correction for long sequencing reads," *Algorithms for Molecular Biology*, vol. 11, no. 10, 2016. 37

[162] E. W. M. John D. Kececioglu, "Combinatorial algorithms for dna sequence assembly," *Algorithmica*, vol. 13, no. 7, 1995. 37

[163] G. T. Mark J. Chaisson, "Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory," *BMC Bioinformatics*, vol. 19, no. 13, p. 238, 2012. 37

[164] E. W. Myers, "An o(nd) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 1-4, pp. 251–266, 1986. 37

[165] D. Galles, "Radix sort," https://www.cs.usfca.edu/ galles/visualization/RadixSort.html. 42

[166] W. Yuan, "Fast parallel radix sort algorithm," http://projects.csail.mit.edu/wiki/pub/SuperTech/ParallelRadixSort/Fast_Parallel_Radix_Sort_Algorithm 42

[167] V. J. Duvanenko, "Parallel in-place radix sort simplified," http://www.drdobbs.com/parallel/parallel-in-place-radix-sort-simplified/229000734. 42

[168] luison9999 (TopCoder member), "Using tries," https://www.topcoder.com/community/data-science/data-science-tutorials/using-tries/. 45

[169] D. Galles, "Trie (prefix tree)," https://www.cs.usfca.edu/ galles/visualization/Trie.html. 45

[170] e. a. Yatish Turakhia, Kevin Jie Zheng, "Darwin: A hardware-acceleration framework for genomic sequence alignment," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/01/24/092171 46

[171] e. a. Michael Roberts, Wayne Hayes, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004. [Online]. Available: https://academic.oup.com/bioinformatics/article/20/18/3363/202143 46

[172] M. K. M. Susan L. Graham, Peter B. Kessler, "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, jun 1982. [Online]. Available: http://doi.acm.org/10.1145/872726.806987 49

[173] Unknown, "Data release: 54x long-read coverage for pacbio-only de novo human genome assembly," https://www.pacb.com/blog/data-release-54x-long-read-coverage-for/, February 2014. 49

[174] ——, "Linux programmer's manual," http://man7.org/linux/man-pages/man2/gettimeofday.2.html, September 2017. 49

[175] ——, "Integer intrinsics," https://docs.nvidia.com/cuda/cuda-math-api/group__CUDA__MATH__INTRINSIC__INT.html, October 2018. 53, 55

[176] B. L. Jason Chin, Christopher Dunn, "Falcon genome assembly tool kit manual," https://github.com/PacificBiosciences/FALCON/wiki/Manual, November 2016. 53

[177] M. Harris, "Using shared memory in cuda c/c++," https://devblogs.nvidia.com/using-shared-memory-cuda-cc/, January 2013. 53

[178] Unknown, "Inline px assembly in cuda," https://docs.nvidia.com/cuda/inline-ptx-assembly/index.html, October 2018. 55

[179] ——, "Paralel thread exection isa," https://docs.nvidia.com/cuda/pdf/ptx_isa_6.0.pdf, September 2017. 55

[180] A. Mac, "." https://devtalk.nvidia.com/default/topic/672658/ptx-isa-predicated-execution-and-the-warp-divergence-issue/, January 2014, forumpost. 55

[181] Unknown, "6.57 other built-in functions provided by gcc," https://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html. 56

[182] e. a. Nauman Ahmed, Hamid Mushtaq, "Gpu accelerated api for alignment of genomics sequencing data," in *Proc. IEEE International Conference on Bioinformatics and Biomedicine*, Kansas City, USA, November 2017. 57

[183] R. Crovella, "Difference between tesla k40c, k40m, k40s," https://devtalk.nvidia.com/default/topic/789809/difference-between-tesla-k40c-k40m-k40s/?offset=5, November 2014. 61

[184] Unknown, "Profiler user's guide," https://docs.nvidia.com/cuda/profiler-users-guide/index.html, October 2018. 61

[185] M. H. Yukiteru Ono, Kiyoshi Asai, "Pbsim: Pacbio reads simulatortoward accurate genome assembly," *Bioinformatics*, vol. 29, no. 1, pp. 119–121, 2013. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bts649 66

List of definitions

sensitivity True Positive rate, the portion of positives that are correctly identified.

specificity True Negative rate, the portion of negatives that are correctly identified.

A