TITLE1
TITLE2
TITLE3

Tong Dong Qiu

CE-MS-2017

Abstract

DALIGNER or Daligner

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 authors acknowledge 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
January 15, 2018

1 Introduction

Background 2

MAIN IS TRUE

2.1 Cell and Molecular 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 [13]. 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 [14]. Examples of protein functions are breaking down molecules like other proteins, fat or carbyhydrates, fighting off foreign particles like viruses or bacteria, and assisting in a chemical reaction, in which case the protein is called an *enzyme* [15]. 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 catalyze certain reactions [16][17].

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 proces 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 eachother in a double helix. Each nucleotide is composed of a nucleobase, a sugar called deoxyribose and a phosphate group [18]. 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.

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.

Creating new proteins from DNA is done via messenger RNA or *mRNA*, which looks

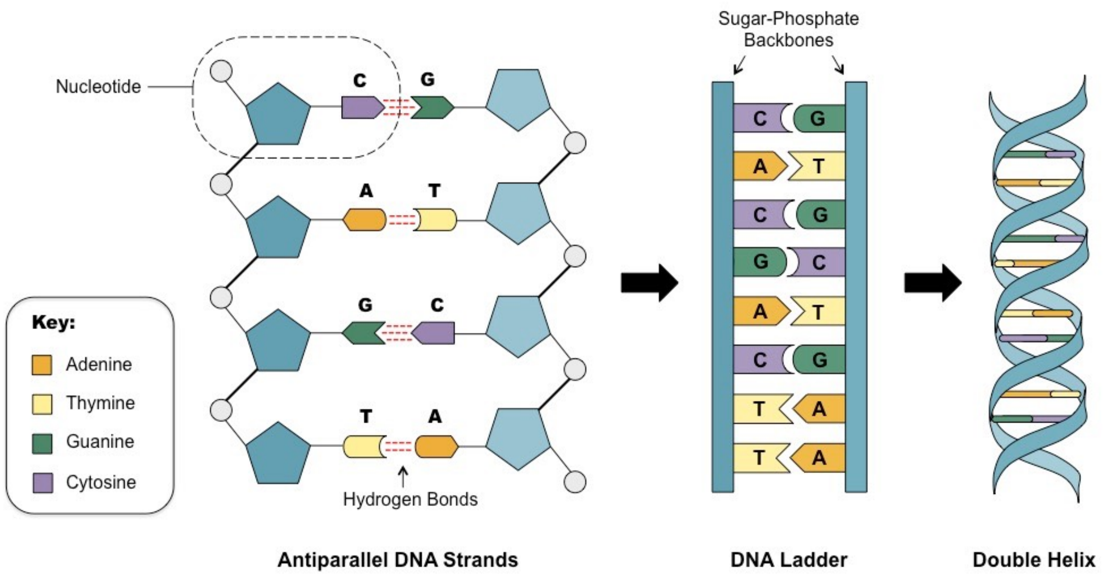

Figure 2.1: Structure of DNA [1]

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) [19].

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 seperated 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 seperated 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 uracyl.

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 [20]. 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 [21]. The used mRNA molecules degrade after they are used [22].

The protein is now a chain of amino acids, linked together by peptide bonds. This is called its *primary structure* [23]. 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 [24], charged amino acids can form ionic bonds [25]. Hydrophobic amino acids can form weaker van der Waals bonds. Most of these bonds noncovalent, which means they do not share electrons [25]. Cysteine is the only amino acid that can form covalent bonds [23].

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 2.2. 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* [23].

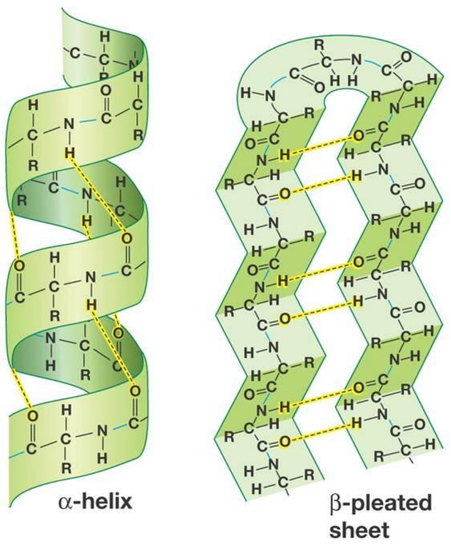

Figure 2.2: Alpha helix and beta sheet visualized, source: [2]

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 [26].

2.2 DNA sequencing

2.2.1 Sanger sequencing

Sanger sequencing, also called chain-termination sequencing, is the first major DNA sequencing technique. It is published in 1977 [27], and later improvements led to the development of commercial DNA sequencing machines in 1991 [6]. It is also the technique used by the Human Genome Project [3], a project to sequence the whole genome of a human. The project start around 1990, and finished in 2003 [28]. Since Sanger sequencing has a maximum chain length of about 900 basepairs [3], 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 [29]. 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 2.3. 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 [30].

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 electrophoresis* (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 seperate 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.

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 [31].

The first NGS method is *pyrosequencing*, developed in 2005 [6]. 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) [32][5], such that each bead has about 10 million identical copies of its fragment [33]. 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

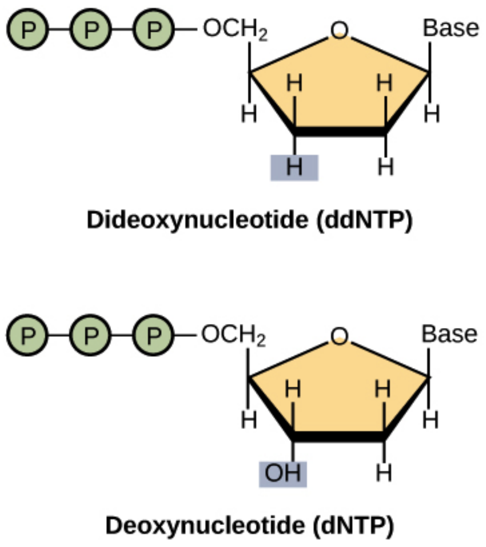

Figure 2.3: Normal nucleotides and dideoxynucleotides, source: [3]

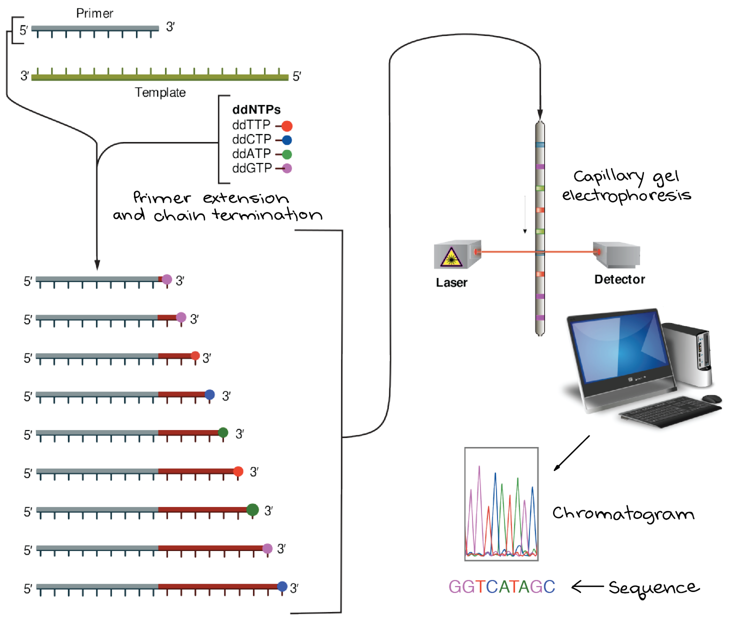

Figure 2.4: Workflow of Sanger sequencing, source: [3]

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 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 2.5.

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

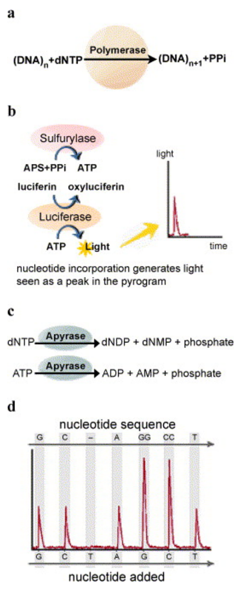

Figure 2.5: Workflow of pyrosequencing, source: [4]

The most popular NGS method is created by Illumina [34], 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 [5]. Bridge PCR is further explained in Figures 2.6 and 2.7. At the end of the PCR step, the reverse strands are washed away [35]. 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 [31][36]).

The third major NGS method is Sequencing by Oligonucleotide Ligation and Detection (SOLiD), released by Applied Biosystems Instruments (ABI), which later became Life Technologies [6]. SOLiD relies on ligation to chain nucleotides. DNA ligase is able to ligate double-stranded DNA [5], whereas DNA polymerase only adds one base to a growing chain, complementary to a template strand [37].

A single strand DNA piece, known as an adapter, is merged with the template strand 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 2.8. 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 leave 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

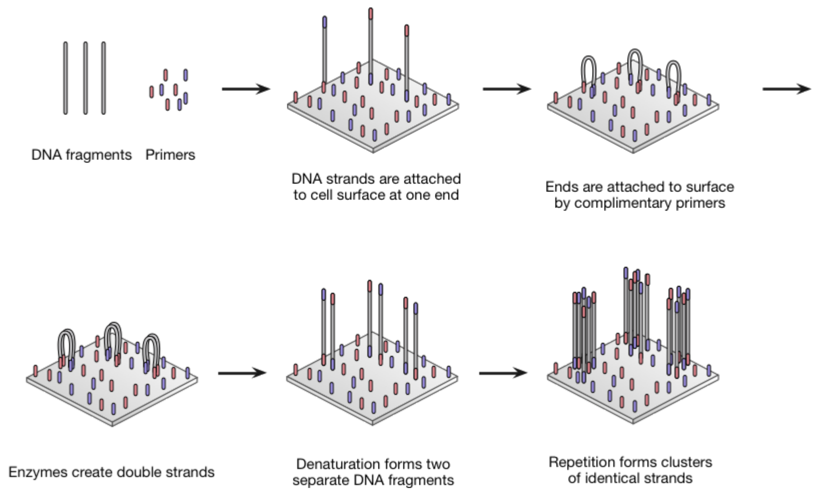
DNA fragments
Primers
DNA strands are attached to cell surface at one end
Ends are attached to surface by complimentary primers
Enzymes create double strands
Denaturation forms two separate DNA fragments
Repetition forms clusters of identical strands

Figure 2.6: Process of bridge PCR, source: [5]

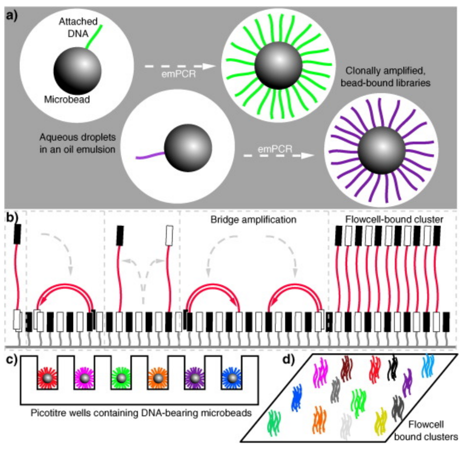
a)
Attached DNA
Microbead
emPCR
Clonally amplified, bead-bound libraries
Aqueous droplets in an oil emulsion
emPCR
b)
Bridge amplification
Flowcell-bound cluster
c)
Picotitre wells containing DNA-bearing microbeads
d)
Flowcell bound clusters

Figure 2.7: (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: [6]

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 **??**. The total workflow in shown in Figure 2.10.

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 [5]. A downside is that the read lengths are usually short, 50 to 75 basepairs [31].

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 [31]. Ion Torrent is very similar to pyrosequencing, but uses hydrogen ions instead of fluorescent dye to detect bases [5]. The number of ions is proportional to the number bases that was attached, however, at large repeating sections the accuracy decreases.

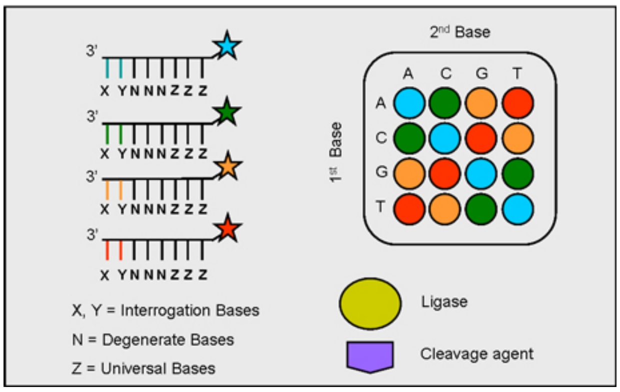

Figure 2.8: Color coding of SOLiD sequencing results, source: [7]

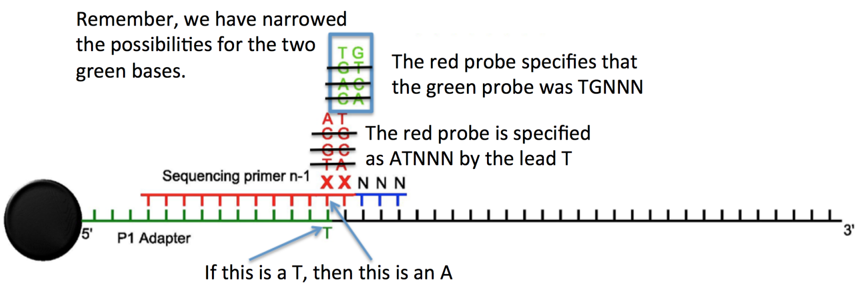
Remember, we have narrowed the possibilities for the two green bases.
The red probe specifies that the green probe was TGNNN
The red probe is specified as ATNNN by the lead T
Sequencing primer n-1
XX NNN
5'
P1 Adapter
3'
If this is a T, then this is an A

Figure 2.9: Color decoding of SOLiD sequencing results, source: [8]

Ion Torrent produces reads of up to 400 basepaires, but the error rate is relatively high at 1.5%.

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 [38]. However, an often used definition is that TGS approaches are able to sequence a single DNA molecule, thus eliminating the need for amplification [6][5].

The first TGS method is created by Helicos Biosciences, a company that went backrupt in 2012 [6]. 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 [38].

The other TGS are roughly placed into three categories: (a) sequencing by synthesis, (b) sequencing by nanopore, (c) direct imaging [38].

Probably the most used TGS platform is the Single Molecule RealTime sequencing platform from Pacific Biosciences.

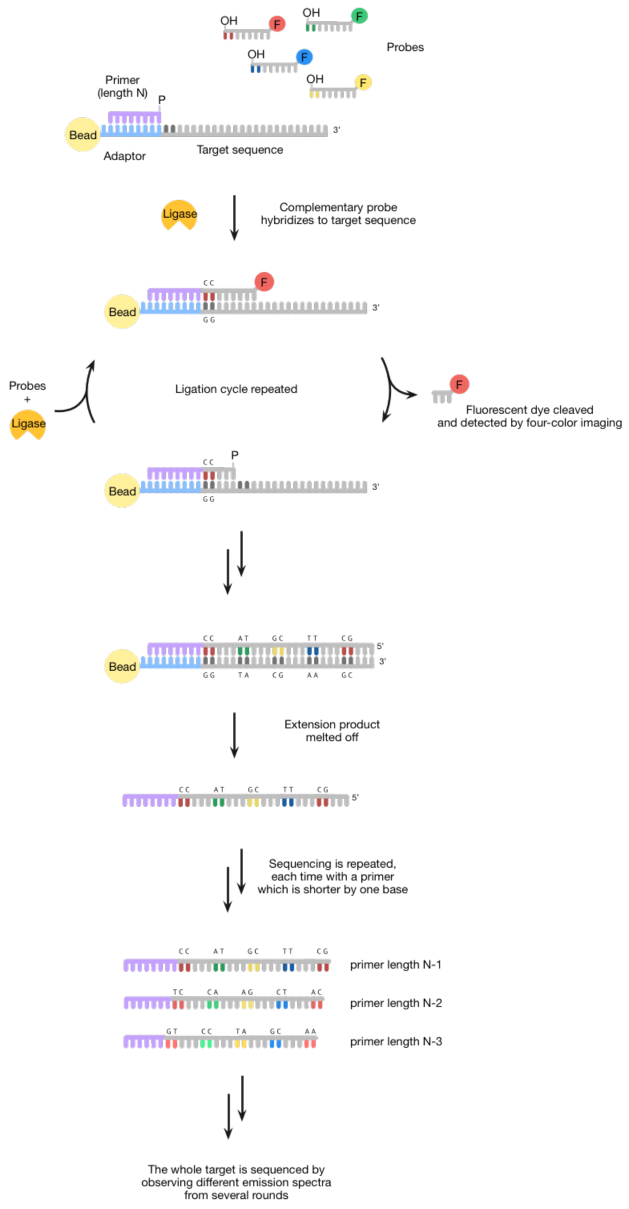
Probes
Primer (length N)
P
Bead
Adaptor
Target sequence
3'
Ligase
Complementary probe hybridizes to target sequence
Probes + Ligase
Ligation cycle repeated
Fluorescent dye cleaved and detected by four-color imaging
Extension product melted off
Sequencing is repeated, each time with a primer which is shorter by one base
primer length N-1
primer length N-2
primer length N-3
The whole target is sequenced by observing different emission spectra from several rounds

Figure 2.10: Workflow of SOLiD sequencing, source: [5]

2.3 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

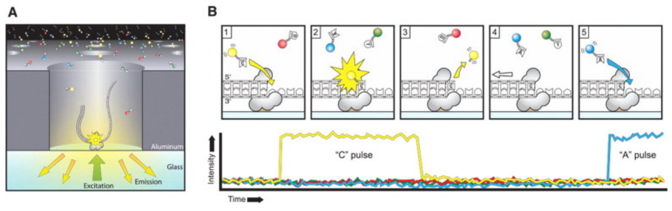

Figure 2.11: A: a ZMW with DNA polymerase and a DNA template strand, B: a C nucleotide is added to the chain, the next one is an A, source: [9]

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 [**?**]

GPUs cannot operate on their own, they must be guided by a CPU. A general workflow using a GPU is shown in Figure **??**.

- Copy data from CPU to GPU
- Let GPU process the data
- Copy results from GPU to 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.

2.4 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 **??** 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 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 eachother, only one memory transaction is needed, since a transaction processes a whole memory line. This is known as coalescing.

2.4.1 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 [10]. Figure 2.13 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.
- Local memory: Each thread has private memory for when registers are not enough. It is stored in global memory, so accesses are very slow.
- 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 **??** 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 [39].

2.4.2 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 low. These operations can be overlapped or pipelined by using streams. Figure 2.14 shows how the execution time can decrease by overlapping copying and computing.

2.5 Heterogeneous computing

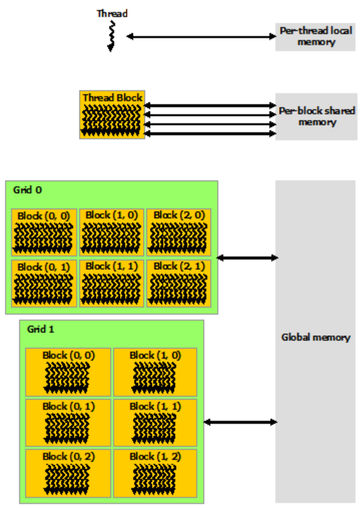

Figure 2.12: Memory hierarchy [10]

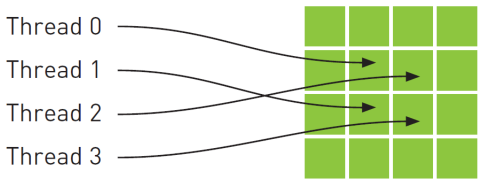

Figure 2.13: This access pattern has spatial locality, and could be cached in the texture cache [11]

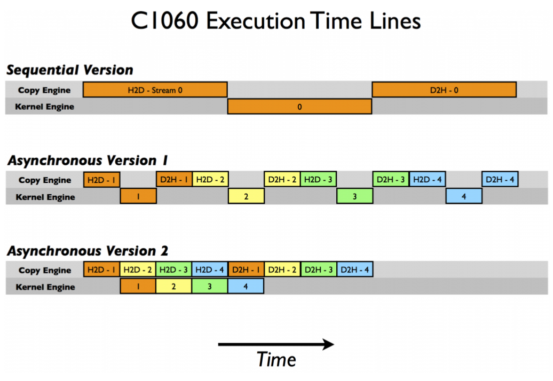

Figure 2.14: Different versions of the same computation [12]

3 Concept

MAIN IS TRUE

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 **NGS!** (**NGS!**) reads [40]. 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 [41]. While the drawback is that these reads have an error rate of 12-15%, this can be compensated by the distribution of these errors [42]. 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 [43]. Secondly, the work of Churchill and Waterman [44] implies that the accuracy of the consensus sequence of k sequences is O(ϵ^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 de novo assembly of the genome is possible, given enough coverage [42].

Important points for de novo 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 **DBG!** (**DBG!**) [45] 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 [46][47].

3.2 Daligner

The first step in an **OLC!** (**OLC!**) assembler is usually finding overlaps between reads [48]. BLASR [49] was the only long read aligner at the time, and inpsired Daligner. It uses the same filtering concept, but with a cache-coherent threaded radix sort to find seeds, instead of a BWT index [50]. 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 [51], combined with heuristic trimming and a datastructure 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 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 $i, j > 0$.

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 refered to as $F(d, k)$, where ρ is implicitely 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 cimputes 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 [42]. 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.

3.3 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 k-mers for every readpair (a, b) with $a \in \mathcal{A}$ and $b \in \mathcal{B}$. Previous methods to match k-mers include Suffix Arrays [52] and BWT indices [50].

Assuming that the k-mer matches are independent, the probablity that a k-mer is conserved while sequencing is $\pi = (1 - 2\epsilon)^k$. The number of conserved k-mers in an alignment of τ basepairs is a Bernouilli distribution with rate π, so an average of $\tau \cdot \pi$ k-mers 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 k-mers is 10. Only .046% of the expected readpairs have 1 or fewer k-mers, and only 0.26% have 2 of fewer. To filter with 99.74% sensitivity, only readpairs with 3 or more k-mer matches need to be examined.

The specificity of the filter is increased in two ways:

- computing the number of k-mer matches in diagonal bands of width 2^s instead of in the whole reads
- thresholding on the number of bases in k-mer matches, instead of the number of k-mers themselves

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

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

- Build a list $List_X = \{(kmer(X_x, i), x, i)\}_{x,i}$ of all k-mers for database $X \in \{\mathcal{A}, \mathcal{B}\}$, where $kmer(R, i)$ is the k-mer $R[i - k + 1, i]$.
- Sort both lists in order of their k-mers.
- 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 k-mer.
- 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 k-mers 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 k-mers in each pair of neighbouring diagonal bands. Note that only bases in matching k-mers are counted, not the number of k-mers. When there are k+1 matching consecutive bases, two k-mers are generated. These are less 'valuable' than two non-overlapping k-mers. 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.

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.

3.4 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 [53][54] 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 [53] 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 $b \in [0, 2^B - 1]$.

```
for i = 0 to N−1 do
{
        b = src[i]_p
        trg[bucket[b]] = src[i]
        bucket[b] += 1
}
```

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 [55][56], 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 $bucket[t][b] = \{i : src[i] < b \text{ or } src[i] = b \text{ and } i/part < t\}$. 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 seperately. 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.

```
int64 MASK = 2^B-1

sort_thread(int t, int bit, int N, int64 *src, int64 *trg, int *bucket, in
{
        for i = t*N to (t+1)*N-1 do
        {
                c = src[i]
                b = c >> bit
                x = bucket[b & MASK] += 1
                trg[x] = c
                next[x/N][(b >> B) & MASK] += 1
        }
}

int64 *radix_sort(int T, int N, int hbit, int64 src[0..N-1], int64 trg[0..
{
        int bucket[0..T-1][0..2^B-1], next[0..T-1][0..T-1][0..2^B-1]
        part = (N-1)/T + 1
        for l = 0 to hbit-1 in steps of B do
        {
                if (l != lbit)
                        bucket[t,b] = Sum_t next[u,t,b]
                else
                        bucket[t,b] = |{i : i/part == t and src[i] & MASK
                bucket[t,b] = Sum_u,(c<b) bucket[u,c] + Sum_(u<t) bucket[u
                next[u,t,b] = 0
                in parallel: sort_thread(t,l,part,src,trg,bucket[t],next[t
                (src,trg) = (trg,src)
        }
        return src
}
```

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 [42]. The optimal number of threads is more complex, because they usually do not have their own caches.

3.5 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 3.1 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 3.1: Local Alignment

```
MASKC  = 1 << (C−1)
W[low−2] = W[hgh+2] = W[hgh+1] = y = yp = −1
for k = low−1 to hgh+1 do
{
        (ym,y,yp) = (y,yp+1,W[d+1]+1)
        if (ym = min(ym,y,yp))
                (y,m,b) = (ym,M[k−1],B[k−1])
        else if (yp = min(ym,y,yp)
                (y,m,b) = (yp,M[k+1],B[k+1])
        else
                (y,m,b) = (y,M[k],B[k])

        if (b & MASKC != 0)
                m −= 1
        b <<= 1
        while (B[y] == A[y+k])
        {
                y += 1
                if (b & MASKC == 0)
                        m += 1
                b = (b << 1) | 1
        }
        (W[k],M[k],B[k]) = (y,m,b)
}
```

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 [42]. 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 [57][58] of all E-bit vectors and taking the minimum of each path of the trie.

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:

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

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

Specification 4

MAIN IS TRUE

4.1 Profiling

MCprof [59][60] was used to profile Daligner, the results are shown in Figures **??** and **??**. The algorithm consists of three main phases: seeding, filtering and aligning. The seeding phase takes part in the lex_sort(), count_thread() and merge_thread() functions. The filter implementation is not very complex, and is combined with the alignment part in report_thread(), this function starts the actual alignment in the form of Local_Alignment() when a suitable seed is found. Each seed is aligned in forward and reverse direction.

Since Local_Alignment() is the most timeconsuming function, that will be the first one the be considered for acceleration. Each Local_Alignment() call originates from a separate readpair A, B, so executing multiple Local_Alignment() instances at the same time should not be a problem. This is why Daligner uses multiple threads, each one running the filter-align function report_thread() on its own list of readpairs.

In order to launch multiple Local_Alignment() functions from one kernel, they have to be prepared.

Experiments 5

Conclusion 6

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.

[2] “Levels of protein structure,” https://canvas.instructure.com/courses/1021937/pages/levels-of-protein-structure.

[3] “Dna sequencing,” https://www.khanacademy.org/science/biology/biotech-dna-technology/dna-sequencing-pcr-electrophoresis/a/dna-sequencing.

[4] 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

[5] Unknown, “Next generation sequencing,” https://www.atdbio.com/content/58/Next-generation-sequencing.

[6] 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

[7] A. E. Ozan Gundogdu, “Next generation sequencing,” http://grf.lshtm.ac.uk/sequencing.htm.

[8] A. Baldor, “Solid - sequencing by ligation,” http://www.anthonybaldor.com/solid-sequencing-by-ligation/, November 2015.

[9] 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

[10] “Cuda c programming guide,” http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[11] 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.

[12] M. Harris, “How to overlap data transfers in cuda c/c++,” https://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/, 2012.

[13] 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

[14] 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

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

[16] Unknown, “Specificity of enzymes,” http://www.worthington-biochem.com/introbiochem/specificity.html.

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

[18] A. Purcell, “Dna,” https://basicbiology.net/micro/genetics/dna/, February 2016.

[19] “The genetic code,” http://www.biology-pages.info/C/Codons.html, December 2016.

[20] “translation / rna translation,” http://www.nature.com/scitable/definition/translation-173, 2014.

[21] 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.

[22] M. Vötsch, “mrna degradation,” http://www.eb.tuebingen.mpg.de/research/research-groups/remco-sprangers/mrna-degradation.html.

[23] Unknown, “Protein structure,” http://www.nature.com/scitable/topicpage/protein-structure-14122136, 2014.

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

[25] “Ionic and covalent bonds,” https://chem.libretexts.org/Core/Organic_Chemistry/Fundamentals/Ionic_and

[26] D. Goodsell, “Chaperones,” https://canvas.instructure.com/courses/1021937/pages/levels-of-protein-structure, August 2002.

[27] 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

[28] 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

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

[30] "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.

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

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

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

[34] 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/

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

[36] 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/

[37] 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.

[38] 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

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

[40] 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

[41] Unknown, "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.

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

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

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

[45] 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.

[46] 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.

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

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

[49] 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.

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

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

[52] G. M. Udi Manber, "Suffix 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

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

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

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

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

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

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

[59] I. Ashraf, "Communication driven mapping of applications on multicore platforms," Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, April 2016.

[60] I. Ashraf and V.M. Sima and K.L.M. Bertels, “Intra-application data-communication characterization,” in *Proc. 1st International Workshop on Communication Architectures at Extreme Scale*, Frankfurt, Germany, July 2015.

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