Speeding up Genotyping through GPU Acceleration

Master's Thesis

Jørgen Wictor Henriksen

15th of May 2023

Abstract

In the last couple of decades, high-throughput sequencing has steadily become more effective and orders of magnitude cheaper. With the potential for millions of genomes being sequenced in the coming years, tools for analysing the large amounts of sequenced data will become increasingly important. Recent work in alignment-free genotyping methods have shown that alignment-free methods where we use statistical methods on analysis of *k*mers from sequenced reads can give competitive accuracies while being significantly faster compared to more established alignment-based methods. A recently published genotyper, KAGE, showed that an alignment-free genotyper implemented in Python could yield competitive accuracies while being more than 10 times faster than any other known method. This thesis explores how parts of KAGE that deals with large matrix- and array-operations can be GPU accelerated, and finally presents GKAGE, a GPU accelerated version of KAGE. GKAGE achieves up to 10 times speed up compared to KAGE and is able to genotype a human individual in only a few minutes on consumer grade hardware.

Contents

1 Introduction **4**

2 Background **5**

2.1 Graphical Processing Units 5

2.1.1 GPUs in Computers 5

2.1.2 Programming Model and CUDA 6

2.2 Biology 7

2.2.1 DNA, Chromosomes and Genomes 7

2.2.2 Variants and Variant Calling 8

2.2.3 Genotype and Genotyping 9

2.2.4 High-Throughput DNA sequencing 10

2.3 Nucleotide Binary Encoding 10

2.4 *k*mer Counting 10

3 Thesis Goal **11**

4 Methods **12**

4.1 GPU Support Directly in Python 12

4.2 Custom CUDA Implementations 12

4.3 Finding Bottleneck Components 12

4.4 Implementation Details 12

4.4.1 CUDA Hash Table for *k*mer Counting 12

5 Results **14**

5.1 *K*mer parsing from raw reads 15

5.2 *K*mer counting 15

5.3 GKAGE vs KAGE 15

6 Discussions **16**

6.1 Drawbacks of Graphical Processing Units 16

7 Conclusion **17**

1 Introduction

...

2 Background

This section will explain necessary preliminaries for the further topics in this thesis.

2.1 Graphical Processing Units

Graphical Processing Units (GPUs) are massively parallel processing units designed for high-throughput parallel computations. This is as opposed to *Central Processing Units* (CPUs), which are designed to quickly perform many serial computations. GPUs were originally developed to accelerate computations performed on images, a highly parallel task where it is commonplace to have millions of relatively small independent computations that must be performed quickly in a single memory buffer. Although GPUs have mainly been used for graphical computations, they have in recent years been adopted in other areas as well with the introduction of the *General Purpose Graphical Processing Unit* (GPGPU) (**GPGPU cite**). The concept of the GPGPU is to use a GPU, which is designed for computer graphics, to perform computations in other domains where CPUs are typically used. Fields such as artificial intelligence (**AI accelerated by GPU cite**) and the broader scientific computing have enjoyed great utility from GPUs, using them to accelerate embarassingly parallel problems, *e.g.*, matrix operations. Despite being similar in power consumption, a GPU can provide much higher instruction throughput and memory bandwidth compared to its CPU competitors. These capability advantages exist in GPUs because they were specifically designed to perform well with regards to these dimensions.

As of 2023, Nvidia control the vast majority of the GPU market share, with only *Advanced Micro Devices* (AMD) and Intel as current serious competitors (**try to find a serious cite**). Furthermore, Nvidia GPUs with their CUDA programming model is considered to be the standard for scientific computing today (**cite**). Although most of the GPUs manufactured by different GPU manufacturing companies are very similar in both architecture and compute models, the term *GPU* will for the remainder of this thesis specifically refer to Nvidia GPUs, as the work presented in this thesis was developed and tested using only Nvidia GPUs.

2.1.1 GPUs in Computers

Two main computer GPU setups are prominent today: *integrated* graphical processing units (iGPUs), and *discrete* graphical processing units (dGPUs). iGPUs are GPUs integrated onto the same

die as a computer's CPU, where the two share the same physical *Random Access Memory* (RAM) unit. dGPUs are dedicated GPU devices that are physically distinct from the host computer's CPU and RAM, and have their own physical RAM. dGPUs are significantly more powerful in terms of compute throughput when compared to iGPUs. However, having their own physical RAM introduces an overhead; Memory buffers with input data have to be copied to the dGPU's RAM before processing, and results have to be copied back from the dGPU's RAM to the host RAM.

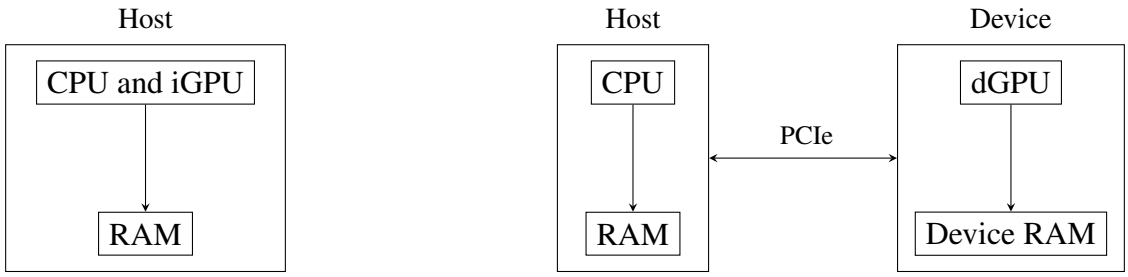

Figure 1: **Left**: A computer setup with a CPU and an iGPU sharing the same die and the same physical RAM. **Right**: A computer setup where a dGPU is connected over PCIe and the dGPU has its own physical RAM, adding the overhead of copying data both to and from the host computer when utilizing the GPU.

For the work presented in this thesis, only dGPUs were utilized. Therefore, the term GPU will from here on out be referring to a dGPU and not an iGPU. This means that all GPU implementations discussed in this thesis will include copying memory back and fourth from the *host* (CPU) RAM and the *device* (GPU) RAM. It is also possible for a single computer to have several connected GPUs, allowing for further parallelization of both memory transfers and compute, however this was not utilized in this thesis' work.

2.1.2 Programming Model and CUDA

Modern GPUs can in effect be considered to be massive *Single Instruction Multiple Data* (SIMD) machines. Strict flow control is therefore important; The same set of instructions should run in the same order for maximum utilization of the GPU's capability.

...

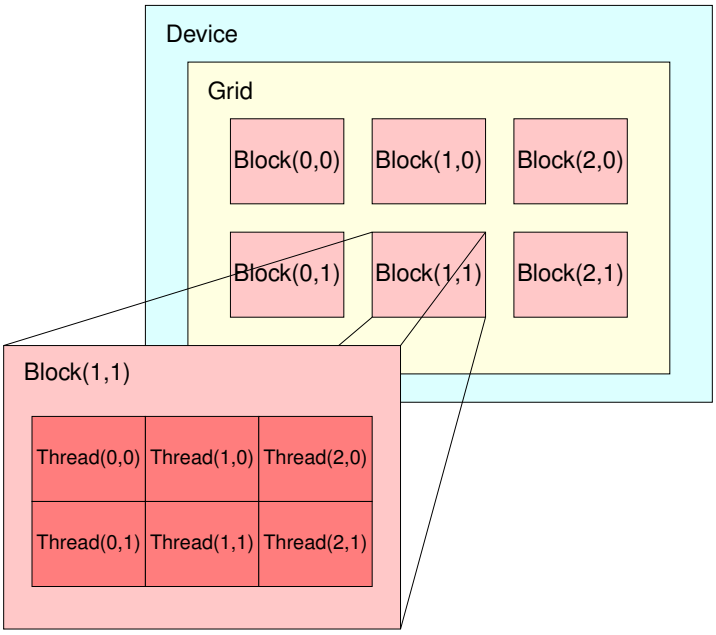

Figure 2: An overview of the CUDA programming model:

2.2 Biology

2.2.1 DNA, Chromosomes and Genomes

DNA, or *deoxyribonucleic acid*, is a type of molecule that contains all the genetic material found in the cells of all known living organisms [1]. The molecule is composed of two complementary strands of *nucleotides* that are twisted together to form a double helix structure, connected by bonds formed between complementary nucleotides. The two strands are in turn composed of the four nucleotide bases: adenine (A), guanine (G), cytosine (C) and thymine (T), where A and T, and C and G are complementary bases [2, p.15]. Furthermore, the two complementary strands of nucleotide bases actually encode the precise same information. This is because with knowledge of just one of the strands' nucleotide sequence, say $strand_1$, we can determine the sequence of the other strand, $strand_2$, by exchanging each nucleotide in $strand_1$ with their complements and finally reversing the strand to determine what $strand_2$'s sequence is.

Relatively small differences in these DNA sequences are what differentiates individuals within the same species from one other. It is therefore interesting to study these sequences of nucleotides encoding organisms' genetic information, as the encoded information can reveal details about both associated physical traits and diseases. In human cells, these DNA strands are estimated to be

Figure 3: A conceptual representation of a DNA molecule made up of two strands. The strands are composed of nucleotides forming base pairs where A (adenine) and T (thymine), and C (cytosine) and G (guanine) are complements of each other.

roughly $3 * 10^9$ bases long [2, p.13].

DNA is organized into structures called *chromosomes*. Humans have 23 chromosome pairs, making up a total of 46 chromosomes. Each of the pairs include one version of the chromosome inherited from the male parent, and one version inherited from the female parent [3].

The term *genome* can be used to refer to the complete genetic material of an organism. In practice, however, the genome of an organism often simply refers to the complete DNA nucleotide sequence of one set of chromosomes for that organism [2, p.13]. Commonly in bioinformatics, one can also encounter the term *reference genome*, referring to a theoretical reconstruction of an organism's genome created by scientists. Such genome reconstructions are commonly used when examining new DNA sequences, often by aligning new DNA sequences to the reference in order to see at which positions their nucleotides differ and what differences are present at those positions [4].

2.2.2 Variants and Variant Calling

When examining the genome of several individuals within the same species, one will find locations along the genome where the nucleotides differ for the different individuals. These distinct nucleotide manifestations are commonly referred to as *variants*. The term *variant calling* is used to refer to the process of determining which variants an individual has. In other words, given a reference genome sequence, where and how does the genome sequence of the individual of interest differ from the reference sequence. This process can abstractly be described in three steps: 1) sequence the genome of interest to get DNA reads (described in section 2.2.4), 2) align the reads to the reference genome by finding where along the reference genome sequence each read fits best, usually using a heuristic determining which location the read actually originates from, and 3) examine the alignments and note where the reads differ from the reference sequence, determining the

variants present in the sequenced genome [5].

2.2.3 Genotype and Genotyping

The term *genotype* refers to the set of variants an individual carries at a particular location along the genome sequence of all of its chromosomes [6]. For humans, who have two of each chromosome, a genotype would refer to two variants, one in each of the two chromosomes. *Genotyping* an individual refers to the process of determining which genotypes an individual carries. In most genotyping software tools today, genotypes are given in a format that specifies whether a particular variant is present in none, one or both of a human's chromosomes. For instance, given a reference genome sequence where a variant site is known to could manifest an A at a particular allele where the reference sequence contains a C, a human individual's genotype for this variant could either be referred to as $0/0$, meaning that the variant is present in neither of the chromosomes, $0/1$, meaning that the variant is present in one of the chromosomes, or $1/1$, meaning that the variant is present in both chromosomes.

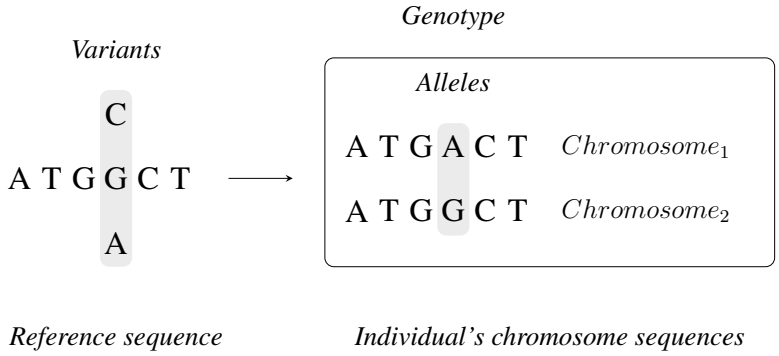

Figure 4: In humans, where there are two chromosomes, a genotype constitutes as a set of two alleles, one in each chromosome at the variant location. Along the reference sequence on the left, several possible variants may be known to occur at a specific location. After examining the sequence of an individual, we try to determine the individual's genotype by scoring which variants are present in each chromosome at the location of interest.

The most established way to genotype an individual today is to align DNA reads to a reference genome sequence and then examine how the reads differ from the reference sequence to determine which variants are present, and which genotypes are most probable at the different locations [4]. However, given how many reads one have come to expect from high-throughput sequencing today [2.2.4] and how time consuming it is to align reads to a $3 * 10^9$ long reference sequence, although accurate, this strategy is very time consuming. A new prominent strategy has emerged in recent years that helps to alleviate the time consumption aspect of genotyping. Statistics based methods

where the variant calling step is skipped altogether, and small parts of the DNA reads called *k*mers are analyzed to then use bayesian models determine which genotypes are most probable given previous knowledge accumulated over years of research [7, 8, 9]. One such baysian genotyper, KAGE, have recently showed that it can genotype a human individual more than 10 times faster than any other known genotyper tool, while still providing competitive accuracy scores [7].

2.2.4 High-Throughput DNA sequencing

...

2.3 Nucleotide Binary Encoding

DNA nucleotide sequences (described in section 2.2.1) inside computer software is commonly represented simply by a sequence of the 8 bit characters A, C, T and G (or alternatively the lowercase a, c, t and g). This representation, however, is cumbersome to operate on and requires large amounts of memory to store. To circumvent these issues, a widely adopted technique is to encode the nucleotides into binary form. This leads to much quicker processing of nucleotide sequences and reduces the memory usage needed to store the sequences by 75%. This is achieved by realizing that only 2 bits, giving $2^2 = 4$ possible unique states, is enough to represent all of the four DNA nucleotides A, C, G and T. The binary encoding can be extended further to represent whole nucleotide sequences in binary arrays. For example, an integer array, if interpreted 2 consecutive bits at a time, can represent such a sequence.

2.4 *k*mer Counting

The *k*mer counting problem is ...

Nucleotide to 2 bit encoding lookup table

A	00
C	01
G	10
T	11

DNA	A	C	C	T	G	T	A	G
	|	|	|	|	|	|	|	|
2 bit represented DNA	00	01	01	11	10	11	00	10

Figure 5: A lookup table showing how nucleotides can be encoded using 2 bits and a DNA nucleotide sequence represented both as plain characters as well as its 2 bit encoded representation. Recall that computers use 8 bits to represent a single nucleotide with a character, whilst the 2 bit encoding only needs 2 bits to represent a nucleotide.

3 Thesis Goal

The goal of this thesis is to explore whether state-of-the-art genotyping can be sped up in any significant way by utilizing GPUs. More specifically, this thesis will investigate whether alignment-free genotyping, which presently is significantly faster compared to alignment-based genotyping, can be sped up by the GPU. In order to investigate this, I will attempt to integrate GPU accelerated functionality into a base genotyper, KAGE [7], which is presently the fastest known genotyper that also yields good results. The resulting GPU accelerated genotyping software, GKAGE, will then be benchmarked against KAGE to account for any potential speed up. Finally, I will discuss elements about the work process and implementations, discuss possible future work and conclude the work presented in this thesis.

4 Methods

The following chapter will describe the methods used and implementation details of how we created GPU support for KAGE.

4.1 GPU Support Directly in Python

...

4.2 Custom CUDA Implementations

...

4.3 Finding Bottleneck Components

...

4.4 Implementation Details

4.4.1 CUDA Hash Table for *k*mer Counting

In order to solve the *k*mer counting problem described in section 2.4, GKAGE uses a GPU accelerated static hash table. The hash table uses open addressing with a simple linear probing scheme for insertions (during initialization), counts and queries.

The hash table is implemented using Nvidia's CUDA framework (described in section 2.1.2. as a C++ class with two arrays, one for the keys and one for the values (counts), allocated in the GPU's global memory. The class implements methods for the three main operations:

1. Insertion: only once during initialization
2. Counting: incrementing the values associated with each *k*mer value in an array
3. Querying: fetching the values associated with each *k*mer value in an array, returning an array of the same size

The methods accept arrays that are either allocated in the GPU's global memory or in the host RAM. When calls are made with the latter, the class method will handle the allocation of GPU

memory, copying of data between host and GPU, and de-allocation of GPU memory when the processing has completed. In the case of queries where the input *k*mers are located in the host RAM, the corresponding counts array will also be located in the host RAM.

All of the operations (insert, count and query) work on a single *k*mer basis. Notably, all of the operations take in an array of *k*mers, and all of the corresponding CUDA implementations launch a single thread for each *k*mer to fulfill the operation that is supposed to happen for that *k*mer. For example, when counting *k*mers, an array of *n* *k*mers will be provided, and *n* CUDA threads will be started, each taking care of incrementing the count of a single *k*mer in the hash table.

$$p_0 = hash(k) \bmod c \qquad\qquad p_i = (p_{i-1} + 1) \bmod c \tag{1}$$

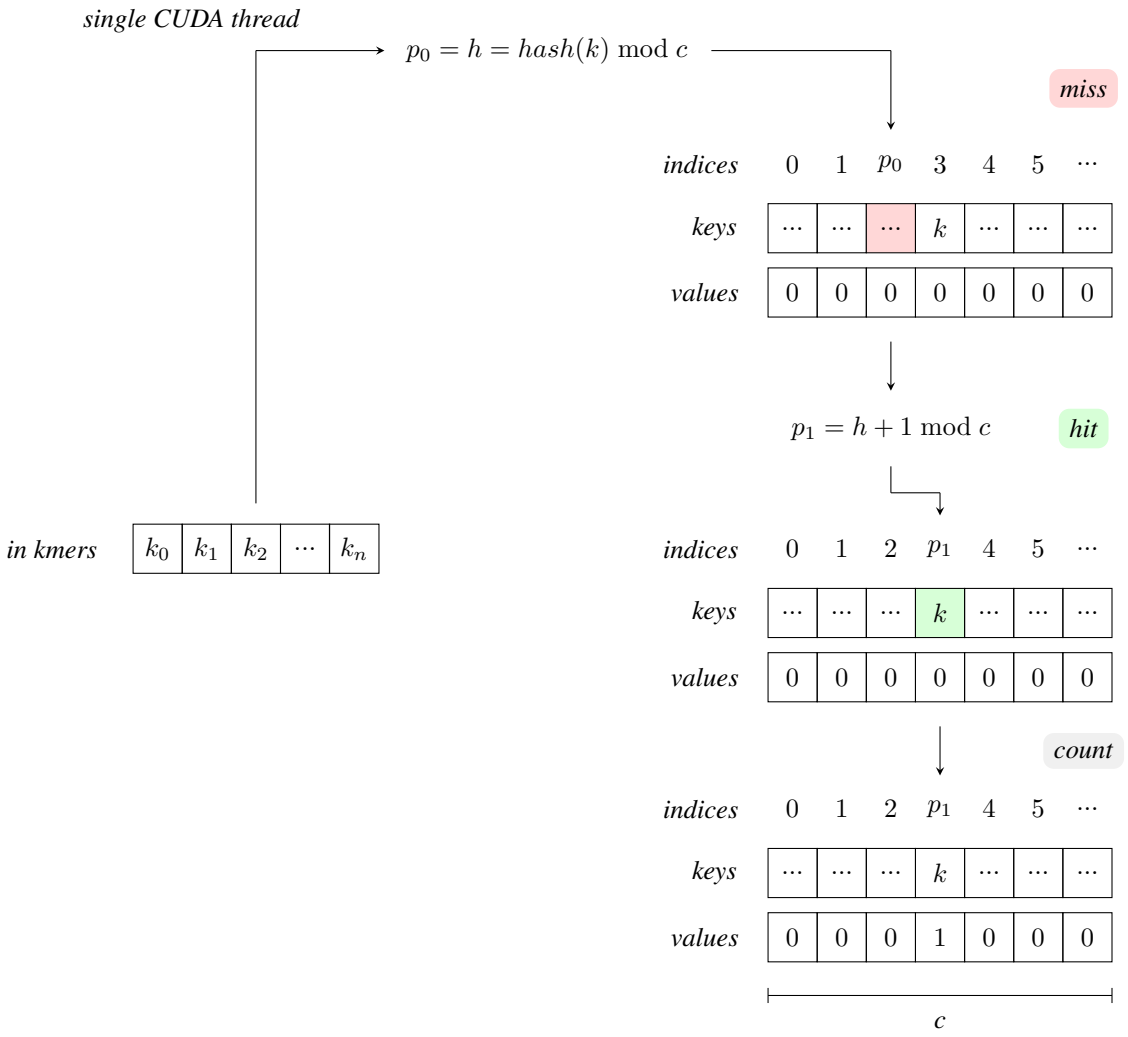

Figure 6: As an array of 64-bit integer encoded kmers are counted by the hash table, each CUDA thread will compute the first probe position p_0 for each individual kmer, and then continue probing by linearly moving up to the next consecutive slot until either an empty slot or the original kmer handled by the thread is observed. If en empty slot is observed, the thread terminates. If the original kmer is observed, the value at the current slot is increased.

5 Results

The following section will describe the benchmarks and results from this master's project. The benchmarking results provided in this chapter were determined by running the associated pieces of software on two systems: one high-end compute server, which will be referred to as *system 1*, and one consumer desktop computer, which will be referred to as *system 2*.

System	CPU	GPU
1: High-end compute server	AMD EPYC 7742	NVIDIA Tesla V100
2: Consumer desktop computer	Intel Core i5-11400F	NVIDIA GTX 1660 SUPER

5.1 *K*mer parsing from raw reads

Section (methods:cupy...) describes how GPU support was implemented for parts of BioNumPy [10] in order to allow for GPU acceleration when when parsing *k*mers from raw reads in FASTA files. In short: BioNumPy reads and parses chunks of *k*mers from FASTA files by 1) reading a chunk of raw bytes from the FASTA file, 2) converting each DNA nucleotide found in the raw read data into 2-bit encoded representations [2.3], and 3) parsing all valid *k*mers of the desired size from the 2-bit encoded reads data.

After copying the raw bytes that were read into memory from the FASTA file directly to the GPU memory, steps 2) and 3) could be performed significantly faster on the GPU for large enough chunk sizes, given the high throughput of the GPU.

...

5.2 *K*mer counting

...

5.3 GKAGE vs KAGE

...

6 Discussions

6.1 Drawbacks of Graphical Processing Units

While GPUs can be excellent for accelerating many problems in scientific computing, they do come with some notable caveats.

Expensive; power hungry; have experienced significant fluctuation in price and ease of access in the last couple of years (although seemingly mostly because of mining?); can be difficult to make effective solutions for someone inexperienced with the GPU compute models and frameworks; not suitable for every problem, only problems that are possible to rephrase as massively parallel, and it can be difficult to judge whether a problem is suitable before trying

7 Conclusion

References

[1] National Human Genome Research Institute. *Deoxyribonucleic Acid (DNA) Fact Sheet*. https://www.genome.gov/about-genomics/fact-sheets/Deoxyribonucleic-Acid-Fact-Sheet. Accessed: 2023-03-30. 2023.

[2] Gautam .B Singh. *Fundamentals of Bioinformatics and Computational Biology. Methods and Exercises in MATLAB*. Springer, 2015.

[3] National Human Genome Research Institute. *Chromosomes Fact Sheet*. https://www.genome.gov/about-genomics/fact-sheets/Chromosomes-Fact-Sheet. Accessed: 2023-03-31. 2023.

[4] Aaron McKenna et al. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data". en. In: *Genome Res* 20.9 (July 2010), pp. 1297–1303.

[5] Stepanka Zverinova and Victor Guryev. "Variant calling: Considerations, practices, and developments". en. In: *Hum Mutat* 43.8 (Dec. 2021), pp. 976–985.

[6] National Human Genome Research Institute. *Genotype*. https://www.genome.gov/genetics-glossary/genotype. Accessed: 2023-03-31. 2023.

[7] Ivar Grytten, Knut Dagestad Rand, and Geir Kjetil Sandve. "KAGE: Fast alignment-free graph-based genotyping of SNPs and short indels". In: *bioRxiv* (2021). DOI: 10.1101/2021.12.03.471074. eprint: https://www.biorxiv.org/content/early/2021/12/20/2021.12.03.471074.full.pdf. Address <https://www.biorxiv.org/content/early/2021/12/20/2021.12.03.471074>.

[8] Luca Denti et al. "MALVA: Genotyping by Mapping-free ALlele Detection of Known VAriants". In: *iScience* 18 (2019). RECOMB-Seq 2019, pp. 20–27. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2019.07.011. Address <https://www.sciencedirect.com/science/article/pii/S2589004219302366>.

[9] 1000 Genomes Project Consortium et al. "A global reference for human genetic variation". In: *Nature* 526.7571 (2015), p. 68.

[10] Knut Dagestad Rand and Ivar Grytten. *bionumpy*. https://github.com/bionumpy/bionumpy. 2022.