Creating De Bruijn graph from Oxford Nanopore's MinION data

Piotr Samborski
Bartłomiej Truszkowski

September 2020

Abstract

Purpose: Try to perform de novo genome assembly on input from Oxford Nanapore's MinION with the use of casual's computer GPU unit. Create De Bruijn graph with accessible format and perform data preprocessing to keep only K-mers with good enough quality .

Results: We managed to create graph from the entire chromosome (size ca. $17GB$) with the value of K up to 31

1 Introduction

1.1 Basic information about DNA

DNA is basic component of every living organism and contains various useful information. For average people knowledge about their DNA could help with disease diagnosis or diet preparation. Unsurprisingly, for biotechnologists this data can be even more useful and interesting.

1.2 DNA sequencing

DNA consists of 4 nucleotides: adenine, cytosine, guanine, thymine. DNA sequencing is the process which tries to determine theirs order within DNA.

1.3 Oxford Nanopore's MinION

MinION is a pocket-sized device which applies nanopore sequencing technology to nucleic acid analyses, with far reaching applications including real-time bacterial metagenomic community analysis, subtyping, and long read scafolding for whole genome sequencing of organisms.[1] MinION returns file in fastq format, which contains multiple reads and every read consists of:

1. Id

2. Sequence containing first letters of mentioned nucleotides, every first letter stands for nucleotide
3. + character
4. Sequence of characters which has the same length as second, contains information about quality of read in ASCII format, ! character is the lowest possible quality, while ∼ character is the highest

2 Reading data

2.1 Starting with data

Data is being read from the file directly into program. File is being open for read in binary mode and the reading is performed line by line. Each four lines are transferred to GPU. There the calculations start: the data is converted into a special code, and only K-mers with an appropriate quality are saved for further calculations.

2.2 Data format

For later calculations data should be as compressed as it is possible. For the compression an own code is used. Each K-mer is transformed to take up little space with the use of following method:

```
unsigned long long get_value(char c)
{
    unsigned long long value = 0;
    if (c == 'A')
        value = 0;
    else if (c == 'C')
        value = 1;
    else if (c == 'T')
        value = 2;
    else if (c == 'G')
        value = 3;
    return value;
}

C = 0; //Decoded value of K-mer
K // length of KMER (K-mer)

for (int i = 0; i < K; i++)
{
    $C \mathrel{+}= get\_value(KMER[i]) * 4^{K-i-1}$
}
```

It is the quad representation of a decimal number. It allows to store up to 31 length K-mer inside one "unsigned long long".

2.3 Compressed data advantages

Such compression gives an opportunity to keep half a billion of K-mers inside $4GB$ array. If we compare it to all amount of K-mers from chromosome, it is about 5%. Using only K-mers with enough quality allows to work on the entire $17GB$ file.

2.4 Step by step solution

1. Read four lines from the file and keep the second and the fourth line (lines which contains important data) inside arrays.
2. Copy data to GPU and start kernel with a loop
3. Loop is indexing by thread indexes inside blocks.
4. For each index (for each K-mer) there are calculating code and average or minimum quality.
5. If the quality of K-mer fulfills expectations, K-mer is being saved into array with good quality data, and will be processing in the next calculations.

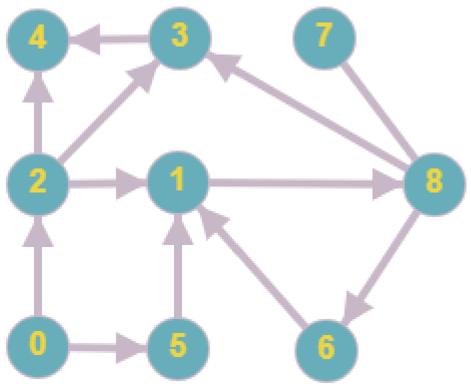
3 De Bruijn graph

3.1 Basic theory

The idea of De Bruijn graph is about splitting one node into two nodes connected with directed edge. For example, the given node (K-mer) AATTGC will be splitted into nodes: AATTG and ATTGC, and there will be directed edge from first to second. More precise information can be found here: [3]. In case of many reads from chromosome, De Bruijn graph gives opportunity to merge all of them. Reads will be split into smaller parts, but repeated nodes will create links between different reads, so it is possible that walk through the entire graph will return proper DNA sequence.

3.2 Compressed Sparse Row

Compressed Sparse Row is effective way to store large data. It can represent graph with only two arrays, which have length $O(number\ of\ edges\ in\ graph)$. First array - named C, contains all edges of the graph, precisely, stores destination nodes from every edge. Second array - named R, contains indexes of the beginning of edges for chosen node. Below is basic example of such conversion.



C =

2	5	8	1	3	4	4	1	1	8	3	6	7
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)

R =

0	2	3	6	7	7	8	9	10	13
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)

More about that format can be found here: [2]

3.3 Creating graph

Assume that K-mers in mentioned data format are already given. Thrust (CUDA library) enables creating De Bruijn graph in Compressed Sparse Row format with the use of GPU in easy way. Below is list of steps needed to prepare graph from those K-mers:

1. Sort all K-mers by value.
2. Reduce all K-mers by key, after that operation there will be one array with unique K-mers and second with numbers of appearances of each.
3. In order to get C array, get second $K-1$ mer from every unique K-mer (in thrust use transform, with function shifting bits).
4. Create similar array with all first $K-1$ mers, notice that this array is also sorted because of used data format, first bit has the biggest value, so skipping last bit will keep that property.
5. Reduce last array by key and keep array with numbers of appearances, name it NoA.

6. In order to get R array, use this recursion:
   $R[0] = 0$
   for $i > 0, R[i] = R[i-1] + NoA[i-1]$

4 Results

The solution has been tested for 5 different K values: $5, 10, 17, 24$ and 31, with two different variants of quality filters: average quality above threshold and quality of every K-mer's nucleotide above threshold. Time analysis was divided into 3 parts: time of allocating memory, time of reading data with conversion to preferable format and time of creating the graph. To perform computations for small values of K, number of bytes had to be reduced (K-mer's value was stored not in long long, but in smaller integer type), otherwise there would be not enough memory. Even with that change there were few cases when program failed to allocate memory. Because of that, some graphs will be lacking a part of the data.

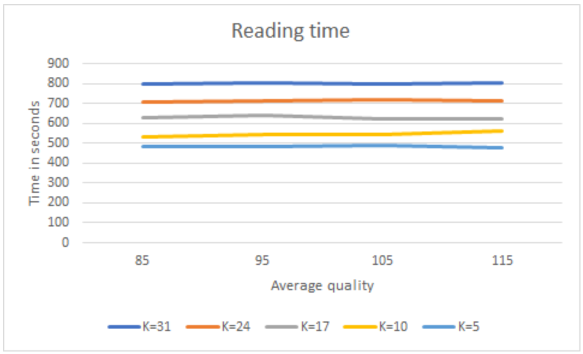
4.1 Time analysis

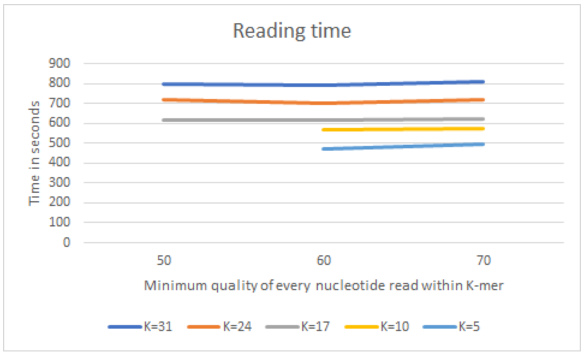
4.1.1 Allocating memory

Time of allocating memory varies between $9.0s$ and $9.5s$ and is similar for every exercised case.

4.1.2 Reading from the file and data conversion

Reading from file and data conversion last longer for higher K values. It is similar for every K value regardless of quality filter. Information is provided below.

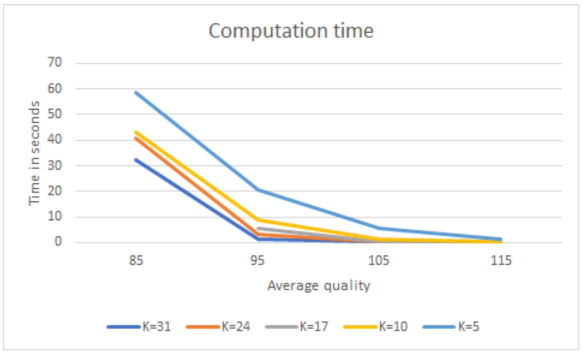


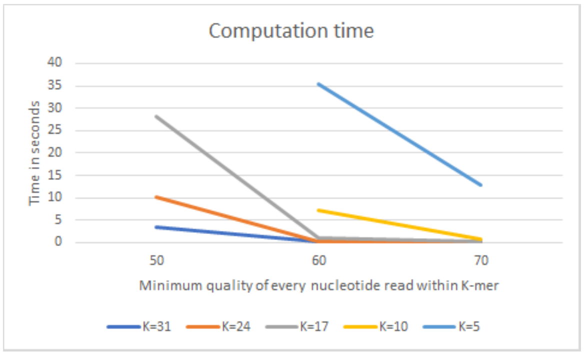


4.1.3 Computation time

Creating output graph lasts significantly less than reading data. It lasts a bit longer for small values of K, size of data to compute is probably a main reason.

Graphs with times are provided below.



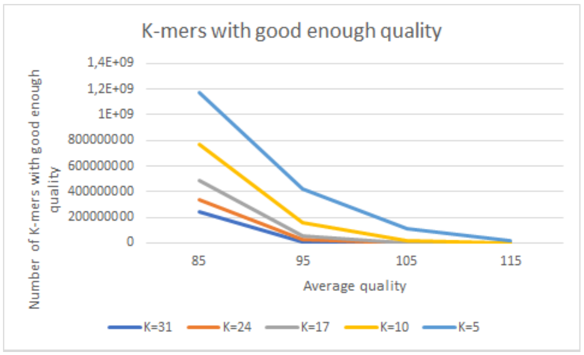


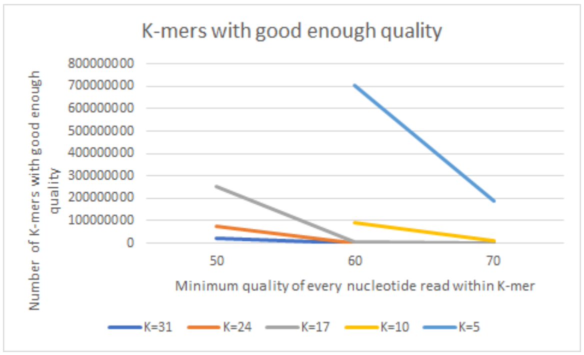
4.2 Results analysis

To start with, numbers of all processed K-mers are similar regardless of K value: 9 billion $+/-$ 2 million. All K-mers were filtered and only these with proper quality were saved for further computations.

4.2.1 All K-mers with good quality

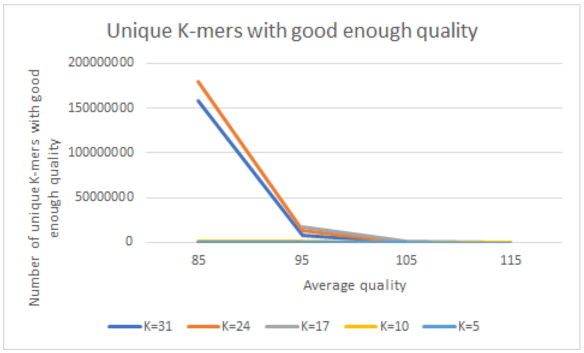
For both quality filters, K-mers with smaller value of K are more numerous. For minimum filter this is expected behaviour - chance of error grows exponentially with the value of K. Case with average quality is definitely more interesting. An average chance of error should be always similar regardless of K value, but, as data below shows - it is not. Possible explanation of this is that MinION could perform reads with very bad quality more often, that reads with very good quality and the mean is undervalued because of them.

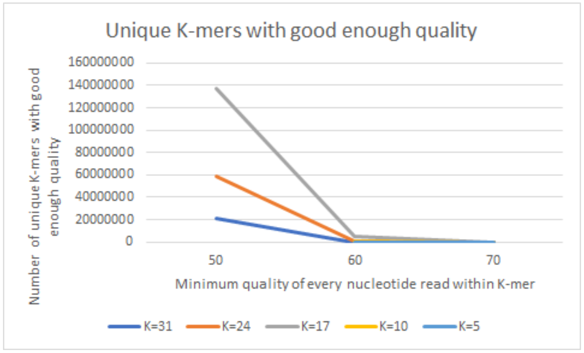




4.2.2 Unique K-mers with good quality

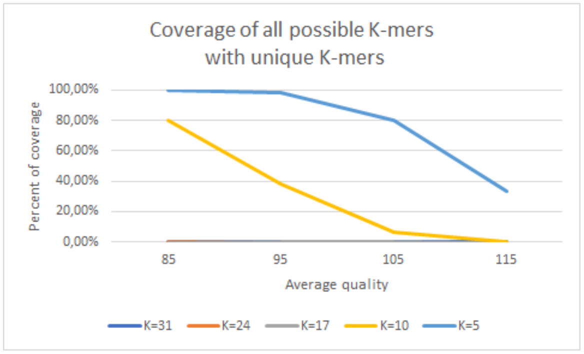
For minimum quality the highest number of unique K-mers is for $K = 17$, which can look surprising at the first glance, but it is not. Highest possible value of unique K-mers for $K = 5$ equals 1024 and for $K = 10$ equals 1048576 which are small in comparison to other values. On the other hand, minimum filter prefers low value K. For average filter the highest number is for $K = 24$, but this is only because measurement for $K = 17$ and $averagequalitythreshold = 85$ did not succeeded. For $threshold = 95$, $K = 17$ has the highest number of unique K-mers, so for 85 similar situation could be expected. Noteworthy is fact that differences in results of K-mers with $K > 10$ are definitely smaller. It is because amount of good quality K-mers is higher for smaller K, but on the other hand, number of all possible K-mers grows exponentially with the value of K.

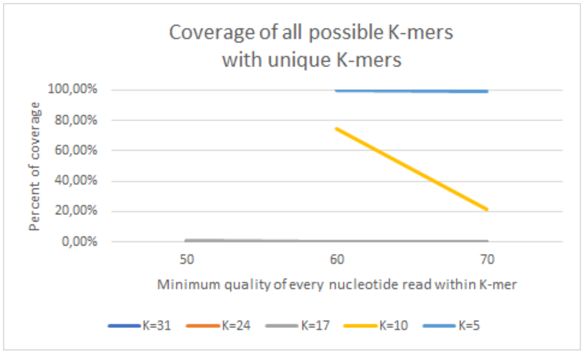




4.2.3 Coverage of all possible K-mers with unique K-mers

Graphs below shows that for $K = 5$ and $K = 10$ great part of all possible K-mers is in De Bruijn graph, while for $K \geq 17$ it is close to 0. It can imply two different conclusions. First is that for high values of K, chromosome contains only small percent of all possible K-mers. Other possible deduction is that chosen data filtering could not work properly for high values of K or exercised thresholds were too high.





4.3 Conclusions

Most interesting thoughts from the work:

1. The most important thing is that you can work on $17GB$ files, but only with strict rules of good quality K-Mers.
2. By using "unsigned long long" data type and special coding, it is possible to increase K-Mer length up to 31.
3. Reading time is related to K-Mer's length, but not to the chosen quality.
4. Computation time depends only on amount of good quality K-Mers.
5. For low K it is possible to get even 100% coverage K-Mers using only "good quality K-Mers".
6. It is very likely, that MinION produces very low quality reads more often, than very good quality reads.
7. It is possible that chromosome contains only very small percent of K-mers where $K \geq 17$

References

[1] Andrea D.Tyler et al. "Evaluation of Oxford Nanopore's MinION Sequencing Device for Microbial Whole Genome Sequencing Applications". In: *Nature* (2018).

[2] Krzysztof Kaczmarski, Piotr Przymus, and Paweł Rzażewski. "Improving High-Performance GPU Graph Traversal with Compression". In: (2014).

[3] Ben Langmead. *De Bruijn Graph assembly.* URL: `https://www.cs.jhu.edu/~langmea/resources/lecture_notes/assembly_dbg.pdf`.