Int. J. Advance Soft Compu. Appl, Vol. 11, No. 1, March 2019
ISSN 2074-8523

Genomic Repeat Detection Using the Knuth-Morris-Pratt Algorithm on R High-Performance-Computing Package

Lala Septem Riza[1], Achmad Banyu Rachmat[1], Munir[1], Topik Hidayat[2], Shah Nazir[3]

[1]Department of Computer Science Education, Universitas Pendidikan Indonesia, Indonesia
e-mail: lala.s.riza@upi, banyurachman95@gmail.com, munir@upi.edu
[2]Department of Biology Education, Universitas Pendidikan Indonesia, Indonesia
e-mail: topikhidayat@upi.edu
[3]Department of Computer Science, University of Swabi, Swabi, Pakistan
e-mail: snshahnzr@gmail.com

Abstract

Genomic repeat, which is to find repeating base pairs in Deoxyribonucleic Acid (DNA) sequences, can be used to detect genetic disease by analyzing the overload or over normal limits of the repetition. Since it takes very high computation cost, this research builds a parallel-computing model and its implementation to solve it. It can be achieved by modifying and implementing the Knuth-Morris-Pratt algorithm (KMP) on the R High-Performance-Computing Package, namely 'pbdMPI'. It contains the following steps: preprocessing and splitting DNA sequence, KMP on parallel computing with 'pbdMPI', combining all indices, and calculating genomic repeats. To validate the model and implementation, 114 experiments involving human DNA sequences are conducted on the standalone and parallel-computing scenarios. The results show that the proposed system can reduce the computation cost, which is more than 100 times faster than the standalone computing. Some comparisons of the computation cost in term of the numbers of batches and numbers of cores are presented along with the existing researches. In summary, the proposed model provides the significant improvement on the computational cost.

Keywords: *DNA, human genom, genomic repeats, string matching, Knuth-Morris-Pratt, high-performance computing.*

1 Introduction

Genome sequencing of many species allows scientists to study all gene devices and their interactions [1]. In the last decade scientists had to conduct laboratory research for 3 years to analyze DNA (i.e., Deoxyribonucleic Acid) [2]. One of the cases of DNA analysis that requires time and effort on such a large scale is to analyze diseases caused by repeated genomic patterns, called genomic repeats [3], such as three recurrent base pairs that can cause disease in the trinucleotide category repeat disorders [4].

Efforts from sequencing have generated enormous amount of data so that it also gave birth to a new field called bioinformatics. With the resulting sequence data, scientists can analyze biological interests by applying computational methods that allow for much more efficient analysis of time and energy than many research laboratories do today [5].

In analyzing the genomic repeats problem, a string matching or pattern matching analysis will be searched for in a large text. The basic algorithm for searching strings or patterns is to match all possibilities contained in the data from the first index in the text until it runs out. This algorithm is known as the brute force (Naïve) algorithm which has complexity with the worst possible is O (mn), which will take very long if more text will be used as string or pattern search object [6].

The need for searching strings or patterns in large data allows scientists to make algorithms more efficient than brute-force algorithms. Therefore, some string matching algorithms were developed such as the Knut-Morris-Pratt algorithm [7]. This most famous string search algorithm ultimately inspires other scientists to continue to develop more efficient algorithms. One of the development algorithms of Knuth-Morris-Pratt is the Ukkonen algorithm [8] and the Fast Hybrid Pattern-Matching Algorithm [9]. However, along with the development of the times and the increasing number of data generated in sequence, scientists should be able to solve computing problems with greater data [5].

Thus, computer scientists create a concept of parallel computing or distributed systems that enable a computing job to be completed by multiple cores, nodes or computers simultaneously. One of them is the MapReduce concept [10], which is the basis of Google's search technology on a large scale and allows scientists to also apply the MapReduce concept to various research cases. Another example is the Package High-Performance Computing in R programming languages, such as ‘Rmpi’ [11] and ‘pbdMPI’ [12] that develop parallel computing with MPI (Message Passing Interface) in the R programming language. Other examples are ‘randomForestSRC’ [13], ‘dclone’ [14], and etc. Moreover, the ‘foreach’ package in R (at https://cran.r-project.org/package=foreach) has been also implemented in many areas, such as parallel particle swarm optimization [15] and parallel exponential smoothing [16].

Various packages have their own procedures in their use as they need to be compiled at the prompt/terminal or can be done inside the console R itself. Also with the concept of programming such as data splitting to be analyzed, and so on. This research will modify and implement the Knuth-Morris-Pratt algorithm as the best string matching algorithm [17] on R Package High-Performance Computing 'pbdMPI' to be used for large datasets.

2 Genomic Repeats and Its Techniques

2.1 DNA and Genomic Repeats

The human genome has a size of approximately 3 billion base pairs out of a total of 23 chromosomes whose research begins with the Human Genome Project (HGP) beginning in 1990 [18], while the history for DNA sequencing was started by Sanger since 1997 [19]. DNA in humans is the same as the animal which is located in the nucleus of cells and mitochondria, in contrast to plants that also have DNA located on the chloroplast. DNA is a double-stranded, helical-shaped nucleic acid molecule composed of nucleotide monomers with deoxyribose sugar [1]. Most of the DNA lies in the nucleus cells but can also be found in mitochondria. DNA information is stored as code and into four chemical bases: adenine ("A"), guanine ("G"), cytosine ("C"), and thymne ("T"). The DNA bases have their respective pairs, "A" with "T" while "C" with "G". Each base is also attached to sugar molecules and phosphate molecules. Simultaneously, they are called nucleotide [20].

Genomic Repeats or repeated sequences are patterns of recurrent nucleic acids in the genome. Based on the sequence, the genomic repeats are divided into minisatellite and microsatellite as explained below:

a. Minisatellite: It is a recurrence of a nucleic acid pattern with 10-60 pairs of repeating bases approximately 5-50 times in a sequence [20]. In humans, the first minisatellite was discovered in 1980 [21].
b. Microsatellite: It is a recurrence of a nucleic acid pattern with 2-5 pairs of bases repeating about 5-50 times in a sequence [22]. This type is also often referred to as simple sequence repeats (SSR) by genetic scientists in plants. Patterns such as "TATATATATATA" are called dinucleotide microsatellite, whereas a pattern like "GTCGTCGTCGTC" is called trinucleotide microsatellite.

Moreover, the looping of short (microsatellite) patterns overload or over normal limits can cause genetic diseases. The study conducted by [23] explained that the three most important pairs of triplets/trinucleotides in human diseases are "CGG", "CCG", "CTG", "CAG", "GAA" and "TTC" in addition to the other 58 trinucleotides. The disease caused by these three repetitive bases is also called trinucleotide repeat disorders.

Basically, trinucleotide repeat disorders are divided into two groups based on recurrent base pairs: “CGG” or “CCG” loops as alternatives and “CAG” or “CTG” loops as alternatives [24]. Now both groups of trinucleotide repeat disorders are divided into Polyglutamine (PolyQ) Diseases and Nonpolyglutamine Diseases. Group of diseases caused by “CAG” bases (polyglutamine) is largely due to toxic protein mutant expansion function [25] as illustrated in Table 1.

Table 1: Trinucleotide repeat disorders included in Polyglutamine [25]

Disease	CAG Repeat Size	
	Normal	Disease
Spinobulbar Muscular Atrophy (Kennedy Disease)	9-36	38-62
Huntington’s Disease	6-35	36-121
Dentatorubral-pallidoluysian Atrophy (Haw-River Syndrome)	6-35	49-88
Spinocerebellar Ataxia Type 1	6-44	49-82
Spinocerebellar Ataxia Type 2	15-31	36-63
Spinocerebellar Ataxia Type 3 (Machado-Joseph Disease)	12-40	55-84
Spinocerebellar Ataxia Type 6	4-18	21-33
Spinocerebellar Ataxia Type 7	4-35	37-306

Table 1 shows some examples of trinucleotide repeat disorders that belong to the Polyglutamine group. An example of Huntington's Disease (HD) has a detectable repetition of disease in 36-121 which means there are loops containing “CAGCAGCAGCAGCAG” at least 36 times. Huntington's Disease which can furthermore be abbreviated as HD is usually a disease carried by offspring that can affect children's age even though. This disease is due to repetition of “CAG” located on N-terminus with the number of repetitions 36-121 times where it should have a normal repetition of 6-34 times [4].

2.2 Techniques on Genomic Repeats

String matching or also commonly called pattern matching or pattern searching is a technique that is included in the information retrieval. The use of current string matching techniques is used for many reasons, especially in information security, bioinformatics, plagiarism detection, text processing and document matching [17]. The use of string matching helps in real-time HTTP packet data rectification which inevitably results in the need for efficient string matching algorithms to be very important [26]. Another area that is not less popular for the use of string matching techniques is bioinformatics. Analyzing DNA sequences using string matching techniques can provide the information needed quickly. This is what causes string matching to be an interesting and important research topic in the field of computer science [27].

The Knuth-Morris-Pratt algorithm [7] was invented by three scientists, named Knuth, Morris and Pratt, to find the given string positions for text-editing programs. This algorithm provides prefix information on the string or pattern to search before searching. The Knuth-Morris-Pratt algorithm is very different from brute force in terms of the algorithm complexity of pattern matching. Meanwhile, the brute force algorithm (i.e., naïve algorithms) has the complexity of O (mn) because it matches all possibilities of each character in the text [28]. The complexity of the KMP-Prefix algorithm is O (m) where the variable 'm' is the sequence length of the pattern to be searched. The complexity for KMP-Search is O (n) where the variable 'n' is the sequence length of the text that becomes the search object. Therefore the overall complexity of the Knuth-Morris-Pratt algorithm is O (m + n). It means that the Knuth-Morris-Pratt algorithm is much faster than the brute force algorithm.

3 Parallel Computing in R

R is a programming language used for statistical analysis and graphics [29]. R was created by Ross Ihaka and Robert Gentlemen at Auckland University, New Zealand. Currently R language is developed by R Development Core Team. The R language has become the de facto standard among statisticians for the development of statistical software and is widely used for the development of statistical software.

The R programming language has many packages for parallel computing. One of them is the 'snow' which stands for Simple Network of Workstation [30] that can be used for simple parallel computing in the R programming language. The other package developed continuously is ‘snowFT’ or ‘snowfall’ [31]. The most striking difference of any package is ‘Rmpi’ using MPI, ‘rpvm’ using PVM, ‘pnmath’ using OpenMP, and ‘biopora’ using sockets. The package ‘pbdMPI’ is one of package for programming with Big Data in R (called ‘pbdR’) [12]. The significant difference between ‘pbdR’ and the usual R code is that ‘pbdR’ focuses on distributing memory systems, where data is distributed across multiple processors and analyzed at each branch by communication performed by MPI which makes it easier to use HPC.

There are two implementations of R on MPI, namely ‘Rmpi’ and ‘pbdMPI’. ‘Rmpi’ uses manager/workers parallelism where one main processor becomes control for the workers while ‘pbdMPI’ uses Single Program Multi Data parallelism (SPMD), where each processor is a worker and has a piece of data. This concept allows each processor to do the same job on different pieces of data for very large data [32].

4 Knuth-Moris-Pratt with the pbdMPI Package

In the implementation of the Knuth-Morris-Pratt algorithm on ‘pbdMPI’, we design a model to run the concept of parallel computing as shown in Fig 1. Detailed explanation can be found in the next subsections.

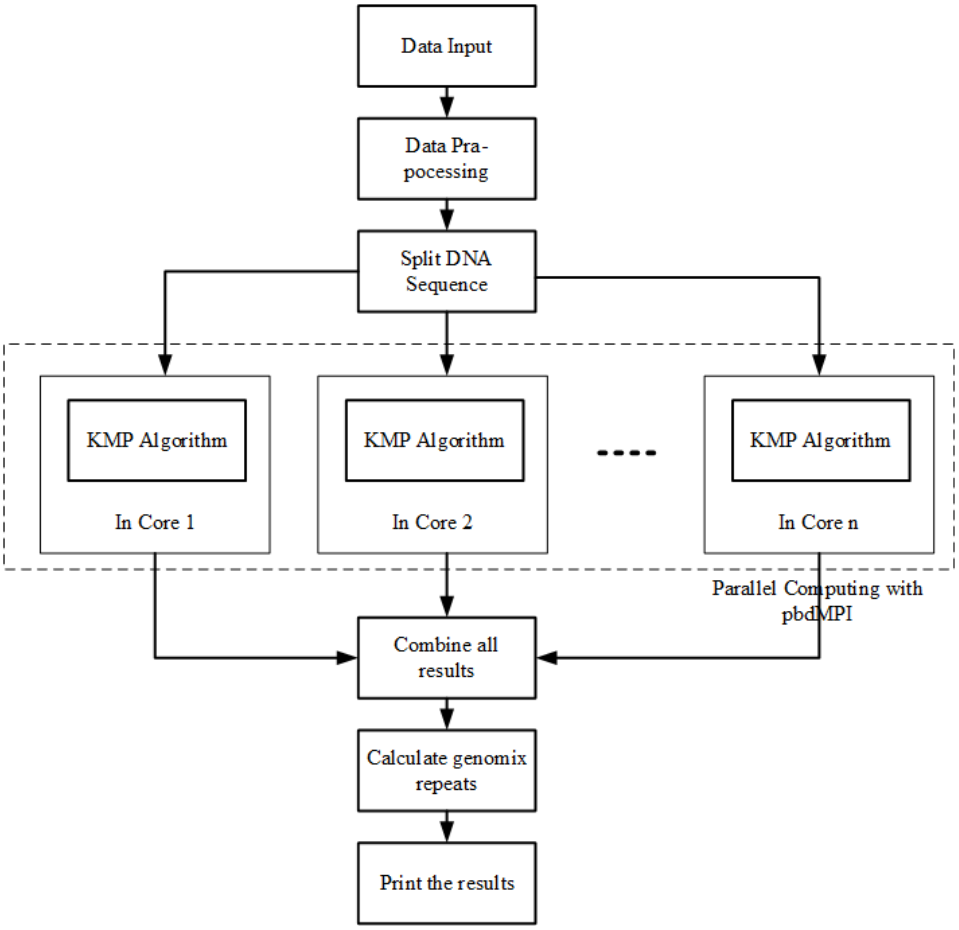

Fig. 1: The proposed model involving the Knutt-Morris-Pratt Algorithm in parallel computing with ‘pbdMPI’

4.1 Data Pre-processing

First, the data file to be used goes through the preprocessing stage to generate a clean string. Previously, the contents of the file has a lot of data information while in this study will only require sequences in it. The sequence in question is the data of base pairs “A”, “C”, “G”, “T” beginning after the word 'ORIGIN' and before the '//' symbol on the contents of the file as illustrated in Fig 2.

```
<……>
COMMENT     All the exons and transcripts in Ensembl are confirmed by similarity to
either protein or cDNA sequences.
ORIGIN
        1 CTACTGCTGC TACATCTGCT
       11 GTCGAT
//
```

Fig. 2: Original file containing DNA sequence

After preprocessing as described previously, the pre-processing results from the above data will be a string ready for use in the next stage, which is "CTACTGCTGCTACATCTGCTGTCGAT".

4.2 Splitting DNA Sequence

Generally, it is an important step in parallel computing since in this phase we have to split the datasets into some batches that are processed by each core. In this case, the datasets is in sequences of DNA. It should be noted that the objective is to find indices of the sequences that are matched with the pattern, which usually is much shorter sequence than the datasets. So, it can be seen that cutting or splitting the datasets could divide matched patterns inside the sequence so that the patterns are not recognized as the matched patterns. For example, we have a sequence containing 26 characters that will be divided into three batches/iterators. So, we can divide the data into 8 characters on the first and second batches and 10 characters for the last one as illustrated in Fig 3.

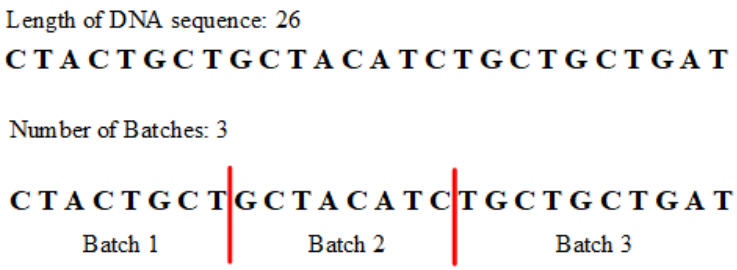

Fig. 3: Splitting DNA sequencing according to the number of batches

If the case is to find the pattern of "TCT" or "ATCT", we can see that the pattern cannot be found on every batch, even though the pattern is found in the complete/long sequence.

Therefore, we makes a mathematical formula of cutting in order to avoid missing patterns as mentioned earlier as follows:

$$L_{batch} = \left\lfloor \frac{L_{sequence}}{N_{batch}} \right\rfloor + (L_{pattern} - 1) \quad (1)$$

where L_{batch}, $L_{sequence}$, $L_{pattern}$, and N_{batch} are length of subsequence on each batch, length of sequence, length of pattern, and numbers of batches, respectively. For example, we try to find the same pattern as pervious, which is "ATCT". By following the equation, we obtain the batches as illustrated in Fig 4.

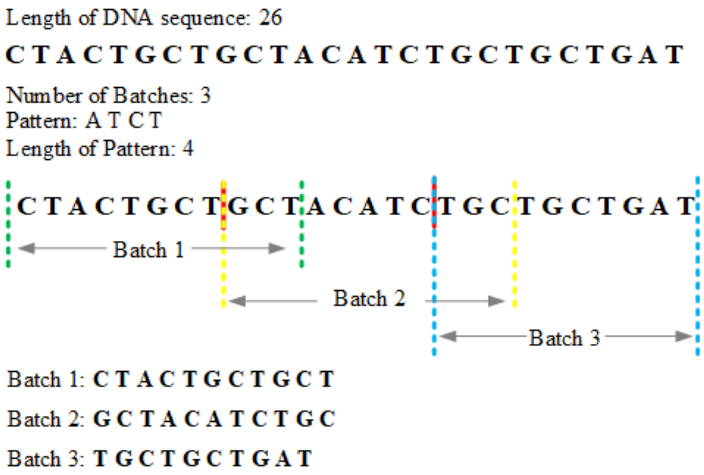

Fig. 4: The new rule on splitting DNA sequence in the proposed model

Now, it can be seen that by defining the new rule to generate each batch, we can find indices of the match pattern on all the batches, which are the same as the complete DNA sequence.

4.3 KMP Algorithm on Parallel Computing with pbdMPI

Basically, in this step we just perform KMP algorithm into the parallel computing with the 'pbdMPI' package. It should be noted that each core perform KMP over each batch. So, the output of the algorithm is a vector of indices of each batch. Moreover, we should increment the values of the indices so that along with the simulation the output of the KMP algorithm are not always begun from 1. Therefore, the authors create a model to change the index on the 2nd cut to the nth piece with the actual index pattern when the string state intact by creating an adder. The variables will participate in the search for the pattern on the string by using the Knuth-Morris-Pratt algorithm and used as the resulting index enhancer.

The number of adders required is as many as 1-piece. The adder value itself is the value of the number of characters used in the calculation as shown in Fig 5 multiplied by the order of each of the pieces that can be formulated as follows:

$$adder[i] = \left\lfloor \frac{L_{sequence}}{N_{batch}} \right\rfloor \times i \tag{2}$$

where, *i* is incremental number (i.e., 1, 2, …). So, for example, the output of each piece is as shown in Fig 5.

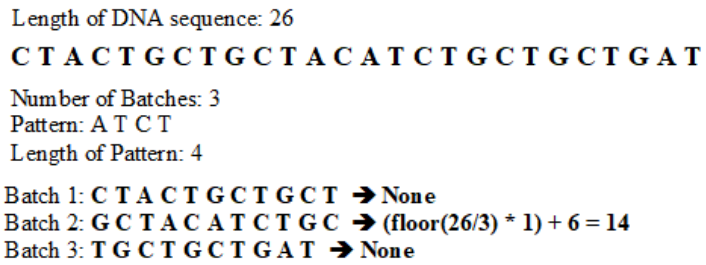

Fig. 5: Calculating indices on the match pattern

4.4 Combining All Indices

After the output of each string searching process using the Knuth-Morris-Pratt algorithm has produced the correct indices, the next step is to combine the result of each slice performed by each core into a list which in the next step will be searched on the indices to obtain how many patterns in the longest sequence occurs.

4.5 Calculating Genomic Repeats

In this final stage we take a look for patterns that appear side by side such as the problem of this research to look for repeating patterns of genetic diseases. In the indices of patterns that are closed are traced. If there is nothing adjacent at all, then the result of this stage will output 0. If it has one repetition sequence of any number, then the indices of patterns will be the result of this stage. If there are more than one, then this stage will release the results of index patterns with the largest number of pattern repetitions. For example on the previous DNA sequence, we obtain the indices of the match pattern with “CTG” are 16, 19, and 22. So the longest loop, which is three times of the repetition pattern “CTG”, is “CTGCTGCTG”.

5 Experimental Design

5.1 Data Gathering

The data used in this study is human DNA sequences that can be downloaded freely on the page ftp://ftp.ensembl.org/pub/release-88/fasta/homo_sapiens/dna/. The data are examples of human DNA sequences in publication number 88 provided on the File Transfer Protocol (FTP) Ensembl site. In this experiments, there are 24 DNA chromosome sequence files that can be seen in Table 2.

Table 2: Files of DNA sequence from FTP Ensembl

File Names	File Capacity (KB)	Length of Sequence
Homo_sapiens.GRCh38.88.chromosome.1.dat	356.015	248.956.422
Homo_sapiens.GRCh38.88.chromosome.2.dat	339.136	242.193.529
…	…	…
Homo_sapiens.GRCh38.88.chromosome.22.dat	77.049	50.818.468
Homo_sapiens.GRCh38.88.chromosome.X.dat	213.977	156.040.895
Homo_sapiens.GRCh38.88.chromosome.Y.dat	69.038	541.06.423
Total	4.028.943	3.085.148.840

The data in Table 2 show an example of a human DNA sequence comprising 24 chromosomes (chromosomes 1-22, X and Y) along with their size in KB and the number of base pairs present in it. The contents of the .dat file are composed of some information and the DNA sequence as the main data.

5.2 Scenarios of Experimentations

In conducting the experiments, we firstly need to design the scenario for the experiment. We perform two experiments. The first is to look for a string pattern using the Knuth-Morris-Pratt algorithm with standalone, then we take a look for patterns on the string using the Knuth-Morris-Pratt algorithm in parallel with the program created with some change of iterator variables and the number of cores used.

The pattern to be used is “CCG”, which is if found repeated as much as 200-900 times then it can be concluded that humans have Fagile XE Syndrome disease where in normal pattern “CCG” only repeated 4-39 times. Furthermore there is a pattern “CAG” which is the cause of the disease that belongs to the category polyglutamine. The differences in the “CCG” and “CAG” patterns lie not only on the difference of one character in the middle, but also have differences in the resulting prefix. The prefix for “CCG” is “0 1 0”, while the prefix for “CAG” is “0 0 0”. So, we investigate the computational cost caused by these different prefixes. Then, there is a pattern “TTAGGG” which is a telomere or the very end of linear DNA. The search “TTAGGG” is intended to see the effect of the pattern length on the difference in computational speed performed. In addition the selection of “TTAGGG” is also because it certainly exists in every human DNA sequence.

5.2.1 Scenario 1: Experiments in Standalone/Single Core

In the first scenario, we perform some experiments on standalone strings with pattern “CCG”, “CAG” and “TTAGGG” as seen in Table 3. It shows the experiments will search for “CCG”, “CAG” and “TTAGGG” patterns on chromosome 1, chromosome 4, and chromosome 14 files on single core/standalone.

Table 3: The first scenario using single core/standalone

No	Patterns	Filenames
1	CCG	Homo_sapiens.GRCh38.88.chromosome.1.dat
2	CCG	Homo_sapiens.GRCh38.88.chromosome.4.dat
3	CCG	Homo_sapiens.GRCh38.88.chromosome.14.dat
4	CAG	Homo_sapiens.GRCh38.88.chromosome.1.dat
5	CAG	Homo_sapiens.GRCh38.88.chromosome.4.dat
6	CAG	Homo_sapiens.GRCh38.88.chromosome.14.dat
7	TTAGGG	Homo_sapiens.GRCh38.88.chromosome.1.dat
8	TTAGGG	Homo_sapiens.GRCh38.88.chromosome.4.dat
9	TTAGGG	Homo_sapiens.GRCh38.88.chromosome.14.dat

5.2.2 Scenario 2: Experiments in parallel computing/multicores

In the second scenario, we conduct 108 attempts with regard to searching three patterns (i.e., “CCG”, “CAG”, and “TTAGGG”) on three files with four different batches performed by three cores: 2 cores, 4 cores and 8 cores as shown in Table 4.

Table 4: The second scenario using parallel computing/multicores

No	Pattern	File	Number of Batches	Core
1	CCG	Chromosome 1	50	2
2	CCG	Chromosome 1	50	4
3	CCG	Chromosome 1	50	8
…	…	…	…	…
21	CCG	Chromosome 4	500	8
22	CCG	Chromosome 4	2.500	2
23	CCG	Chromosome 4	2.500	4
…	….	…	…	…
28	CCG	Chromosome 14	100	2
29	CCG	Chromosome 14	100	4
30	CCG	Chromosome 14	100	8
…	…	…	…	…
107	TTAGGG	Chromosome 14	2.500	4
108	TTAGGG	Chromosome 14	2.500	8

6 Results and Discussion

After doing two experimental scenarios, the author gets the results that will be presented in the following subsections.

6.1 Results on the 1st Scenario (Standalone)

The results of the standalone experiments can be seen in Table 5. The 'Total Pattern' column shows the number of patterns found in the DNA sequence, while 'Longest Pattern Repeats' indicates the longest recurrence of the pattern in the DNA sequence. The 'time' column shows the amount of time to complete each computation in seconds.

Table 5: Results of experiments with standalone

No	Patterns	Files	Total Patterns	Longest Pattern Repeats	Time (s)
1	CCG	Chromosome 1	669.612	12	1821,48
2	CCG	Chromosome 4	386.822	10	667,84
3	CCG	Chromosome 14	243.060	9	340,31
4	CAG	Chromosome 1	4.852.390	12	81.439,60
5	CAG	Chromosome 4	3.456.386	19	42.836,94
6	CAG	Chromosome 14	1847242	11	11.931,92
7	TTAGGG	Chromosome 1	43.719	10	344,02
8	TTAGGG	Chromosome 4	35.503	11	302,90
9	TTAGGG	Chromosome 14	17.018	4	207,00

6.2 Results on the 2nd Scenario (Parallel Computing/Multicore)

Results from the second scenario, which is in parallel computing, can be seen in Table 6. It can be seen that the 'Total Pattern' column shows the number of patterns in the file/DNA sequence. While 'Longest Pattern Repeats' indicates the longest recurrence of the pattern in the file followed by 'Index Pattern Repeats' is the index of the pattern referred to in 'Longest Pattern Repeats'. The 'time' column shows the amount of time to complete each computation in seconds.

Table 6: Results of experiments with parallel computing

No	Patterns	Files	The numbers of Batches	Cores	Total Patterns	Longest Pattern Repeats	Time (s)
1	CCG	Chromosome 1	50	2	669.612	12	130,10
2	CCG	Chromosome 1	50	4	669.612	12	48,42
3	CCG	Chromosome 1	50	8	669.612	12	36,01
4	CCG	Chromosome 1	100	2	669.612	12	161,11
5	CCG	Chromosome 1	100	4	669.612	12	43,48
6	CCG	Chromosome 1	100	8	669.612	12	32,39
…	…	…	…	…	…	…	…
105	TTAGGG	Chromosome 14	500	8	17.018	4	12,57
106	TTAGGG	Chromosome 14	2.500	2	17.018	4	105,49
107	TTAGGG	Chromosome 14	2.500	4	17.018	4	26,78
108	TTAGGG	Chromosome 14	2.500	8	17.018	4	17,26

6.3 Discussion

From the experimental results obtained, we are able to analyze some aspects that will be presented in the following sections.

6.3.1 Comparison of Computation Cost among the Numbers of Batches

In this part, we attempt to analyze the effect of the numbers of batches on the computation cost. To provide better illustration, we only take into account the simulation with 8 cores. As illustrated in Fig 6, it can be seen that basically the computation costs of all simulations are not linear along with the batch numbers. Intuitively, we can state that the optimal batch number is 500 since in the average the computation time becomes small for all simulations.

6.3.2 Comparison of Computation Cost among the Numbers of Cores

Furthermore, we analyze the computation times in term of the numbers of cores as seen in Fig 7. It should be noted that Fig 7 was generated when the number of batches is 500. It can be seen that all simulations involving three different patterns (i.e., “CCG”, “CAG”, and “TTAGGG”) and three different cores (i.e., 2, 4, and 8) have the same trend. It shows that the higher number of cores have the faster computation. It means that the proposed model and its implementation have been successful to reduce the computation cost.

Moreover, we can also compare standalone with parallel computing. For example, the pattern matching of “CCG” on the file of the chromosome 1 took 1,821.48

seconds, which was more than 10, 37, and 50 times on parallel computing using 2, 4, and 8 cores in the average (i.e., 166.21, 48.78, and 35.30 seconds), respectively. In matching on "CAG", the comparisons of the standalone experiments with all the results of parallel computing experiments are very significant. In the chromosome 1, the standalone scenario took time more than 184, 528, and 809 times on the average time on 2, 4, and 8 cores. As for the search "TTAGGG", the experimental results of standalone and parallel computing are not too significant as the experiments for two previous patterns. The parallel-computing experiments using 8 cores had only ten times faster than the standalone in the average.

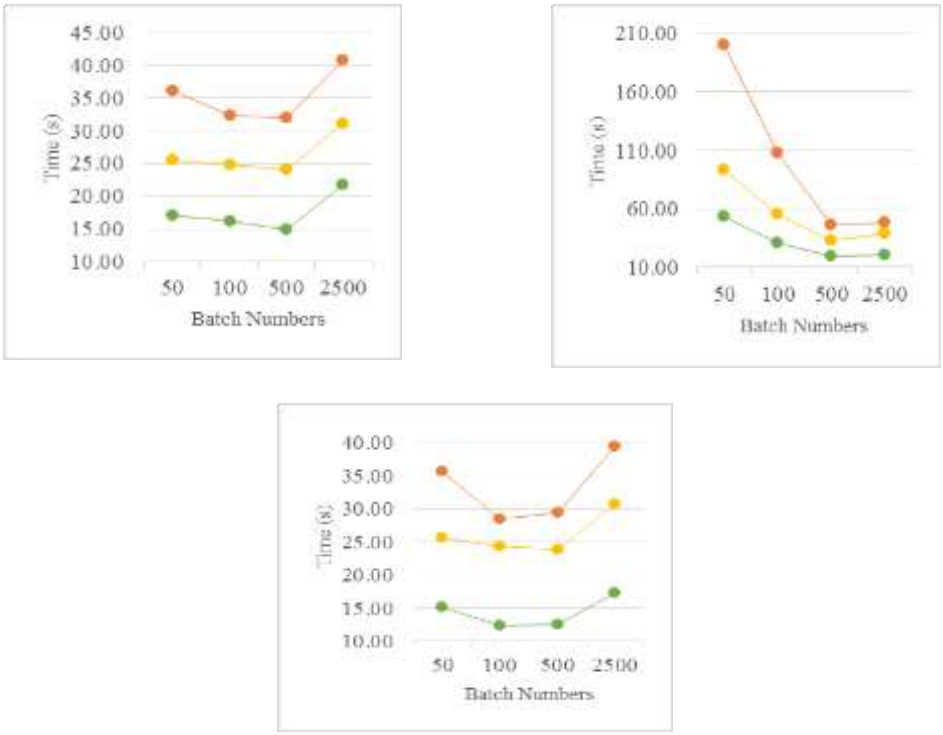

Fig. 6: Profiles of computation cost along with some batch numbers on the pattern matching of "CGG" (on the top left), "CAG" (on the top right), and "TTAGGG" (on the bottom) on the following chromosome 1 (orange line), 4 (yellow line), and 14 (green line).

6.3.3 Comparisons with the Previous Research

First, the previous research conducted by [33] used the suffix of patterns, unlike in this study which used prefix in the Knuth-Morris-Pratt algorithm itself. The research used data from DNA Data Bank Japan (DDBJ) at random. The results with two computers that run in parallel need to spend more than 100 seconds just to process a DNA sequence that has 1,000 base pairs while in this research using 2 cores only takes 86.99 seconds to search the sequence which has 107,043,718

base pairs. It means that this research can give a very good contribution on the computational cost.

In addition, the research by [34] performed a search pattern on DNA sequences and English text using 'OpenMPI'. The algorithm used in the research was the Boyer Moore string search algorithm. The study revealed that experiments using DNA data gave more stable results than English text. However, according to the research, the ratio of the computing time between standalone and parallel computing is 1:3. It means that the proposed model and its implementation in this research are better since it obtained the best ratio of 1:809.

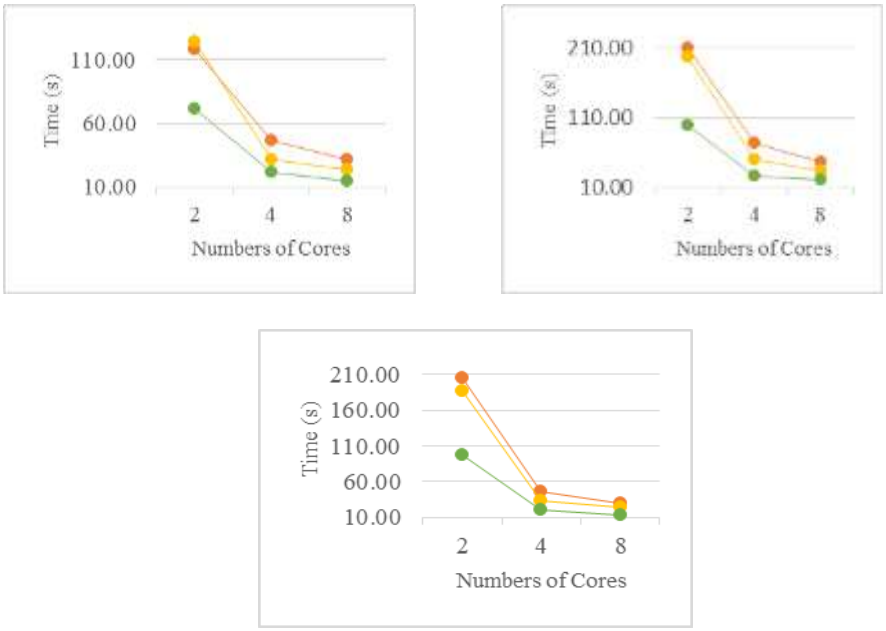

Fig. 7: Profiles of computation cost along with numbers of cores on the pattern matching of "CGG" (on the top left), "CAG" (on the top right), and "TTAGGG" (on the bottom) on the following chromosome 1 (orange line), 4 (yellow line), and 14 (green line).

7 Conclusions

After performing the research on genomic repeats detection program with Knuth-Morris-Pratt algorithm implementation on R package high-performance computing 'pbdMPI', we draw some conclusions as follows:

1. The proposed model has the concept of cutting the string so that no miss occurs. The model also introduces adder variables as an addition of the index value on the parallel computing.

2. The research has conducted 117 experiments divided into the standalone scenario (i.e., 9 simulations) and the parallel-computing scenario containing 108 simulations.
3. Analysis of experimental results has been presented. For example on matching on "CAG", the comparisons of the standalone experiments with all the results of parallel-computing experiments are very significant. In the chromosome 1, the standalone scenario took 184, 528, and 809 times slower than on the average time on 2, 4, and 8 cores.

References

[1] Campbell, N. A., and Reece, J. B. (2008). *Biology: Eight Edition.* San Francisco: Pearson Benjamin Cummings.

[2] Pahadia, M., Srivastava, A., Srivastava, D., and Patil, D. N. (2015). Genome Data Analysis using MapReduce Paradigm. *2015 Second International Conference on Advances in Computing and Communication Engineering* (pp. 556-559). IEEE.

[3] Edgar, R. C., and Myers, E. W. (2005). PILER: Identification and Classication of Genomic Repeats. *Bioinformatics, 21*(1), 152-158.

[4] Orr, H. T., and Zoghbi, H. Y. (2007). Trinucleotide Repeat Disorders. *Annual Review of Neuroscience, 30*, 575-621.

[5] Liu, B., Li, J., Chen, C., Tan, W., Chen, Q., and Zhou, M. (2015). Efficient Motif Discovery for Large-Scale Time Series in Healthcare. *IEEE Transactional on Industrial Informatics, 11*(3), 583-590.

[6] Kindhi, B. A., and Sardjono, T. A. (2015). Pattern Matching Performance Comparison as Big Data Analysis Recomendations for Hepatitis C Virus (HCV) Sequence DNA. *2015 Third International Conference on Artificial Intelligence, Modelling and Simulation* (pp. 99-104). IEEE.

[7] Knuth, D. E., Morris, J. H., and Pratt, V. R. (1977). Fast Pattern Matching In Strings. *SIAM Journal on Computing, 6*(2), 323-350.

[8] Ukkonen, E. (1985). Finding Approximate Patterns in Strings. *Journal of Algorithms, 6*, 132-137.

[9] Franek, F., Jennings, C. G., and Smyth, W. F. (2005). A Simple Fast Hybrid Pattern Matching Algorithm. *CPM*, 288-297.

[10] Dean, J., and Ghemawat, S. (2010). MapReduce: A Flexible Data Processing Tool. *Communications of The ACM, 53*, 72-77.

[11] Yu, H. (2002). Rmpi: Parallel Statistical Computing in R. *R News, 2*(2), 10-14.

[12] Ostrouchov, G., Chen, W.-C., Schmidt, D., and Patel, P. (2012). *Programming with Big Data in R*. Retrieved from http://www.r-pbd.org

[13] Ishwaran, H., and Kogalur, U. (2007). Random Survival Forests for R. *R News, 7*(2), 25-31.

[14] dclone: Data Cloning in R. (2010). *The R Journal, 2*(2), 29-37.

[15] Riza, L. S., Asyari, A. H., Prabawa, H. W., Kusnendar, J., & Rahman, E. F. (2018a). Parallel Particle Swarm Optimization for Determining Pressure on Water Distribution Systems in R. Advanced Science Letters, 24(10), 7501-7506.

[16] Riza, L. S., Utama, J. A., Putra, S. M., Simatupang, F. M., & Nugroho, E. P. (2018b). Parallel Exponential Smoothing Using the Bootstrap Method in R for Forecasting Asteroid's Orbital Elements. *Pertanika Journal of Science & Technology, 26*(1), 441 - 462.

[17] Vijayarani, D., and Janani, M. (2017). String Matching Algorithms For Reteriving Information From Desktop – Comparative Analysis. *International Conference on Inventive Computation Technologies (ICICT).* IEEE.

[18] Venter, J. C., Adams, M., Myers, E., Li, P., Mural, R., Sutton, G., . . . Skupski, M. (2001). The Sequence of the Human Genome. *Science, 291*(5507), 1304–1351.

[19] Sanger, F., Air, G. M., Barrell, B. G., Brown, N. L., Coulson, A. R., Fiddes, J. C., ... & Smith, M. (1977). Nucleotide sequence of bacteriophage φX174 DNA. nature, 265(5596), 687.

[20] U.S. National Library of Medicine. (n.d.). *What is DNA?* Retrieved Februari 8, 2017, from Genetics Home Reference: https://ghr.nlm.nih.gov/primer/basics/dna

[21] Wyman, A. R., and White, R. (1980). A Highly Polymorphic Locus in Human DNA. *Proc. Natl. Acad. Sci. U.S.A., 77*(11), 6754-6758.

[22] Turnpenny P, E. S. (2005). *Emery's Elements of Medical Genetics (12th ed.).* London: Elsevier.

[23] Calladine, C. R., Drew, H. R., Luisi, B. F., and Travers, A. A. (2004). *Understanding DNA: The Molecule and How It Works (Third Edition).* San Diego: Elsevier Academic Press.

[24] Claude T., Ashley, J., and Warre, S. T. (1995). Trinucleotide Repeat Expansion and Human Disease. *Annual Reviews, 29*, 703-728.

[25] Lutz, R. E. (2007). Trinucleotide Repeat Disorders. *Seminars in Pediatric Neurology, 14*, 26-33.

[26] Zhang, L., Peng, Y., Liang, J., Liu, X., Yi, J., and Wen, Z. (2015). An Improved String Matching Algorithm for HTTP Data Reduction. *2015 International Conference on Intelligent Information Hiding and Multimedia Signal Processing* (pp. 345-348). IEEE.

[27] Yangjun Chen, Y. W. (2016). On the Massive String Matching Problem. *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)* (pp. 350-355). IEEE.

[28] Barth, G. (1981). An Alternative For The Implementation of The Knuth-Morris-Pratt Algorithm. *Information Processing Letters, 13*, 134-137.

[29] Ihaka, R., & Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, *5*(3), 299-314.

[30] Rossini, A., Tierney, L., dan Li, N. (2007). Simple Parallel Statistical Computing in R. *Journal of Computational and Graphical Statistics, 16*(2), 399–420.

[31] Knaus, J. (2008). sfCluster/snowfall: Managing Parallel Execution of R Programs on a Compute Cluster. *useR!*

[32] Houston, M. (2007). *Folding@Home - GPGPU*. Retrieved from http://graphics.stanford.edu/~mhouston/

[33] Cheng, L.-L., Cheung, D. W., dan Yiu, S.-M. (2003). Approximate String Matching in DNA Sequences. *8th International Conference on Database Systems for Advanced Applications.* Kyoto, Japan: IEEE.

[34] Al-Dabbagh, S. S., Barnouti, N. H., Naser, M. A., dan Ali, Z. G. (2016). Parallel Quick Search Algorithm for the Exact String Matching Problem Using OpenMP. *Journal of Computer and Communications, 4*, 1-11.