CoreID JOURNAL

Vol. 2 No. 2, July 2024 (52-59)
ISSN: 2987-6990
DOI: https://doi.org/10.60005/coreid.v2i2.34

Investigating the Performance of Serial and Parallel Smith-Waterman Algorithm Implementations for Genetic Sequence Alignment Using OpenMPI

Nita Dwi Fitriani[1], Silmi Rahma Amelia[2], Fahdzi Muttaqien[3], Atthar Luqman Ivansyah[4]
[1,2,3,4]Master Program in Computational Science, Faculty of Mathematics and Natural Sciences,
Bandung Institute of Technology, Bandung, 40132, Indonesia

Article Info

Article history:

Received June 30, 2024
Revised July 20, 2024
Accepted July 26, 2024

Keywords:

Deoxyribonucleic acid (DNA)
Parallel
Sequence Alignment
Serial
Smith-Waterman

ABSTRACT

Deoxyribonucleic acid (DNA) is composed of nucleotide chains containing nitrogenous bases, phosphate groups, and pentose sugars, with variations primarily occurring in the sequence of nitrogenous bases. The analysis of DNA sequences often employs the Smith-Waterman algorithm for sequence alignment, a fundamental technique in bioinformatics. This research evaluates the performance of the Smith-Waterman algorithm across varying sequence lengths (10, 10^2, 10^3, and 10^4) using both serial and parallel implementations with the OpenMPI library. The study focuses on measuring execution times and speedup on 4, 6, 10, 12, and 24 cores. Results indicate that while execution times increase with longer sequences, parallelization significantly reduces processing time for sequences longer than 10^2. However, smaller sequences exhibit higher overhead on shorter lengths. The findings underscore the importance of efficient parallel programming and task allocation strategies in optimizing computational performance for DNA sequence analysis.

Corresponding Author:

Nita Dwi Fitriani
Master Program in Computational Science, Faculty of Mathematics and Natural Science,
Institut Teknologi Bandung, Jalan Ganesha No. 10, Bandung, 40132, Indonesia.
Email: 20922302@mahasiswa.itb.ac.id

1. INTRODUCTION

Deoxyribonucleic acid (DNA) is a polymer composed of repeating nucleotide monomers connected by the phosphate group with the pentose sugar through carbon number 5 by a covalent ester bond, as shown in figure. 1. Each DNA consists of two polymer strands twisted to form a helix structure [1], [2].

As a constituent of nucleic acids, DNA is a genetic information repository. The information in DNA is encoded in a sequence of nitrogenous bases, which is a combination of purine bases consisting of adenine (A) and guanine (G) and pyrimidine bases consisting of thymine (T) and cytosine (C)). DNA sequences can be aligned to study and identify the similarity of the differences between two DNA sequences [3], [4]. This technique is called sequence alignment. The analysis data can be the basis for steps in other biological processes, such as predicting the structure and analyzing the DNA function of a particular organism [5] so that no errors occur in handling a case. In research conducted by Mohamed Issa and Mohamed Abd Elaziz, the coronavirus (COVID-19), a new virus, was identified by aligning the DNA sequence of COVID-19 with other viruses [6]. Other researchers use the results of this research to find effective drugs that can inhibit or kill the activity of COVID-19.

DNA sequences from various organisms have been decoded and stored in a database. With the growing amount of data, the number of DNA databases has increased, making manual analysis of DNA sequences impractical as it can take much time and is sensitive to analysis errors. Therefore, fast tools and algorithms are needed to overcome this problem.

To find a good sequence alignment, the gaps and nitrogenous bases that do not match when two DNA sequences are compared have a minimum number, and the nitrogenous bases that match the reference DNA sequence have a maximum number. The algorithm that can be used to solve sequence alignment problems is a dynamic programming algorithm [7]–[9]. This algorithm is classified into two types global and local. The Needleman-Wunsch algorithm is generally used for the entire sequence (global) analysis, which was discovered by Saul B. Needleman and Christian D. Wunsch in 1970. Meanwhile, to find the best subsequence (local) match between two sequences, Temple F. Smith and Michael S. Waterman developed the local alignment algorithm in 1981, known as the Smith-Waterman algorithm [10]. Compared to global alignment, local alignment is more suitable for DNA sequence alignment because DNA is a molecule that tends to have many repeats, so there is a possibility that the two DNA sequences being compared have similarities only in certain parts, and an extensive database will be used in the alignment process, so global alignment will take longer and require more computational cost than local alignment.

The results of research conducted by Ernawati., et al. show that the calculation time for comparing Wilms tumor DNA at local AF484671 and locus AK290854 using the Smith-Waterman algorithm serially is 0.178 seconds with a relatively small sequence size, which is 100 x 100 [1]. Comparison of sequences DNA using the smith-waterman algorithm was also carried out by Chaibou, Amadou, and Sie, Oumarou serially and in parallel using OpenMP and Cuda C, and the resulting parallel calculations significantly reduced calculation time [11]. Parallelization with OpenMP is also used by Muhama, FN et al. in analyzing the performance of the Needleman-Wunsch algorithm and the Smith-waterman algorithm in the sequence alignment [12]. The results show that with four cores, the execution time of the code may be decreased by around 60% because of the parallelization capabilities of OpenMP directives, and the execution time of DNA sequencing using the Smith-waterman algorithm is faster than using the Needleman-Wunch algorithm[13], [14].

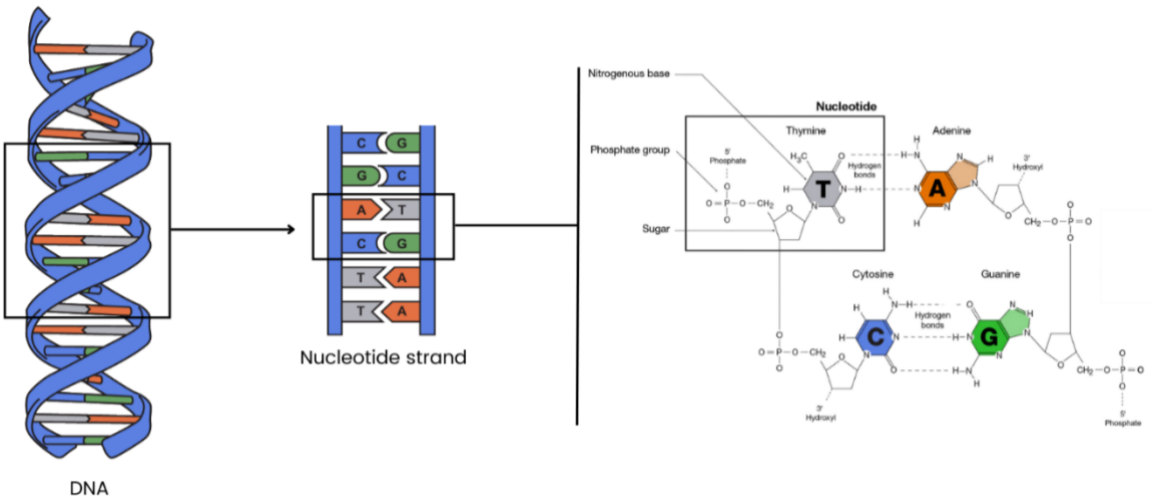

Figure. 1 Structure of DNA

So, the serial calculation process in sequence alignment analysis generally takes a long time for extensive data because the data depend on each other. To solve the dependency data, antidiagonal layout can used for parallelization technic so the data to be independent and the calculation can be faster than serial calculation. This parallel calculation process can use OpenMPI, an open-source Message Passing Interface (MPI) implementation[15]. However, although it can be accessed quickly and for free, OpenMPI is for the parallelization process of sequence alignment in the Smith-Waterman algorithm, so far, it has yet to be done. Therefore, in this study, DNA sequence alignment using the Smith-Waterman algorithm was performed serial and parallel using OpenMPI[16], [17]. In addition to the alignment results, serial and parallel calculation times were identified. In addition to the alignment results, serial and parallel calculation times in processor cores 4, 6, 10, 12, and 24 were also identified.

2. METHOD

The Smith-Waterman algorithm consists of three main stages initialization, matrix filling, and traceback, as shown in Figure.2 The difference between the serial and parallel calculation procedures lies in using OpenMPI in the parallel program as a library so that it can be run on systems with multiple processors [13], [18], [19].

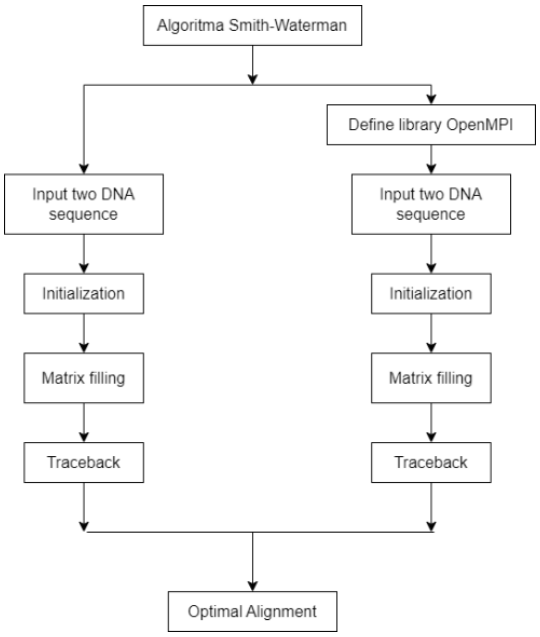

Figure. 2 Ilustration of Smith-Waterman Algorithm for serial and parallel program

2.1 Serial Program

The Smith-Waterman algorithm compares all possible pairs and assigns a score to determine the similarity between two DNA sequences (S and T). Let $S = s_1, s_2, \dots, s_m$ with length m and $T = t_1, t_2, \dots, t_n$ with length n be the two sequences being compared. A matrix M of size $(m+1) \times (n+1)$ is constructed to calculate the maximum score between two sequences, S and T. M is indexed by i and j, where each index corresponds to one sequence. Specifically, cell M_{ij} can be recursively constructed using repeated equations (1) and (2)[20].

$$for\ 0 \le i \le m, 0 \le j \le n,$$
$$M_{i0} = M_{0j} = 0 \quad (1)$$
$$for\ 1 \le i \le m\ dan\ 1 \le j \le n,$$
$$M_{ij} = max \begin{cases} M_{(i-1)(j-1)} + Sbt\ (s_i, t_j), \\ M_{(i-1)j} - gap\ cost\ t, \\ M_{i(j-1)} - gap\ cost\ t. \end{cases} \quad (2)$$

In addition to constructing matrix D of size $(m+1) \times (n+1)$, the initialization stage also involves populating cells using equation (1). The $gap\ cost\ t$ in equation (2) represents the penalty value if one character from the two sequences being compared is shifted and replaced with the $gap\ cost\ t$ character. The $gap\ cost\ t$ value must be negative because it is assumed to require additional effort to shift those characters after the gap is inserted. The substitution matrix value $Sbt\ (s_i, t_j)$ in matrix D represents the score for matching and mismatching. The match value is used when two characters are found to be the same, so its value must be set as positive, and conversely for the mismatch value, which is negative and given when the two characters being compared are different. In this study, the match value is set to 3, the mismatch value is -3, and the gap value is -2[18], [19].

Using equation (2), the second stage, matrix filling, can be performed by taking the highest value (maximum value) of the three calculated values. If a negative value is found, it is replaced with 0. Thus, this filling process will only focus on regions of similar sequences. The final step, the traceback, is done by tracing back the maximum value from the bottom right corner until 0. All these calculation steps are shown on the flowchart in Figure.3

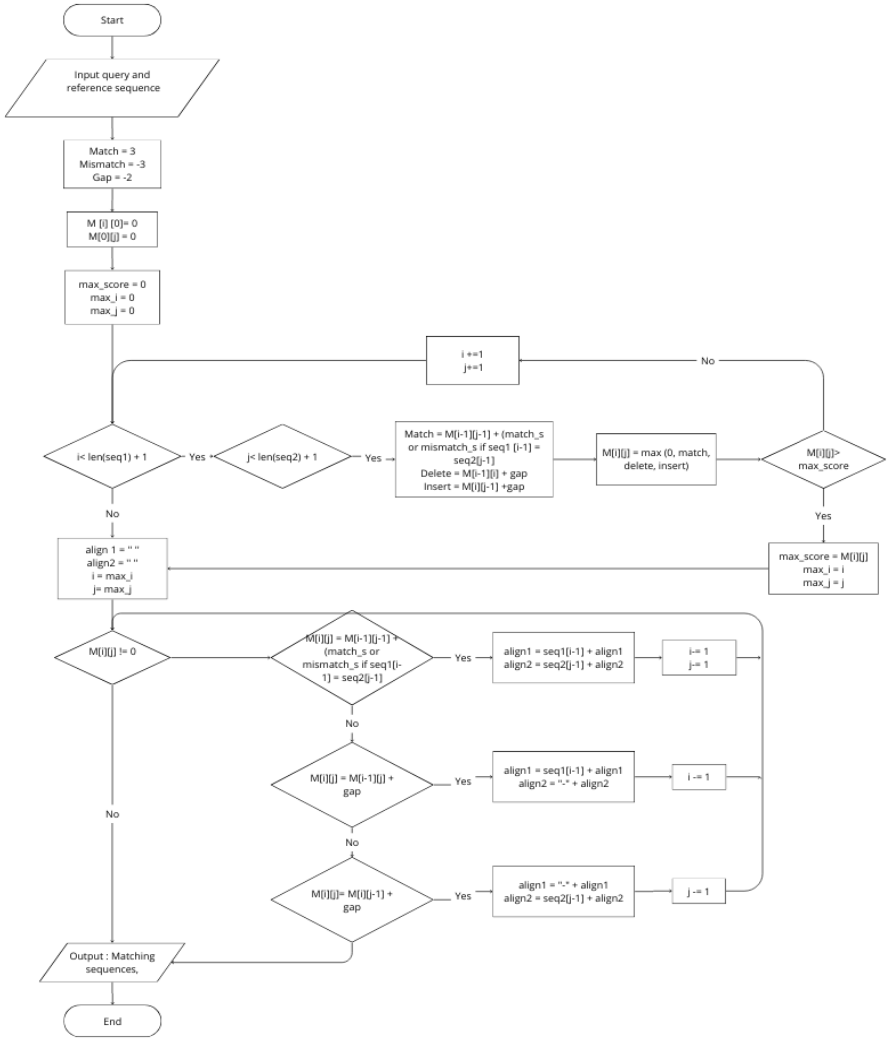

Figure. 3 Flowchart of Smith-Waterman Algorithm

2.2 Parallel Program

The parallel program that can be used in the Smith-Waterman algorithm is the anti-diagonal layout, first introduced by Wozniak in 1997. Figure 3 shows the calculation of M_{ij} and $M_{(i-1)(j+1)}$. We can see that the two cells in the anti-diagonal position are independent by evaluating the data dependencies in the figure. So that, theoretically, the calculation of the two cells can be calculated in parallel computation [20].

The length of S and T sequences, as in the serial part, is $m \times n$. The diagonal index (d) is defined in equation (3). For diagonal d, the starting row index (row_s) and the last row index (row_e) can be calculated in equation (4)[12].

$$d = x + y - 1 (1 \leq d \leq m + n - 1) \quad (3)$$

$$row_s = max(1, d - m) \quad (4)$$

$$row_e = min(d, m)$$

$$col_s = d - row_s \quad (5)$$

$$col_e = d - row_e$$

Based on equation (4), the starting column index (col_s) and the ending column index (col_e) can be calculated using equation (5). At the same time, the number of Nd cells on the diagonal d is $row_s - row_e + 1$. That way, the for-loop executed in the total calculation is d times with each for-loop the chunk of data Nd also needs to be calculated. Thus, several dummy symbols are added which do not affect the final result of the query and reference sequence to avoid Nd, which sometimes cannot be divided by the number of elements processed in the SIMD register. When counting each cell (i, j) on the d diagonal, the value (i−1,j−1) on the diagonal d − 2 and the value (i,j−1), (i,j−1), (i−1, j) and (i−1,j) on the diagonal d − 1 are required (Figure. 4) [18].

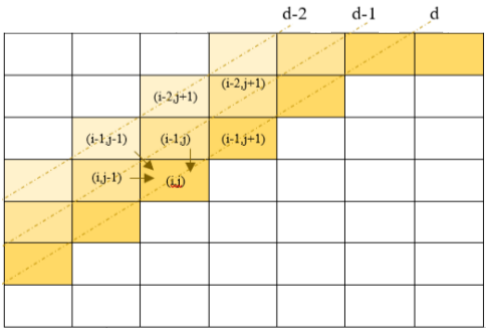

Figure. 4 Data dependency of antidiagonal layout in Smith-waterman algorithm

3. RESULT AND DISCUSSION

3.1 Analysis of Execution Time on Serial and Parallel Programs

This study used the Smith-Waterman algorithm to determine the similarity between two nitrogenous base sequences present in DNA. In this algorithm, a score matrix is used to find similarities between two sequences, where each cell in the matrix represents a similarity score of two characters in the sequence. In each iteration, scores within the matrix are calculated and used to populate new cells. Once the whole score matrix is calculated, the algorithm performs a traceback to determine which pair of characters gives the highest similarity score. On Figure. 5(a) and 5(b) show the results of comparing two DNA sequences run in serial and parallel programs.

a) DNA Reference: TAGAAGAATT

DNA Query: GCGAAGAAAA

		G	C	G	A	A	G	A	A	A	A
T	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	5	5	3	5	5	5	5
G	0	5	3	5	3	5	10	8	6	5	5
A	0	3	5	3	10	8	8	15	13	11	10
A	0	1	3	5	8	15	13	13	20	18	16
G	0	5	3	8	6	13	20	18	18	20	18
A	0	3	5	6	13	11	18	25	23	23	25
A	0	1	3	5	11	18	16	23	30	28	28
T	0	0	1	3	9	16	18	21	28	30	28

Matching sequence :
['T', 'A', 'G', 'A', 'A', 'G', 'A', 'A']

[' C', 'G', 'G', 'A', 'A', 'G', 'A', 'A']

b) DNA Reference: AACTAATATG

DNA Query: CACGGGGGGC

		C	A	C	G	G	G	G	G	G	C
G	0	0	0	0	0	0	0	0	0	0	0
A	0	0	5	3	1	0	0	0	0	0	0
A	0	0	5	5	3	1	0	0	0	0	0
C	0	5	3	10	8	6	4	2	0	0	5
T	0	3	5	8	10	8	6	4	2	0	3
A	0	1	8	6	8	10	8	6	4	2	1
A	0	0	6	8	6	8	10	8	6	4	2
T	0	0	4	6	8	6	8	10	8	6	4
A	0	0	5	4	6	8	6	8	10	8	6
T	0	0	3	5	4	6	8	6	8	10	8

Matching sequence :
['A', 'A', 'C', 'T', 'A', 'A', 'T', 'A', 'T', 'G']

['C', 'A', 'C', 'G', 'G', 'G', 'G', 'G', 'G', 'C']

Figure. 5 Result of sequence alignment in a) serial and b) parallel program

Although in Figure 5, only the results of the length of sequence 10×10 were displayed to identify the effect of DNA sequence size on execution time, experiments were also carried out on DNA sequences with sizes 10, 102, 103, and 104. Execution time analysis measures the program's speed in completing the alignment process of the two sequences compared. Table 1 and Figure. 6 compare the execution time between serial programs using one processor core and parallel programs using processor cores 4, 6, 10, 12, and 24.

In the serial program, the calculation process is carried out sequentially from one cell to another from left to right. While in the parallel program, in this study carried out with an antidiagonal layout, the calculation process is divided into several parts used, each calculated simultaneously. Thus, the serial execution time is longer than parallel except for the DNA sequence, which is 10 in size.

At a length of the sequence of 10, the serial program can complete the calculation for 0.0005 seconds, while in the serial program, the calculation is completed for more than 0.001 seconds. It happens because 10 is a small data size, so data processing in serial programs can be done faster than in parallel. After all, there is little data to be processed. While in parallel programs, there is overhead in sharing and merging data which can take time.

However, when the sequence length is enlarged (more than 102), the advantages of parallel programs begin to be seen with faster execution times than serial. After sharing the data with the provided cores, calculations can be performed by several processes simultaneously. Thus, by dividing an enormous task into several smaller tasks and completing them in parallel, execution time can be saved, and program performance can be significantly improved. It is seen in sequence size 104 with an execution time on the serial for more than 1.5 hours, but in parallel programs with core 24, the execution time is only 0.027 hours. Thus, using parallel programs in sequence alignment with the Smith-Waterman algorithm can reduce computational costs in terms of execution time.

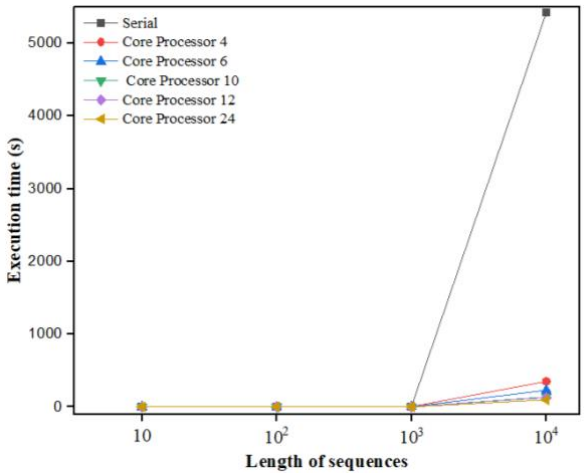

Figure. 6 The effect of sequence length on execution time

Table 1. Comparison the execution time between serial (1) and parallel program in Smith-waterman algorithm

Length of Sequences	Execution time (s)					
	1	Parallel (core)				
		4	6	10	12	24
10	0.0005	0.0009	0.0010	0.0011	0.0021	0.0039
10^2	0.0325	0.0227	0.0152	0.0103	0.0160	0.0187
10^3	3.2635	3.1908	1.8730	1.0244	1.2398	0.7584
10^4	5420.4654	350.2373	228.3099	137.1735	129.0008	97.0002

3.2 Speed-up Analysis on Parallel Programs

Speedup is the ratio between serial program execution time and parallel program execution time (equation 6). The speedup concept shows how much of an advantage it has to use parallel processing over serial processing. The greater the speedup value, the faster the program is executed in parallel versus serially.

$$speed - up = \frac{t_{serial}}{t_{parallel}}$$

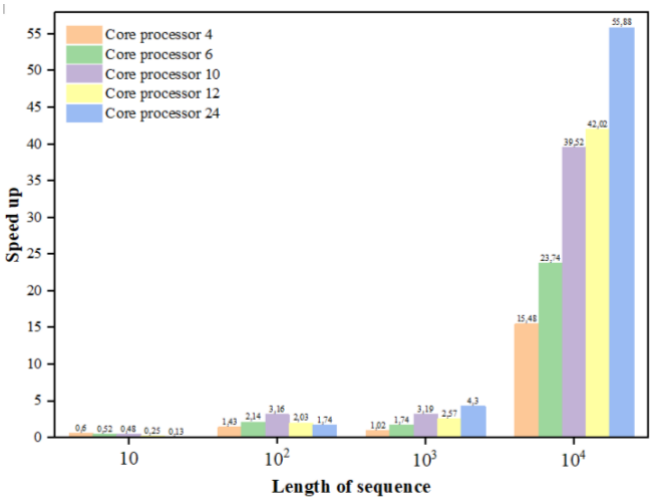

Figure. 7 The effect of sequence length on speed-up in each core.

(6) Figure.7 shows that parallelization speed up on several cores cores 4, 6, 10, 12, 24. It is done to find out how the number of cores affects speed-up. How much influence the number of cores on speed-up is influenced by several factors, (i) the length of sequence the larger the sequence length, the greater the benefits of parallelization, and when the sequence length is small, the overhead of parallelization can be greater than

the benefits of parallelization itself, (ii) Algorithm structure algorithms that are too heavy on communication and synchronization between cores will result in large overheads and reduce the benefits of parallelization, (iii) Computational structure If computing has dependencies between elements, then parallelization will become difficult, and the overhead will be even more significant.

Computational structure factors have been addressed using antidiagonal layouts, where calculations are not performed from left to rigth cells but diagonally, as shown in Figure.4, That way, the dependence between cells has been appropriately handled. The algorithm structure of the Smith-Waterman algorithm is quite simple, consisting of only three steps, as described in the methods section. So, overhead due to the complicated algorithm structure can be prevented. In the case of the three factors that affect the number of cores to speed-up, the sequence length is the main influential factor.

It can be seen in figure. The difference in the sequence length on each core causes a difference in speed-up. At a sequence of 10, classified as a small problem, the speed-up is less than one, which means that parallel programs take longer than serial programs. It indicates an overhead in using parallelism, as described earlier.

At a length of the sequence of 10^2, the most remarkable speedup is at core 10. it happens because the distribution of tasks on each core can be done more efficiently and evenly than on other cores. At the length of sequence 10^2, the number of cores 12 and 24 is too much, which worsens the performance of parallel programs or, in other words, overhead. A more even division of core 10 occurs at a length of the sequence of 10^3 compared to core 12.

However, the data is more prominent when the sequence length is 10^4 than when it is 10^2 and 10^3. The overhead associated with task division and synchronization becomes progressively smaller relative to the overall execution time. In this condition, using more cores can speed-up execution time because tasks can be divided more evenly, and more can be done simultaneously. So the more cores used, the higher the speedup obtained.

4. CONCLUSION

This paper proposes enhancement for time efficiency of the Smith-Waterman algorithm using OpenMPI as a library for paralyzation. Variations in the length of sequence used by 10, 102, 103, and 104 affect the execution time both in serial and parallel, where the more significant the sequence length, the longer it takes to complete the sequence. In addition to the sequence length, the number of cores used also affects the execution time and speed-up. Core 24, with a sequence length of 104, has the highest speed-up compared to the other cores, which is 55.88 times faster than the serial.

ACKNOWLEDGEMENTS

This work is supported by the Master Program in Computational Science, Faculty of Mathematics and Natural Science, Institut Teknologi Bandung.

REFERENCE

[1] E. Ernawati, D. Puspitaningrum, and A. Pravitasari, "Implementasi Algoritma Smith–Waterman Pada Local Alignment Dalam Pencarian Kesamaan Pensejajaran Barisan DNA (Studi Kasus: DNA Tumor Wilms)," *Jurnal Pseudocode*, vol. 1, no. 2. 2014.

[2] A. Vologodskii and M. D. Frank-Kamenetskii, "Strong bending of the DNA double helix," *Nucleic Acids Res.*, vol. 41, no. 14, pp. 6785–6792, Aug. 2013.

[3] M. I. Khan, M. S. Kamal, and L. Chowdhury, "MSuPDA: A Memory Efficient Algorithm for Sequence Alignment," *Interdiscip. Sci. – Comput. Life Sci.*, vol. 8, no. 1, pp. 84–94, 2016.

[4] G. Jun *et al.*, "Detecting and Estimating Contamination of Human DNA Samples in Sequencing and Array-Based Genotype Data," *Am. J. Hum. Genet.*, vol. 91, no. 5, pp. 839–848, Nov. 2012.

[5] E. F. Kirkness *et al.*, "The dog genome: Survey sequencing and comparative analysis," *Science (80-.).*, vol. 301, no. 5641, pp. 1898–1903, 2003.

[6] M. Issa and M. A. Elaziz, "Analyzing COVID-19 virus based on enhanced fragmented biological Local Aligner using improved Ions Motion Optimization algorithm," *Appl. Soft Comput. J.*, vol. 96, p. 106683, 2020.

[7] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Brief. Bioinform.*, vol. 11, no. 5, pp. 473–483, Sep. 2010.

[8] Y. Xiao and A. Konak, "A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem," *J. Clean. Prod.*, vol. 167, pp. 1450–1463, Nov. 2017.

[9] E. Pruesse, J. Peplies, and F. O. Glöckner, "SINA: Accurate high-throughput multiple sequence alignment of ribosomal RNA genes," *Bioinformatics*, vol. 28, no. 14, pp. 1823–1829, Jul. 2012.

[10] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. od Mol. Biol.*, no. 147, pp. 195–197, 1981.

[11] A. Chaibou and O. Sie, "Comparative Study of the Parallelization of the Smith-Waterman Algorithm

on OpenMP and Cuda C," *J. Comput. Commun.*, vol. 03, no. 06, pp. 107–117, 2015.
[12] F. N. Muhamad, R. B. Ahmad, S. M. Asi, and M. N. Murad, "Performance Analysis of Needleman-Wunsch Algorithm (Global) and Smith-Waterman Algorithm (Local) in Reducing Search Space and Time for Dna Sequence Alignment," *J. Phys. Conf. Ser.*, vol. 1019, no. 1, 2018.
[13] G. S. Sadasivam and G. Baktavatchalam, "A novel approach to multiple sequence alignment using hadoop data grids," in *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, 2010, pp. 1–7.
[14] P. Vadapalli, "Label Encoder vs One Hot Encoder in Machine Learning [2023]," *upgrad*, 2022. [Online]. Available: https://www.upgrad.com/blog/label-encoder-vs-one-hot-encoder/.
[15] B. Chen, Y. Xu, J. Yang, and H. Jiang, "A new parallel method of smith-waterman algorithm on a heterogeneous platform," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6081 LNCS, no. PART 1, pp. 79–90, 2010.
[16] M. Malik, S. Malhotra, and N. Prasanth, "Time Improvement of Smith-Waterman Algorithm Using OpenMP and SIMD," 2020, pp. 686–697.
[17] G. M. Striemer and A. Akoglu, "Sequence alignment with GPU: Performance and design challenges," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–10.
[18] E.-S. Orabi, M. A. Assal, M. A. Azim, and Y. Kamal, "DNA fingerprint using smith waterman algorithm by grid computing," in *2014 9th International Conference on Informatics and Systems*, 2014, p. PDC-74-PDC-79.
[19] S. K. Zahid, L. Hasan, A. A. Khan, and S. Ullah, "A novel structure of the Smith-Waterman Algorithm for efficient sequence alignment," in *2015 Third International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*, 2015, pp. 6–9.
[20] Fa Zhang, Xiang-Zhen Qiao, and Zhi-Yong Liu, "A parallel Smith-Waterman algorithm based on divide and conquer," in *Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings.*, pp. 162–169.