computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

A WORK DISTRIBUTION STRATEGY FOR GLOBAL SEQUENCE ALIGNMENT

Kailash Kalare [1], Jitendra Tembhurne [2]

[1] PDPM Indian Institute of Information Technology, Design & Manufacturing, Jabalpur, India, 1611701@iiitdmj.ac.in
[2] Indian Institute of Information Technology, Nagpur, India, jitendra.tembhurne@cse.iiitn.ac.in

Paper history:
Received 13 December 2018
Received in revised form 27 February 2019
Accepted 11 March 2019
Available online 31 March 2019

Keywords:
Global alignment;
N-W algorithm;
Global memory;
Optimization.

Abstract: The sequence alignment comprises to identify similarities and dissimilarities between two given sequences. In this paper, we propose a work distribution strategy for the implementation of DNA global sequence alignment. The main objective of this work is to minimize the execution time required for DNA global alignment of large biological sequences. The proposed approach dealt the issues with the memory optimizations and minimization of execution time. We considered the biological sequences of different size to fit into the global memory of the system. The proposed strategy is implemented in shared memory architecture using OpenMP programming for large biological sequences. Parallelization using OpenMP directive is relatively easy and execute the code fast. We experimented on the Dell Precision Tower 7910 with Intel Xeon processor with 32GB RAM and 28 CPU cores. The efficient use of global memory and cache memory optimization dominate the results of execution time. The results demonstrate the significantly high speed up using OpenMP as compared with other implementations.

1. INTRODUCTION

One of the principal applications of edit distance algorithm in Bioinformatics is to find out the similarity of macromolecules such as DNA sequence composed of letters A, T, C and G. The replicas of DNA, which shows imperfection gets changed by mutations at random places [1]. These mutations can grow exponentially with large sequences. The mutations can cause the transformations to both the sequences. Such transformations are;

1) Insertion of character x before position m denoted by $\rightarrow mx$

2) Deletion of character x at position m denoted by $x \rightarrow i\_$ and,

3) Substitution of character x to character y at position m denoted by $x \rightarrow my$.

The transformations due to mutations cause weights defined by a predefined weight function W. The following assumptions are made for weight functions [1, 2] is shown in Table 1.

The series of transformations forms a metric for both $Seq_{\#1}$ and $Seq_{\#2}$ which is referred to as *Score Matrix* (SM). Thus, the optimum alignment is to find minimum weight alignment with minimum weight transformation. One of the optimal alignments can be shown in Example 1.

Example 1: $Seq_{\#1}$ = GACTAC and $Seq_{\#2}$ = ACGC with weight function W = {*Match* (+1), *Mismatch* (0), *Gap* (-1)}. The alignment is given as:

$$\begin{matrix} - & A & C & G & - & C \\ G & A & C & T & A & C \end{matrix}.$$

In Example 1 the substitution is done at 4th position, while two gaps are added to $seq_{\#1}$ at 1st position and 5th position. The score was calculated as follows:

3 matches, compute $3 \times 1 = 3$,
2 gaps, compute $- 2 \times 1 = - 2$,
The total score is $3 - 2 = 1$.

Table 1. **Weight Function for Transformation.**

Sr. No.	Weight Function
1	$W(Seq_{\#1})+W(Seq_{\#2})=W(T_{total})$ where T_{total} is the number of total transformations
2	$W(X \rightarrow Y) = W(Y \rightarrow X)$
3	$W(X \rightarrow X) = 0$
4	$W(X \rightarrow Y) + W(Y \rightarrow Z) \geq W(X \rightarrow Z)$.

A pair of characters in a position is called aligned pair. The weights of the series of transformations are the sum of weights of aligned pairs. Global alignment between two large DNA sequences is the problem of finding the optimum alignment under the given scoring scheme. The BLAST [3] is used for sequence comparisons. It compares the sequences and finds out statistical information. FASTA [4] provides sequence similarity searching against protein databases. An adaptive grid implementation of the DNA sequence alignment proposed by C. Chen at. el [5], the author described a dynamic programming algorithm to compute k non-intersecting near-optimal alignments in linear space. In order to reduce runtime significantly, a hierarchical grid system as the computing platform and Static as well as dynamic load balancing techniques were applied. In [6], the work for graphical representation and alignment of DNA sequences are presented. The graphical alignment approach outlined, which is both conceptually and computationally not involved, designed to quickly find the two best global alignments. An optimization approach and its application to compare DNA sequences were proposed in [7]. It uses linear programming analysis methods based on the LZ algorithm and the Phylogenetic tree obtained by ClustalW using 48HEV sequences to compare strings. Parallel architecture for DNA sequence inexact matching with Burrows-Wheeler Transform was by [8] proposed on novel hardware architecture to parallelize the inexact matching algorithm based on BWT, and implements it on FPGAs. F. Saeed at. el [9] proposed a high-performance multiple sequence alignment system for pyrosequencing reads from multiple reference genomes. This work was based on domain decomposition to align such a large number of reads from single or multiple reference genomes. The alignment algorithm accurately aligns the erroneous reads and has been implemented on a cluster of workstations using MPI library. A tiling based sequence alignment is proposed in [10]. The combination of OpenMP and MPI paradigm is utilized for load balancing on parallel architecture. The proposed algorithm targets the metrics for DNA sequence alignment were time, speedup and efficiency.

An optimized technique for DNA sequence data compression using OpenMP and MPI was illustrated in [11]. This compression is vital in massive data storage and transmission. A parallel algorithm for DNA sequencing on heterogeneous platform using supervised machine learning approach is described in [12]. FED based a parallel algorithm was proposed by Q. Xue et. al. using Message Passing Interface (MPI). The results reported the matching in the given sequence and improved speedup using MPI [13]. The performance evaluation of DNA sequencing using OpenMP and CUDA is presented in [14]. A parallel approach for solving the k-differences prime problem is presented and speedup achieved up to 5.6 and 72.8 on OpenMP and GPU respectively.

2. METHOD AND MATERIALS

2.1. NEEDLEMAN-WUNSCH ALGORITHM

The N-W algorithm is a programming model for efficiently implementing recursion dynamically. This algorithm takes two input sequences $seq_{\#1}$ and $seq_{\#2}$, builds a score matrix SM, where SM $[n,m]$ represents the score of optimal alignment of $seq_{\#1}[1..n]$ and $seq_{\#2}[1..m]$ where n is the length of $seq_{\#1}$ and m is the length of $seq_{\#2}$. The recursion is given [15, 16] by

$$SM[i,j] = \max \begin{Bmatrix} SM[i-1,j-1] + S_{scheme}\left(seq_{\#1}[i], seq_{\#2}[i]\right) \\ SM[i-1,j] - GAP_{penalty} \\ SM[i,j-1] - GAP_{penalty} \end{Bmatrix}. \quad (1)$$

Initially, it fills $SM[0,0] = 0$ and then proceed to fill the matrix from the top left corner to the bottom right corner by applying the recursion equation (1) on each i and j. The S_{scheme} is a predefined matrix that compares characters at individual positions and assigns weights. In this paper, we used the S_{scheme} shown in Table 2. The various scoring schemes are presented in the literature such as BLOSUM and PAM, for more detail see [1]. The N-W algorithm also creates Direction Matrix (DM), which stores direction of movement (pointer) that can be used for finding the optimal alignment. The purpose of the DM is to backtrack for finding the maximum value given in recursion. The backtracking is the reverse of the score calculation. It starts from the bottom right corner and proceeds towards the top left corner [15, 17].

Table 2. **S_{scheme} used by our approach.**

	A	T	G	C
A	+2	-1	-1	-1
T	-1	+2	-1	-1
G	-1	-1	+2	-1
C	-1	-1	-1	+2

2.2. PARALLELIZATION OF NEEDLEMAN-WUNSCH ALGORITHM

Various parallel techniques were proposed in the literature for the implementation of the N-W algorithm used in DNA sequence alignment. The thread level parallelization using pthreads is proposed in [18, 19], achieves high performance. However, synchronization among threads and overhead to shift control from one thread to another thread becomes the bottleneck for these methods. Fig. 1 and Fig. 2 illustrate the typical tiling implementation techniques utilized for the various algorithms after dependency analysis. The tiling is an optimization that has been used to obtain huge performance gains on selecting the proper tile size to fit into the cache memory. The tiling is the compiler-based optimization, divides the original task into sub-tasks and computation of these sub-tasks are assigned to different threads to be performed in parallel.

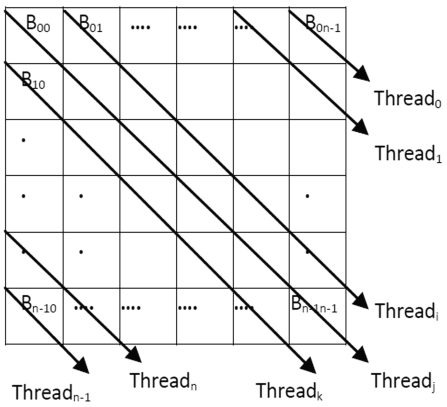

Figure 1 – Diagonal Blocks Processing

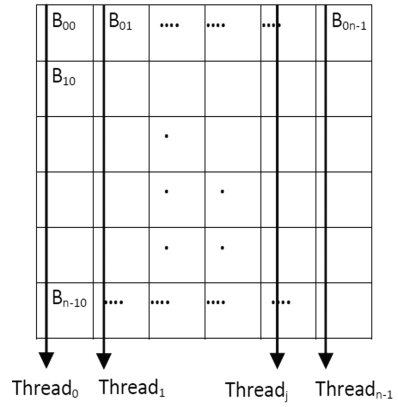

Figure 2 – Vertical Blocks Processing

Fig. 1 and Fig. 2 shows, blocks from B_{00} to $B_{n\text{-}1n\text{-}1}$ are referred as sub-task of the original task, which are independent of each other as per order of their execution. In Fig. 1, the blocks are processed diagonally as these blocks are independent to the other blocks in the computation. In Fig. 2, the blocks are executed in parallel in vertical direction to complete the computation. The tiling provides efficiency to the algorithm for large sequences. However, the number of threads required is large [20] and huge computation performed by the threads. But, managing large number of threads increases the overheads such as the cost of computation.

The best alignment for two sequences *n* and *m* is determined by applying the three steps i.e. 1) initialization, 2) scoring and, 3) trace back. In our approach, we have used scoring criteria for match = +2, mismatch = -1 and gap = -1. The initialization of sequences *n* × *m* is shown in Table 3. The trace back or Global alignment is demonstrated in Table 4. The last right point is utilized for back trace in the matrix. The next point is identified by moving diagonal or left or up as per the computed value from the start point.

Table 3. Score Matrix Table.

		SEQUENCE 1								
			A	C	G	T	T	G	C	A
		0	-1	-2	-3	-4	-5	-6	-7	-8
S	C	-1	-1	+1	0	-1	-2	-3	-4	-5
E	C	-2	-2	+1	0	-1	-2	-3	-1	-2
Q	A	-3	0	0	0	-1	-2	-3	-2	+1
U	T	-4	-1	-1	-1	+2	+1	0	-1	0
E	G	-5	-2	-2	+1	+1	+1	+3	+2	+1
N	C	-6	-3	0	0	0	0	+2	+5	+4
C	G	-7	-4	-1	+2	+1	0	+2	+4	+4
E	A	-8	-5	-2	+1	+1	0	+1	+3	+6
2										

Table 4. Backtracking in Global Sequence Alignment.

		SEQUENCE 1								
			A	C	G	T	T	G	C	A
		0	-1	-2	-3	-4	-5	-6	-7	-8
S	C	-1	-1	+1	0	-1	-2	-3	-4	-5
E	C	-2	-2	+1	0	-1	-2	-3	-1	-2
Q	A	-3	0	0	0	-1	-2	-3	-2	+1
U	T	-4	-1	-1	-1	+2	+1	0	-1	0
E	G	-5	-2	-2	+1	+1	+1	+3	+2	+1
N	C	-6	-3	0	0	0	0	+2	+5	+4
C	G	-7	-4	-1	+2	+1	0	+2	+4	+4
E	A	-8	-5	-2	+1	+1	0	+1	+3	+6
2										

The alignment of sequences *n* and *m* is illustrated in Fig. 3.

***Seq*#1 (*n*)**: A C G T T G C _ G

| | | | | | | | |

***Seq*#2 (*m*)**: C C _ A T G C G A

Figure 3 – Optimal Global Sequence Alignment

3. WORK DISTRIBUTION STRATEGY

In this section, we introduced the nomenclature, scheduling algorithm for global alignment and the OpenMP implementation.

Table 5 shows the list of abbreviations and their meanings used in this paper.

Table 5. Abbreviations and Meanings

Variable	Meaning
$Seq_{\#1}$	First input sequence
$Seq_{\#2}$	Second input sequence
MatchScore, MissMatchScore, $GAP_{penalty}$	S_{scheme} elements
ScoreMatrix (*SM*)	Score matrix
BS	Block size
BlockID	Block id

The parallel algorithm for finding global alignment is represented in Algorithm 1.

Algorithm 1: Scheduling for Global Alignment

1. Find the length of strings.
2. Define the BS.
3. Divide the string to form a block matrix.
4. Create a block matrix with entries such as 1, 2, 3, ….. .
5. Find StartRowID, EndRowID, StartColID, EndColID from BlockID.
6. Computes scores for individual blocks assigned to each thread.
7. Assign threads to the blocks in Columnar, Horizontal and Vertical direction, with threadID form 1, 2, and 3 respectively.
8. Find BlockID for blocks down and left of the diagonal block.
9. Compute the blocks successively to the block down to diagonal block. Also, compute the blocks successively to the block left of the diagonal block. The two independent threads were assigned to complete these block computations. This process will repeat until all the diagonal blocks were consumed.

Fig. 4 shows a typical block matrix addresses for the computation of score matrix. In this block matrix, every block is about the same size. Score computation for every element will be within that block. The starting and ending elements are computed separately.

Fig. 5 demonstrates the blocks execution by the different threads. Initially, the $block_0$ is executed by $thread_0$ then $block_1$ and $block_{10}$ is executed by $thread_1$ and $thread_2$ and this process carried till left diagonal block computation finished and thread disband starts thereafter.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
33	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Figure 4 – A typical Block Matrix

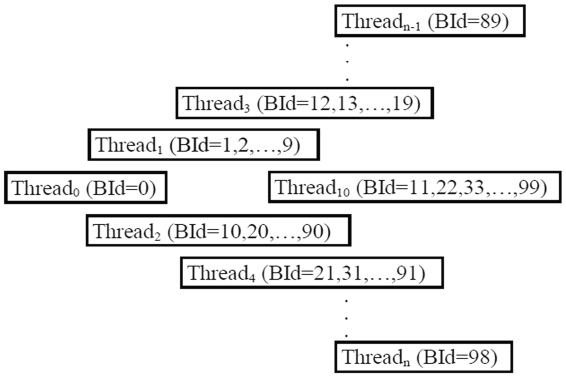

Figure 5 – Threads execution order

Fig. 6 illustrates the proposed block computation and work distribution approach for the computation of the score matrix for the global DNA alignment. The *SM* computation starts with the computation of diagonal blocks from B_{00} to $B_{n\text{-}1n\text{-}1}$. This computation is done by using a single thread. Once the diagonal block computation is finished, two new threads become active to start the computation of blocks down and left to the diagonal block. This computation process will continue till last diagonal computation is finished.

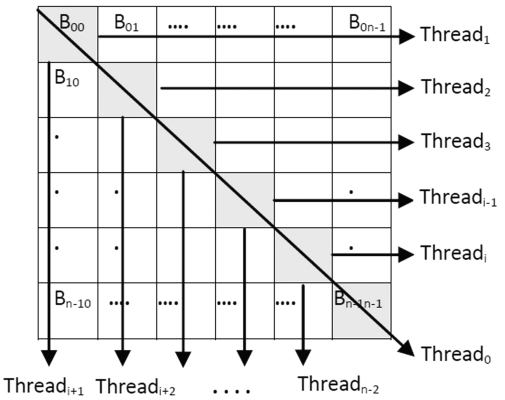

Figure 6 – Proposed Load Distribution Strategy

3.1. OPENMP IMPLEMENTATION

The Listing 1 illustrates the high level description of OpenMP [20] code with load distribution strategy for the computation of score matrix computation using N-W algorithm.

Listing 1: High level description of N-W algorithm.

```
1.  #pragma omp synchronized parallel{
2.   #pragma omp sections{
3.     #pragma omp section
4.       for(a = 1; a < noofcolsBM; a++)
5.          Compute Scores for every block with
           blockid(a, 0);
6.     #pragma omp section
7.       for(b = 1; b < noofrowsBM; b++)
8.          Compute Scores for every block with
           blockid(0, b);
9.     #pragma omp section{
10.      #pragma omp parallel{
11.        for(q = 1; q < noofrowsBM; q++) {
12.          t = q;
13.          Compute scores for every block
             with blockid(t, q);
14.      #pragma omp parallel{
15.        #pragma omp sections{
16.          //section 1 inside section 3 for every
             q and w
17.          #pragma omp section{
18.            d = t;
19.            for(c = q + 1; c < noofrowsBM;
                 c++){
20.              Compute Scores for every
                 block with blockid(c, d);
21.          }
22.        }
23.    //section 1 inside section 3 for every q and w
24.    #pragma omp section{
25.     f = q;
26.     for(e = t + 1; c < noofrowsBM; e++){
27.       Compute Scores for every block with
          blockid(e, f);
28.     }
29.    }
30.   }
31.  }
32. }
33. }
```

The Line No. 3 and Line No. 7 in Listing 1 can be performed in parallel. The computations of block with block addresses specified are corresponding to the first column in the Fig. 3 for Line No. 3 and computation of block addresses indicated in the first row in the Table 3 for Line No. 7. These parallel computations will begin after the computation of block '0'. However, once block '1' and block '10' finished the computation, the Line No. 11 becomes active and the computation is distributed to the 3 threads. The nested parallelism is activated and utilized when one block finished the computation, then two new sections are initiated to compute in parallel as indicated in Line No. 17. For example, once the block '11' is processed completely as shown in Fig.3, 4 threads were active to begin to perform the computations of the blocks '20', '21', '2' and '12' in parallel. The process of sections creation and assigning the blocks to them will continue for every diagonal block execution as indicated by Line No. 17. With this approach more threads will be available for the computation at the successive stages of block computation. Hence, this will increase the overall performance of the N-W algorithm computation in parallel.

4. RESULTS AND DISCUSSION

The performance is measured only for the computation of the score matrix. The time required for the sequential algorithm is compared with the time required for parallel algorithm. We compared genomes of equal residues ranging from 10000 to 100000 on Dell Precision Tower 7910 with Intel Xeon processor with 32GB RAM and 28 CPU cores. The *OMP_NESTED* environment variable is set to true. The performance measure includes the metrics; score matrix computation and different DNA lengths.

We first evaluated the time required for computation of the score matrix in sequential and parallel. We have considered the different set of threads for performance measurement such as 8, 12 and 28 threads. Fig. 7 to Fig. 9 shows the comparison between time required for the sequential and parallel implementation of the proposed approach by using a different set of threads and different lengths of DNA sequences respectively. The *x-axis* represents the sequence lengths in characters, and the *y-axis* represents the time for computation in seconds. The speedup achieved for the DNA sequence alignment for the different DNA lengths listed in this section.

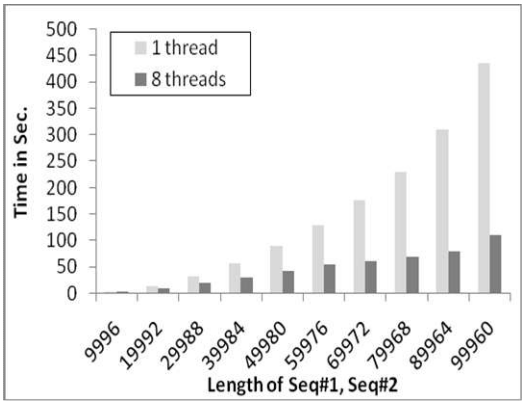

Figure 7 – Computation time of 8 threads Vs single thread

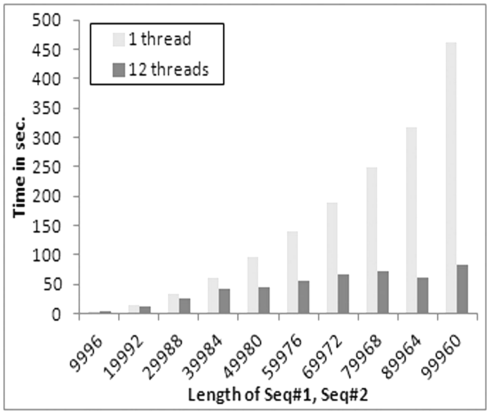

Figure 8 – Computation time of 12 threads Vs single thread

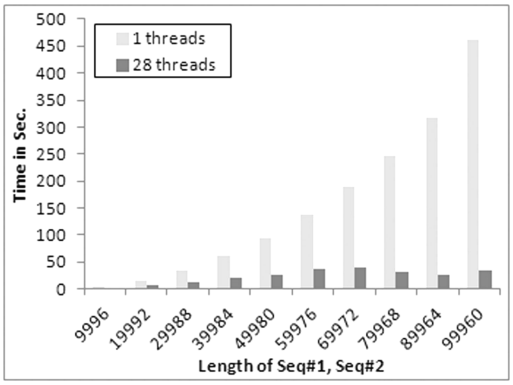

Figure 9 – Computation time of 28 threads Vs single thread

The speedup is computed by using the formula, $Speedup = time_{sequential} / time_{parallel}$. We evaluated the speedup for all the above computations. The speedup achieved in the range of 1.5x to 4x on the use of 8 cores, the speedup ranging from 1.2x to 5.5x is achieved on the use of 12 cores, and the speedup achieved range from 2x to 13x on the use of 28 cores. The maximum speedup is achieved for string length of 100000 as 13x on the 28 core system. It is also possible to get a higher speedup for larger sequences for higher configuration systems.

We have compared our work distribution approach with [21-24]. The performance analysis shows that the speedup achieved by [21] is 2.63x, [22] is 3x, [23] is 7x, [24] is 2x and 13x by our proposed approach. As highlighted in Table 6 the proposed strategy in this paper outperform [21-24] by achieving highest speedup. In this work, we mainly focused on the memory optimizations and minimization of execution time.

Table 6. Comparison with other Approaches

Ref	Speedup	Method highlights
[21]	2.63x	• cell values computation is parallelized in the matrix • anti-diagonal elements are computed in parallel
[22]	3x	• distribution of the DNA sequence comparisons • DNA sequence are handled by multiple compute units such as concurrent processes, thread group and threads • calculation of score matrix of comparison between sequences performed by each compute unit
[23]	7x	• tiling based multithreaded approach • DNA sequences are divided in smaller chunks • chunks are concurrently processed by the different threads
[24]	2x	• parallelization of computation of the values of cells in the score matrix • score matrix is reorganized: all cells in a column can be computed concurrently • each thread perform several calculations
our method	**13x**	• Computation divided in to blocks • blocks execution by the different threads • OpenMP nested parallelism is utilized • load distribution on CPU using different threads

5. CONCLUSION

In this paper, we have presented the work distribution strategy using OpenMP to speed up the global sequence alignment for DNA. We have been using the OpenMP nested parallelism strategy for our algorithm implementation. This nested parallelism supported by OpenMP increases the efficiency of the system and achieves high speed up. The speed up is computed for DNA sequence of different size ranging from 10000 to 100000. The performance evaluation shows that our algorithm achieved high speed up over the sequential algorithm. The speedup will increase for large sequences. The backtracking process is not considered in this work. In addition, backtracking can be considered for future work and the approach can be well suitable for MPI and CUDA implementation.

6. REFERENCES

[1] M. Rosenberg, *Sequence Alignment, Methods, Concepts and Strategies*, University of California Press, 2011.

[2] K. Sharma, *Bioinformatics: Sequence Alignment and Markov Models*, McGraw-Hill, 2008.

[3] Blast Library, 2018, [Online]. Available: http://blast.ncbi.nlm.nih.gov/Blast.cgi.

[4] FASTA Tool, 2018, [Online]. Available: http://www.ebi.ac.uk/Tools/fasta33/index.html.

[5] C. Chen, B. Schmidt, "An adaptive grid implementation of DNA sequence alignment," *Journal of Future Generation Computer Systems*, vol. 21, issue 7, pp. 988-1003, 2005.

[6] M. Randic, J. Zupan, D. Vikic and D. Plavsic, "A novel unexpected use of a graphical representation of DNA: graphical alignment of DNA sequences," *Journal of Chemical Physics Letters*, vol. 431, isuess (4-6), pp. 375-379, 2006.

[7] L. Liu, C. Li, F. Bai, Q. Zhao and Y. Wang, "An optimization approach and its application to compare DNA sequences," *Journal of*

Molecular Structure, vol. 1082, pp. 49-55, 2015.

[8] Y. Xin, B. Liu, B. Min, W. Li, R.C.C. Cheung, A.S. Fong and T.F. Chan, "Parallel architecture for DNA sequence inexact matching with Burrows-Wheeler transform," *Microelectronics Journal*, vol. 44, issue 8, pp. 670-682, 2013.

[9] F. Saeed, A.P. Rathke, J. Gwarnicki, T.B. Wolf, A. Khokhar, "A high performance multiple sequence alignment system for pyrosequencing reads from multiple reference genomes," *Journal of Parallel Distributed Computing*, vol. 72, issue 1, pp. 83-93, 2012.

[10] D.D. Shrimankar and S.R. Sathe, "Analysis of parallel algorithms on SMP node and cluster of workstations using parallel programming models with new tile-based method for large biological datasets," *Bioinformatics and Biology Insights*, vol. 10, pp. 255-265, 2016.

[11] C. Li, Z. Ji, F. Gu, "Efficient parallel design for BWT-based DNA sequences data multi-compression algorithm," *Proceedings of International Conference on Automatic Control and Artificial Intelligence*, Xiamen, China, 3-5 March, 2012, pp. 967-970.

[12] S. Memeti and S. Pllana, "A machine learning approach for accelerating DNA sequence analysis," *International Journal of High Performance Computing Applications*, vol. 32, issue 3, pp. 363-379, 2016.

[13] Q. Xue, J. Xie, J. Shu, H. Zhang, D. Dai, X. Wu, W. Zhang, "A parallel algorithm for DNA sequences alignment based on MPI," *Proceedings of International Conference on Information Science, Electronics and Electrical Engineering*, Sapporo, Japan, 26-28 April, 2014, pp. 786-789.

[14] L. Feuser and N. Moreano, "Parallel solutions to the k-difference primer problem," *Proceedings of the International Conference on Computational Science, ICCS'2018, Lecture Notes in Computer Science*, Vol. 10860, Springer, Cham, 2018.

[15] S.B. Needleman, C.D. Wunch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, pp. 443-453, 1970.

[16] T.F. Smith, M.S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195-197, 1981.

[17] V. Bharadwaj, W.H. Min, "Handling biological sequence alignment on networked computing systems," *Journal of Parallel Distributed Computing*, vol. 69, issue 10, pp. 854-865, 2009.

[18] T. Almeida, N. Roma, "A parallel programming framework for multi-core DNA sequence alignment," *Proceedings of International Conference on Complex, Intelligent and Software Intensive Systems*, Krakow, Poland, 15-18 February, 2010, pp. 907-912.

[19] E. Ruccii, A. De Giusti, F. Chichizola, M. Naiouf, L. De Giustil, "DNA sequence alignment: hybrid parallel programming on a multicore cluster," *Proceedings of the International Conference on Computers, Digital Communications and Computing, Recent Advances in Computers, Communications, Applied Social Science and Mathematics*, Barcelona, Spain, 2011, pp 183-190.

[20] B. Chapman, G. Jost and R.V.D. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, MIT Press, 2007.

[21] A.A. Khan, L. Hassan, S. Ullah, "OpenMP based parallel and scalable genetic sequence alignment," *Journal of Engineering and Applied Science*, vol. 34, issue 2, pp. 29-34, 2015.

[22] P. Borovska and M. Lazarova, "Parallel models for sequence alignment on CPU and GPU," *Proceedings of International Conference on Computer Systems and Technologies – CompSysTech-2011*, Vienna, Austria, June 16–17, 2011, pp. 210-215.

[23] S.R. Sathe, D.D. Shrimankar, "Parallelization of DNA sequence alignment using OpenMP," *Proceedings of International Conference on Communication, Computing & Security*, Rourkela, Odisha, India, February 12-14, 2011, pp. 200-203.

[24] A. Chaibou1 and O. Sie, "Comparative study of the parallelization of the Smith-Waterman algorithm on OpenMP and Cuda C," *Journal of Computer and Communications*, vol. 3, pp. 107-117, 2015.

***Kailash W. Kalare**, M. Tech. (CSE) from VNIT, Nagpur and currently is a Ph.D scholar at PDPM IIITDM, Jabalpur, India. His areas of interest are Deep Learning, High Performance Computing, and Image Reconstruction.*

***Jitendra V. Tembhurne,** Ph.D from VNIT, Nagpur and currently working as an Assistant Professor in the department of Computer Science & Engineering, Indian Institute of Information Technology, Nagpur. His areas of interest are Parallel Computing, Data Mining, Deep Learning, ML and Data Science.*