International Journal of Research in Computer Science
eISSN 2249-8265 Volume 2 Issue 3 (2012) pp. 1-6
© White Globe Publications
www.ijorcs.org

A COMPARISON OF COMPUTATION TECHNIQUES FOR DNA SEQUENCE COMPARISON

Harshita G. Patil[1], Manish Narnaware[2]
*M-Tech (CSE) G.H.Raisoni College of Engineering, Hingana Road, Nagpur, India
[1]Email: patil.harshita27@rediffmail.com,
[2]Email: manish.narnaware@yahoo.com

Abstract: *This Project shows a comparison survey done on DNA sequence comparison techniques. The various techniques implemented are sequential comparison, multithreading on a single computer and multithreading using parallel processing. This Project shows the issues involved in implementing a dynamic programming algorithm for biological sequence comparison on a general purpose parallel computing platform Tiling is an important technique for extraction of parallelism. Informally, tiling consists of partitioning the iteration space into several chunks of computation called tiles (blocks) such that sequential traversal of the tiles covers the entire iteration space. The idea behind tiling is to increase the granularity of computation and decrease the amount of communication incurred between processors. This makes tiling more suitable for distributed memory architectures where communication startup costs are very high and hence frequent communication is undesirable. Our work to develop sequence-comparison mechanism and software supports the identification of sequences of DNA.*

Keywords: *Dynamic Programming Algorithms, FASTA, Sequences Alignment, Tiling.*

I. INTRODUCTION

Comparing DNA sequences is one of the basic tasks in computational biology. In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences of nucleotide or amino acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns.

DNA (deoxyribonucleic acid) is the chemical material in a cell that carries the genetic codes for living organisms. Its structure is a double helix consisting of two sequences of letters from a four-letter alphabet (A, T, C, G), such that A is paired with T, and C with G. The letters represent the nucleotides or bases known as adenine, thymine, cytosine and guanine. Since the bases are paired, they are referred to as base pairs. All the DNA of a living organism is called its genome. The size of a genome can vary from millions of base pairs for bacteria to several billions base pairs in the case of mammals.

The nearly exponential growth rate of biological sequence database threatens to overwhelm existing computational methods for biological data analysis and searching. The computational demand needed to explore and analyze the data contained in these databases is quickly becoming a great concern. To meet these demands, we must use high performance computing systems, such as parallel computers. Biological sequences can be treated as strings over a fixed alphabet of characters, *a*, *c*, *t* and *g*. An alignment is a way of stacking one sequence above the other and matching characters from the two sequences that lie in the same position. From the alignment, we can find two subsequences contained respectively in two sequences that have the most similarity. The problem of the biological sequence alignment is the most faced in the exploring and analyzing the data, no matter in sequence assembly, comparison of homology, finding of gene coding region and prediction of protein's structure and function.

In this paper we consider the parallelization of this implementation, since parallelization of an iterative implementation of the algorithm would not be feasible. There has been significant recent work on the parallelization of dynamic programming algorithms in computational biology including implementations suitable for computational grids. What distinguishes this work is the data-driven recursive implementation, with resulting dynamically allocated tasks. The rest of this paper is organized as follows. In section 2, we provide Needleman-Wunsch and Smith Waterman algorithm. Section 3 provides parallel processing techniques. The parallel computation technique using multi-core architecture for DNA sequence comparison is shown in section 4, and conclusions about the completed work are discussed in section 5.

II. NEEDLEMAN-WUNSCH AND SMITH WATERMAN ALGORITHM

There are several methods for alignment of two biological sequences. The dynamic programming is probably the most popular programming method in sequences alignment. The Needleman-Wunsch algorithm, the first algorithm applying the dynamic programming to comparing biological sequences, was proposed by Needleman and Wunsch in 1970. Later, Smith and Waterman improved the Needleman-Wunsch algorithm and proposed the well-known Smith-Waterman algorithm. The time complexity of these algorithms is $O(mn)$, where m, n are the lengths of the two sequences respectively. Because the cores of these algorithms are dynamic programming, all algorithms need to manipulate an $(n+1)$ $(m+1)$ matrix, named dynamic programming matrix. The most time spent in these algorithms is calculating the dynamic programming matrix, so research work on parallelization of two sequences alignment focuses mostly on the calculation of the matrix. As the growth of biological sequence database, the length of sequences often becomes very long, and the size of the matrix becomes very large. Thus, not only the execution time of these algorithms needs to be very long, the memory space needed in the algorithm becomes very large. Even in some cases the size of the matrix is bigger than the size of memory space in one processor.

Two input sequence files are compared sequentially and final score is computed**.**

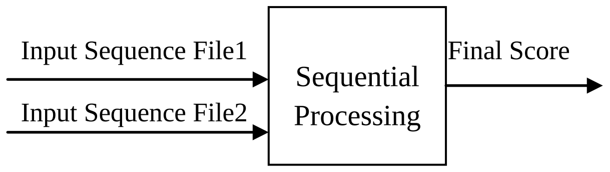
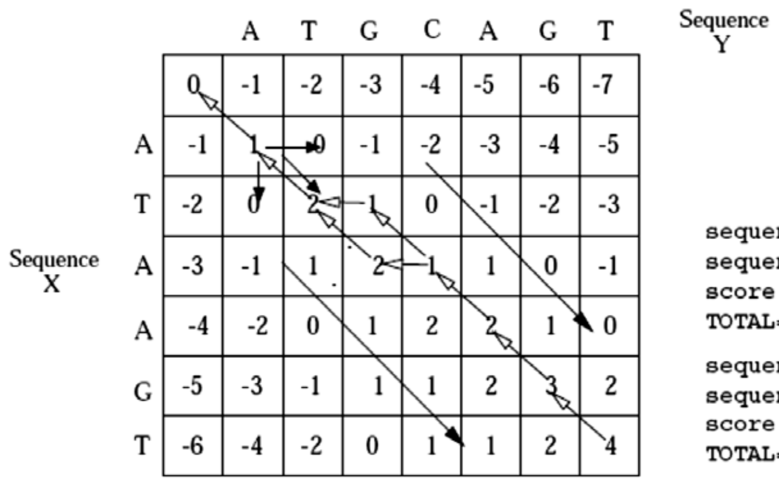


Figure 1: Sequential Processing

A. Sequence comparison using similarity matrix

This consists of two parts: the calculation of the total score indicating the similarity between the two given sequences, and the identification of the alignment(s) that lead to the score. In this paper we will concentrate on the calculation of the score, since this is the most computationally expensive part. The idea behind using dynamic programming is to build up the solution by using previous solutions for smaller subsequences. The comparison of the two sequences X and Y, using the dynamic programming mechanism, is illustrated in Figure 2. This finds global alignments by comparing entire sequences. The sequences are placed along the left margin (X) and on the top (Y). A similarity matrix is initialized with decreasing values (0,-1,-2,-3,…) along the first row and first column to penalize for consecutive gaps (insertions or deletions).

The other elements of the matrix are calculated by finding the maximum value among the following three values: the left element plus gap penalty, the upper-left element plus the score of substituting the horizontal symbol for the vertical symbol, and the upper element plus the gap penalty.



```
sequence X  A T  - A A G T
sequence Y  A T  G C A G T
score       1 1 -1 0 1 1 1
TOTAL= 4

sequence X  A T A  - A G T
sequence Y  A T G  C A G T
score       1 1 0 -1 1 1 1
TOTAL= 4
```

Figure 2: Similarity Matrix

For the general case where X = x1,…, xi and Y = y1,…, yj, for i = 1,.., n and j = 1,…,m, the similarity matrix SM[n;m] is built by applying the following recurrence equation, where gp is the gap penalty and ss is the substitution score:

$$SM[i,j] = max \begin{cases} SM[i,j-1] + gp \\ SM[i-1,j-1] + ss \\ SM[i-1,j] + gp \end{cases} \quad (1)$$

In our example, gp is -1, and ss is 1 if the elements match and 0 otherwise. However, other general values can be used instead. Following this recurrence equation, the matrix is filled from top left to bottom right with entry [i; j] requiring the entries [i, j -1], [i – 1, j -1], and [i-1, j]. Notice that SM[i; j] corresponds to the best score of the subsequences x1,…, xi and y1,…, yj. Since global alignment takes into account the entire sequences, the final score will always be found in the bottom right hand corner of the matrix. In our example, the final score 4 gives us a measure of how similar the two sequences is. Figure 2 shows the similarity matrix and the two possible alignments (arrows going up and left).

III. PARALLEL COMPUTATION

A parallel version of the sequence comparison algorithm using dynamic programming must handle the data dependences presented by this method, yet it should perform as many operations as possible independently. This may present a serious challenge for efficient parallel execution on current general purpose parallel computers, i.e., MIMD (Multiple Instruction stream, Multiple Data stream computers).

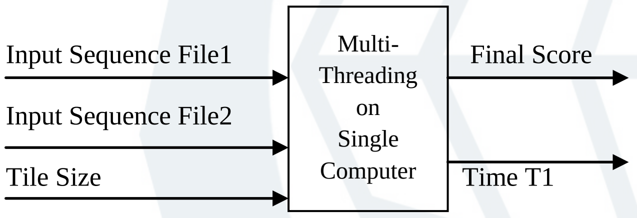


Figure 3: Multi -Threading on Single Computer

Input is two sequence files and tile size. Two input sequence files are compared. The final score is computed using the concept of multithreading on a single computer.

One solution to this problem is to divide the similarity matrix into rectangular blocks, as shown in Figure 4(a). In this example, the program would compute block 1 first, followed by 2 and 5, etc. If each block has q rows and r columns, then the computation of a given block requires only the row segment immediately above the block, the column segment to its immediate left, and the element above and to the left … a total of q +r +1 elements. For instance, if each block has 4 rows and 4 columns, then each block has to compute 16 maxima after receiving 9 input values. The communication-to-computation ratio drops from 3:1 to 9:16, an 81% reduction!

Note that this blocking will decrease the maximum achievable parallelism somewhat, by introducing some sequential dependence in the code. However, given the sizes of the current problems and the parallel machines currently used, this potential loss will not be a limiting factor.

The load-balancing problem can be addressed by putting several rows of blocks (or “strips”) on the same processor. Figure 4(b) illustrates this approach when four processors are used. The first and fifth strips are assigned to processor 1, the second and sixth strips are assigned to processor 2 and so on. This helps to keep all processors busy through most of the computation. For example, processor 1 initially works with the first strip, then simultaneously with the first and fifth strip, then finally only with the fifth strip. The processor utilization rises to 75%.

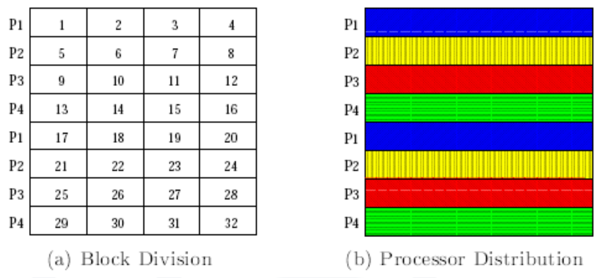


Figure 4: Partition of the similarity matrix

IV. PARALLEL COMPUTATION TECHNIQUE USING MULTI-CORE ARCHITECTURE

The input DNA sequences are collected in FASTA format. The FASTA files are converted to sequential sequence file using the convertor. The converted files are then compared on the distributed environment to compute the final score. This process is shown in Figure 5.

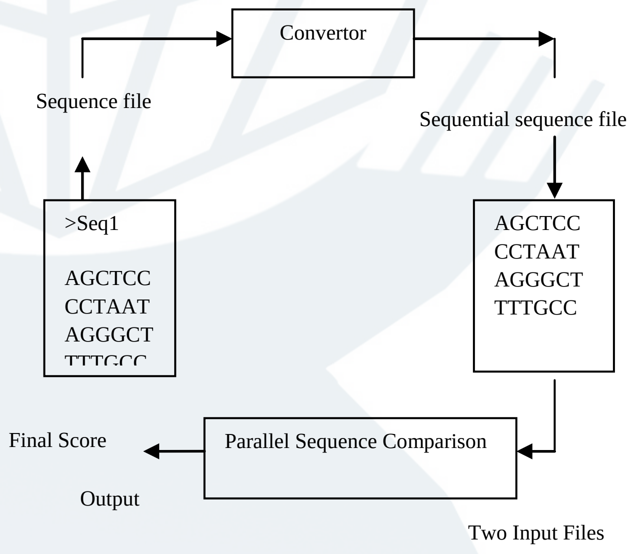


Figure 5: Parallel Computation Technique for DNA Sequence Comparison

A. *FASTA format*

In bioinformatics, FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences. The format originates from the FASTA software package.

Example of a simple FASTA file

> seq1 This is the description of my first sequence.
AGTACGTAGTAGCTGCTGCTACGTGCGCTAGCT
AGTACGACGTAGATGCTAGCTGACTCGATGC
> seq2 This is a description of my second sequence.
CGATCGATCGTACGTCGACTGATCGTAGCTAC
GTCGTCATCGTCAGTTACTGCATGCTCG

B. Parallel Algorithm Models

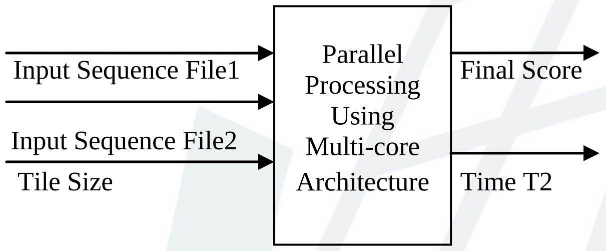


Figure 6: Parallel processing using multi-core architecture

Complete project will take the input as two sequence file and tile size from the user and system will calculate the final score by comparing sequences of given file by using multiple machine connected in network**.**

Master-Slave Model: One or more processes generate work and allocate it to worker processes shown in Figure 7.

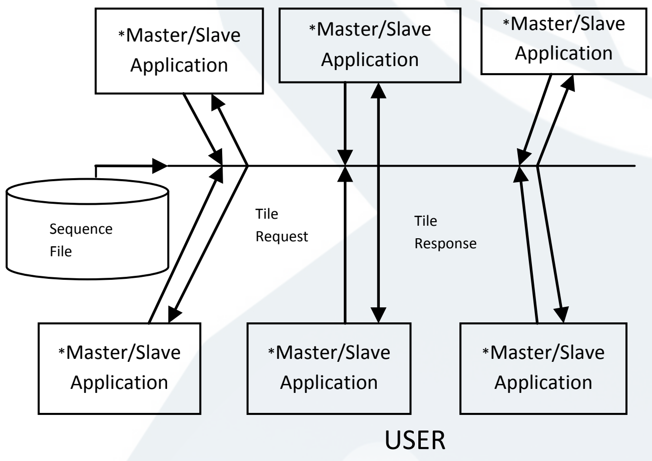

*Master/Slave Application
*Master/Slave Application
*Master/Slave Application
Sequence File
Tile Request
Tile Response
*Master/Slave Application
*Master/Slave Application
*Master/Slave Application
USER

Figure 7: Distributed System used for comparison

When user operates the application, the application acts as master to perform the task of sequence file comparison and other machines in which the application is installed will act as slave- returns the result of tile matrix requested by master. Any machine connected in network can act as master or slave. The interaction between master and slave is shown in Figure 8.

Master	Slave
1. Load Sequential Sequence File (Sequence X file)	
2. Load next Sequential Sequence File (Sequence Y file)	
3. Input file size i.e. tile width and tile height.	
4. Calculate the number of tiles required to calculate the final score.	
5. Calculate the number of diagonal rows required to calculate.	
6. Read sequence X data for tile(0,0) and send request to slave 1.	→
	1. Calculate the matrix of required tile in different thread.
7. Repeat step 6 for n number of slave and wait for the tile response.	
←	2. Return the file to the master.
8. Accept tile response from the slave.	
9. Collect all tiles of each diagonal row.	
10. Repeat step 6 to 9 for next diagonal row.	

Figure 8: Interaction between Master and Slave

C. Similarity Matrix

The similarity matrix used for sequential comparison is used in this approach also for DNA comparison but it is performed parallel on different machines using tiling technique.

D. Tiling Technique

Tiling is an important technique for extraction of parallelism. Informally, tiling consists of partitioning the iteration space into several chunks of computation called tiles (blocks) such that sequential traversal of the tiles covers the entire iteration space. The idea behind tiling is to increase the granularity of computation and decrease the amount of communication incurred between processors. This makes tiling more suitable for distributed memory architectures where communication startup costs are very high and hence frequent communication is undesirable.

Tiling is a well-established technique to enhance data locality or coarsen the grain of parallelism in loop programs. The iteration space of the program is covered with (usually congruent) tiles and the enumeration of the iteration points is changed so as to enumerate the tiles in the outer dimensions (i.e. loops) and the points within each such tile in the inner dimensions. The shape and size of the tiles is usually chosen dependent on the dependence vectors to minimize communication startups and the volume of the data communicated, especially in the context of distributed-memory architectures. For shared-memory systems, the number of startups and the volume are less of a concern, as long as the transfer time of the data between cores stays small compared to the computation time for each tile.

The Sequential Sequence File will be arranged in similarity matrix and the matrix will be divided into tiles. One tile will be allocated to one machine for computation. The computations will be done diagonally. Number of tiles coming under one diagonal line will be allocated to machine in network depending upon the number of machines available in the network. When computation of one diagonal row is complete then only the computation for the next diagonal line will start. In this way the complete matrix will be filled and the last cell of the matrix is the final score which will be equal to the size of sequential sequence file in case of match and if final score is half of the size we can say 50% of the DNA is matched and if final score is negative it indicates mismatch.

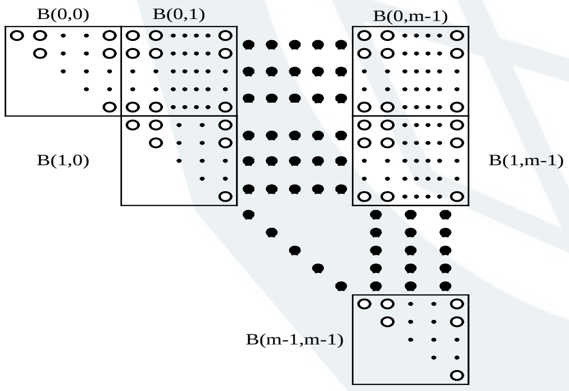


Figure 9: Tiling Technique

V. CONCLUSION

The aim of this study is to show the difference in computation to communication ratio using three different techniques. This study shows the computational power of parallel computers to speed up the process of comparing sequences. We looked at the dynamic programming mechanism and presented a multithreaded parallel implementation. The implementation uses similarity matrix method but takes advantage of the tiling technique under multithreading model. The result of implementation of parallel processing on a single machine is shown in Figure 10 (a) and using parallel processing in multi-core multithreading architecture is shown in Figure 10 (b) which shows computation performed per sec.

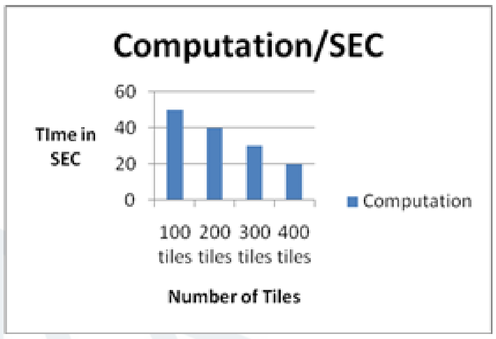


Figure10 (a): Computation on Single machine using Parallel Processing

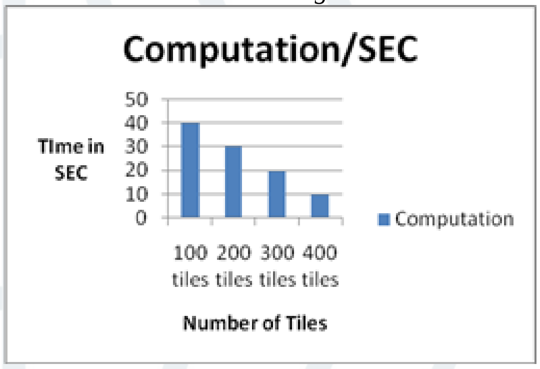


Figure 10 (b): Computation on distributed system using mullti-core achitecture

VI. REFERENCES

Conferences

[1] Sudha Gunturu*, Xiaolin Li*, and Laurence Tianruo Yang** "Load Scheduling Strategies for Parallel DNA Sequencing Applications" 11th IEEE International Conference on High Performance Computing and Communications 2009.

[2] Armin Gr¨oßlinger "Some Experiments on Tiling Loop Programs for Shared-Memory Multicore Architectures" Dagstuhl Seminar Proceedings 07361 Programming Models for Ubiquitous Parallelism 2008.

[3] Nasreddine Hireche, J.M. Pierre Langlois and Gabriela Nicolescu Département de Génie Informatique, École Polytechnique de Montréal ''Survey of Biological High Performance Computing: Algorithms, Implementations and Outlook Research'' IEEE CCECE/CCGEI, Ottawa, May 2006. doi:10.1109/CCECE.2006.277302

[4] Friman S´anchez, Esther Salam´ı, Alex Ramirez and Mateo Valero HiPEAC European Network of Excellence Universitat Polit`ecnica de Catalunya (UPC), Barcelona, Spain "Parallel Processing in Biological Sequence Comparison Using General Purpose Processors" 2005 IEEE. doi:10.1109/IISWC.2005.1526005

[5] Matteo Canella - Filippo Miglioli Universit`a di Ferrara (Italy) Alessandro Bogliolo Universit`a di Urbino (Italy) Enrico Petraglio - Eduardo Sanchez Ecole Polytechnique F´ed´erale de Lausanne EPFL-LSL,Lausanne (Switzerland)" Performing DNA Comparison on a Bio-Inspired Tissue of FPGAs" Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03) 2003 IEEE. doi:10.1109/IPDPS.2003.1213358

[6] N. F. Almeida Jr ,C. E. R. Alves, E. N. Caceres, S. W.Song ”Comparison of Genomes using High-Performance Parallel Computing” Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD’03) 2003 IEEE. doi:10.1109/CAHPC.2003.1250332

[7] Fa Zhang, Xiang-Zhen Qiao and Zhi-Yong Liu Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, *National Natural Science Foundation of China, Beijing, 100083”*A Parallel Smith-Waterman Algorithm Based on Divide and Conquer” Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP.02) 2002 IEEE. doi:10.1109/ICAPP.2002.1173568

[8] W.S Martins, J.B Del Cuvillo, F.J.Useche, K.B Theobald, G.R.Gao Department of Electrical and Computer Engineering University of Delaware, Newark DE19716, USA” A Multithreaded Parallel Implementation of a Dynamic Programming Algorithm for Sequence Comparison” Pacific Symposium on Biocomputing 6:311-322 (2001).

[9] Subhra Sundar Bandyopadhyay, Somnath Paul and Amit Konar Electronics and Telecommunication Department Jadavpur University, Kolkata, India “Improved Algorithms for DNA Sequence Alignment and Revision of Scoring.

How to cite

Harshita G. Patil, Manish Narnaware, "A Comparison of Computation Techniques for DNA Sequence Comparison". *International Journal of Research in Computer Science*, 2 (3): pp. 1-6, April 2012. doi:10.7815/ijorcs.23.2012.021