Sai Venkatesh Aravapalli, 2021, 9:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

Parallelization of Local Sequence Alignment

Sai Venkatesh Aravapalli
Department of Computer Science and Engineering,
Vellore Institute of Technology,
Vellore, Tamil Nadu, India.
aravapalli100@gmail.com.

Abstract- Everybody would agree that serial computing is easy to implement and use, but simply not efficient enough for industry-level purposes. To solve this problem different types of approaches are being experimented on both hardware and software. One of the approaches among them is parallel computing. It is one of the successful approaches that enables the user to use the processor in such a way that no core remains idle. The allocation of the work depends on the processes. Due to this reason, day by day higher number of industries are providing and using cloud solutions which work on the basis on parallel and distributed computing. For instance, Amazon's AWS or Google's Google Cloud platform are becoming the center for development, may it be in the field of web development or in the field of data analytics. To cope up with the fast-paced improvement in technology, one must also become familiarized with this domain, and hence this work.

Keywords: - Parallelization, Local Sequence Alignment, Smith-Waterman Algorithm.

I. INTRODUCTION

Genome[a] is an emerging field, constantly presenting many new challenges to researchers in both biological and computational aspect of application. Sequence comparison is a very essential and important operation. They detect similar or identical parts between two sequences called the query sequence and the reference sequence. The global and local alignments are the most prevalent kinds of sequence alignment.

In global alignment, we find the superior counterpart between parts of the sequences. On the other hand, local alignment algorithms try to match parts of sequences and not the entirety of them Local alignment is faster than global alignment, due to the lack of need to align the entire sequences. In our work, we would be implementing the Smith-Waterman Algorithm in a serial and parallel manner to for comparison and analysis.

As common sense suggests, the parallel implementation should execute and provide the triggered a revolution in discovery-based research and systems biology to facilitate understanding of even the most complex biological systems such as the brain.

II. LITERATURE REVIEW

Searching databases of DNA and protein sequences is one of the fundamental tasks in bioinformatics.

The Smith-Waterman algorithm guarantees the maximal sensitivity for local sequence alignments, but it is slow. It should be further considered that biological databases are growing at a very fast exponential rate, which is greater than the rate of improvement of microprocessors. This trend results in longer time and/or more expensive hardware to manage the problem.

Special purpose hardware implementations, as for instance super-computers or field programmable gate arrays (FPGAs) are certainly interesting options, but they tend to be very expensive and not suitable for many users.

For the above reasons, many widespread solutions running on common microprocessors now use some heuristic approaches to reduce the computational cost of sequence alignment.

Thus, a reduced execution time is reached at the expense of sensitivity. FASTA **(Pearson and Lipman, 1988) and BLAST (Altschul et al., 1997)** are up to 40 times faster than the best known straight forward CPU implementation of Smith-Waterman. A number of efforts have also been made to obtain faster implementations of the Smith-Waterman algorithm on commodity hardware. Farrar exploits Intel SSE2, which is the multimedia extension of the CPU. Its implementation is up to 13 times faster than SSEARCH (a quasi-standard implementation of Smith-Waterman).

To our knowledge, the only previous attempt to implement Smith-Waterman on a GPU was done by **W. Liu et al. (2006).** Their solution relies on OpenGL that has some intrinsic limits as it is based on the graphics pipeline. Thus, a conversion of the problem to the graphical domain is needed, as well as a reverse procedure to convert back the results. Although that approach is up to 5 times faster than SSEARCH, it is considerably slower than BLAST.

There are many existing tools for sequence alignment. Among those, FASTA2 and BLAST3 are two commonly used ones, where the time complexity has been reduced through some heuristic algorithms. These heuristics algorithms obtain efficiency, however, at the expense of sensitivity. As a result, a distantly related sequence may not be found in a search using the above tools.

Researchers have been worked on this issue through different approaches. For example, **Fa Zhang, Xiang Zhen Qiao, and Zhi-Yong Liu** presented a parallel Smith Waterman algorithm based on divide and conquer that can reduce running time and memory requirement. However, their method is also at the cost of losing sensitivity.

Other methods that apply standard computer systems such as high-performance supercomputers and computer clusters with software solutions for conducting the Smith-Waterman algorithm, although can achieve high sensitivity and reduce running time, are with extremely high cost. With the advance technology in the FPGA, a cost-efficient parallel implementation for the Smith-Waterman algorithm can be obtained.

III. ALGORITHM

Smith-Waterman algorithm calculates the local alignment of two sequences. It guarantees to find out the best possible local alignment taking into account the specified scoring system. This includes a substitution matrix and a gap- scoring method. Scores consider match, mismatch and substitution. To measure the comparison between two sequences, a score is calculated as follows.
Given an alignment between sequences S0 and S1, the following values must be assigned, for each column:

- ma = (+5) [Match]
- mi = (-3) [Mismatch]
- G = (-4) [Gap]

IV. PROCEDURE

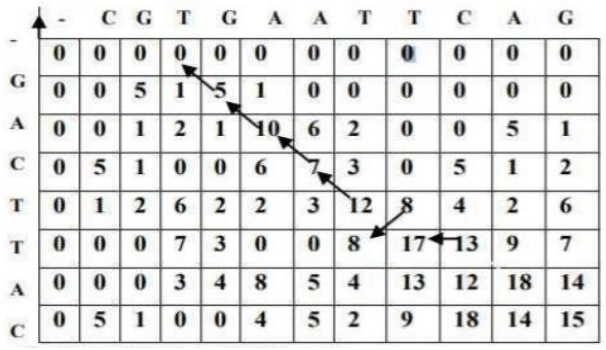
Initialization of the matrix and consider two sequences A and B. Matrix filling with the suitable scores. The two sequences are set in a matrix form by means of A+ 1 column and B+1 row. The value in the first row and first column are set to zero.

$$= Max \begin{cases} M_{i-1,j-1} + S_{i,j}, \\ M_{i,j-1} + W, \\ M_{i-1,j} + W, \\ 0 \end{cases}$$

The second and essential step of the algorithm is filling to entire matrix. To fill each and every cell it is important to know the diagonal values.

Trace back the sequence for an appropriate alignment is trace backing; before that the maximum score obtained in the entire matrix has to be detected for the local alignment of the sequences.

It is likely to those maximum scores can be present in one or more than one cell, in such case there may be option of two or more alignments, and the best alignment can be obtained by scoring it. Tracing back begins from the position which has the highest value, pointing back with the pointers, consequently find out the possible predecessor, then go to next predecessor and continue until it reaches the score 0.



	-	C	G	T	G	A	A	T	T	C	A	G
-	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	5	1	5	1	0	0	0	0	0	0
A	0	0	1	2	1	10	6	2	0	0	5	1
C	0	5	1	0	0	6	7	3	0	5	1	2
T	0	1	2	6	2	2	3	12	8	4	2	6
T	0	0	0	7	3	0	0	8	17	13	9	7
A	0	0	0	3	4	8	5	4	13	12	18	14
C	0	5	1	0	0	4	5	2	9	18	14	15

Fig 1. Trace Back of possible alignment.

V. EXISTING TOOL

There are few existing tools which have a parallel implementation of the Smith- Waterman algorithm, but the most prominent one is Clustal W [EMBOSS WATER]. EMBOSS Water uses the Smith-Waterman algorithm (modified for speed enhancements) to calculate the local alignment of two sequences. We can perform the alignment for protein, DNA or RNA sequences.

VI. PROPOSED METHOD

The problem at hand was tackled with a modular approach. Eight functions were constructed, each of which would be explained as follows:

1. n-Element:
This function is used to calculate the number of elements that have been found by the Smith Waterman Algorithm. Three conditions are given: One of which is to find out if the number of elements in the diagonal are increasing, decreasing or stable.

2. Calc First Diag Element:
This function is used to calculate the position of the maximum scored value in the matrix. This value needs to be found because the algorithm suggests that the backtracking to find the path should be started from this particular point.

3. Similarity Score:
This function is used to find out the optimal order of execution based on three conditions, which are used to calculate the new values of left, upper and the diagonal elements.

- If the diagonal element > maximum element, Move diagonally upwards.
- If upper element > maximum element, move upwards.
- If left element > maximum element, move leftwards every iteration, the values of maximum element are updated and inserting into the similarity and predecessor matrices.

4. Match Mis-match Score:
This function is used to calculate a similarity function or the alphabet for a match or mismatch. If the value of the two elements is equal, it is a match, otherwise it's a mismatch.

5. Backtrack:
The purpose of this function is to modify the matrix that needs to be printed and helps us identify the path that needs to be taken to get the most optimum solution Print Matrix: It's a looped iteration implementation to display the matrix.

6. Print Predecessor Matrix:
It is in this function in which we print the arrows depicting the path of local alignment.

7. Generate:
This function generates the two sequences A and B which would be locally aligned with each other. A random seed is used to ensure the reproducible nature of the output.

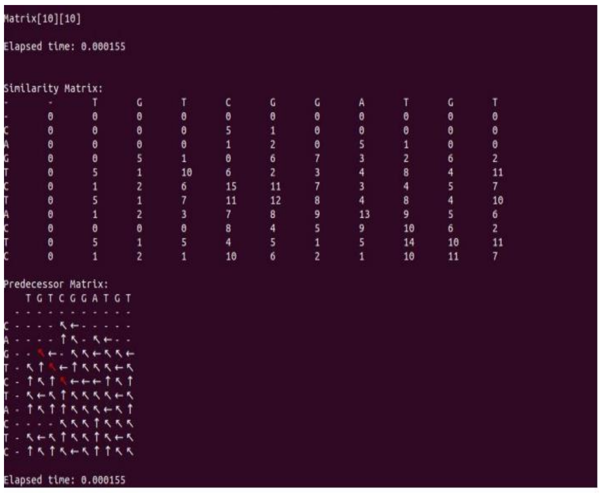
8. Packages required for implementation:
<stdio.h>
The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library
<stdlib.h>
is the header of the general-purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others. It is compatible with C++ and is known as cstdlib in C++. The name "stdlib" stands for "standard library"
<math.h>
The math.h header defines various mathematical functions and one macro. All the functions available in this library take double as an argument and return double as the result.
<omp.h>

It is a library that allows memory multiprocessing programming in C.

<time.h>

In C programming language time.h (used as ctime in C++) is a header file defined in the C Standard Library that contains time and date function declarations to provide standardized access to time/date manipulation and formatting.

VII. RESULTS



```
Matrix[10][10]

Elapsed time: 0.000155


Similarity Matrix:
-       -       T       G       T       C       G       G       A       T       G       T
-       0       0       0       0       0       0       0       0       0       0       0
C       0       0       0       0       5       1       0       0       0       0       0
A       0       0       0       0       1       2       0       5       1       0       0
G       0       0       5       1       0       6       7       3       2       6       2
T       0       5       1       10      6       2       3       4       8       4       11
C       0       1       2       6       15      11      7       3       4       5       7
T       0       5       1       7       11      12      8       4       8       4       10
A       0       1       2       3       7       8       9       13      9       5       6
C       0       0       0       0       8       4       5       9       10      6       2
T       0       5       1       5       4       5       1       5       14      10      11
C       0       1       2       1       10      6       2       1       10      11      7

Predecessor Matrix:
   T G T C G G A T G T
 - - - - - - - - - - -
C - - - - ↖ ← - - - - -
A - - - - ↑ ↖ - ↖ ← - -
G - - ↖ ← - ↖ ↖ ← ↖ ↖ ←
T - ↖ ↑ ↖ ← ↑ ↖ ↖ ↖ ← ↖
C - ↑ ↖ ↑ ↖ ← ← ← ↑ ↖ ↑
T - ↖ ← ↖ ↑ ↖ ↖ ↖ ↖ ← ↖
A - ↑ ↖ ↑ ↑ ↖ ↖ ↖ ← ↖ ↑
C - - - - ↖ ↖ ↖ ↑ ↖ ↖ ↖
T - ↖ ← ↖ ↑ ↖ ↖ ↑ ↖ ← ↖
C - ↑ ↖ ↑ ↖ ← ↖ ↑ ↑ ↖ ↖

Elapsed time: 0.000155
```

Fig 2. Output Result.

1. Analysis of Results:

The code was run for various lengths of sequences and the elapsed time was recorded in each case. After tabulating all the executions, we get:

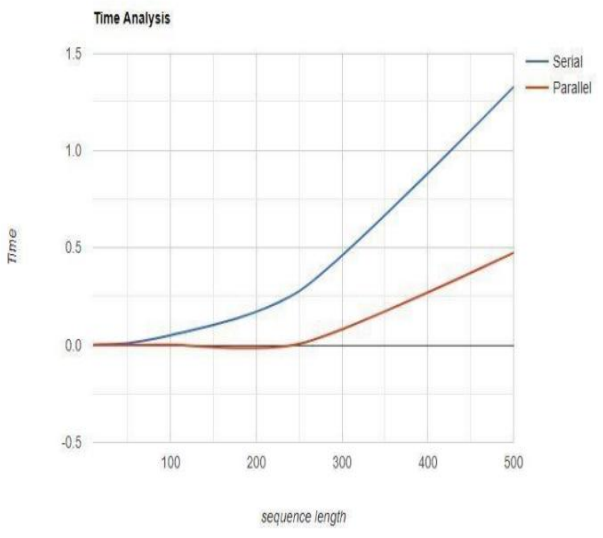


Fig 3. Graph of difference between serial and parallel with respective time.

Table 1. Result Analysis between serial and Parallel.

Matrix Size	Serial	Parallel
10 x 10	0.001308	0.000352
20 x 20	0.00273	0.00170
50 x 50	0.0099	0.000374
100 x 100	0.0506	0.002353
250 x 250	0.27707	0.006117
500 x 500	1.324	0.047371
1200 x 1200	11.62	0.166454

VIII. CONCLUSION AND FUTURE WORK

As we can see in the above table for comparison, for small lengths of the sequence, serial and parallel execution times are having a slight difference in time but the time variation goes on increasing with the increase in the sequence length.

The serial execution time has a rapid increase in graph while the parallel execution time has been almost constant for lower sequence length and increases slightly with increase in sequence length.

However, the parallel implementation remains stable with not a high rise in the execution time because of the parallel execution of the task with two threads, making the process faster than its serial counterparts.

REFERENCES

[1] Cuong Cao Dang, Vincent Lefort, Vinh Sy Le, Quang Si Le, and Olivier Gascuel, “Maximum likelihood estimation of amino acid replacement rate matrix”, Bioinformatics. 2011, 27(19):2758-60.

[2] Frank Keul, Martin Hess, Michael Goesele and Kay Hamacher, “PFASUM: a substitution matrix from Pfam structural alignments”, June 5 2017.

[3] S Henikoff and J G Henikoff, “Amino acid substitution matrices from protein blocks.”, Proc Natl Acad Sci U S A. 1992 Nov 15; 89(22): 10915–10919.

[4] Gary Benson, Yozen Hernandez and Joshua Loving,” A Bit-Parallel, General Integer-Scoring Sequence Alignment Algorithm”, 2004.

[5] Vincent Ranwez and Yang Zhang,” Two Simple and Efficient Algorithms to Compute the SP-

Score Objective Function of a Multiple Sequence Alignment", PLoS One, 2016.

[6] Robert C. Edgar1 and Kimmen Sjölander," A comparison of scoring functions for protein sequence profile alignment ", May, 2004.

[7] Cheng Ling, Khaled Benkrid, Ahmet T. Erdogan, "High performance Intra-task parallelization of Multiple Sequence Alignments on CUDA-compatible GPUs", Adaptive Hardware and Systems (AHS) 2011 NASA/ESA Conference on, pp. 360- 366, 2011.

[8] Chao-Chin Wu, Jenn-Yang Ke, Heshan Lin, Wu-chun Feng, "Optimizing Dynamic Programming on Graphics Processing Units via Adaptive Thread-Level Parallelism", Parallel and Distributed Systems (ICPADS) 2011 IEEE 17th International Conference on, pp. 96-103, 2011.