*J. Engg. and Appl. Sci. Vol. 34 No. 2 July - December 2015* *ISSN 1023-862X*

# OPEN MP-BASED PARALLEL AND SCALABLE GENETIC SEQUENCE ALIGNMENT

Asif Ali Khan*, Laiq Hassan*, Salim Ullah*

**ABSTRACT:**

*In bioinformatics, sequence alignment is a common and insistent task. Biologists align genome sequences to find important similarities and dissimilarities in them. Multiple heuristics and dynamic programming based approaches are available for sequence alignment. Smith-Waterman (SW), an exact algorithm for local alignment, is the most accurate of them all. However, the space and time complexity of the SW algorithm is quadratic. It is imperative to use parallelism and distributed computing techniques in order to speed up this process. In this paper, we discuss and evaluate an OpenMP based implementation of SW algorithm. All the experiments have been performed on a Linux based multi-core machine thereby reducing the overall complexity of the SW algorithm from quadratic to linear. The results obtained with various input sequences demonstrate that the parallel version of the SW algorithm runs 2.63 times faster than its sequential counterpart.*

**KEYWORDS:** *Smith-waterman, Sequence Alignment, OpenMp, DNA, Application Program Interface (API), Dynamic Programming, FASTA, BLAST*

## INTRODUCTION

Sequence alignment has been a longstanding research topic in the area of molecular biology. For making inferences about newly discovered genes, biologists compare their DNA sequences with the existing ones i.e. genes of some known functionality to find structural and functional similarities between them. Bioinformaticians use sequence alignment to find evolutionary trend in different species, study diseases and their inheritance in a more efficient and improved manner. These and many more applications make sequence alignment an active bioinformatics research area.

Biological sequence alignment is the process of comparing DNA, RNA or protein sequences to find similarities between them. This simple comparison has become a challenge because of the volume of available genetic data which is getting doubled every six months[1], much higher than advancement in computing power.

Many Dynamic Programming (DP) based algorithms were proposed for computing optimal genome sequences alignment. Among them, Smith-Waterman (SW) algorithm[2], proposed by Smith and Waterman, is the most accurate one. However, the time and space complexity of the SW algorithm is quadratic with respect to the length of the sequences to be compared and hence results in a large computational time. Heuristics based approaches e.g. BLAST[3,4] were proposed to minimize the computational time at the cost of reduction in accuracy i.e. the lesser time efficient heuristics give more accurate results and vice versa.

Parallel and distributed computing based techniques were applied to accelerate computationally expensive sequence alignment algorithms. Many proposed systems execute SW on clusters[5-7] and grids[8]. Parallel programming paradigms like Open Mp and MPI have been used to parallelize the global alignment algorithm namely Needleman-Wunsch (N-W) in the recent past[9,12]. These solutions significantly reduced the overall processing time.

In this paper, we present the parallel implementation of a local sequence alignment algorithm Smith Waterman on a multi-core machine and analyze its performance gain.

### Sequence alignment

Sequence alignment is of utmost importance to the biologists. By aligning the sequences of the entire genome, biologists find important matches and mismatches in them. From biological perspective, a match means similar structure, conserved regulatory regions, while mismatch means functional differences and diverged regions etc.

Different methods have been proposed by researchers

* *Department Computer Systems Engineering, University of Engineering and Technology, Peshawar, Pakistan.*

for the optimal alignment of genome sequences. These methods are broadly classified into two categories i.e. global and local. Global methods attempt to find maximum possible match from end to end while local methods find small similar stretches in sequences[11]. The fundamental and well known algorithms for sequence alignment are; Needleman-Wunsch for global alignment and Smith-Waterman for local alignment with the latter being used most commonly in computational bioinformatics. Both these algorithms are based on well-known technique called dynamic programming. DP based algorithms give accurate comparison results but are computationally expensive.

Heuristic based approaches like FASTA and BLAST operate in linear time. BLAST is amongst the most commonly used bioinformatics tools because of its computational power. BLAST is around 50 times faster than dynamic programing based algorithms which is remarkable. However, its accuracy is not up to the mark. FASTA is a software package, developed by David J. Lipman and William R. Pearson for aligning DNA and Protein sequences.

Smith Waterman algorithm align DNA sequences using similarity matrix. Calculation of each element in the matrix as shown in Figure 1 is dependent on its 3 neighbors' values: left (west) element, upper (North) element, and diagonal (North-West) element. These intrinsic dependencies limit the processing power of the SW algorithm.

DNA sequences are represented as strings composed of elements of the alphabet S = [A, C, G, T]. In order to find similarity or matching pattern in DNA sequences, we have to identify the optimal alignment in them because they are rarely identical. Sequence alignment means finding one to one correspondence between characters of the two sequences. Gaps can also be inserted at different locations such that the sequences end up with the best possible match.

smith-waterman algorithm

SW is the core of dynamic programming algorithms. SW being an exact method gives optimal result at the cost of increased computational complexity. To align two sequences of size 'n' and 'm', this complexity becomes O(mn).

SW algorithm finds similar regions/stretches in the input sequences, namely subject and query sequence, by finding the distance characterized by minimal cost of transformation and performing two elementary operations; insertion/deletion (gap operation) and substitution.

Considering the two sequences to be compared are 'sub' and 'qry' with size 'm' and 'n'. SW finds the similar subsequences by computing a matrix *'H'* using Equation 1. The scoring scheme associated with the SW algorithm is as follows.

Scoring scheme =

'S' is the score associated if the two characters are similar (match); 'D' is penalty, if the two characters are different (mismatch); 'g' is gap penalty, if any gap is inserted.

$$H(i, j) = \max \begin{cases} S\ (match) \\ D\ (mismatch) \\ g\ (gap\ penalty) \end{cases} \tag{1}$$

Implementation of SW algorithm is a 3 step process given below:

**Initialization**

$$\begin{cases} 0 \\ H(i\_1, J\_1) + S/D \\ H(i\_1, j)\_g \\ H(i, j\_1)\_g \end{cases}$$

The first step in implementing SW algorithm is the initialization of the first row and first column of a matrix to 0. The pseudo code for initialization is as follow:

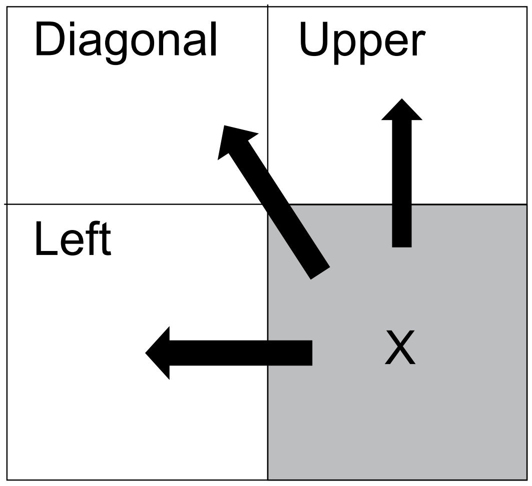

Figure 1. Data dependencies in Smith-Waterman Algorithm

$$H(0,j) = 0$$

$$H(i,0) = 0$$

**Matrix Fill-up**

In matrix fill-up stage, the entire matrix is filled up using Equation 1. The pseudo code for computing the matrix is as follows:

```
for i = 1 to m
    for j = 1 to n
        H(i,j) = max as a per Equation 1
```

**Trace back**

This step is performed to obtain the local alignment result. Since SW is a local alignment algorithm, its traceback starts from the cell with maximum value and following the arrow as shown in Figure 2 till a minimum threshold is reached which in most of the cases is zero. The arrow direction shows the origin of the value. A diagonal arrow means that in both sequences the character at that location is similar (match). An arrow in vertical direction means a gap should be inserted in the horizontal sequence and a horizontal arrow means, insertion of a gap in vertical sequence.

| | * | C | A | G | C | G | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | ↖ 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 0 | ↖ 4 | ← 2 | 2 | 0 | 0 | 2 |
| G | 0 | 0 | 0 | 2 | 2 | ↖ 4 | 2 | 0 | 2 |
| T | 0 | 0 | 0 | 0 | 1 | 2 | ↖ 6 | 4 | 2 |
| A | 0 | 0 | 2 | 0 | 0 | 0 | 4 | 5 | 3 |
| C | 0 | 2 | 0 | 1 | 2 | 0 | 2 | 3 | 4 |

Figure 2. Similarity matrix for two sequences ‘sub’ and ‘qry’

Since SW is a local alignment algorithm, it starts from the maximum score in the matrix (corresponding character T) and gives aligned subsequence. In global alignment algorithms, this process starts from the last cell in the matrix and reaches the first cell i.e. end to end. A sample alignment example is shown in Figure 3, where the scoring scheme used is: S = +1, D = -1 and g = 2.

| | | | | | |
|---|---|---|---|---|---|
| Sub: | A | G | C | G | T |
| Qry: | A | G | -- | G | T |
| Sore: | +1 | +1 | -2 | +1 | +1 |

Figure 3. Sequence alignment example

## PROPOSED APPROACH

In SW algorithm, the matrix fill-up stage is the most crucial one from computational perspective. All computations and comparisons for finding the maximum value are carried out in this stage. This step can be expedited if we parallelize the process of computing the cell values. This seems complex at first due to the intrinsic data dependencies. Computation of each cell value depends on the values of 3 neighboring cells as discussed previously i.e. is dependent on the values of cells , and which in turn are dependent on their neighbors and so on.

By properly observing this overall process, it is clear that elements in anti-diagonals of the similarity matrix can be calculated simultaneously because they are independent of each other and their predecessors' values have already been calculated as shown in Figure 4. This parallel computation of anti-diagonal elements significantly reduces the time complexity of an matrix from to , i.e. from quadratic to linear.

Exploiting this nature of the SW algorithm, we have developed a parallel version of the algorithm in Open Multi-Processing (OpenMP)[12], an Application Program Interface (API) that supports multi-platform shared memory multi-processing in C, C++ and FORTRAN. In OpenMP, workload is distributed between threads in such a way that they can communicate by sharing variables and can be scheduled differently.

In general, for two sequences of size M and N, the total number of anti-diagonals are and the number of elements in the longest diagonal is.

The pseudo code for parallelizing the SW algorithm

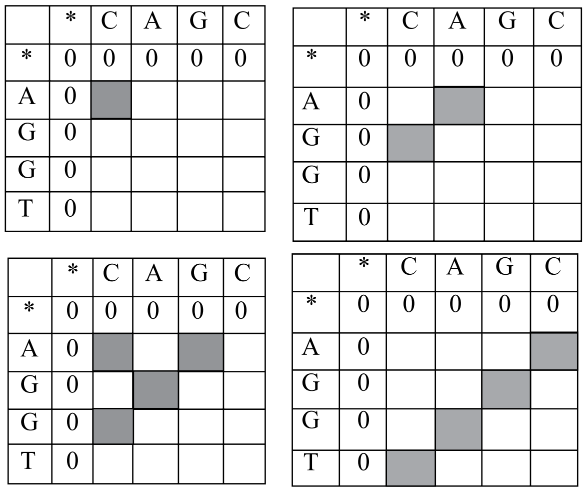

Figure 4. Independent streams in similarity matrix

using OpenMP programming paradigm is shown in Figure 5.

```
For x = 1 to M + N − 1
{ # Prgma omp parallel for
For temp = 1 to length of diagonal
Computer the element}
```

Figure 5. Pseudo code for parallelizing (for) loop in SW

The outer ‘for loop’ cannot be parallelized because the anti-diagonals are dependent on each other and it will iterate M + N-1 times. The inner loop compute the elements of each anti diagonal which are mutually independent and hence can be computed in parallel. The OpenMP ‘for construct’ has been used to parallelize this loop.

## EXPERIMENTAL RESULTS

The experimental results described in this section were obtained by running the SW algorithm on a Linux based machine having four Intel cores of 2.6 GHz clock frequency each. The following subsections present the results for both sequential and parallel implementations on the same platform.

### A. Sequential Implementation

The experimental results of sequential SW algorithm with input sequences of various lengths and average execution time are summarized in Table I. The first column in Table I shows the query length whereas the second column gives the average execution time in milli seconds (ms). We have used the term Average Execution Time as the experiment was repeated 20 times for each query sequence and the average time is reported in the Table 1. The results show that the execution time is directly proportional to the number of sequences and query length.

**Table 1. Sequential Execution Time**

| Query Length | Average Execution Time (ms) |
|---|---|
| 100 | 170 |
| 200 | 340 |
| 300 | 500 |
| 400 | 660 |
| 0.3 | 37.88 |
| 0.25 | 65.62 |
| 0.2 | 99.05 |

### B. OpenMP Based Parallel implementation

As discussed previously, the performance of the sequential code is directly dependent on the query size. For parallel implementation, the number of threads created are equally important. The parallel code if executed with a single thread is equivalent to its serial counterpart. The effect of varying number of threads while

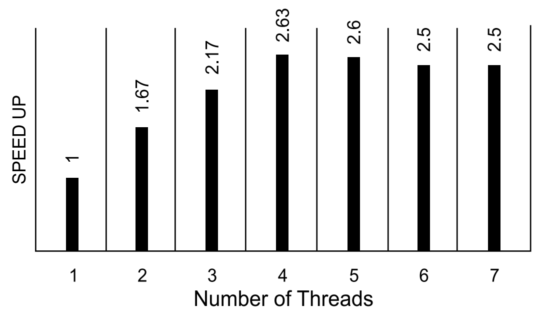

Figure 6. Number of threads vs speedup

**Table 2. Openmp Execution Time (4 Threads)**

| Query Length | Execution Time (ms) |
|---|---|
| 100 | 64 |
| 200 | 129 |
| 300 | 190 |
| 400 | 250 |

keeping the query length unchanged on performance gain is reported in Figure 6. The maximum performance as shown in the figure is achieved by setting the number of threads to 4.

Table 2 presents the average execution time of the parallel implementation of SW algorithm for sequences of various length. All the experiments have been repeated 20 times and the average execution time is presented. The number of threads in all the experiments is 4 as it gives optimal results as demonstrated in Firgure 6.

### C. Sequential VS Parallel Implementation

Figure 7 presents the performance comparison of sequential and parallel implementations. The OpenMP based parallel implementation as shown in Figure 6 gives the best performance when the number of threads is 4. The results demonstrate that on the average the OpenMP based parallel implementation performs 2.63 times better than its sequential counterpart.

## CONCLUSION AND FUTURE WORK

We have presented a scalable and parallel implementation of SW algorithm using OpenMP programming paradigm. The results show notable improvement i.e.

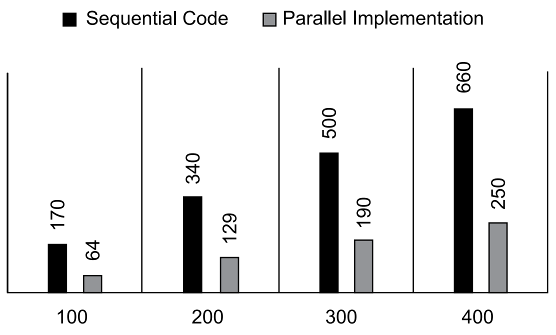

Figure 7. Sequential vs. Parallel SW Execution Time

2.63 times faster than its serial counterpart which is significant. The technique presented here can easily be ported to any available multi-core machine.

As a future work, the authors plan to run the same code on a High Performance Computing (HPC) cluster having 160 cores and 640 GB of main memory established in Ghulam Ishaq Khan Institute (GIKI) of Engineering Sciences in Khyber Pakhtunkhwa, Pakistan.

## REFERENCES

1. *Ramdas T and Egan. G, 2005. A survey of FPGAs for acceleration of high performance computing and their application to computational molecular biology," in Proc. 10th Intl. Conf. TENCON '05, Melbourne Australia, November, pp. 1-6.*

2. *Smith T.F, Waterman. M. S, 1981. Identification of common molecular subsequences", Journal of Molecular Biology, vol. 147: pp. 195–197.*

3. *Altschul S.F. , Gish W, Miller W, E. Myers W, Lipman D.J, 1990. Basic local alignment search tool," Journal of Molecular Biology, vol. 215: pp. 403-410.*

4. *Chao K.M, Zhang J, Ostell J, Miller W, 1994. A local alignment tool for long DNA sequences", Computer Applications in the Biosciences, vol. 11(2): pp. 147-153.*

5. *Chen C. and Schmidt B. 2003. Computing large-scale alignments on a multi-cluster", in Proc. IEEE Int. Conference on Cluster Computing, pages 38–45.*

6. *Rajko S. and Aluru S, 2004. Space and time optimal parallel sequence alignments", IEEE Transactions on Parallel and Distributed Systems, vol. 15(12): pp. 1070–1081.*

7. *Almeida T. J. and Roma N. F. V. 2010. A Parallel programming framework for multi-core DNA sequence alignment", proc. Intl. Conf. on Complex, Intelligent and Software Intensive Systems. pp.907-912.*

8. *Sousa M. S, Melo A. C. M. A, and Boukerche A, 2010. An adaptive multi-policy grid service for biological sequence comparison", Journal of Parallel*

*and Distributed Computing, vol. 70(2): pp. 160–172.*

9. *Sathe S. R. and Shrimankar D.D, 2010. Parallelization of DNA Sequence Alignment using OpenMP", ICCCS, pp. 200-203.*

10. *Wang L. Z, Xue W, Jie W, 2002. Running MPI Application in the Hierarchical Grid Environment", in Proc. International Conference on Scientific & Engineering Computation[IC-SEC], pp. 762-765.*

11. *L. Hasan, 2011. Acceleration of Bioinformatics Sequence Alignment, A Hardware perspective, Delft, The Netherlands.*

12. *OpenMP Standards: http://www.openmp.org/. (Last Accessed: January 22, 2014).*