Acceleration of Algorithm of Smith-Waterman Using Recursive Variable Expansion.

Hassan Kehinde Bello and Kazeem Alagbe Gbolagade

Abstract— Biological sequence alignment is becoming popular and interesting field to researchers especially the Bioinformatists. Two sequences with similar or varying lengths can be aligned using any alignment algorithm like Smith-Waterman Algorithm (SWA), Needleman-Wunsch Algorithm (NWA), BLAST, FASTA etc. Some of these algorithms are fast but lack accuracy (like FASTA and BLAST) while some are accurate at the expense of time (like SWA). SWA uses a dynamic programming approach with time and space constraint. Various methods (like systolic array method, implementation of the algorithm on FPGA etc.) have been applied on algorithm by various researchers to reduce or eliminate the computational complexity. This paper focuses on Recursive Variable Expansion (RVE) using parallel approach on the algorithm of Smith-Waterman to tackle the problem of time constraints in the algorithm and compare the result with another researcher's work.

***Index Terms*— Smith-Waterman Algorithm, Recursive Variable Expansion, Dynamic programming, Systolic array, Parallel approach, Time constraint, Space constraint, Sequence alignment.**

I. INTRODUCTION

In computational biology, sequence alignment is an important aspect in which known and unknown functionalities of biological sequences are compared [1]. The main objective is to determine the optimal alignment between the two sequences whose lengths may be similar or vary. Alignment score is computed based on the total number of gap penalty , matches and mismatches in the alignment. Biological sequence alignment is classified into local and global alignment. In local sequence alignment, optimal value is calculated from the most similar sub-region common to both sequences while in global alignment, the two sequences must be of similar length and the optimal value is computed from beginning to the end of the sequences [2]. The most common biological sequence alignment algorithms are Needleman-Wunsch Agorithm (NWA) and Smith-Waterman Algorithms (SWA). These two algorithms are commonly used to find global and local alignment respectively [3,4,5]. The two alignments are based on dynamic programming technique with space and time complexity of O(mn) [6] where *m* and *n* are the lengths of the two sequences being considered for alignment.

Apart from SWA and NWA mentioned above, there are other alignment algorithms like BLAST and FASTA. These algorithms use heuristic technique [7,8,9]. They are very fast at the expense of accuracy. Contrarily, SWA is accurate at the expense of time. However, it is the most accurate biological sequence alignment available but the computational complexity makes it slow in real applications [10]. Researchers have attempted various approaches to improve the acceleration of SWA. Some are based on software and some parts are implemented on hardware [11, 12,13]

The remaining part of the paper is organized as follows; in section 2, we present the background, in section 3, we discuss recursive variable expansion, in section 4, we present our results and discussion and finally the paper concluded in section 5.

***Manuscript received April , 2018*.**

***Hassan Kehinde Bello,** Department of Computer Science, Federal Polytechnic, Offa. Nigeria.*

***Kazeem Alagbe Gbolagade,** Department of Computer Science, Kwara state university Malete, Nigeria*

II. BACKGROUND

SWA is used for biological sequence alignment based on dynamic programming approach. It identifies common regions in sequence that share local similarity characteristics [14].

A. The Smith Waterman Algorithm

The optimal local alignment of two sequences X an Y is given by algorithm of SW in equation (1) for the computation of local alignment of matrix $M_{i,j}$

$$M(i,j)=\text{Max}\begin{cases}0\\ M(i-1,j-1)+S(x_i,y_j) & \text{match/mismatch}\\ M(i-1,j)+g\\ M(i,j-1)+g\end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

where $M(0,0) = 0$, $M(0,j) = g \times j$ and $M(i,0) = g \times i$, for $1 \leq i \leq n, 1 \leq j \leq m$. The g is the penalty for inserting a gap in any of the sequence and M(i,j) is the score for match/mismatch, depending upon whether $X[i] = Y[j]$ or $X[i] \neq Y[j]$.

The time complexity of the initialization step is O(*m* + *n*), where *m* is the number of rows and *n* is the number of columns in the matrix M.

1) Steps in the algorithm of Smith Waterman

Step 1: Initialization:

At the initialization step, the matrix M(i,j) will be initialized with $M_{0,j} = 0$ and $M_{i,0} = 0$, for all i and j as shown in fig 1. At this step, the time complexity is given by is O(*m* + *n*), where *m* is the total number of rows and *n* is the total number of columns in the sequences X and Y respectively.

Step 2: Matrix filling

The equation 1 will be used to fill up all the entries in the matrix Mi,j based on given scoring parameter (fig. 3). At this step, the time complexity is equal to the number of cells in the matrix i.e O(*mn*).

Step 3: Trace back

At the end of step 2 above, the cell with the highest score will be located at the bottom right of the matrix and traced back to get the

optimal local alignment (fig. 4). The time complexity of the trace back is given by O(m + n).

At the end of step 3, the total time complexity of Smith-Waterman algorithm is given by O(m+n) + O(mn) + O(m+n) = O(mn). Also, the space complexity of the algorithm is given by O(mn) because the size of the matrix is m x n.

Example:

Given the sequences X: AGGTCA and Y: CGTGTAA with match = 2, mismatch = 1 and gap = 1

		A	G	G	T	C	A
	0	0	0	0	0	0	0
C	0						
G	0						
T	0						
G	0						
C	0						
A	0						
A	0						

Fig. 1: initialization step

To parallelize SW algorithm we consider data dependency will reduce the O(mn) complexity in the filling of the matrix, we use a parallel calculation in which $M_{i,j}$ entry depends on the values of three neighboring entries $M_{i,j-1}$, $M_{i-1,j}$ and $M_{i-1,j-1}$, where each of those entries is also depending on the values of three neighboring entries, which makes data dependency extends to every other entry in the region [14]. The entries (i-1,j-1), (i,j-1) and (i-1,j) will be computed before the entry (i,j) is computed due to data dependency. This makes all elements in the anti-diagonal to be calculated simultaneously since they fall outside each data dependency region.

The following points in parallelism are worth noted:

- The degree of parallelism is constrained to the number of elements in the anti diagonal.
- The number of elements in the longest anti diagonal is LD = min(m,n)
- In m x n matrix, the number of anti diagonal require to reach the bottom right is (m+n-1)

		A	G	G	T	C	A
	0	0	0	0	0	0	0
C	0	1	1	1	1	2	1
G	0	1	3	3	2	2	2
T	0	1	2	4	5	4	3
G	0	1	3	4	5	6	5
C	0	1	2	4	6	6	7
A	0	2	2	3	2	7	8
A	0	2	3	3	4	6	9

Fig 2: Sample data dependency matrix $M_{7,6}$

Fig. 2 is a sample matrix with m = 7 and n = 6, the long arrow shows the direction in which computation progresses [15]. At least 12 cycles (anti diagonal) are involved in this computation and only six cells can be computed in parallel. The processing elements and the number of elements in the longest anti-diagonal (LD) is given to be 6 in fig. 2.

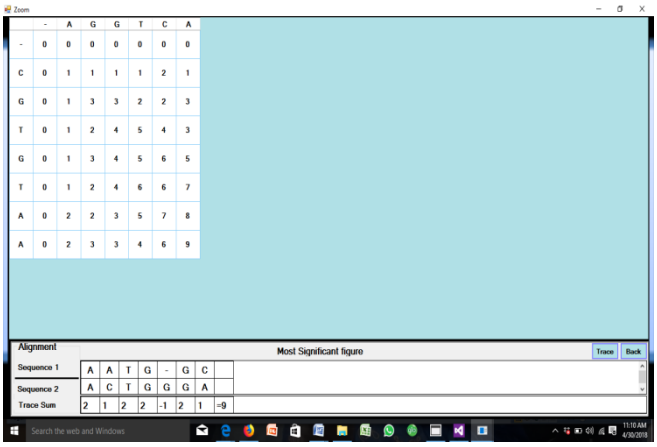

Fig 3. Matrix filling step

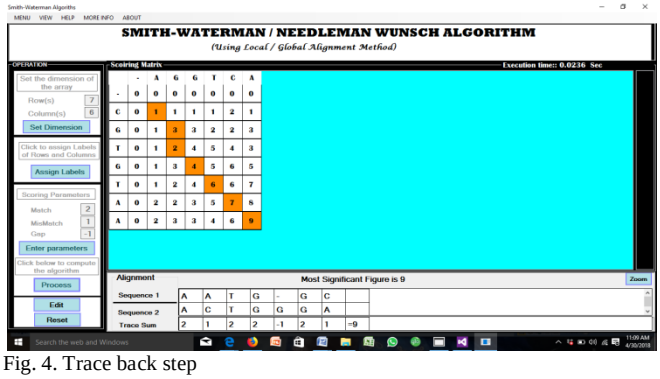

Fig. 4. Trace back step

Fig. 5 represents an implementation used to compute an element in matrix Mi,j. in the scheme, there is one Look-up Table (LUT), three comparators (^) and three adders (+). It will take 4 cycles time to compute one element.

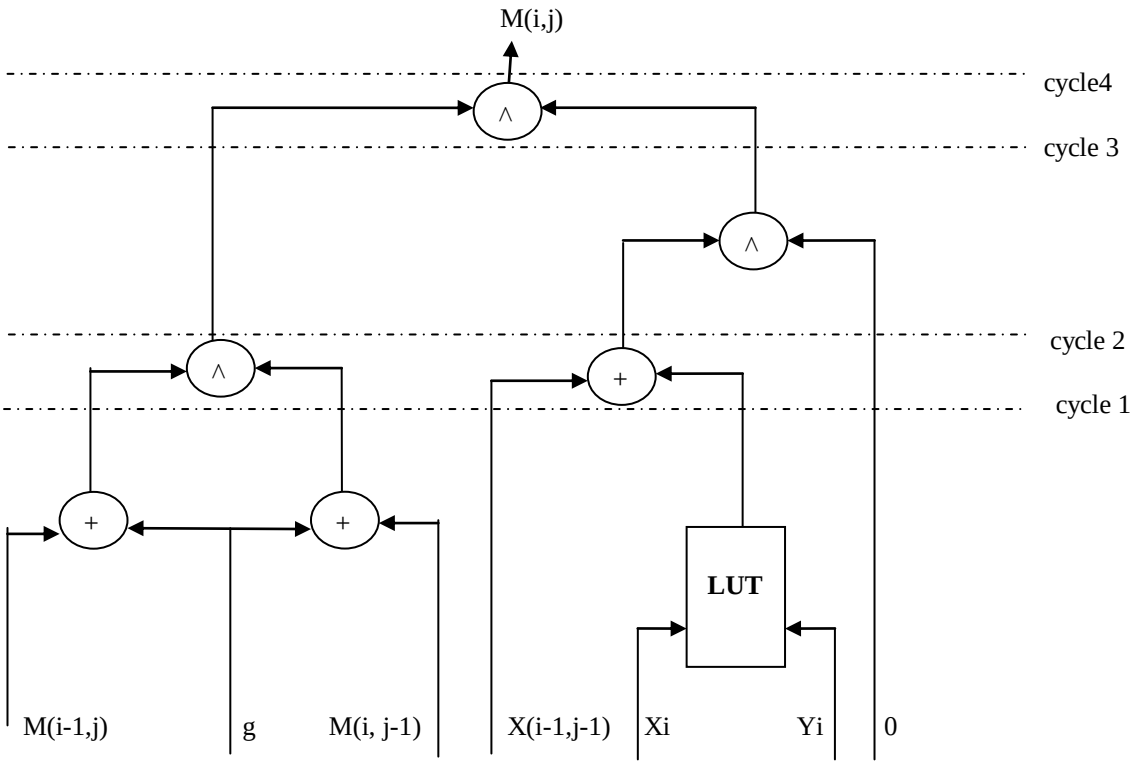

Fig. 5: Computational circuit for matrix $M_{i,j}$

Fig. 3 above contains 3 adders, 2 comparators (∧) and 1 look up table (LUT). This makes it possible to simultaneously compute all the elements in each anti diagonal, since they fall outside each other's data dependency regions [15]. We can reduce the complexity O(*mn*) at the fill up stage by computing the entries of the matrix in parallel.

III. RECURSIVE VARIABLE EXPANSION

The method of eliminating data dependencies from a program for the sake of parallelization is known as Recursive Variable Expansion (RVE). is a technique which removes the loop carried data dependencies among the various statements of the program to execute every statement in parallel [16],[17],[18],[19] For example, if a statement (Si) depends on another statement (Wj) instead of waiting for the completion of Wj before executing statement Si, all occurrence of variables in Si that create dependency can be replaced with the computation of their variables in Wj. By this, there will be no need to wait for the completion of statement of Wj, statement Si can then be executed independent of Wj [20].

Example, Consider the following

```
C(i) =1
     for  i = 2 to 6
C(i) = C(i-1) + i
      end for
```

RVE can be applied to the above algorithm as

C(6) = C(5) + 6
= C(4) + 5+6
= C(3) + 4+5+6
= C(2) + 3+4+5+6
= C(1) + 2+3+4+5+6
= 1+ 2+3+4+5+6

B. Approach to our scheme

Fig.6 is a structure derived when RVE is applied to compute the maximum value of Mi,j in equation 1. It reduces to sub equations which also reduce to more sub equation before arriving on the maximum value of Mi,j.

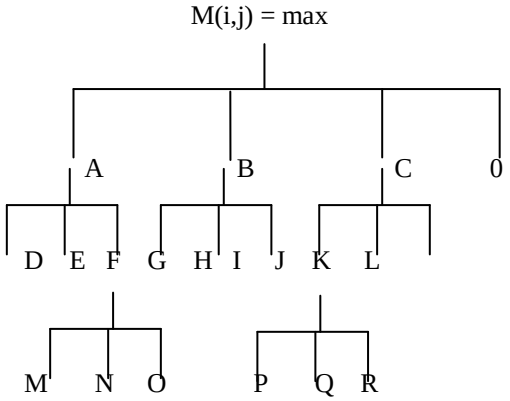

Fig. 6: derived structure

Where,
A= M(i,j-1) + g
B = M(i-1,j-1) + s(i,j)
C = M(i-1,j) + g
D = H(i,j−2)+2g
E= H(i−1,j−2)+g+s(i,j−1)
F= H(i−1,j−1)+2g

G= H(i−1,j−2)+g+s(i,j)
H = H(i−2,j−2)+s(i−1,j−1)+s(i,j)
I= H(i−2,j−1)+g+s(i,j)
J= H(i−1,j−1)+2g
K= H(i−2,j−1)+g+s(i−1,j)
L= H(i−2,j)+2g
M= H(i−1,j−2)+3g
N= H(i−2,j−2)+2g+s(i−1,j−1)
O = H(i−2,j−1)+3g
P = H(i−1,j−2)+3g
Q= H(i−2,j−2)+ 2g +s(i−1,j−1)
R= H(i−2,j−1)+3g

From equation 2 which is the extraction of Fig. 6 (derived from equation 1), If recursive variable expansion is applied. Equation 2 is written as the maximum of Mi,j. Thirteen independent equations are obtained from fig 6. These sub equations can be computed in parallel to obtain the maximum VALUE OF Hi,j. The thirteen equations will require four levels as $\log_2 13 = 4$ (fig 5).

$$M(i,j) = \text{Max} \begin{cases} 0 \\ D = M(i,j-2)+2g \\ E = M(i-1,j-2)+g+s(i,j-1) \\ M = M(i-1,j-2)+3g \\ N=M(i-2,j-2)+2g+s(i-1,j-1) \\ O = M(i-2,j-1)+3g \\ G = M(i-1,j-2)+g+s(i,j) \quad \ldots\ldots\ldots (2) \\ H = M(i-2,j-2)+s(i-1,j-1)+s(i,j) \\ I = M(i-2,j-1)+g+s(i,j) \\ P = M(i-1,j-2)+3g \\ Q = M(i-2,j-2)+ 2g +s(i-1,j-1) \\ R = M(i-2,j-1)+3g \\ K == M(i-2,j-1)+g+s(i-1,j) \\ L = M(i-2,j)+2g \end{cases}$$

As the value of gap (g) is -2, equation 2 becomes

$$M(i,j) = \text{Max} \begin{cases} 0 \\ D = M(i,j-2)-4 \\ E = M(i-1,j-2)+s(i,j-1) -2 \\ M = M(i-1,j-2)-6 \\ N = M(i-2,j-2)+s(i-1,j-1) -4 \\ O = M(i-2,j-1)-6 \\ G = M(i-1,j-2)+s(i,j) -2 \quad \ldots\ldots\ldots (3) \\ H = M(i-2,j-2)+s(i-1,j-1)+s(i,j) \\ I = M(i-2,j-1)+s(i,j) -2 \\ P = M(i-1,j-2)-6 \\ Q = M(i-2,j-2)+s(i-1,j-1) -4 \\ R = M(i-2,j-1)-6 \\ K = M(i-2,j-1)+s(i-1,j) -2 \\ L = M(i-2,j)-4 \end{cases}$$

From equation (3), M(i,j) can be computed by eliminating repeated equations since they add no value to it.

For the value of s(i,j) = -1, i-1 and j-1[1] are present in E, M,G and P given as

$$\text{Max } M(i,j) = \begin{cases} 0 \\ E = M(i-1,j-2)-1 -2 = M(i-1,j-2)-3 \\ M = M(i-1,j-2)-6 \\ G = M(i-1,j-2)-1 -2 = M(i-1,j-2)-3 \\ P = M(i-1,j-2)-6 \quad \ldots\ldots\ldots\ldots (4) \end{cases}$$

It is clear that neither M nor P can be maximum, therefore the maximum is between E and G. Equation 2 can now be reduced to the equation 4 and equation 5.

$$M(i,j) = \text{Max} \begin{cases} 0 \\ D = M(i,j-2)-4 \\ E = M(i-1,j-2)+s(i,j-1) -2 \\ G = M(i-1,j-2)+s(i,j) -2 \\ H = M(i-2,j-2)+s(i-1,j-1)+s(i,j) \\ K = M(i-2,j-1)+s(i-1,j) -2 \\ I = M(i-2,j-1)+s(i,j) -2 \\ L = M(i-2,j)-4 \quad \ldots\ldots\ldots\ldots\ldots (5) \end{cases}$$

In order to compute the maximum of equation (4), we use $\log_2 7 = 3$ levels and this is better than 4 levels. From equation 4, the following fig. 7(a), 7(b), 7(c) and 7(d) can be calculated. To compute O1 for a block of 3x3 elements we shall use i, ii, iii, iv and v (fig. 7a). Other blocks O2,O3 and O4 can be calculated as well using a block of 3x3 elements in a similar way as shown in fig 7(b), 7(c) and 7(d) respectively.

	i-2	i-1	i
J-2	iii	ii	i
J-1	iv	O4	O2
J	v	O3	O1

Fig 7(a)

	i-2	i-1	i
j-2	iii	Ii	i
j-1	iv	O4	O2
J			

Fig 7(b)

	i-2	i-1	i
j-2	ii	i	
j-1	iii	O4	
J	iv	O3	

Fig 7(c)

	i-2	i-1	i
j-2	ii	i	
j-1	iii	O4	
J			

fig. 7(d)

In M(i,j), O2 is calculated using i,ii,iii and iv; O3 is calculated using i,ii,iii and iv and finally O4 is calculated using i, ii and iii

C. Calculation of Time

In a matrix of 3x3 blocks, the values M(i,j), M(i-1,j), M(i,j-1) and M(i-1,j-1) can be computed in parallel since all the values are independent of one another. The total time taken to compute all the variables is the same as the time

taken to compute M(i,j) in parallel.

Table 1: Time estimation for 3 block

	Processing time	**Level**
1	computation of maximum of seven equations:	2 levels
2	Computation of s(i,j) in parallel:	1 level
3	Time used by the adders: (3 adders)	2 levels
4	Total number of time required to compute one element	5 levels
	Total time to compute 3 blocks in serial diagonal: 3 x 5 level	**15 levels**

Table 2: Time estimation to compute 7 lines

	Processing time	**Level**
1	Time to compute maximum of three elements:	3 levels
2	Time taken to compute s(i,j) in parallel:	1 level
3	Time used by the adders:	2 levels
4	Total number of time required to compute one element:	6 levels
5	Total time taken to compute 7 lines serially 7 x 6 levels	42 levels
	Speed up obtained by our scheme = $\frac{42}{15} = 2.8$	

Table 3: Comparison of acceleration with value of m = n = 100

	Values	Speed Up	Speed blocking factor
Serial approach	4mn 40000	1	-
Hardware acceleration	4(m+n-1) 796	50	-
Rec. Var. Exp with b = 3 T a	$7(\frac{m}{3} + \frac{n}{3} - 1)$ 460	87	1.74

Table 4: comparison of result

	Block size	Speed blocking factor
[14]	2	1.36
Our scheme	3	2.8
	% speed	51%

III. Results and Discussion

Table 1 shows processing in 3 blocks and table2 computes 7 lines serially. The results obtained by this method clearly confirmed that a reasonable amount of speed (51%) is gained when the block size is increased as shown in table 3 and table 4. In either case of table 3 and table 4 (when block size = 3), we gained more acceleration than when block size =2. The final result with block size =3 is compared with [14] when block size = 2, this shows that acceleration is achieved with increment in block size.

IV. Conclusion

In this paper we presented acceleration of SWA using Recursive Variable Expansion and better acceleration is achieved when the block size is increased. Our result is compared with other researcher and we concluded that block size determines acceleration in the algorithm of Smith-Waterman.

References

[1] [1] Hassan K. B., Kazeem A. G. "Residue Number System: An Important Application in Bioinformatics", International Journal of Computer Applications (0975 – 8887) Volume 179 – No.10, January 2018

[2] Hassan K. B., Kazeem A. G." Application of Smith-Wateman and NeedlemanWunsch Algorithm in Pairwise Sequence Alignment of Deoxyribonucleic Acid", Proc. of the 1st International Conference of IEEE Nigeria Computer Chapter In collaboration with Dept. of Computer Science, University of Ilorin, Ilorin, Nigeria – 2016.

[3] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," J. Mol Biol., vol. 48, pp. 443–453, 1970

[4] S.F Altschul etal. Basic Alignment Search Tool. *J.Mol.Biol.*, pages403-410,1990

[5] T. Smith and M. Waterman, "Identification of common molecular subsequences," J. Mol. Biol., vol. 147, pp. 195–197, 1981.

[6] Zubair, N Zaid A, Koen B, M<udassir S. " Acceleration of Smith-Waterman using Recursive Variable Expansion" conference paper DOI: 10.1109/DSD.2008.32

[7] Lipman, DJ; Pearson, WR (1985). "Rapid and sensitive protein similarity searches" Science 227 (4693): 1435–41. doi:10.1126/science.2983426. PMID 2983426.

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," Journal of Molecular Biology, vol. 215, no. 3, pp. 403–410, 1990. View at Publisher · View at Google Scholar · View at Scopus [12] S. F. Altschul, T. L. Madden, A.

[9] Schäffer et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," Nucleic Acids Research, vol. 25, no. 17, pp. 3389–3402, 1997. View at Publisher · View at Google Scholar · View at Scopus

[10] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences", Journal of Molecular Biology, vol. 147, pp: 195–197,

[11] Laiq Hasan and Zaid Al-Ars, "Performance Improvement of the Smith-Waterman Algorithm", Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2007), November 29–30, 2007, Veldhoven, The Netherlands.

[12] Y. Yamaguchi, Y. Miyajima, T. Maruyama, and A. Konagaya, "High Speed Homology Search Using Run-Time Reconfiguration", FPL 2002.
[13] A. Di Blas et. al., "The UCSC Kestrel Parallel Processor", IEEE Transactions on Parallel and Distributed Systems, vol. 16(1), pp: 80–92, 2005.
[14] Laiq Hasan Zaid Al-Ars Zubair Nawaz Koen Bertels "Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion"
[15] Laiq Hasan Zaid Al-Ars Zubair Nawaz "A Novel Approach for Accelerating the Smith-Waterman Algorithm using Recursive Variable Expansion" Delft University of Technology Computer Engineering Laboratory Mekelweg 4, 2628 CD Delft, The Netherlands, 2014
[16] W. R. Pearson and D. J. Lipman, "Rapid and Sensitive Protein Simlarity Searches",Science, vol. 227, pp: 1435–1441, 1985.
[17] S. F. Altschul, Gish, W. Miller, W. Myers and D. J. Lipman, "A Basic Local Alignment Search Tool", Journal of Molecular Biology, vol. 215, pp: 403–410, 1990.
[18] J. Chiang, M. Studniberg, J. Shaw, S. Seto and K. Truong, "Hardware Accelerator for Genomic Sequence Alignment", Proceedings of the 28th IEEE EMBS Annual International Conference, Aug 30–Sept 3, 2006, New York City, USA.
[19] Y. Yamaguchi, Y. Miyajima, T. Maruyama, and A. Konagaya, "High Speed Homology Search Using Run-Time Reconfiguration", FPL 2002.
[20] Zubair Nawazl, Mudassir Shabbir, Zaid AlArs,etal " Acceleration of Biological Sequence Alignment using Recursive Variable Expansion.