Solving Sequence Alignment Problem using Pipeline Approach

Pankaj Agarwal[1] and **S. A. M. Rizvi**[2]

Abstract - ***This paper presents two models based on pipeline approach for determining pair-wise sequence alignment of two molecular sequences. One of the models considers a variation of Needleman-Wunsch method as a basic algorithm and other is based on the use of scoring matrix for alignment. The basic purpose of using the pipelines is to reduce the time-complexity of alignment significantly. Paper also discusses the design & implementation of the basic linear version of the algorithms in our software tool by the name "Sequence Comparison and Analysis Tool [SCAT]". Our tool also provides the option of sequence alignment on the basis of common grouping like chemical, functional & structural etc. The software tool is implemented using Visual Basic-6 package with user-friendly windows environment.***

Index Terms - Sequence Alignment, Pipeline, Needleman-Wunsch Algorithm, Scoring Matrix etc.

1. INTRODUCTION

Sequence comparison can be defined as the problem of finding, which parts of the sequences are similar and which parts are different **[1,4,5]**. It is regarded as the building block for many other, more complex problems such as multiple alignments (the comparison of a group of related sequences) and the construction of phylogenetic trees that explain the evolutionary relationship among species. Sequence comparison is actually a well-know problem in computer science. For the computer scientist, bimolecular sequences are just another source of data. Indeed, one that has experienced a tremendous growth in interest to the point that it has spawned an interdisciplinary field of its own; generally know as *bioinformatics*, *computational molecular biology* or just *computational biology* **[4,5]**. As biological databases grow in size, faster algorithms and tools are needed **[6-15]**.

Our interest is to identify similarities and differences between two sequences by comparing them with each other. Generally, a measure of how similar they are is also desirable. A typical approach to solve this problem is to find a good and plausible *alignment* between the two sequences. If two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps as indels (that is, insertion or deletion mutations) introduced in one or both lineages in the time since they diverged from one another. The objective is to *match* identical subsequences as far as possible. An alignment can bee seen as a way of transforming one sequence into the other. Once the alignment is produced, a *score* can be assigned to each pair of aligned letters, called *aligned pair*, according to some chosen scoring scheme such as PAM and BLOSUM **[4,5]** that take into account physicochemical properties or evolutionary knowledge of the sequences being aligned.

Computational approaches to sequence alignment generally fall into two categories: *global alignments* and *local alignments*. Calculating a global alignment is a form of global optimization that "forces" the alignment to span the entire length of all query sequences. By contrast, local alignments identify regions of similarity within long sequences that are often widely divergent overall. Local alignments are often preferable, but can be more difficult to calculate because of the additional challenge of identifying the regions of similarity.

[1]*Asst. Professor, Krishna Institute of Engineering and Technology, Department of Computer Science and Engineering, Ghaziabad, U.P*

[2]*Associate Professor, Department of Computer Science, Jamia Millia Islamia Central University, New Delhi*

2. BACKGROUND

In our proposed method we have applied a multi-Pipeline approach to the standard global alignment algorithm referred as Needleman-Wunsch method. So let us first understand the working principle behind Needleman-Wunsch algorithm **[2]**. It computes the similarity between two sequences A and B of lengths m and n, respectively, using a dynamic programming approach. Dynamic

Pogramming is a strategy of building a solution gradually using simple recurrences **[3]**. The key observation for the alignment problem is that the similarity between sequences A[1..n] and B[1..m] can be computed by taking the maximum of the three following values:

1. The similarity of A[1..n −1] and B[1..m −1] plus the score of substituting A[n] for B[m];
2. The similarity of A[1..n −1] and B[1..m] plus the score of deleting aligning A[n];
3. The similarity of A[1..n] and B[1..m −1] plus the score of inserting B[m].

From this observation, the following recurrence can be derived:

match (A[1..i], B[1..j]) = match (A[1..i −1], B[1..j −1]) + sub (A[i], B[j]);

max{match (A[1..i −1], B[1..j]) + Del (A[i]);

match (A[1..i], B[1..j −1]) + Ins (B[j]) }

Where match (A, B) is a function that gives the similarity of two sequences A and B, and sub (a, b), Del (c) and Ins (c) are scoring functions that give the score of a substitution of character 'a' for character 'b', a deletion of character 'c', and an insertion of character 'c', respectively.

This recurrence is complete with the following base case:

match (A[0], B[0]) = 0; where A[0] and B[0] are defined as empty strings.

To solve the problem with this recurrence, the algorithm generally builds an (n +1) × (m +1) matrix where each M[i, j] represents the similarity between sequences A[1..i] and B[1..j]. The first row and the first column represent alignments of one

sequence with spaces. M[0, 0] represents the alignment of two empty strings, and is set to zero. All other entries are computed with the following formula:

M[i, j] = M[i −1, j −1] + Substitute (A[i], B[j]); // *if A[i]=B[j]*
Max{ M[i −1, j] + Del (A[i] ; M [i, j −1] + Ins (B[j] }
// *if A[i]<>B[j]*

The matrix can be computed either row by row (left to right) or column by column (top to bottom). In the end, M[n, m] will contain the similarity score of the two sequences. Since there are (m+1) · (n+1) positions to compute and each take a constant amount of work, this algorithm has time complexity **[3]** of $O(n^2)$. Clearly, it has also quadratic space complexity since it needs to keep the entire matrix in memory.

Once the matrix has been computed, the actual alignment can be retrieved by tracing a path in the matrix from the last position to the first. The trace is a simple procedure that compares the value at each M[i, j] to the values of its left, top and diagonal entries according to the formula given above. For instance, if M[i, j] = M [i, j −1] + Ins (B[j]), the trace reports an insertion of character B[j] and proceeds to entry M[i, j −1]. Alternatively, pointers can be saved on each entry during the computation of the matrix so that this evaluation step can be avoided at the cost of more memory usage. Since the path can be as long as $O(m + n)$, this procedure has linear time complexity. Note that sometimes more than one path can be traversed and, as a result, multiple high-scoring alignments can be produced. In the matrix of Figure 1, two optimal alignments can be retrieved

A = A C A A G A C A G – C G T
B = A G A A C A – A G G C G T

It is often useful to see the dynamic programming solution for the sequence alignment problem as a directed weighted graph with (n +1) × (m +1) nodes representing each entry (i, j) of the matrix, and having the following edges:

- ((i −1, j −1), (i, j)) with weight equals to sub (A[i], B[j]);
- ((i −1, j), (i, j)) with weight equals to Del (A[i]);
- ((i, j −1), (i, j)) with weight equals to Ins (B[j]);

A path from node (0, 0) to (n, m) in the *alignment graph* corresponds to an alignment between the two sequences, and the problem of retrieving an optimal alignment is converted to the problem of finding a path in the graph with highest weight.

Needleman-Wunsch method works fine for short sequences but for longer sequences the performance of the algorithm degrades quite considerably due to its $O(n^2)$ behavior. Our proposed method improves the time complexity to O(n) which is a significant improvement.

3. PROPOSED METHODOLOGY

Problem 3.1: Sequence Alignment of two molecular sequences

In recent years use of parallel algorithms and methods **[18,19,20]** has gained a lot of attention by researchers particularly in the area of sequence comparison related problems in molecular biology. We have proposed a multi-Pipeline strategy with two-stages per pipeline for alignment of two sequences. A delay of one unit time is inserted in each of the successive pipelines as each next pipeline is data dependent on its previous pipeline and thus delay enables the availability of data for each successive pipeline. Thus pipelines do not work concurrently with each other; rather they follow a sequential order while execution i.e. with the start of initial clock pulse pipeline-1 comes into play; at the second clock pulse pipeline-2 takes off and in similar fashion each of the other pipelines starts in successive clock pulses following a delay of one unit every time.

In spite of this forced delay of one unit in each successive pipeline time-complexity of the algorithm improves significantly. The computation involved in the two stages employed in each pipeline is given below with a general assumption that each stage consumes one unit of cycle time.

The time complexity of the general algorithm given as below will take O(m*n) which becomes quite significant as the size of the sequences grows and thus is not feasible at all.

```
For i=1 to m
   For j= 1 to n
      If A[i]=B[j] then
      M[i,j]=M[i-1,j-1]+Sub[A(i),B(j)],
        Else
      M[i,j]=Max{M[i-1,j]+Del[A(i)], M[i,j-1]+Ins[B(j)}
```

Consider two short sequences
ACAAG----------length 5
AGAAC----------Length 5
We need to compute M[5,5]
M[0,j] and M[i,0] are initialized

Figure 3 shows the result of applying the general algorithm which in this case will take 25 units of time to align two sequences each of length 5. Figure 4 shows how the matrix of order O (m*n) is filled by applying the proposed method allowing a delay of one unit at the beginning of each pipeline. Use of five pipelines has been depicted. Clearly there is significant improvement in the time complexity where it only takes 10 units of cycle-time to complete the process. In general the time complexity can be given as O(c*n) where 'c' is a constant term which is a very significant improvement over O(m*n)

Figure 5 given above shows the general architecture of the proposed pipeline-model with N functional units including Fetch units [F_i], Decoders [D_i], Execution unit with Adders [A_i], Comparators [C_i] and Storage units [S_i]

Problem 2: Determining the longest Continuous Subsequence with no gaps in given two sequences.

Some times we are more interested in finding the longest conserved region from two given molecular sequences. The proposed model based on pipeline approach is an attempt to solve the above stated problem. Again we propose a two stage multi-Pipeline model. Input to the pipeline is two DNA sequences which are converted in all the six-reading frames into corresponding protein sequences and thus resulting in six pairs of amino acid sequences. For each of the sequence pairs; matrix of order m*n is constructed based on some scoring matrix where m & n are the lengths of the sequences respectively.

Here we have proposed the use of six-pipelines each with two stages where all the six pairs of obtained sequences are input to one pipeline. All the six-pairs of sequences can be aligned concurrently with each other and thus improving the time complexity significantly. Figure 9 shows how the pipeline works for the given prepared matrix in figure 7.

Traditional algorithms would have taken O(6*n*m) time in the worst case and even the best algorithm would have taken O(6*n) time-complexity. However our strategy provides a better time complexity of O(n) in the worst-case with some overhead on the required resources in the form of multiple functional units. This is indeed a very significant improvement. Method does require the existence of multiple functional units like loaders, adders etc.

All the six pairs of obtained sequences can be mapped on to the six pipelines simultaneously as shown in figure 9 (here we have not shown the six pairs of obtained sequences converted in all the six reading frames). Scoring matrices are constructed for each pair of all the six sequences where values in the matrix are identified by the taken variables a_{ij}, b_{ij}, c_{ij}, d_{ij}, e_{ij} and f_{ij}. Each of the pipelines has global variables by the names S_i, T_i, Q_i, W_i, X_i and Z_i respectively that computes the sum starting from the residues positions a_{i1} to a_{ik} for each of the six sequences. Then we look for the maximum of the obtained sum values in each of the sequence pairs. For example in the above taken sample sequence the sum $S2=S2+a_{21} + a_{32} + a_{43} + a_{54}=10+7+6+8=31$ is the maximum sum among the sum values S1, S2, S3, S4, and S5.

The best alignment corresponding one of the obtained pairs of sequences (one of the six reading frames) is a_{21}, a_{32}, a_{43}, a_{54} i.e. at pipeline 1.

D A L T N
| | | |
T **D A L T**

Where aligned characters are marked by pipe symbols. Similarly the alignment for the other pairs of sequences can be obtained simultaneously reducing the time complexity of the algorithm significantly.

X_i and Z_i respectively that computes the sum starting from the residues positions a_{i1} to a_{ik} for each of the six sequences. Then we look for the maximum of the obtained sum values in each of the sequence pairs.]

4. IMPLEMENTATION

Here we have shown the screen formats of the implementation of the linear versions of the presented algorithms in our tool named as '*Sequence Comparison and Analysis Tool*'. The tool actually provides the solution to number of sequence comparison problems prevalent in molecular biology. Figure 10 show the interface that captures all the input details for aligning two sequences. As it can be seen sequence alignment can be done in four ways i.e. between nucleotide-to-nucleotide, nucleotide to proteins, Proteins to proteins and proteins to nucleotide. For a given input DNA sequence, one can not only consider it's upper & lower strands but also the reverse strand in either case. Alignment can be done for all the sequences obtained in six reading frames. Both the local and global alignments are possible. One can also provide the values for residue match mismatch and gap value. A number of algorithms including standard and self developed {algorithms are a part of our research papers already published in various journals and conferences **[21-27]** } are implemented in our tool (description of these algorithms are beyond the scope of this paper). One can align the sequences based on various scoring matrices also such as PAM & BLOSUM . Four types of alignment have been considered i.e. exact alignment, gap alignment, alignment based on groupings and ends-free alignment. Result window is quite user-friendly showing the alignment score and % of matched residues.

5. CONCLUSIONS AND FUTURE WORK

The proposed models can be easily implemented on parallel computers with multiple functional pipelines and will improve the time complexity of aligning the sequences. Assumption of multiple pipelines and functional unit improves the time complexity of the standard algorithms quite considerably from $O(n^2)$ to O(n). The most significant part of the algorithm is its ability to align more than one pair of sequences simultaneously with no additional overhead. Use of data-flow computers can be quite useful for the discussed sequence alignment problem and can provide even a better solution for sequence comparison types of jobs. we hope to come up with a better solution in our next paper by using the strategy data flow computing.

6. REFERENCES

[1] Apostolico, A. and R. Giancarlo, "Sequence Alignment in Molecular Biology", Purdue University Technical Report, PURDUE CS TR 95-075, 1975.

[2] Needleman, S. B. and C. D. Wunsch, "A General Method Applicable to Search for Similarities in the Amino Acid Sequence of Two Proteins", Journal of Molecular Biology, 48:443-453, 1970.

[3] Cormen, T. H., C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms", Second edition, MIT Press, 2001.

[4] Lesk, A., "Computational Molecular Biology", Oxford University Press, 1988.

[5] Setubal, J. C. and J. Meidanis, "Introduction to Computational Molecular Biology", PWS Publishing Company, 1997.

[6] Altschul, S., W. Gish and W. Miller, E. W. Myers and D. Lipman, "A Basic Local Alignment Search Tool", J. Molecular Biology, 215:403-10, 1990.

[7] Masek, W.J., and M.S. Paterson, "A faster algorithm for computing string edit distances", J. Comput. Syst. Sci., 20, 18{31 (1980).

[8] Sheik, S. S., Aggarwal, S. K., Poddar, A., Balakrishnan, N. and Sekar, K., "A FAST pattern matching algorithm." *J. Chem. Inf. Comput. Sci.*, 2004, 44, 1251–1256.

[9] Charras, C. and Lecroq, T., *Handbook of Exact String matching algorithms* (available at the website: http://www-igm.univ-mlv.fr/ ~lecroq/string/).

[10] Needleman, S.B. and C.D. Wunsch, "A general method applicable to the search of similarities in the amino acid

sequences of two proteins", J. Molec. Biol. 48(1970) 443-453.

[11] Smith, T.F. and M.S. Waterman, "Identification of common molecular subsequences", J. Molecular Biology; 147 (1981) 195-197.

[12] Hirschberg, D.S. "A Linear Space Algorithm for computing maximal Common Subsequences". CACM V18 No6 p431-343 1975.

[13] Myers, E. W. "An O (nd) Difference algorithm and its variations" Algorithmica, vol.2, 1986, pp.251-226.

[14] Edmiston, E.W., Core, N.G., Saltz, J.H and R.M. Smith, "Parallel processing of biological Sequence Comparison Algorithms", Internet. J. Parallel Programming 17(3) (1988) 259-275.

[15] Delcher, A.L. et al.(2002). Fast Algorithms for Large scale Genome Alignment and Comparison. Nucleic Acids Research, Nuclei Acids Research, 30(11), 2002, pp.2478-2483

[16] Gusfield, D., Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.

[17] Durbin, R., S. Eddy, A. Krogh and G. Mitchison, Biological Sequence Analysis, Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, 1998.

[18] Huang, X. "A space-efficient Parallel Sequence Comparison algorithm for a message-passing multiprocessor", Internat.J. Parallel Programming 18(3) (1989) 223-239.

[19] Edmiston, E.W., Core, N.G., Saltz, J.H and R.M. Smith, "Parallel processing of biological Sequence Comparison Algorithms", Internet. J. Parallel Programming 17(3) (1988) 259-275.

[20] Aluru, S., and N. Futamura, "Parallel biological sequence comparison using prefix computations", J. Parallel Distrib. Computt.63 (2003) 264-272.

[21] Pankaj Agarwal and S.A.M. Rizvi; "*The Future: Integrated Environment For Bioinformatics*"; Proceedings of Bioinformatics National Conference (BIOCON-2006), Sirsa, Haryana (2006).

[22] Pankaj Agarwal and S.A.M. Rizvi; " *Three New Approaches For Finding Exact Patterns In DNA Sequences*"; Proceedings of Bioinformatics National Conference (BIOCON-2006), Sirsa, Haryana (2006).

[23] Pankaj Agarwal and S.A.M. Rizvi, "*A New Index-Based Parallel Algorithm for finding Longest Common Subsequence in Multiple DNA Sequences*" ; Proceedings of 7th International Conference on Cognitive Systems, New Delhi, India (2005).

[24] Pankaj Agarwal and S.A.M. Rizvi; "*A New Bucket-Based Algorithm For Finding LCS From Two Given Molecular Sequences*"; IEEE Proceedings in 4th IEEE International Conference of Information and Technology, Las Vegas, USA (2006);

[25] Pankaj Agarwal and S.A.M. Rizvi, "*Prediction Of Secondary Structure Of Proteins Using An Artificial Intelligence Technique: The Nearest Neighbor Strategy*"; Proceedings of International Conference on Bioinformatics, New Delhi, India (2006); [Abstract Published]

[26] Pankaj Agarwal and S.A.M. Rizvi; " *A New Parallel Technique for Alignment of Two Amino Acid Sequences* "; Accepted at Computer Science & IT Education Conference to be hosted by Institute of Technology, Mauritius in November 2007.

[27] Pankaj Agarwal and S.A.M. Rizvi, " *Neural Network For Prediction Of Initiation Site Of mRNA Sequences* "; Published at 5th Annual CISTM Conference on Information Science Technology and Management to be held in July 2007 at Hyderabad.

		0	1	2	3	4	5	6	7	8	9	10	11	12
		-	A	G	A	A	C	A	A	G	G	C	G	T
0	-	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
1	A	-1	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
2	C	-2	0	0	-1	-2	-1	-2	-3	-4	-5	-6	-7	-8
3	A	-3	-1	-1	1	0	-1	0	-1	-2	-3	-4	-5	-6
4	A	-4	-2	-2	0	2	1	0	1	0	-1	-2	-3	-4
5	G	-5	-3	-1	-1	1	1	0	0	2	1	0	-1	-2
6	A	-6	-4	-2	0	0	0	2	1	1	1	0	-1	0
7	C	-7	-5	-3	-1	-1	1	1	1	0	0	2	1	0
8	A	-8	-6	-4	-2	0	0	2	2	1	0	1	1	2
9	G	-9	-7	-5	-3	-1	-1	1	1	3	2	1	2	1
10	C	-10	-8	-6	-4	-2	0	0	0	2	2	3	2	1
11	G	-11	-9	-7	-5	-3	-1	-1	-1	1	3	2	4	3
12	T	-12	-10	-8	-6	-4	-2	0	0	0	2	2	3	5

Figure 1: Standard dynamic programming matrix for the global alignment of sequences A=ACAAGACAGCGT and B=AGAACAAGGCGT with paths to retrieve optimal alignments indicated with arrows.

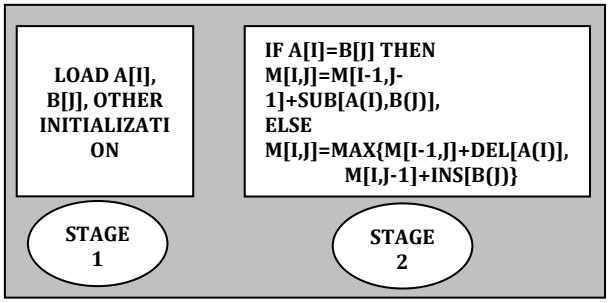

Figure 2: Two stage single pipeline

A_i/B_j	--	A	G	A	A	C
--	0	-1	-2	-3	-4	-5
A	-1	1	0	-1	-2	-3
C	-2	0	0	-1	-2	-1
A	-3	-1	-1	1	0	-1
A	-4	-2	-2	0	2	1
G	-5	-3	-1	-1	1	1

Figure 3: Result of alignment [Algorithm will take 25(5*5)

		1	2	3	4	5	6	7	8	9	10	
P1	S1	Load A[1],B[1]	Load A[1],B[2]	Load A[1],B[3]	Load A[1],B[4]		Load [1],B[5]					
	S2		M[1,1]=1	M[1,2]=0	M[1,3]=-1	M[1,4]=-2	M[1,5]=-3					
P2	S1		Load A[2],B[1]	Load A[2],B[2]	Load A[2],B[3]	Load A[2],B[4]	Load [2],B[5]					
	S2			M[2,1]=0	M[2,2]=0	M[2,3]=-1	M[2,4]=-2	M[2,5]=-1				
P3	S1			Load A[3],B[1]	Load A[3],B[2]	Load A[3],B[3]	Load A[3],B[4]	Load A[3],B[5]				
	S2				M[3,1]=-1	M[3,2]=-1	M[3,3]=1	M[3,4]=-2	M[3,5]=-3			
P4	S1				Load A[4],B[1]	Load A[4],B[2]	Load A[4],B[3]	Load A[4],B[4]	Load [4],B[5]			
	S2					M[4,1]=-2	M[4,2]=-2	M[4,3]=0	M[4,4]=2	M[4,5]=1		
P5	S1					Load A[5],B[1]	Load A[5],B[2]	Load A[5],B[3]	Load A[5],B[4]	Load A[5],B[5]		
	S2						M[5,1]=-3	M[5,2]=-1	M[5,3]=-1	M[5,4]=1	M[5,5]=1	

Figure 4: Result of the proposed model with one unit of delay in each successive pipeline

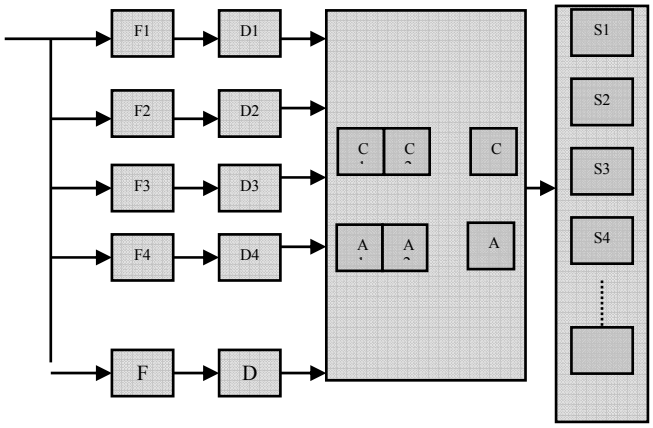

Figure 5: General architecture of the proposed-Pipeline [F: Fetch unit, D: Decode Unit, C: Comparator, A: adders, S: store units]

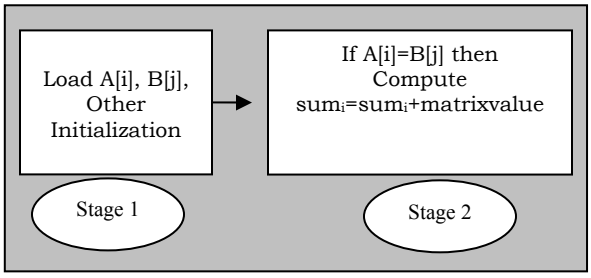

Figure 6: Two stage single pipeline.

A_i/B_j	D	A	L	T	N
T	$-2[a_{11}]$	$0[a_{12}]$	$-3[a_{13}]$	$8[a_{14}]$	$0[a_{15}]$
D	$10[a_{21}]$	$-3[a_{22}]$	$-7[a_{23}]$	$-2[a_{24}]$	$2[a_{25}]$
A	$-3[a_{31}]$	$7[a_{32}]$	$-3[a_{33}]$	$0[a_{34}]$	$-3[a_{35}]$
L	$-7[a_{41}]$	$-3[a_{42}]$	$6[a_{43}]$	$-3[a_{44}]$	$-6[a_{45}]$
T	$-2[a_{51}]$	$0[a_{52}]$	$-3[a_{53}]$	$8[a_{54}]$	$0[a_{55}]$

Figure 7: Alignment scores using BLOSUM-80

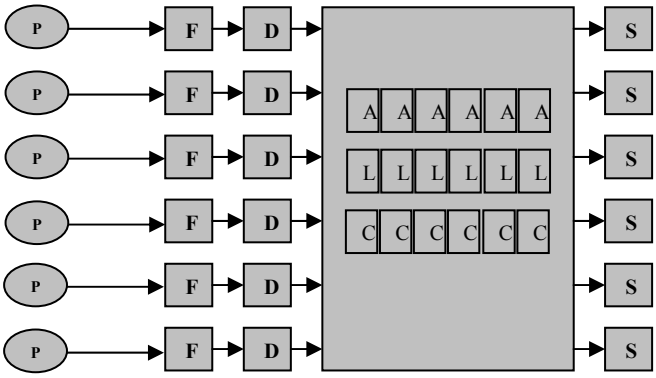

Figure 8: Architecture with multiple Functional Unit {P: Pipeline, F: Fetch Unit, D: Decode unit, A: adder, L: Loader, C: Comparator, S: storage unit}

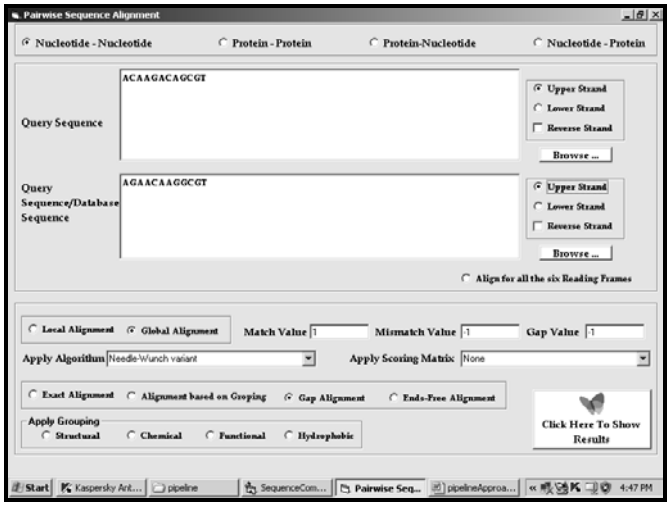

Figure 10: Interface that captures the inputs for aligning

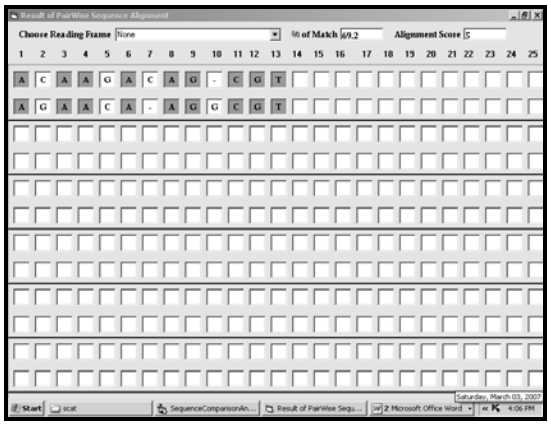

Figure 11: Showing alignment of the two input sequences with alignment score.

	1	2	3	4	5
Pipeline 1	Load a_{11} S1=0	Load a_{21} S2=0	Load a_{31} S3=0	Load a_{41} S4=0	Load a_{51} S5=0
		Load a_{22} S1=S1+ a_{22}	Load a_{32} S2=S2+ a_{32}	Load a_{42} S3=S3+ a_{42}	Load a_{52} S4=S4+ a_{52}
			Load a_{33} S1=S1+ a_{33}	Load a_{43} S2=S2+ a_{43}	Load a_{53} S3=S3+ a_{53}
				Load a_{44} S1=S1+ a_{44}	Load a_{54} S2=S2+ a_{54}
					Load a_{55} S1=S1+ a_{55}
Pipeline 2	Load b_{11} T1=0	Load b_{21} T2=0	Load b_{31} T3=0	Load b_{41} T4=0	Load b_{51} T5=0
		Load b_{22} T1=T1+ a_{22}	Load b_{32} T2=T2+ b_{32}	Load b_{42} T3=T3+ b_{42}	Load b_{52} T4=T4+ a_{52}
			Load b_{33} T1=T1+ b_{33}	Load b_{43} T2=T2+ b_{43}	Load b_{53} T3=T3+ a_{53}
				Load b_{44} T1=T1+ b_{44}	Load b_{54} T2=T2+ b_{54}
					Load b_{55} T1=T1+ b_{55}
Pipeline 3	Load c_{11} Q1=0	Load c_{21} Q2=0	Load c_{31} Q3=0	Load c_{41} Q4=0	Load c_{51} Q5=0
		Load c_{22} Q1=Q1+ c_{22}	Load c_{32} Q2=Q2+ c_{32}	Load c_{42} Q3=Q3+ c_{42}	Load c_{52} Q4=Q4+ c_{52}
			Load c_{33} Q1=Q1+ c_{33}	Load c_{43} Q2=Q2+ c_{43}	Load c_{53} Q3=Q3+ c_{53}
				Load c_{44} Q1=Q1+ c_{44}	Load c_{54} Q2=Q2+ c_{54}
					Load c_{55} Q1=Q1+ c_{55}
Pipeline 4	Load d_{11} W1=0	Load d_{21} W2=0	Load d_{31} W3=0	Load d_{41} W4=0	Load d_{51} W5=0
		Load d_{22} W1=W1+ d_{22}	Load d_{32} W2=W2+ d_{32}	Load d_{42} W3=W3+ d_{42}	Load d_{52} W4=W4+ d_{52}
			Load d_{33} W1=W1+ d_{33}	Load d_{43} W2=W2+ d_{43}	Load d_{53} W3=W3+ d_{53}
				Load d_{44} W1=W1+ d_{44}	Load d_{54} W2=W2+ d_{54}
					Load c_{55} W1=W1+ c_{55}
Pipeline 5	Load e_{11} X1=0	Load e_{21} X2=0	Load e_{31} X3=0	Load e_{41} X4=0	Load e_{51} X5=0
		Load e_{22} X1=X1+ e_{22}	Load e_{32} X2=X2+ e_{32}	Load e_{42} X3=X3+ e_{42}	Load e_{52} X4=X4+ e_{52}
			Load e_{33} X1=X1+ e_{33}	Load e_{43} X2=X2+ e_{43}	Load e_{53} X3=X3+ e_{53}
				Load e_{44} X1=X1+ e_{44}	Load e_{54} X2=X2+ e_{54}
					Load c_{55} X1=X1+ c_{55}
Pipeline 6	Load f_{11} Z1=0	Load f_{21} Z2=0	Load f_{31} Z3=0	Load f_{41} Z4=0	Load f_{51} Z5=0
		Load f_{22} Z1=Z1+ f_{22}	Load f_{32} Z2=Z2+ f_{32}	Load f_{42} Z3=Z3+ f_{42}	Load f_{52} Z4=Z4+ f_{52}
			Load f_{33} Z1=Z1+ f_{33}	Load f_{43} Z2=Z2+ f_{43}	Load f_{53} Z3=Z3+ f_{53}
				Load f_{44} Z1=Z1+ f_{44}	Load f_{54} Z2=Z2+ f_{54}
					Load c_{55} Z1=Z1+ c_{55}

Figure 9: Working of the Proposed Pipeline [Each of the pipelines has global variables by the names S_i, T_i, Q_i, W_i,