A Parallel Pairwise Local Sequence Alignment Algorithm

Sanghamitra Bandyopadhyay, *Senior Member*, *IEEE*, Ramkrishna Mitra

***Abstract*—Researchers are compelled to use heuristic based pairwise sequence alignment tools instead of Smith-Waterman (SW) due to space and time constraints, thereby losing significant amount of sensitivity. Parallelization is a possible solution, though till date the parallelization is restricted to database searching through database fragmentation. In this article, the power of a cluster computer is utilized for developing a parallel algorithm, RPAlign, involving, firstly, detection of regions that are potentially alignable (RPAs), followed by their actual alignment. RPAlign is found to reduce the timing requirement by a factor of upto 9 and 78 when used with BLAST and SW respectively, while keeping the sensitivity similar to the corresponding method. For distantly related sequences, that remain undetected by BLAST, RPAlign with SW can be used. Again, for megabase scale sequences, when SW becomes computationally intractable, the proposed method can still align them reasonably fast with high sensitivity.**

***Index Terms*—BLAST, message passing interface (MPI), parallel computing, Smith-Waterman.**

I. Introduction

Pairwise sequence alignment is a challenging task because of the exponential growth of genomic information, necessitating large scale comparison of two input strings. The size of GenBank /EMBL/DDBJ nucleotide database is now doubling in every 15 months [1]. When searching databases to find out sequences similar to a given query sequence, the search programs compute an alignment score for every sequence in the database. This score represents the degree of similarity between the query and database sequence. A dynamic programming algorithm for computing the optimal local alignment score was first described by Smith and Waterman [2], and later improved by [3] for linear gap penalty functions. Though dynamic programming is the best alignment procedure so far, it is not suitable for large strings in terms of both time and space. For two strings of length m and n, the time and space complexities of the Smith and Waterman (SW) algorithm are $O(mn)$.

The time and space complexity had been improved to $O(rn)$ by [4], where r is the amount of allowed error, by considering only the useful part of the distance matrix. However, for large error rates, r is $O(m)$, so the complexity is still $O(mn)$. Later on, the space complexity of SW was improved to $O(n)$ [5]. Dynamic programming has been accelerated through GLASS by finding exactly matching long substrings first, but the time and space complexity are still high [6]. LAGAN[7] is another implementation of dynamic programming but is not applicable on a genome scale without prior information ("anchors") that directs comparison to orthologous regions.

There are many heuristic based search tools and they can be categorized into hash-table based search tools and suffix-tree based tools. FASTA [8], BLAST [9][10], MegaBLAST [11], BL2SEQ [12], WU-BLAST [13], SENSEI [14], FLASH [15], PipMaker [16], Pattern Hunter [17], BLAT [18], SSAHA [19], are methods that belong to the category of hash-table-based tools. These are basically achieved by "Seed-and-extend" methods. In a Seed-and-extend method, one or more exactly matching k-mers ("Seeds" or "hot-spots") provide initial evidence of possible similarity. These seeds are then extended to compute the final sequence alignments. The extension step is more accurate than the seeding step, but it is computationally expensive. These methods quickly abandon most candidate similarities because they don't immediately yield alignments that are likely to be statistically significant. Current hash-table-based search tools handle short queries well, but become very inefficient, in terms of both time and space, for long queries. The limitation of seed-and-extend methods have been overcome in [20], [21], [22]. In [22] a parallel technique called Pash was designed to compare genome-sized datasets. However it is not the best choice when indels are prevalent. As mentioned in [22], Pash is relatively inefficient when mapping a relatively small dataset onto a relatively larger one.

Suffix tree is another efficient approach on which various search tools have been developed. These include MUMmer [23], QUASAR [24], REPuter [25], AVID [26]. There are many significant problems with the suffix-tree based approach: they manage mismatches inefficiently (they are good for highly similar strings, but fail to recognize more distant homologies) and they have a high space overhead.

Most of the tools mentioned above require data structures larger than the database, some of them more than two orders of magnitude larger. Recent advances in parallelization makes it possible to implement BLAST (http://www.ncbi.nlm.nih.gov/BLAST) in a parallel setup as well. Earlier works on parallel sequence search mostly focus on distributing the query set across several cluster nodes [27], [28], [29] each of which executes a serial job. Throughput is increased, but the time for a particular query to complete is unchanged. Other existing parallel techniques have been focusing mostly on database segmentation. In this approach the database is partitioned among cluster nodes and an assigned part of the database is searched for the same query [30], [31]. This approach of database splitting was developed in the mpiBLAST [32]. Among the several published parallel BLAST codes, mpiBLAST reported the highest speed up, underwent the largest scalability tests, and has been directly integrated with the NCBI toolkit. A subsequent efficient algorithm pioBLAST [33] was developed which has reduced non-search overheads of mpiBLAST by focusing on the use of collective I/O and dynamic database partitioning. Parallel BLAST has also been implemented on supercomputers like the IBM Blue Gene/L [34]. It is based on optimally splitting up the set of queries as well as databases. It reduced the I/O, thereby delivering a fast, high throughput BLAST. This method is capable of performing at least 2 million BLAST searches per day against a database of 2.5 million protein sequences. The other useful works on the parallelization of BLAST are ParAlign [35], pp-Blast [36], ScalaBLAST [37]. All the above mentioned parallel BLAST implementations are based on searching database sequences in parallel by segmenting and distributing the set of query sequences or database sequences.

Although BLAST is widely used in the Bioinformatics community, it is well known to suffer from low sensitivity as compared to SW. In particular for the sequences which are distantly related BLAST may be unable to throw up any hit, a problem that SW can overcome. However, SW is known to be unable to compare two large DNA sequences due to its computational complexity. Some attempts in developing faster, parallel, implementations of the SW algorithm can be found in [38], [39] but these are essentially database searching algorithm. In [39] a vector implementation of SW makes the rigorous Smith-Waterman competitive with BLAST (within a factor of 5 or less) but for large scale DNA sequence it is not practical. Few attempts have been made for developing parallel algorithms for comparing a pair of large scale sequences. This requires proper fragmentation of the two sequences, and distribution of the fragments to the different nodes of a parallel computer. Not much work is available in this direction probably because it has been difficult to parallely identify those subsequences which are actually alignable in the two sequences though some sequential algorithm have been attempted in this regard[21].

In this article, we propose an efficient algorithm which can overcome this problem and can align two DNA or protein sequences in parallel by identifying regions that are potentially alignable (RPAs). Once this is done parallely, the task of aligning these subsequences can be easily parallelized resulting in a gain in computation time. Such a parallel algorithm, referred to as RPAlign, is developed in this article. It employs frequency counts in windows to detect the RPAs in the two subsequences. A cluster computer is utilized for implementing RPAlign using Message Passing Interface (MPI). Note that our task is not to propose a new alignment algorithm, but to improve the time requirement, through the use of judicious parallelism, of any pairwise local sequence alignment method.

II. Systems And Methods

The code is written in C using Message Passing Interface (MPI). A cluster of 18 nodes is used with Linux WS 3.0 standard operating system. Master node consists of Intel Xeon 2.8 GHz single CPU and 1 GB RAM. Each Slave or Worker node consists of Pentium IV 2.8 GHz CPU and 512 MB RAM. The bl2seq module of NCBI BLAST toolkit (version 2.2.15) is used for both DNA and protein sequences.

III. Algorithm And Implementation

The detection of RPA between two DNA or protein sequences is based on the computation of the frequency of each type of element. The system incorporates one master processor (*MP*) and *n*-1 worker processors (*WPs*). The proposed algorithm is described below in detail.

A. Efficient Data Handling forParallelProcessing

The *MP* and *WPs* parallely read the two input sequences S_l and S_s, assuming $|S_l| > |S_s|$, and determine their lengths. Each processor (including the *MP*, that is treated as WP_1 in the following discussion) then extracts one overlapping subsequence from the larger sequence S_l. Considering the length of the overlapping window to be denoted by *w*, the length of each subsequence or fragment F_i, i=1, 2, …, *n*, is given by

$$\frac{|S_l|}{n} + w,$$

where *n* is the number of nodes in the cluster. Therefore the start and end positions of fragment F_i denoted by $Start_i$ and End_i are given by,

$$Start_i = (i-1) * \frac{|S_l|}{n} + 1$$

and

$$End_i = i * \frac{|S_l|}{n} + w \cdot$$

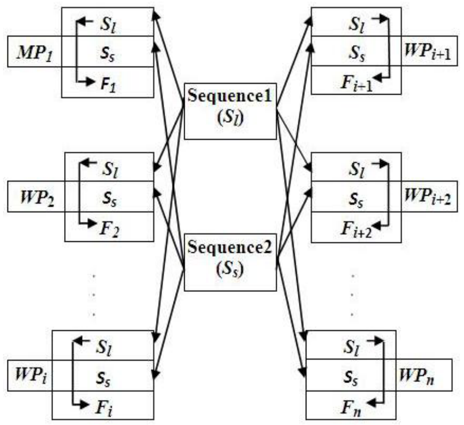

Fig. 1. Parallel I/O and dynamic partition of the larger sequence (Here S_l).

Parallel file I/O in a shared memory framework is used through which load balancing is performed and copying overhead is reduced.

B. *Computing Frequencies and Composite Scores*

The tasks performed by processor P_i, i=1, 2, …, n are outlined in Fig. 2. These are now described in detail.

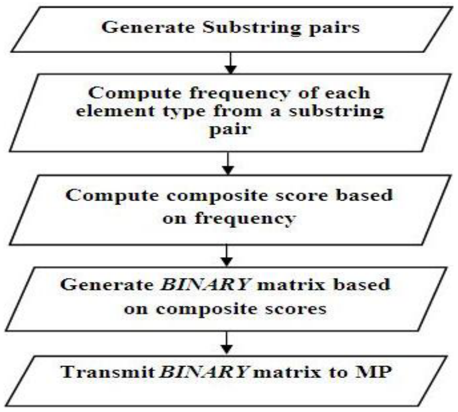
Generate Substring pairs
Compute frequency of each element type from a substring pair
Compute composite score based on frequency
Generate *BINARY* matrix based on composite scores
Transmit *BINARY* matrix to MP

Fig. 2. Flow chart of *BINARY* matrix formation in each processor.

F_i, the fragment of S_l read by P_i, is further divided into substrings of length w by sliding it one letter at a time to generate substrings F_{ij}, j=1, 2, …, $|Fi|$-w+1. The second sequence S_s is also divided into substrings of length w by shifting w letters at a time to yield S_{sk} substrings, where k = 1, 2, …, $\lceil |S_s|/w \rceil$. Then for every possible substring F_{ij} or S_{sk}, the frequencies of each type of element are determined as $f_e(F_{ij})$ or $f_e(S_{sk})$ for e=1, 2, …, 5 for DNA and e=1, 2, …, 20 for protein sequence. For the DNA sequence, elements are A, T, G, C and N where N stands for the unknown and for protein sequence there are 20 different types of amino acids present. A score is then computed on the basis of f for a substring pair. On the basis of this score RPA will be detected. As protein sequences are more complex in nature than the DNA sequences, and as substitution matrix plays an important role for the alignment, computation of RPA for protein sequences is much more complicated than for DNA sequences. This is first described below. On the other hand as DNA sequences are usually much larger than protein sequences, efficient data handling is essential. For this reason an optimization technique has been developed.

1) Protein Sequences: In the proposed algorithm BLOSUM 62 [40], a substitution matrix generally used for comparison of two protein sequences, is considered though other standard substitution matrices can also be implemented. According to this matrix, two identical residues can generate a score from +4 to +11 depending on the residue type; while for two non-identical residues the score can be generated from –4 to +3. On the basis of the scores of two amino acid residues, as provided by BLOSUM 62, amino acid pairs have been classified into 15 categories which are given in Table I. Note that the score generated by category 1 amino acid pair is ~3 times more than the score of category 7 amino acid pair. From category 1 to category 7 all amino acid pairs are identical and other categories provide the score of non identical amino acid pairs.

Let *Cat* (*Ae, Ae′*), return the category of amino acid pair [*Ae, Ae′*] from Table I. Then we define

$$BL\_Score(Cat(A_e, A_{e'}))$$

as the BLOSUM 62 score for the respective category as obtained from Table I. After computing the frequency *fe, e*=1, 2, …, 20 of all the substrings, composite scores (*CS*) are now computed as follows:

$$CS(F_{ij}, S_{sk}) = \sum_{e=1}^{20} \sum_{e'=e}^{20} \underset{e,e'}{Min}(f_e(F_{ij}), f_{e'}(S_{sk})) * \gamma \\ (BL\_Score\,(Cat\,(A_e, A_{e'}))) \tag{1}$$

where the function γ(x) is defined as

$$\gamma(x) = x \qquad \text{if } x > 0$$
$$\gamma(x) = 0 \qquad \text{if } x <= 0$$

Note that Eqn. 1 considers all those amino acid pairs which provide a positive BLOSUM 62 score.

2) DNA Sequences: Here the minimum frequencies of A, T, G, C, and N corresponding to each pair F_{ij} and S_{sk} are computed. On the basis of those frequencies, f_e, e=1, 2, …, 5, *CS* is now computed as,

$$CS(F_{ij}, S_{sk}) = \sum_{e=1}^{5} Min\,(f_e(F_{ij}), f_e(S_{sk})) \tag{2}$$

C. *Generating the BINARY Matrix*

It may be noted that *CS* is a gross over estimation of the actual alignment score of the two substrings. This is done on purpose to ensure that even after such an over estimation, if $CS < \theta$, where θ is a threshold value, then the corresponding substrings need not be considered as they are not alignable. Now a matrix called *BINARY* of dimension $\lceil |S_s|/w \rceil$ X $\lceil |F_i|/w \rceil$ is generated in node i, where each row and column represents w length of non overlapping letters of S_s and F_i respectively. Initialize *BINARY* matrix to all 0's. For each S_{sk}, k=1, 2, …, $\lceil |S_s|/w \rceil$, w consecutive fragments from F_i are used to compute w different *CS* values. If any of these *CS* values exceeds a threshold θ, then cells (k, j) and (k, j+1) of *BINARY* matrix are set to 1 as w consecutive *CS* values cover 2w-1 letters. Fig. 3 states this process formally.

Note that all the w *CS* values need not be computed. As soon as a value exceeding θ is obtained, the remaining substring pairs are not considered any further. Since DNA sequences can be extremely long leading to high computational cost for comparing all the *CS* values with θ, a procedure for optimizing this computation is described below.

```
For $k$= 1, 2, …, $\lceil |S_s|/w \rceil$
  For $j$=1, 2, …, $\lceil |F_i|/w \rceil$
    Compare $F_{ij}$, $j$= ($j$-1)$w$+1, …, $jw$,
    and $S_{sk}$ to provide $CS^1$, $CS^2$, …, $CS^w$.
    If max ($CS^k$, $k$=1,…, $w$) > $\theta$ then
      set $BINARY(k, j) = BINARY(k, j+1) =1$
    End If
  End
End
```

Fig. 3. Computation of *BINARY* matrix in processor *Pi*

D. Merging and Redistribution

After each WP_i completes the computation of the *BINARY* matrix these are transmitted to the *MP*. Here the matrices are collated side by side to yield a matrix called *RPA_DETECTION* matrix. In this matrix, diagonals that are strings of all 1's are found. The start and the end positions of these diagonals define the RPAs. Note that, there is a possibility for multiple overlapping surfaces of similarity (diagonals) in *RPA_DETECTION* matrix. As a result there is a chance that a particular sequence fragment may be included in multiple aligned segments. In RPAlign, this problem is reduced by merging the adjacent overlapping diagonals. But the overlapping diagonals which are not adjacent are also considered as RPAs, the reason being that prior knowledge about which RPA will provide the best score is absent. The diagonal which is a subset of another diagonal is not considered as an RPA. Finally all these RPAs are redistributed to each node including the *MP* in such a way that the load balancing will be achieved. Thereafter each node performs the actual alignment either by BLAST (providing RPAlign BLAST) or by SW implementation (providing RPAlign SW) and alignment output is stored in a shared memory space.

TABLE I
SCORE WISE CATEGORIES OF AMINO ACID PAIRS ON THE BASIS OF SUBSTITUTION MATRIX (BLOSUM62)

Score	Category	Amino Acid pairs
11	1	[w, w]
9	2	[c, c]
8	3	[h, h]
7	4	[p, p][y, y]
6	5	[d, d][f, f][g, g] [n, n]
5	6	[e, e][k, k][m, m][q, q][r, r][t, t]
4	7	[a, a][i, i][l, l][s, s][v, v]
3	8	[i, v][f, y]
2	9	[d, e][e, q][h, y][k, r][l, m][i, l][w, y]
1	10	[s, t][a, s][n, s][d, n][e, k][q, r][k, q][i, m][f, w][h, n][l, v][m, v]
0	11	[a, c][g, s][d, s][e, s][k, s][n, t][a, g][e, n][q, n][n, r][k, n][d, q][e, h][e, r][h, q][m, q][h, r][f, i][f, l][a, t][a, v][g, n][q,s][f, m][t, v]
-1	12	[c, s][c, t][c, m][c, i][c, l][c, v][p, s][h, s][r, s][m, s][r, t][m, t][a, p][d, p][e, p][p, q][k, p][a, e][a, q][a, r][a, k][a, m] [a, i][a, l] [d, g][d, h][d, k][d, t][e, t][q, y][i, t][l, t][k, t][q, t][h, k][f, h][m, r][k, m][m, y][m, w][p, t][i, y][l, y][f, v][v, y]
-2	13	[c, f][c, y][i, s][l, s][s, v][f, s][s, y][f, t][t, y][g, p][h, p][n, p][h, t][p, r][m, p][p, v][a, d][a, h][a, f][a, y][e, g][g, q][g, h][g, r] [g, k][g, t][g, w][m, n][n, y][d, r][e, m][e, y][l, q][e, v][k, v][q, v][q, w][h, m][h, w][t, w][l, r][r, y][k, l][k, y][c, w][l, w][a, n]
-3	14	[c, p][c, g][c, n][c, d][c, q][c, h][c, r][c, k][s, w][i, p][l, p][p, y][a, w][g, m][f, g][g, y][i, n][l, n][n, v][f, n][d, m][d, i][d, v,][d, f] [d, y][e, i][e, l][e, f][e, w][i, q][f, q][h, i][h, l][i, r][r, f][r, v][r, w][i, k][f, k][k, w][i, w][v, w][g, v][h, v]
-4	15	[c, e],[f, p][p, w][g, i][g, l][n, w][d, l][d, w]

The substring F_{ij+1} is generated by sliding F_{ij} by one letter to the right. That is, to generate a new substring, F_{ij+1}, one letter is removed from the left most position and one new letter is inserted at the right most position of the current substring. It can generate two possible effects on F_{ij+1}.

(a) If the inserted and deleted letters are the same then *f* vector of F_{ij} and F_{ij+1} are the same.

(b) If the inserted and deleted letters are not identical then the frequency of one element type (A, T, C, G or N) is decreased by one, and the other one is increased by one.

On the basis of the above logic it is clear that *CS* values for DNA sequences can be changed by at most 1. Now for computing the value in cell (k, j) and (k, j+1) of the *BINARY* matrix, *CS* is first computed between S_{sk} and F_{ij}. If $CS > \theta$, then as earlier, computation is discontinued and $BINARY(k, j) = BINARY(k, j+1) =1$. However, if $CS < \theta$, then let $\theta' = \theta - CS$. In this case the next comparison needs to be made between $F_{i(j+\theta'+1)}$ and S_{sk}. In this way when, $(j+\theta'+1) > w$, then the search is discontinued, since under no circumstance can any of the *CS* value exceed θ. If $(j+\theta'+1) < w$, then the process repeats.

IV. RESULTS

The speed and more importantly, the quality of alignment of the proposed method is evaluated using a set of eight amino acid, eight DNA and seven megabase scale DNA sequence pairs. The length of the input sequences ranged from less than 66 to 8797 residues for amino acid sequences and from 723 bp to 11.1 mb for nucleotide and genome sequences. The nucleotide sequences are described in Table II and the protein sequences are described in Table V. The window length *w* is chosen, in general, as min ($\frac{|S_l|}{40}$, 25000). The variation of the performance of RPAlign is studied for different values of the threshold θ in terms of the window size (or, different values of θ/w).

TABLE II
NUCLEOTIDE SEQUENCE PAIRS AND THEIR RESPECTIVE ID AND LENGTH

Pair	Sequence 1		Sequence 2	
	GI Number	Length(bp)	GI Number	Length (bp)
P1	92296557	723	114157166	1549
P2	118562368	6485	89142743	6736
P3	13273284	16571	508206	13246
P4	28876381	38206	303969	34214
P5	209811	35937	28876316	41796
P6	28876316	41796	28876437	40014
P7	112806880	44237	114804244	158484
P8	41179002	203828	114804244	158484

The performance of RPAlign is compared in terms of speed and quality with SW implementation and BLAST (bl2seq with –F F option to disable the filter query sequence option). For each pair of input sequences the time of the fastest of three consecutive runs is recorded.

A. *Sensitivity Analysis*

Measurement of the sensitivity of BLAST, RPAlign BLAST and RPAlign SW are computed based on the alignment provided by SW. Experiments with RPAlign are conducted for different fractions of θ/w, namely, 0.8, 0.85, 0.9 and 0.95. Note that here for RPAlign SW, the choice of an appropriate θ/w for a particular sequence pair depends on the characteristics of the sequences, their relationships, i.e. whether they are closely or distantly related. Not only that, their length is also an important consideration. Lower θ/w value is not suitable for the two closely related same length input sequences, as the proposed method will detect only one long diagonal whose start point is near the top left cell of the *RPA_DETECTION* matrix and end point is near the bottom right cell. As all other minor diagonals are subset of this diagonal, they are not considered as RPAs. Thus it will not be possible to parallelize the algorithm effectively. The situation will be the same as performing BLAST or SW in a stand alone mode. But for the sequences, which have less homology or have significant homology but their lengths are quite different from each other, θ/w= 0.8 is ideal in terms of sensitivity. For two input sequences of length m and n, SW needs $O(mn)$ in both time and space. But for the case where $m>>n$, the effective search space will be at most ~ $O(n^2)$, since the length of the aligned portion in the larger sequence will be of the order of the length of the shorter sequence. RPAlign first detects the RPAs in the two sequences, which will be of length at most ~ $O(n)$. Thereafter, it can immediately prune (m-n) elements from the larger sequence, thereby providing significant time gain for alignment. Thus θ/w=0.8 in such cases is effective. For example for P8, where the length of the two sequences are quite different, the TG obtained is 61.39%, while sensitivity is still 100% (See Table III and Table IV, last row). For the remaining sequence pairs except for P6, θ/w=0.8 produces 100% sensitivity and TG range is 0.23% - 19.04%. θ/w=0.85 and 0.9 appear to be reasonable choices in RPAlign as these provide high sensitivity, while TG is also reasonably high. Except for the pair P6, θ/w=0.85 and 0.9 provide sensitivity ranges from 85.79% -100% and 62.43% - 98.75%, respectively, while TG range is from 5.21% - 79.48% and 56.61% – 91.83%, respectively. θ/w=0.95 or more is recommended when the two input sequences are homologous or closely related. Here RPAlign detects many small RPAs as a result of which full parallelism can be obtained. In the process some amount of sensitivity is lost. From Table III it is found that for P2, P3, P4 and P5, RPAlign SW with θ/w=0.95 produces sensitivity ranges from 40.33% - 51.69% which are still much higher than BLAST (0.65% - 3.61%). For the pairs P6, P7 and P8 where significant similarity is found, BLAST produces the sensitivity ranges from 20.12% - 29.44% with respect to SW whereas RPAlign SW with θ/w=0.95 produces 65.76% - 99.7%. Here it is clear that to gain speed up if we consider θ/w=0.95 even then we can gain a significant amount of sensitivity than BLAST. Moreover sometimes BLAST is unable to align two distantly related sequences as is the case for pair P1 (denoted by UA under BLAST in Table III). But for P1 also RPAlign with θ/w=0.95 produces significant sensitivity (76.97%). As can be noted from Table III and Table IV, as θ/w is increased, the sensitivity decreases but the time gain (TG) increases. To measure the sensitivity for protein sequences, eight pairs are considered out of which first four are taken from ASTRAL database (http://astral.berkeley.edu/) where sequences are less than 40% similar in nature (see Table V). For protein sequences, RPAlign is executed with θ/w= 7.5 or less to obtain 100% sensitivity. Here threshold value θ is always higher than window length w because BLOSUM 62 score is considered to detect RPAs. From R1 to R8, SW and RPAlign SW produces the same results. For R1 to R4, which are ASTRAL domain sequences and have less than 40% domain conservedness as mentioned in the ASTRAL database [Chandonia *et al.,* 2004], BLAST is unable to align the sequences whereas for the remaining pairs it's sensitivity range with respect to SW is from 5.33% – 30.48%. This reflects that BLAST produces significantly lower quality of alignment than RPAlign SW.

B. *Timing Analysis*

Parallel implementation of the proposed method makes it efficient in terms of speedup of the computation. The timing analysis is provided here based on the same set of DNA sequences considered earlier. Fig. 4 shows the speed comparison among the different algorithms. It is in fact a graphical representation of the timing results in Table IV. As expected, RPAlign BLAST requires the lowest computation time followed by that for BLAST. Again SW always requires the largest computation time, while RPAlign SW with different values of θ/w, in general, provides an improvement over the SW time. Among the latter, RPAlign SW with θ/w= 0.95 provides the largest speedup. P6 presents an interesting case where no time gain (TG) is observed for θ/w = 0.8 and 0.85 since the sequences are similar. In fact, here the alignment time is the same as that for SW. The additional

RPA detection time results in an overall negative time gain. Moreover it becomes evident from this result that the detection time of RPAs is only a very small factor of the alignment time. As can be obtained from Table V, for protein sequence pairs R1 to R4, the execution times for SW and RPAlign SW are almost the same since the sequences are very small. However for R5 to R8 a significant TG is obtained. For example for R5 to R8, SW needs 9.68, 8.62, 7.76 and 6.33 seconds, respectively, whereas RPAlign SW needs 1.81, 0.11, 0.15 and 1.91 seconds, respectively. Thus the TG obtained ranges from 69.82% – 98.72% with 100% sensitivity as compared with SW. For the same set of data TG of BLAST over SW and RPAlign SW ranges from 99.14% – 99.38% and 36.36% – 97.17% respectively but with significantly lower sensitivity values viz., 5.33% – 30.48%.

TABLE III
SENSITIVITY COMPARISON OF DNA SEQUENCES BY SW, RPALIGN SW, BLAST AND RPALIGN BLAST

Pair	Sensitivity (%) with respect to SW						
	SW	BLAST	RPAlign BLAST	RPAlign SW with θ/w=			
				0.95	0.9	0.85	0.8
P1	100	UA	UA	76.97	94.19	97.71	100
P2	100	3.61	3.61	51.69	74.34	88.42	100
P3	100	1.63	1.63	41.62	69.21	100	100
P4	100	0.65	0.65	43	62.43	85.79	100
P5	100	0.75	0.75	40.33	68.06	86.14	100
P6	100	29.44	29.44	99.7	99.7	100	100
P7	100	20.12	20.12	65.76	81.21	91.04	100
P8	100	26.69	26.69	93.22	98.75	100	100

TABLE V
SENSITIVITY COMPARISON OF PROTEIN SEQUENCES BY SW, BLAST AND RPALIGN SW
(I: IDENTITY, S: SIMILARITY IN SW, P: POSITIVE IN BLAST, G: GAP)

Pair	Sequence ID	Length (aa)	Program	I	S/ P	G
R1	d1idra_ a.1.1.1	127	SW	3	4	0
	g1pnb.1 a.52.1.3	107	BLAST	-	-	-
			RPAlign SW	3	4	0
R2	d1s69a_ a.1.1.1	123	SW	5	9	0
	d2tct_1 a.4.1.9	66	BLAST	-	-	-
			RPAlign SW	5	9	0
R3	d1allb_ a.1.1.3	161	SW	25	44	37
	d1wmub_ a.1.1.2	146	BLAST	-	-	-
			RPAlign SW	25	44	37
R4	d1jbob_ a.1.1.3	172	SW	13	30	18
	d3sdha_ a.1.1.2	145	BLAST	-	-	-
			RPAlign SW	13	30	18
R5	119568124	8797	SW	450	768	987
	125983774	4181	BLAST	24	30	2
			RPAlign SW	450	768	987
R6	119568112	8757	SW	92	165	193
	118085751	3728	BLAST	28	28	4
			RPAlign SW	92	165	193
R7	119568122	8779	SW	66	100	86
	125853858	3461	BLAST	12	21	0
			RPAlign SW	66	100	86
R8	47221249	6015	SW	509	900	1058
	125983774	4181	BLAST	38	53	11
			RPAlign SW	509	900	1058

Table IV shows that BLAST achieves a TG of 99.39 – 99.95% as compared to SW. RPAlign BLAST further improves upon the time of BLAST while providing the same quality of alignment. For example the TG of RPAlign BLAST over BLAST is 23.8% – 82.28% and over SW is 99.59% – 99.99%.

Fig. 5 shows the speed comparison among RPAlign SW, BLAST and RPAlign BLAST for seven pairs of magabase scale DNA sequences. For these sequence pairs SW is unable to perform the alignment because of the sequence sizes. The figure shows that the proposed method efficiently enhances the performance of BLAST by a significant margin. The time for RPAlign SW, though greater than that for BLAST, is still manageable. Interestingly, it was observed that BLAST show a tendency to come closer. This presents an interesting application of the proposed method which can be used in conjunction with SW for megabase scale sequences with high sensitivity, as characteristic of SW, but with significantly reduced time requirement.

V. DISCUSSION AND CONCLUSION

An MPI based parallel algorithm for performing pairwise local alignment through the detection of regions that are potentially alignable has been proposed. It has been observed that the proposed method can provide an alignment quality comparable
to that of the SW algorithm while requiring significantly less time. Although BLAST has reduced the running time compared with the best known SW implementation, it has significantly low sensitivity particularly for the sequences which are distantly related and thus does not reflect the actual biological evidence. The characteristic of the proposed algorithm is that if it is used with BLAST it runs much faster with the same quality of output. If it is used with SW implementation, then the required time is much smaller and sometimes comparable to that of BLAST, but sensitivity is comparable to that of SW. The RPAlign algorithm thus allows the researchers to obtain SW-like sensitivity, while requiring significantly less time. Our aim is to enhance the existing methodologies in terms of speed without losing the sensitivity provided by them by

utilizing the power of parallel processing. It can be used with any pairwise alignment algorithm like BLAT, BLASTZ etc. It can efficiently align not only the large DNA sequences but also the more complex protein sequences which are less similar in nature. The efficiency of the proposed method derives from the fact that it is able to appropriately prune those regions of the sequence pair which will not take part in the final alignment (which the SW algorithm unnecessarily tries to align). This in turn results from the detection of the regions where alignment is possible. In the case of distantly related sequence pairs, the gain is much more, since in these cases a large amount of effective pruning is possible. RPAlign SW is 8-78 times faster than SW and RPAlign BLAST is hundreds to thousands times faster than SW. For megabase scale sequences RPAlign BLAST is found to be 3-9 times faster than BLAST.

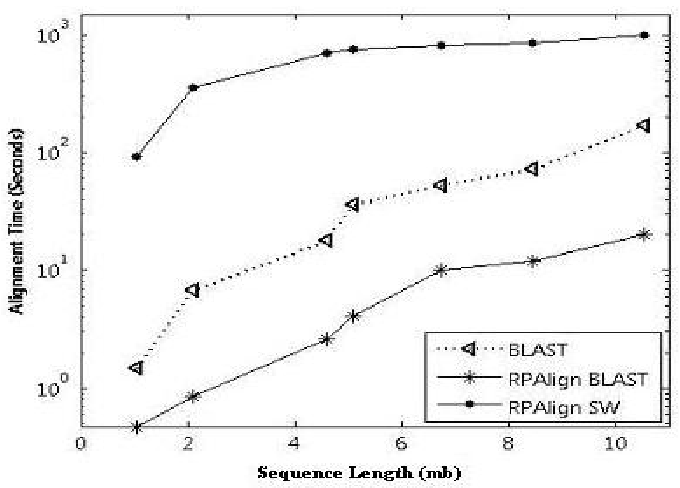

Fig. 5. Speed comparison among RPAlign SW, RPAlign BLAST, and BLAST.

TABLE IV
TIME COMPARISON OF DNA SEQUENCES BY SW, RPALIGN SW, BLAST AND RPALIGN BLAST

Time (Seconds)														
Pair	SW	RPAlign SW								BLAST		RPAlign BLAST		
		θ/w=0.95	TG (%)	θ/w=0.9	TG (%)	θ/w=0.85	TG (%)	θ/w=0.8	TG (%)		TG (%)	θ/w= 0.9	TG w.r.t. BLAST	TG w.r.t. SW
P1	0.098	0.012	87.75	0.023	76.53	0.039	60.2	0.086	12.24	UA	UA	UA	UA	UA
P2	2.73	0.06	97.8	0.55	79.83	0.56	79.48	2.21	19.04	0.025	99.45	.011	56	99.59
P3	12.85	0.26	97.97	3.128	75.65	12.18	5.21	12.82	0.23	0.078	99.39	.032	58.97	99.75
P4	82.79	1.06	98.71	11.8	85.74	39.07	52.8	82.52	0.32	0.071	99.91	.053	25.35	99.93
P5	92.78	2.54	97.26	8.21	91.83	33.87	63.49	90.09	2.89	0.12	99.87	.047	60.83	99.94
P6	96.02	95.36	0.68	95.36	0.68	96.03	-0.01	96.03	-0.01	0.084	99.91	.064	23.8	99.94
P7	112.69	6.28	94.42	48.89	56.61	87.67	22.2	96.03	14.78	0.146	99.87	.065	55.47	99.94
P8	4347.32	239.72	91.81	1656.38	61.89	1678.4	61.39	1678.4	61.39	1.349	99.95	.239	82.28	99.99

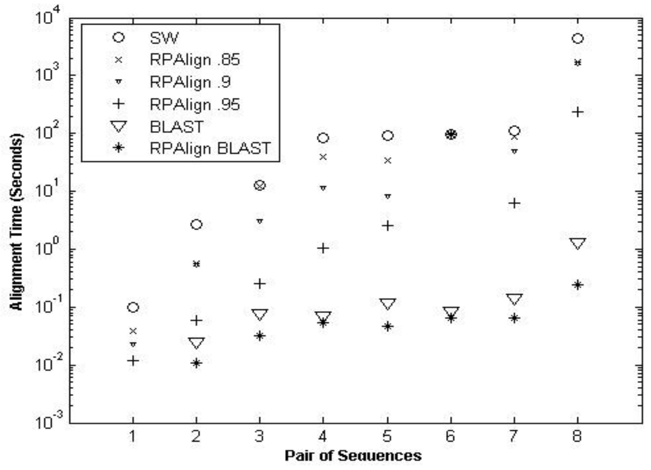

Fig. 4. Speed comparison among RPAlign SW, SW, RPAlign BLAST and BLAST.

As RPAlign produces high quality of alignment in a significantly lesser time, we are currently investigating new multiple sequence alignment techniques in a parallel framework. In the near future we will utilize the power of RPA detection for database searching using SW in parallel. Database size can be reduced by removing the sequences which have no RPAs before doing the actual search. Again use of SW will result in a stronger biological significance.

ACKNOWLEDGEMENT

The authors thank Dr. Susmita Sur-Kolay, Associate Professor, ACMU, Indian Statistical Institute, Kolkata, India for providing access to the cluster computer.

REFERENCES

[1] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. F. F. Ouellette, B. A. Rapp and D. L. Wheeler, "GenBank," ***Nucleic Acids Res***., 28, 15-18, 2000.
[2] T. F. Smith and M.S. Waterman "Identification of common molecular s-

ubsequences," *J. Mol. Biol*., 147, 195–197, 1981.

[3] O. Gotoh, " An improved algorithm for matching biological sequences," *J. Mol.Biol*., 162, 705-708, 1982.

[4] E. W. Myers, "An *O(ND)* difference algorithm and its variations," *orithmica*, 1, 251–266, 1986.

[5] F.W. Myers, W. Miller, " Optimal alignments in linear space," *Bioinformatics oxford journal*, 4**,** 11-17, 1988.

[6] S. Batzoglou, L. Pachter, J. P. Mesirov, B. Berger and E. S. Lander, " Human and mouse gene structure: comparative analysis and application to exon prediction," *Genome Res*., 10, 950-958, 2000.

[7] M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, NISC Comparative Sequencing Program, E. D. Green, A. Sidow and S. Batzoglou, "LAGAN and Multi-LAGAN: Efficient tools for large-scale Multiple Alignment of Genomic DNA," *Genome Res*. 13, 721-731, 2003.

[8] W. Pearson and D. Lipman, "Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci., USA*, 85, 2444-2488, 1988.

[9] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, " Basic local alignment search tool," *J. Mol Biol*., 215, 403-410, 1990.

[10] S. F. Altschul, T.L. Madden, A. A. Schäffer, J. Zhang, W. Miller, D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res*., 25, 3389-3402, 1997.

[11] Z. Zhang, S. Schwartz, L. Wagner and W. Miller, "A greedy algorithm for aligning DNA sequences", *J. Comput. Biol.*, 7, 203-214, 2000.

[12] T. A. Tatusova and T. L. Madden, "BLAST 2 SEQUENCES, a new tool for comparing protein and nucleotide sequences," *FEMS Microbiol Lett.,* 174, 247-250, 1999.

[13] W. Gish, "WU-BLAST" http://blast.wustl.edu/, 1995.

[14] D. J. States and P. Agarwal, "Compact encoding strategies for DNA sequence similarity search," In *ISMB*. AAAI, St Louis, MO, 211-217, 1996.

[15] A. Califano and I. Rigoutsos, "FLASH: A fast look-up algorithm for string homology," In *ISMB*, Bethesda, MD, 56–64, 1993

[16] S. Schwartz, Z. Zhang, K. A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, W. Miller, "PipMaker- a web server for aligning two genomic DNA sequences," *Genome Res*., 10, 577–586, 2000.

[17] Bin Ma, John Tromp, Ming Li " Pattern Hunter : Faster and more sensitive homology search," *Bioinformatics*, 18, 440–445, 2002

[18] W. J. Kent, "BLAT-the BLAST-like alignment tool," *Genome Res.*, 12, 656-664, 2002.

[19] Z. Ning, A. J. Cox and J. C. Mullikin, " SSAHA: A fast search method for large DNA databases," *Genome Res*., 11, 1725-1729, 2001

[20] T. Kahveci, V. Ljosa and A. K. Singh, " Speeding up whole-genome alignment by indexing frequency vectors," *Bioinformatics,* 20**,** 2122- 2134, 2004.

[21] K.R. Rasmussen, J. Stoye and E.W. Myers In *Proc. of the 9th Conf. on Computational Molecular Biology (RECOMB'05)*, pp189-203, Cambridge, MA, 2005.

[22] K. J. Kalafus, A. R. Jackson and A. Milosavljevic, " Pash: Efficient Genome-scale sequence anchoring by positional hashing," *Genome* Res, 14, 672-678, 2004.

[23] A.L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White and S. L. Salzberg, " Alignment of whole genomes," *Nucleic Acids Res.*, 27, 2369 -2376, 1999.

[24] S. Burkhardt, A. Cramer, P. Ferragina, H. Lenhof, E. Rivals, M. Vingron " Q-gram based database searching using a suffix array (QUASAR) ," *In RECOMB*. ACM Press, France, 77–83, 1999.

[25] S. Kurtz C. Schleiermacher, " REPuter: fast computation of maximal repeats in complete genomes," *Bioinformatics*, 15, 426–427, 1999.

[26] N. Bray, I. Dubchak and L. Pachter, " AVID: a global alignment program," G*enome Res.*, 13, 97–102, 2003.

[27] R. Braun, K. Pedretti, T. Casavant, T. Scheetz, C. Birkett, C. Roberts, " Parallelization of local blast service on workstation clusters," *Future Generation Computer Systems*, 17, 745-754, 2001.

[28] N. Camp, H. Cofer and R Gomperts, " High-throughput blast, *http://www.sgi.com/industries/sciences/chembio/resources/papers/ HTBlast/HTWhitepaper.html*", 1998.

[29] E. H. Chi, E. Shoop, J. Carlis, E. Retzel, J. Riedl, "Efficiency of shared-memory multiprocessors for a genetic sequence similarity search algorithm," *Technical Report, University of Minnesota, CS department*, vol. TR97 -05, 1997.

[30] R. D. Bjornson, A. H. Sherman, S. B. Weston, N. Willard, J. Wing, " Turboblast (r): A parallel implementation of blast built on turbohub," *Proceedings of the International Parallel andDistributed Processing Symposium,* 2002.

[31] D. Mathog, " Parallel blast on split databases," *Bioinformat ics*, 19, 1865 –1866, 2003.

[32] A. E. Darling, L. Carey, W. Feng, "The design, implementation, and evaluation of mpi-BLAST," In Proceedings of the Cluster World Conference and Expo, in conjunction with the 4th international Conference on Linux Clusters: *The HPC Revolution*, San Jose, CA, 2003.

[33] H. Lin, X. Ma, P. Chandramohan, A. Geist and N. Sarnatova, " Efficient data access for parallel blast," *International Parallel and Distributed Processing Symposium,* 2005..

[34] H. Rangwala, E. Lantz, R. Musselman, K. Pinnow, B. Smith, B. Wallenfelt, " Massively Parallel BLAST for the Blue Gene/ L. *HAPCW* ", 2005.

[35] T. Rognes, " ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches," *Nucleic Acid Research,* **29**, 1647- 1652, 2001.

[36] E.C. Osório, J.E. de Souza, A.C. Zaiats, P.S.L. de Oliveira and S. J. De Souza, " pp-Blast: a pseudo-parallel " Blast, *Braz. J Med Biol Res*., 36, 463-64, 2003.

[37] C. Oehmen and J. Nieplocha, "ScalaBLAST: A scalable implementation of BLAST for high-performance data-intensive Bioinformatics anal*ysis," IEEE Transactions on parallel and distributed systems*, 17, 740-747, 2006.

[38] T. Rognes, and E. Seeberg, "Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics*, 16, 699-706, 2000.

[39] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, 23, 156-161, 2007.

[40] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc Natl Acad. Sci. USA*, 89, 10915-10919, 1992.