Staircase Method: A Novel Method for Parallelizing S-W Algorithm

Muralidhara B L
Department of Computer Science
Bangalore University
Bangalore - 560056

ABSTRACT
Sequence comparison is a basic operation in DNA sequencing projects, and most of sequence comparison methods are based on heuristics, which are fast but not sensitive. The Dynamic Programming Algorithm, Smith-Waterman, obtains the best alignment, but at the expense of computational time. Unfortunately, the inefficiency in the performance of the Smith-Waterman algorithm limits its applications in the real world. A possible way out of this is to use parallelization methods for decreasing the time taken to execute the algorithm. In this paper, we present a two master method and a novel parallel technique called staircase method to improve the performance of the Smith-Waterman algorithm.

General Terms
Bioinformatics, Parallel Processing.

Keywords
Sequence alignment, Efficiency, Load balance, Speedup, staircase method

1. INTRODUCTION
Sequence alignment is the procedure of comparing two or more DNA or protein sequences by searching for a series of individual characters or character patterns that are in the same order in the sequence. If the sequence comparison process involves more than two sequences the process is called multiple sequence alignment. Otherwise, it is called pairwise alignment. For the purpose of this paper, pairwise alignment involving only two sequences is considered. Two sequences are aligned by writing them across a page in two rows, identical or similar characters are placed in the same column, and non-identical characters are placed either in the same column as a mismatch or opposite a gap in the other sequence. In an optimal alignment, non-identical characters and gaps are placed to bring as many identical or similar characters as possible into vertical registers. Sequence alignment of biological sequences is useful for discovering functional, structural, and evolutionary relationship information in DNA or protein sequences.

The important types of sequence comparison problems are global and local. To solve the global alignment problem, one has to find the best match between entire sequences. In local alignment algorithms, one must find the best match (matches) between parts of sequences. In local alignment, stretches of sequences with the highest density of matches are aligned, thus generating one or more islands of matches or subalignments in the aligned sequences. There is also a third kind of sequence alignment where the alignment is not on arbitrary substrings, but prefixes and suffixes of the given sequences [21]. Figure 1 compares the Global alignment and Local alignment process.

```
A C G G C A T C A G C T G G A
|     | | |     | | |     |
A G C G C A A T A G C C T G T

- - - G C A - - A G C - - - -
      | | |     | | |
- - - G C A - - A G C - - - -
```

Figure 1: Difference between Local Alignment and Global Alignment (the top diagram is the global alignment and the bottom is the local alignment).

Biological sequences are known to mutate as they evolve from one generation to the next, and a useful algorithm that gives a measure of similarity between two sequences should consider changes like substitution, deletion, and insertion of residues. Given a pair of sequences, there are many possible alignments, and each alignment of residues can be assigned a score using a matrix which rewards exact letter matches and also penalizes substitution (i.e., where a letter in one sequence is mapped to a different letter in the other sequence) and gaps (i.e., where aligned letters are the same, but they occur in different relative positions due to insertions or deletions in the sequences).

Given an alignment between two sequences X and Y with length m and n respectively, a score is associated for them as follows: For each column, we associate +1 if the two letters are identical, -1 if the letters are different, and -2 if one of them is a space. The score is the sum of the values computed for each column. The maximum score is the similarity between the two sequences. There can be many alignments with maximum score. Figure 2 shows one such alignment of sequence X and Y, with scores for each column. In this case, there are two columns with identical characters; one with distinct character and two columns with a space, giving a total score of -3.

X	G	A	C	G	-	
Y	-	A	T	G	C	
Score	-2	+1	-1	+1	-2	= -3

Figure 2: Alignment of the sequences X= GACG and Y= ATGC.

Sequences that are very much alike, or "similar" in the parlance of sequence analysis, probably have the same function. If two sequences from different organisms are similar, there may have been a common ancestor sequence, and the sequences are then defined as being homologous. Various algorithms have been developed for Sequence alignment. They include Smith-Waterman [22], Needleman-

Wunch [15], BLAST [1],[2], FASTA [18], MUMmer [5], [6], REPuter [11], BLAT [10], PatternHunter [13], Bayer block aligner [24], PipMaker [20], SIM [8], SIM4 [7], and GeneSeqer [23].

The Smith-Waterman dynamic programming algorithm which can find the match between two remotely related sequences consists of two parts: calculation of the total score indicating the similarity between the two given sequences, and the identification of the optimal alignment(s) with traces left by the highest scores along the matrix. Given two sequences, $X = x_1\, x_2 .. x_i$ and $Y = y_1\, y_2 .. y_j$ for $i = 1..n$ and $j = 1..m$, the Similarity Matrix $SM[m, n]$ is built by applying the following recurrence relation:

$$SM[i, j] = \max \begin{cases} SM[i, j-1] + gp \\ SM[i-1, j-1] + ss \\ SM[i-1, j] + gp \end{cases} \quad (1)$$

The task of parallel pairwise sequence alignment of biological sequences can be divided into the following stages: (1) Broadcasting two sequences by the master to participating slaves, (2) Computation of the Partial Similarity Matrix by each processor at the local node using the sequential Smith-Waterman algorithm, (3) collection of local alignment and scores by the master from participating slaves, and (4) final Alignment of sequences by the master. In the four stages, there is little scope for stages (1), (2) and (4) for parallelization. The Partial Similarity Matrix can be computed in parallel. Stage (2), computation of Partial Similarity Matrix, is a potential stage for parallelization. Each processor may be assigned a set of rows and columns, called as band, which is a Partial Similarity Matrix. This Partial Similarity Matrix is computed by the process at the local node using the Sequential Smith-Waterman Algorithm, sequentially. Processors can compute a Partial Similarity Matrix in parallel. But the order of computation of a Partial Similarity Matrix is strict and this order of computation has to be maintained by all processors. The order of computation is strict because for computing each Partial Similarity Matrix, the north-vector (north values of a Partial Similarity Matrix), west-vector (west values of a Partial Similarity Matrix) and the north-west values of a Partial Similarity Matrix are required.

2. TWO MASTER METHOD AND STAIRCASE METHOD

Several attempts have been made to increase the performance of Smith-Waterman algorithms performances [3], [4], [9], [12], [14], [16], [17], [19]. In the existing parallel methods, a single master is used to broadcast sequences to participating slaves and collect the local alignments from slaves: Master broadcasts to slaves, sequences to be aligned and West-Vectors for computing a Partial Similarity Matrix (SM). The master collects from slaves, local alignments and the best alignment score. The many number of tasks computed by the master "stall" the slave processors. The slave processors have to wait for the west-vectors to be received by them before computing the Partial Similarity Matrix. To overcome these "stalls", the tasks performed by the master can be split into two, (1) the broadcasting job and (2) the results collection job. These two tasks can then be independently handled by two different masters; we can call these masters, the collection master and the broadcasting master.

To illustrate the strategy, let the size of the system, number of processors participating in the system, be 17 (one master and 16 slaves). Let the length of the first sequence n, length of the second sequence m be 16K each, and number of columns in the band be 64. Each processor is assigned $\frac{16K}{64*16}$ i.e., 16 bands to compute. The first band (first Partial Similarity Matrix) is assigned to the first processor, second band to the second processor and so on, the 17th band to the first processor, 18th band to the second processor and so on. The order of communication is strict: the first processor to compute the 17th band (first slave processors' second band for computation) has to receive the west-vector of the sixteenth band (computed by the 16th processor, which sends the west-vector to master). At this point, the master is also involved in collecting local results from all sixteen slave processors. After Master-I collect local alignments from slaves, it uses the weighted tree method to find the final alignment. The master has to collect local alignments from all processors before it sends the west-vector and the north-west value to the first slave. The master has to execute sixteen local alignment "receive" functions, "receive" the west-vector from the last slave processors (16th processor), and then execute the "send west-vector" to the first slave function. The first processor (slave) at this point is "waiting" to execute the function, and "receive" the west-vector from the master. Until this "receive" is complete, the first processor cannot proceed with the computation of the next Partial Similarity Matrix. As long as the first slave completes computation of a Partial Similarity Matrix, the other processors cannot continue with computation. This delay caused by the master to send the west-vector to the first slave, stalls computation. The delay causes load imbalance, and decreases performance of overall computation. To overcome this problem, the two-master strategy is suggested. In the proposed system, one master is used for partial result collection and the other master for broadcasting sequences and other vectors required for computing a Partial Similarity Matrix.

In the proposed strategy, the task of initiating and collecting is separated. The job of initiating and sending the west vectors to the first processor is handled by Master-I and the local alignment collection from all the processors is handled by Master-II, who we then call the Collection Master. In this strategy, Processor 0 is Master-I which broadcasts sequences to all processors except to itself and processor $size - 1$. Master-I also collects the boundary values from the processor $size - 2$. Processor $size - 1$ is Master-II, which is the collection master. Both the rows and columns are divided into equal size, which we call, band_size. We assume that $m > size$, $n > size$, and $size > 2$, which means there are at least 3 processors participating in the computation, the first one being Master-I, the second one being Master-II and the third one, the slave node. The algorithm has five modules: the modules for Master-I, Master-II, processor 1, processor $size - 2$, and the one for all other processors. Master-I, copies the two sequences, which are in FASTA format and stores those in the data structures, X and Y which are of length n and m respectively. Master-I computes the gap penalties, 0th row and 0th column of SM and stores it in the data structures, X_gap and X_gap respectively. All these data structures, X ,Y, X_gap, X_gap and are stored in the local memory of the Broadcast Master. These data structures are "broadcast" to all slave processors. band_size, the number of columns and rows to be considered in the band, is accepted from the user. The total number of column-bands available for computation then will be $n/band\_size$ and number of row-bands in each column will be $m/band\_size$.

Each processor, except the two masters, is assigned an entire column band for computation at a time. Once the processor

completes the computation of a column band, column + (size − 1) column band is assigned to the processor and the assignment continues till there are no column bands available for computation. Within each column band, a processor computes one row-band at a time. After the slave completes a Partial Similarity Matrix, it "sends" west-vectors, and the north-west value to the next process before proceeding to the next Partial Similarity Matrix in the column. These vectors are received by the next processor through the function "receive" from the previous processor. The process of computing the row-band is continued till all row-bands of the column band are computed by the processor. Each row-band computed (partial SM) is of the size, band_size × band_size. To compute this part_SM of the size, band_size × band_size , the processor needs the north vector of size, band_size, the west vector of size, band_size, and a north-west value. For a total of $2 \times \text{band\_size} + 1$ values received by the slave, band_size × band_size computations are done. A processor after completing a band $B_{l,m}$, for some row band l, and column band m, aligns the partial sequences using S-W trace-back alignment algorithm, stores the partial alignment score in the data structure align_summary, and partial alignment in the data structure align_matrix in its local memory along with its similarity score. The processor after aligning sequences of band $B_{l+1,m}$, compares the alignment score with the previous alignment. The processor keeps only the best alignments and discards the other alignments. After completing the column band, the processor sends align_matrix and align_summary to the Collection Master.

2.1 Staircase Method With Two Masters

In this section, we discuss a novel method to improve the load balance of the S-W parallel algorithm. We call this method, the "staircase" method. The method is called, the Staircase Method since the computation in this method goes like the steps of a staircase. Though the previous strategy, Two Master Method, improves the performance of the S-W parallel algorithm, it fails to balance the load in the system. The last slave processor P_{size-2}'s first computation is delayed till this processor receives the west-vector from processor P_{size-3}, which can happen only when the processor P_1completes $size - 2$ row-bands, processor P_2 completes $size - 3$ row-bands, until processor P_{size-3} finishes its first row-band. The delayed start for processor P_{size-2} is roughly $O(size^2)$. As size and band_size increase, the delayed start also increases and leads to load imbalance. We achieve better load balance in Staircase Strategy by decreasing the delayed start.

It is noticed that when band_size is small at the initial stages (at least till the computation reaches the first principal diagonal), the delayed start overhead can be reduced. We have achieved this by computing the Similarity Matrix in two stages, Stage I and Stage II. In Stage I, the computation of the Similarity Matrix is handled by including all slave processors in the computation till the computation reaches the first positive slope $size - 2$. The size of the row-band and the column-band is reduced to half. The size of the row-band and column-band will be $band\_size/2$ for initial computations. We call this band_size as half_band_size . In Stage II, the computation of the Similarity Matrix is continued with the same size, just as we have done it in the first strategy but with a few modifications. In Stage I, processors compute half-Similarity-Matrix h_SM. The size of the h_SM is h_band_size × h_band_size. That is half the size of the Partial SM we have considered in the first strategy, which is of the size, band_size × band_size. All the processors are involved in the computation of h_SM. Therefore the delayed start problem of the previous methods have been efficiently handled. Within the first stage, since the band_size is reduced, by half of Stage II, the delayed start for the last processor is further reduced. In Stage I, all processors will take one column band each except P_{size-2} which takes two column-bands. But the number of row-bands computed by each processor is different. P_1 computes $size - 2$ row-bands, P_{size-2}computes two row-bands, and other processor computes $size - rank$ number of row-bands. The computation of band, $b_{i,j}$ for some i and j, receives west and north-west vectors from the previous processors. The north-values are anyway available for them in their own local memory. When processors complete the computation of all row-bands, they send their last row of the last band as the north-vectors for the next stage. The north vectors are sent to processor $rank/2 + rank \% 2$. The even numbered processors will also have to send the west-vectors as well as the north-west values to Stage II. The last column of the last band $b_{i,j}$ for some i and j is sent to $2 + (rank/2) - 1$ as the first west-vector and the last column of the previous band $b_{i-1,j}$ is sent to the processor $2 + (rank/2) - 1$ as the next west-vector. The first element of the last column's first value of previous band $b_{i-1,j}$ and the first element of the last column $b_{i,j}$ values are sent to the processor $2 + (rank/2) - 1$. Stage I need not store the border Ghost values, as the border values are directly sent to the processors participating in Stage II. Processors in Stage II will receive north vector values, west vector, and north-west values from the previous stage and starts computing bands the usual way. The band_size for Stage II will be double that of Stage I.

3. RESULTS AND DISCUSSIONS

The Alignment Score and the Alignment was the same as compared to the Sequential SW Algorithm. It was also proved that the algorithm suggested had followed the "Strict Computational Order" for computing the Similarity Matrix and had computed the Similarity Matrix in Parallel. The Two Master Method and the Staircase Method were executed starting with the Sequence size, m and n as 256, Band_Size as 4 and Processor Size as 4. Computation time T_c, Communication Time T_m, Initialization and Finalization time T_f for each processor were noted for the analysis. A Graphical Snapshot of the Communication Matrix, and a Graphical Profile data was also recorded for analysis. The experiments were repeated for various combinations of sequence length, Band_Size and Processor Size. The Sequence length of 256, 512 and 1024, Band_Size of 4, 8 and 16 and Processor Size of 4, 8 and 16 were considered for experimentations. Minimum sequence size of 256 was chosen because any sequence size smaller than 256 for Band_Size of 16 and Processor Size of 16 would result in completion of execution in one pass. And processors would get ≤ 1 column band for computation. One pass computation would not help in analyzing the performance metrics: Speedup and Efficiency. Minimum of Band_Size of 4 was chosen keeping in mind the execution of the Staircase Method. In the Staircase Method, we considered Half_Band_Size which is half the size of the Band_Size. A Band_Size less than 4 would lead to sequential computation. A minimum Processing size of 4 was chosen because two processors, the Send Master, P_s, and the Collection Master, P_r would not be participating in the Computing Similarity Matrix SM. And less than 4 processors would lead to sequential computation. The selected range of processors size, Band_Size and Sequence length would give sufficient and different combination of result set for analysis.

Let m be a Sequence Size, bs be the Band_Size and N be the machine size. Input Set I described by 3-tuple (m, bs, N).

$$I = (m, bs, N) \quad (2)$$

The total number of column band to be computed $B_c$is defined as:

$$B_c = m/Band\_Size \quad (3)$$

Where m is the size of the Sequence, the number of participating nodes, $N_p$which participate in computing SM is total nodes of the system minus 2, because two nodes, Send Master and Collection Master do not participate in computing SM. N_p is defined as follows:

$$N_p = N - 2. \quad (4)$$

The average number of column bands assigned for each participating node, average column band per participating node, $B_c^{Np}$are defined as follows:

$$B_c^{Np} = Bc/Np \quad (5)$$

N is the total number of Processors involved in the computation. T_c is the Computation Time, T_m is the Communication Time, T_f is the Initialization and Finalization Time and $T_t$is the Total Time Taken. T_t is defined as:

$$T_t = T_c + T_m + T_f. \quad (6)$$

Experiments were conducted on Two Master Method for 27 different combinations of Processors Size, Band_Size and Sequence Size. For all Methods in the experiments, the time spent for MPI_Comm_Rank() and MPI_Comm_Size() functions were ignored. These two functions consume significantly less time. The following equations are used for computing T_c, T_m and T_f.

T_c = Average time spent in executing user defined functions, main() and maximum() (7)

T_m= Average time spent in executing MPI_Bcast(), MPI_Ssent() and MPI_Recv() (8)

T_f= Average time spent in executing MPI_Init() and MPI_Finalize() (9)

For $I = (256,4,4)$, T_t=6.42. When N is doubled, theoretical expectations were that Total Computation Time, T_t would be reduced by half and T_t would be 3.21. For $I = (256,8,8)$, T_t=3.45. T_t was reduced almost close to half. The Computation time T_c and communication time, T_m would also be reduced by half when the processor size was doubled. But there was marginal increase in the Initialization and Finalization Time, T_f. The increase in T_f is understandable as N increases. The performance was achieved as the Band_Size was also doubled from 4 to 8. Had the Band_Size been kept at 4 and the Processor size doubled, results would not have been good. Reasons for these phenomena are the following: Increase in the processors size with the same band_size, and sequence size would delay the starting of the later half of the processors and this would result in load imbalance. The increase in T_t was more when the sequence size was small. The significant reduction in T_t was noted for m=512 and m=1024 as well. A notable decrease in T_t could be found in the following cases: For $I = (1024,4,4)$, T_t=94.01. And for $I = (1024,8,8)$, T_t=36.94. The performance of Two Master Method for various I is shown in Figure 3.

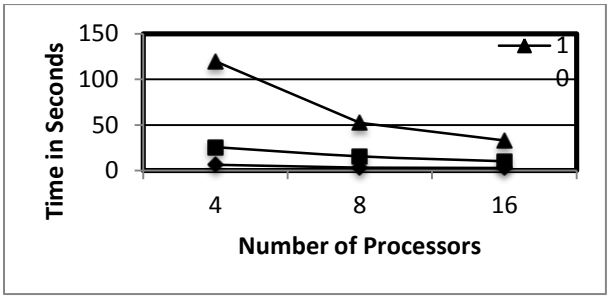

Figure 3: Graph showing the performance of Two Master Method for various processor size and Sequence length.

For all the three cases of sequence size 256, 512 and 1024 the Total Time taken T_t was almost reduced by half as N was doubled from 4 to 8, but the same kind of reduction in T_t was not observed when N was doubled from 8 to 16.

Second set of observed results was noted. The difference between second set of observed values with the first set is that in the later set T_t, T_c, $T_m$and $T_f$were recorded when Band_Size was the same as the Number of Processors but in the former set Band_Size was 2 times N. The Total Computation time, T_t, did not show a significant reduction when the Band_Size and N were doubled for sequence size 256 and 512, but for $I = (1024,8,4)$, T_t=22.01, when N and Band_Size were doubled, T_t,=10.10, a reduction of more than half.

The experiments were conducted on the following environment: Mobile Intel(R) Pentium(R) 4 CPU 3.20GHz, 700MB Memory, 1MB Cache On Linux Operating System. Because experiments were conducted on a single machine, the communication overhead $T_c$and Initialization and Finalization time T_f could be ignored for computing Total Computing Time, T_t. Only T_c is considered for computing the total time T_t.

It was observed that for $I = (256,4,4)$, $I = (256,8,4)$, $I = (512,8,8)$, $I = (512,8,4)$, $I = (1024,4,4)$, and $I = (1024,8,4)$, the average column band per participating node, B_c^{Np} is a whole number. When B_c^{Np} is a whole number, all the processors are assigned equal number of column bands to be computed and the system achieves a perfect load balance. For other values of I, the average column band per participating node, B_c^{Np} is a fractional number. The percentage of processors participating in computing in the last pass of Computing SM is the fractional part. When $I = (256,8,8)$, $N_p = 6$, and $B_c^{Np} = 5.33$, which shows that for the 5 passes, all the participating nodes are participating in computing SM, but in the 6th pass, which is the last pass, only 33% of participating nodes are assigned column bands for computation. Only 2 participating nodes are assigned column bands in the last pass resulting in load imbalance in the last pass. It could be assumed that if there was load balance in the system, the system would perform better than otherwise.

Let us consider Input Sets, I where perfect load balance was achieved. When $I = (256,4,4)$, N-node Speedup, $P_n = 1.93$, and N-node Efficiency, $E_n = 0.48$. But when bs was doubled keeping m and N as the same as in the previous case, i.e., when $I = (256,8,4)$, the N-node Speedup, $P_n = 4.36$, and N-node Efficiency, $E_n = 1.09$. A Superliner Speedup was achieved. The N-node Efficiency was also increased. A speedup of more than 4 for a 4 node Machine size and efficiency of more than 1 was an encouraging result. It was

observed that for any m and N, increasing the bs resulted in a better performance. The better performance was not by chance. It was also observed for the other I as well. When When $I = (512,4,4)$, $P_n = 1.16$, and $E_n = 0.27$ and when $I = (512,8,4)$, $P_n = 2.43$, and $E_n = 0.61$. Better results were achieved by doubling bs. The phenomenon was observed for m=1024 as well, i.e., for $I = (102,4,4)$, $P_n = 0.41$, and $E_n = 0.10$ and for $I = (1024,8,4)$, $P_n = 1.72$, and $E_n = 0.43$. It was observed that as bs was doubled for some N and m, there was a perfect load balance in the system and better $P_n$and $E_n$were achieved. It could be concluded that load balance alone would not contribute to the achievement of better performance, Band_Size, bs plays an important role. Reasons for better performance as bs was doubled by keeping m and N was that for greater values of bs, the average column band assigned for a processor, B_c^{Np} was low i.e less number of column bands were assigned to processors. By assigning less number of column bands, elements of the column band to be accessed by the process was available in the cache itself and the page fault in each process was decreased because of locality of reference. This decrease in the access time of an element resulted in increase of the performance of the computation. The other reason could be that when the processor switches computation from one column band to the other column band, the operating system had to load the column band from the memory to the cache, which resulted in delayed computation. If B_c^{Np} was more, there could be an increase in time for loading of columns by the Operation System from memory to cache. The superliner speedup and the good efficiency achieved when $I = (256,4,4)$ could be because the sequence size was small, B_c^{Np} was low, there was perfect load balance and perhaps all the data structure needed was available in the cache itself. But as N was increased for some m and bs, the performance was decreasing. Reasons for the low performance could be attributed to the memory size and cache size, locality of reference and the amount of message passed from one processor to the other. When the size of the sequence increases, memory requirement for the data structures:SM[m][n],North_Vector[bs], West_Vector[bs],Gost_Vector[n], artial_Align_Matrix[2][$2 * bs$],Master_Align_Matrix[2][$2 * m$],and alignment_Score[Bc][5] increases. The experiments were conducted on a single processor, and allocation of memory for these data structures, and loading the required data structures into cache could have caused delay. This high paging activity could have resulted in increase of page fault rate thereby increasing the accessing time resulting in reduced performance.

In summary, the Two Master Method was proposed to separate Sending jobs and collection jobs to the Smith-Waterman Wavefront method. Separation was proposed as Wavefront Method suffered from load imbalance. In the SW Wavefront method, the master was involved in both sending (or broadcasting) and in the collection of results from the participating processors, N_p. In Two Master method, a separate Send Master, P_s and a separate Receiving Master, P_r were assigned the job of Sending and Receiving. Comparable results were achieved though the Two Master Method. Two observations were noticed which could be generalized. The first generalization was that for some m, when $bs = N$ and the Total Time, T_t, a reduction of half of T_t was achieved for $I = (m, 2 * bs, N)$. The second generalization was that for some m, when $bs = N$ and the Total Time, T_t, a reduction of half of T_t was achieved for $I = (m, 2 * bs, 2 * N)$. When,$I = (256,8,4)$ the N-node Speedup, $P_n = 4.36$, and N-node Efficiency, $E_n = 1.09$. A Superliner Speedup was achieved. It was also observed that load balance alone would not contribute to the achievement of better performance; Band_Size, bs plays an important role. Low performance of the Two Master Method could be attributed to larger size of sequences causing page fault, large data structure to be loaded and accessed, and the large amount of message passed from one processor to the other.

Experiments were conducted on Staircase Method for 27 different combinations of Processors Size, Band_Size and Sequence Size. For analysis, the time spent for MPI_Comm_Rank() and MPI_Comm_Size() functions were ignored. These two functions consume significantly less time. The equations (7), (8) and (9) were used for computingT_c, T_m and T_f.

For $I = (256,4,4)$, T_t=6.62. When N was doubled, theoretical expectations were that Total Computation Time, T_t would be reduced by half and T_t would be 3.31. For $I = (256,8,8)$, T_t=3.58. T_t was reduced to almost close to half. The Computation time T_c and communication time, T_m would also reduced by half as the processor size was doubled. But there was marginal increase in the Initialization and Finalization Time, T_f. The increase in T_f is understandable and expected as N increases. The performance was achieved as the Band_Size was also doubled from 4 to 8. Had the Band_Size been kept at 4 and the Processor size doubled, results would not have been good. Reasons for these phenomena are the same as discussed in the Two Master Method.

A significant reduction in T_t was noted for m=512 and m=1024 as well. A notable decrease in T_t can be found in the following cases: For $I = (1024,4,4)$, T_t=80.83. And for $I = (1024,8,8)$, T_t=36.00. The performance of Staircase Method for the various I is shown in Figure 4.

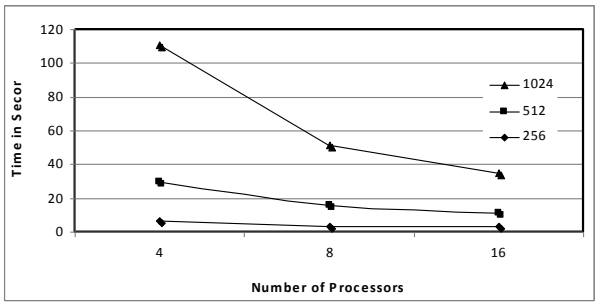

Figure 4: Graph Showing the performance of Staircase Method for various size of processors and Sequence length.

For all the three cases of sequence size 256, 512 and 1024 the Total Time taken T_t was almost reduced by half as N was doubled from 4 to 8, but the same kind of reduction in T_t was not observed when N was doubled from 8 to 16.

Second set of observed results were noted. The difference between second set of observed values with the first set is that in the later set T_t, T_c, $T_m$and T_f were recorded when $bs = N$ but in the former set, $bs = 2 * N$. The Total Computation time, T_t, does show a reduction by $2/3$ when Band_Size and N were doubled for sequence size 256, 512, 1024. For the Two Master Method, when $I = (1024,8,4)$, T_t=22.01, and when N and Band_Size were doubled, T_t,=10.10. A reduction of more than half in T_t was noticed, but for the same set of Input, the reduction in T_t was $2/3$. Reasons for this

performance degradation could be due to the fact that as the Band Size was increased for the Staircase Method, the communication time was increasing.

It was observed that for $I = (256,4,4)$, $I = (256,8,4)$, $I = (512,8,8)$, $I = (512,8,4)$, $I = (1024,4,4)$, and $I = (1024,8,4)$, the average column band per participating node, B_c^{Np} was a whole number. When B_c^{Np} was a whole number, all the processors were assigned equal number of column bands to be computed, and the system achieved a perfect load balance. For other values of I, the average column band per participating node, B_c^{Np} was a fractional number. The percentage of processors participating in computing in the last pass of computing SM was the fractional part. When $I = (512,8,8)$, $N_p = 6$, and $B_c^{Np} = 10.67$, showing that for the first 10 passes all the participating nodes were participating in computing SM, but in the 6th pass, which is the last pass, only 67% of participating nodes were assigned column bands for computation. Only 4 participating nodes were assigned column bands in the last pass resulting in load imbalance in the last pass. It could be assumed that if there was load balance in the system, the system would perform better than otherwise.

Consider Input Sets, I where perfect load balance is achieved. When $I = (256,4,4)$, N-node Speedup, $P_n = 1.87$, and N-node Efficiency, $E_n = 0.47$. But when bs was doubled keeping m and N as the same as in the previous case, i.e., when $I = (256,8,4)$, the N-node Speedup, $P_n = 4.25$, and N-node Efficiency, $E_n = 1.06$. Like in the Two Master Method, the Staircase Method also achieves A Superliner Speedup. The n-Node Efficiency was also increasing. A speedup of more than 4 for a 4 node Machine size and efficiency of more than 1 was an encouraging result. It was observed that for any m and N, increasing the bs resulted in a better performance. The better performance was not by chance. It was also observed for the other I as well. When When $I = (512,4,4)$, $P_n = 0.96$, and $E_n = 0.24$ and When $I = (512,8,4)$, $P_n = 2.32$, and $E_n = 0.58$. Better results were achieved by doubling bs. The phenomenon was observed for m=1024 as well, i.e., when $I = (102,4,4)$, $P_n = 0.48$, and $E_n = 0.12$ and for $I = (1024,8,4)$, $P_n = 1.71$, and $E_n = 0.43$. It was observed that when bs was doubled for some,N and m, there was a perfect load balance in the system and better $P_n$and E_n were achieved. It could be concluded that load balance alone would not contribute to the achievement of better performance; Band_Size, bs plays an important role. Reasons for better performance as bs is doubled by keeping m and N have already been discussed for the Two Master Method and the same reasons hold good for the Staircase Method as well. A better performance was achieved for the Staircase Method compared to the Two Master Method.

It was observed that when $I = (512,4,16)$, the Total Computation Time, T_t for the Staircase Method had been reduced marginally compared to the Two Master Method. When the bs was doubled i.e., when $I = (512,8,16)$ the gap in Total Computation Time, T_t between Two Master Method and Staircase Method was increasing. When the sequence length m was increased from 512 to 1024, i.e., when $I = (1024,4,4)$, the Total Computation Time T_t was 94.01 for Two Master Method and T_t was 80.83 for Staircase Method. But when $I = (1024,8,8)$, Total Computation Time T_t was 36.94 for Two Master Method and T_t was 36.00 for Staircase Method; the gap in T_t had reduced. It could be concluded that when for some m and N , when bs was increased there was a better performance, but the performance degrades for larger m values. Reasons for the phenomenon could be that when m is small, there was better load balance in the first pass and the computation was completed in a few passes. As m was increased, there was good load balance in the first pass, but the computational bands, B_c had increased, thereby increasing the number of passes (for smaller N) resulting in the contribution of the load balance in the first pass not contributing significantly to the overall performance of the system.

4. CONCLUSION

In summary, the Staircase Method was proposed because in the Two Master Method, for the first pass computation, the last slave processor P_{size-2}'s first computation is delayed till the processor receives the west-vector from processor P_{size-3}, which happens only when the processor P_1completes $size - 2$ row-bands, processor P_2 completes $size - 3$ row-bands, until processor P_{size-3} finishes its first row-band. The delayed start for processor P_{size-2} is roughly $O(size^2)$. As size and band_size increase, the delayed start would also increase leading to load imbalance. It was noticed that when band_size was small at the initial stages (at least till the computation reaches the first principal diagonal), the delayed start overhead could be reduced. Delayed start was reduced by computing the Similarity-Matrix in two stages, Stage I and Stage II. In Stage I, Similarity Matrix was computed by including all slave processors in the computation till the computation reaches first positive slope $size - 2$. The Algorithm, reduce both the row-band and column-band by half. The size of the row-band and column-band would be $band\_size/2$ for initial computations. This band was called, h_band_size. In Stage II, the computation of Similarity Matrix was continued with the same size, as it computed in the Two Master Method with a few modifications. As in the Two Master Method, Send Master, P_s and Receiving Master P_r were assigned the job of Sending and Receiving. Comparable results were achieved through the Staircase Method. Two observations were noticed which can be generalized. The first generalization was that for some m and when $bs = N$ and Total Time,T_t, a reduction of half of the T_t was achieved for $I = (m, 2 * bs, N)$. The second generalization was that for some m and when $bs = N$ and a Total Time,T_t, a reduction of half of the T_t was achieved for $I = (m, 2 * bs, 2 * N)$. When,$I = (256,8,4)$, the N-node Speedup, $P_n = 4.25$, and N-node Efficiency, $E_n = 1.06$. A Superliner Speedup was achieved. It was also observed that load balance alone would not contribute to the achievement of better performance; Band_Size, bs plays an important role. The Staircase Method performance was better when for some m and N and the bs were increased. But the performance of the Staircase Method compared to the Two Master Method degrades for a larger size of m. Low performance of the Staircase Method in some cases could be attributed to the larger size of sequences, page fault, loading overhead for the large data structure and the large amount of message passed from one processor to the other.

5. ACKNOWLEDGMENTS

The author acknowledges Dr.Srinivas Bhogle, Director and Country Head, TEOCO Software Private Ltd, Bangalore and Dr. Pradeep G Siddheshwar, Bangalore University for their critical comments and valuable inputs.

6. REFERENCES

[1] Altschul F Stephen, Warren Gish, Webb Miller, Eugene W. Myers & David J. Lipman. 1990. "Basic Local

Alignment Search Tool". Journal of Molecular Biology, 215, 403-410.

[2] Altschul F Stepehn, Tomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller and David J. Lipman. 1997. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". Nucleic Acids Research, Vol.25, No.17, 3389-3402.

[3] Batista Rodolfo Bezerra, Debora Nery Silva, Alba Cristina Magalhaes Alves de Melo, and Li Weigang. 2004. "Using a DSM Application to Locally Align DNA Sequences". 2004IEEE International Symposium on Cluster Computing and the Grid, IEEE Computer Society.

[4] Boukerche Azzedine, Alba Cristina Magalhaes Alves De Melo, Maria Emilia Telles Walter, Renata Cristina Faray Melo, Marcelo Nardelli Pinto Santana, Rodolfo Bezerra Batista. 2004. "A Performance Evaluation of a Local DNA Sequence Alignment Algorithm on a Cluster of Workstations". Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04).

[5] Delcher L Arthur, Adam Phillippy, Jane Carlton and Steven L. Salzberg. 2002. "Fast algorithms for large-scale genome alignment and comparison". Nucleic Acids Research, Vol.30, No.11, 2478-2483.

[6] Delcher L Arthur, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White and Steven L Salzberg. 1999. "Alignment of whole genomes". Nucleic Acids Research, Vol.27, No.11, 2369-2376.

[7] Florea L, Hartzell G, Zhang Z, Rubin G M, and Webb Miller W. 1998. "A computer program for aligning a cDNA sequence with a genomic DNA sequence". Genome Res. Vol.8, pp.967-974.

[8] Huang X, Hardison R C, and Miller W. 1990. "A space-efficient algorithm for local similarities". CABIOS, Vol.6, 373-381.

[9] Hughey Richard. 1996. "Parallel Hardware for Sequence Comparison and Alignment". CABIOS, Vol.12, No.6, pp.473-479.

[10] Kent James. W. 2002. "BLAT – The BLAST-Like Alignment Tool". Genome Research, Vol. 12, pp.656-664.

[11] Kurtz Atefan and Chris Schleiermacher. 1999. "REPuter: fast computation on maximal repeats in complete genomes". Bioinformatics, Vol.15, no.5, 426-427.

[12] Liao Hsien-Yu, Meng-Lai Yin, and Yi Cheng. 2004. "A Parallel Implementation of the Smith-Waterman for Massive sequences Searching". Proceedings of the 26th Annual International Conference of the IEEE EMBS, pp.2817-2820.

[13] Ma Bin, Joh Tromp and Ming Li. "Pattern Hunter: faster and more sensitive homology search". 2002. Bioinformatics, Vol. 18, no. 3, pp. 440-445.

[14] Martins W.S, J.B. Del Cuvillo, F.J. Useche, K.B. Theobald, and G.R. Gao. 2001. "A Multithreaded Parallel implementation of a Dynamic Programming Algorithm for Sequence comparison". Int. Symposium on Computer Architecture and HPC (SBAC-PAD), 1-8.

[15] Needleman Saul B & Christian D. Wunsch. 1970. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins". Journal of Molecular Biology, 48, 443-453.

[16] Pekurovsky D, I. N. Shindyalov and P.E. Bourne. 2004. "A Case study of high-throughput biological data processing on parallel platforms". Bioinformatics, Vol.20, no.12, pp.1940-1947.

[17] Pellicer Stephen, Nova Ahmed, Yi Pan and Yao Zheng. 2005. "Gene Sequence Alignment on a Public Computing Platform". Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW'05).

[18] Person W R and Miller W. 1992. "Dynamic Programming algorithm for biological sequence comparison". Methods Enzymol, Vol.210, pp.575-601.

[19] Rognes Torbjorn & Erling Seeberg. 1981. "Six-fold speedup of Smith-Waterman sequence database searches using parallel processing on common microprocessors". Bioinformatics. Vol.16, no.8, 699-706.

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

[21] Setubal and Meidanis. 1997. Introduction to Computational Molecular Biology. Brooks/Cole Publishing Company.

[22] Smith T. F, & M.S. Waterman. 1981. "Identification of Common Molecular Sequences". Journal of Molecular Biology, 147, 195-197.

[23] Usuka J, Zhu W, and Brendel V. 2000. "Optimal spiced alignment of homologous cDNA to a genomic DNA template". Bioinformatics, Vol.16, pp.203-211.

[24] Zhu J, Liu J S, and Lawrence C E. 1998. "Bayesian adaptive sequence alignment algorithms". Bioinformatics, Vol.14, pp.25-39.