Parallel Two Master Method to Improve BLAST Algorithm's Performance

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

ABSTRACT
The BLAST heuristic algorithm is one of the widely used algorithms for finding similar sequences in sequence databases. The paper acquires importance as traditional approaches to sequence homology searches using BLAST have proven to be too slow to keep up with the current rate of sequence acquisition. The use of the BLAST application on a single processor has become too costly, inefficient, and time-consuming for many life science laboratories. An obvious improvement that has already been suggested is the use of database segmentation to speedup BLAST. A two master method to improve the performance of the parallel BLAST algorithm is presented here. It is found that the two master method has performed better than the single master method.

General Terms
Bioinformatics, Parallel Processing.

Keywords
Sequence alignment, BLAST, Database segment, Query segment, Efficiency, Speedup

1. INTRODUCTION
This paper aligns the letters of the sequences belonging to two classes of biological sequence: a nucleotide sequence in which each letter represents one of the four DNA bases namely A, T, G, and C, and a peptide or a protein sequence, in which each letter represents one of the twenty amino-acid residues. 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. The papers considers pairwise alignment involving only two sequences. 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 one 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). The match score is usually specified by a matrix called the substitution matrix whose entries reflect the biological significance of the corresponding matches or substitutions. Gaps are penalized using a formula that depends on the number, position, and length of each gap. 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 1 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 1: Alignment of the sequences X= GACG and Y= ATGC.

Various algorithms have been developed for Sequence alignment. They include Smith-Waterman [19], Needleman-Wunch [17], BLAST [1], [2], FASTA [18], MUMmer [7], [8], REPuter [14], BLAT [13], PatternHunter [15].

1.1 The BLAST Algorithm
The BLAST search heuristic [1], [2], [3] indexes both the query and target sequence into words of chosen size (11 nucleotides or 3 residues by default). It then searches for matching word pairs (hits) with a score of at least T and extends the match along the diagonal. Gapped BLAST consists of several modifications to the previous algorithm that results in both increased sensitivity and decreased execution time. Gapped BLAST moves down the sequences until it has found two hits, each with a score of at least T, within A letter of each other. An ungapped extension is performed on the second hit, generating a high-scoring segment pair (HSP). If the HSP score exceeds a second cutoff, a gapped extension is triggered simultaneously forward and backward. Standard BLAST output consists of a set of local gapped alignments found within each query sequence, the alignment's score, an alignment of the query and database sequence, and a measure of the likelihood that the alignment is a random match between the query and database (e-value). Factors such as query length, number of queries, total database size, length of database entries, and sequence similarity between the query and database entries affect the amount of time consumed by the BLAST algorithm [6].

BLAST searches a query sequence containing nucleotides (DNA) or peptide (amino acids) against a database of known nucleotide or peptide sequences. Since peptide sequences result from ribosomal translations of nucleotides, comparisons can be made between nucleotide and peptide sequences. BLAST provides the capability to compare all possible

combinations of query and database sequence types by translating sequences on the fly, as shown in Table. 1 [9].

The BLAST programs essentially consist of three parts: preparatory calculations, the sequence comparison, and the combination of the individual results. The second part, where the database search is performed, is the most time consuming piece of the program. The comparison between the query sequence and a database sequence is completely independent of the comparison of the query sequence to other database sequences; therefore, this part of the BLAST programs is very well suited for parallelization. The first and the third part of the programs do not benefit from parallelization. The time spent for the database search is directly proportional to the size of the database; therefore, the comparison part becomes an ever increasing portion of the overall program. Also the BLAST programs are highly scalar codes with very few floating point instructions [12]. The BLAST runs fastest when its databases remain cached in memory, and further acceleration is achieved by threading, which allows more than one CPU at a time to process the data [10].

Table 1: BLAST search types

Search Name	Query Type	Database Type	Translation
blastn	Nucleotide	Nucleotide	None
blastp	Peptide	Peptide	None
blastx	Nucleotide	Peptide	Query
tblastn	Peptide	Nucleotide	Database
tblastx	Nucleotide	Nucleotide	Query and Database

Query segmentation provides the most natural parallelization of BLAST by splitting up a query such that each compute node in a cluster searches a fraction of the sequence database. Thus, multiple BLAST searches can execute in parallel on different queries. However, such an approach typically requires that the entire database be replicated on each compute nodes local storage system [5]. If the database to be searched is larger than core memory, then query-segmented searches suffer from the same adverse effects of disk I/O as in traditional BLAST. When the database fits in core memory, however, query segmentation can achieve nearly linear scaling for all BLAST types [9]. Local BLAST program is one of the programs which use query segmentation approach [5].

Database segmentation is an orthogonal approach to query segmentation. Database segmentation keeps the query intact and distributes individual database segments to each node for the query to be searched upon. Database segmentation performance is better than query segmentation as database segmentation eliminates the high overhead of disk I/O. Database segmentation permits each node to search a smaller portion of the database, thus reducing extraneous disk I/O [6]. One of the biggest challenges of this approach is to ensure that the statistical scoring is properly produced as it depends on the size of the database, a database that database segmentation chops up. The programs such as Turbo BLAST [4], mpiBLAST [6], Bioinfomagic [9], BeoBLAST [10], and parallelBLAST [16] uses the database segmentation.

The NCBI BLAST performs best when the database contains a small number of sequences, but NCBI BLAST performance suffers when confronted with low memory because as the database size exceeds the system memory, running time and average blocks read increase. TurboBLAST, parallel deployment of NCBI BLAST achieves speedup by reducing the paging overhead. A speed up of 7.51 and 13.85 is achieved for 8 and 16 processors cluster. BeoBLAST distributes individual BLAST jobs across nodes of Beowulf cluster can simultaneously align multiple databases with multiple queries. Parallel BLAST can greatly improve speed at almost no cost of sensitivity by restricting alignments to sequence seed rather than an alignment seed. The parallel BLAST is 14-18 times faster than single machine but Job scheduling overhead, merge time limit its performance and moreover it is not directly integrated with the NCBI toolkit.

mpiBLAST achieves superlinear speedup for multiple nodes even when the database is larger than the core memory of a single node. mpiBLAST does not produce heavy intercommunication between nodes, allowing it to continue achieving super-linear speedup over hundreds of nodes. But mpiBLAST efficiency decreases as the number of workers increases, contradictory to the theoretical results. One reason could be its load imbalance. One possible solution would be to segment the database into a equal sized large number of small fragments, and then all workers can complete the job almost at the same time, resulting in load balancing. Across four nodes Bioinfomagic's efficiency is 2.31 and drops all the way down to 1.33 with 128 nodes. Even though the efficiency decreases as the nodes increase, Bioinfomagic still achieves super-linear speed-up. The other advantages are that (1) Bioinfomagic is an open source, and (2) it directly interfaces with the NCBI development library to provide an output format of NCBI-BLAST.

It is found that the Database segmentation performs better than query segmentation as query segmentation suffers from paging activity when the local nodes can not hold the entire database in its memory. Among the models investigated, the Bioinfomagic's performance is relatively more impressive.

2. A CASE FOR TWO MASTERS

Parallel BLAST pairwise sequence alignment algorithms can be divided into the following stages: (1) Loading Query/Database sequence into processors, depending on the segmentation technique used: Query or Database (2) Slicing the Query/Database (depending on the segmentation techniques used) into small size sequences sending it to the participating nodes for computation (3) Aligning the sequences using NCBI BLAST program by the participating processors (4) Receiving the aligned sequences from the participating nodes (5) Merging the received alignments into a final alignment. Step (3) and Step (5) have little scope for parallelization. But Step (1), (2) and (4) can be parallelized for improving the performance.

If m is the length of the database sequence and n is the length of the query sequence, and if $m \gg n$ and the word size for computation is W, the query segmentation suffers from page faults. It can be observed in the query segmentation technique that the database sequence of size m has to be loaded into the memory of all the processors and a search tree data structure has to be created for $(m - W)$ elements each with size W. The space complexity is $O(m)$ and the time taken to create the search tree is of the complexity $O(m \log_2 m)$. The computational time to compare the sliced query sequence with

the large database sequence also increases. It is therefore recommended that for better use of space and time, Database Segmentation Technique is a better technique than Query Segmentation Technique. In Database Segmentation Technique, Query Sequence is loaded to all the processors, a search tree is created for the query sequence and the database is sliced into a size of sz which is the Slice_Size. The comparison is between this small sz with a reasonably sized Query Sequence. The page faults can be reduced using Database Segmentation since very small size sequences are loaded into memory and these perhaps can be loaded into the cache as well. Database Segmentation Technique is used for the experiments. The programmmes were written in C High Level Language using MPICH [11]. The programs were executed on Mobile Intel(R) Pentium(R) 4 CPU 3.20GHz, 700MB Memory, 1MB Cache On Linux Operating System.

Let m be the length of the length of the database sequence, n be the length of the query sequence, sz be the Slice_Size and Machine Size N. The Input for the BLAST algorithm can be defined by 4-tuple (m, n, sz, N).

$$I = (m, n, sz, N) \quad (1)$$

The total number of database Slices, S_c of the database sequence can be defined as:

$$S_c = m/sz \quad (2)$$

Let N_p be the number of participating nodes participating in aligning two sequences: $N_p = N - 1$ for Single Master Method and $N_p = N - 2$ for Two Master Method.

Let S_c^{Np} be the Number of database Slice per participating node, N_p. S_c^{Np} is defined by

$$S_c^{Np} = S_c/N_p \quad (3)$$

In the existing parallel method 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 One Master Method for BLAST Algorithm

In the existing Single Master Method out of the N processors in the system, one processor is the Master which is P_0. The Processor, P_0 loads the database sequence into its memory and computes the number of slices of database sequence S_c, builds 64 combinations of 3 letter DNA sequences into Index_Vector[64][3] and computes the job_remaining. P_0 sends Available signals to P_1 toP_{size-1} and receives Ready signals from P_1 toP_{size-1}. Then, depending on the availability of the job and Ready signals from the participating nodes N_p, send Selected signals to P_1 toP_{size-1}. To all the Selected processors, P_0 sends Slice_Vector[sz + 1]. Once the computations are completed by the Selected processors, the Selected processors send the Result_Vector[sz ∗ 2][sz]. The above process is continued till all the database slices are considered for computation. After receiving all Result_Vector from the slaves, the Master computes the final alignment.

Algorithm for BLAST Single Master Method

Copy the Query Sequence residues from Secondary Storage device to the Vector X[n]
Build 64 combination of 3 letter DNA sequence into Index_Vector[64][3]

If the processor rank = 0
  Copy the Database sequence into the Vector Y[m]
  Compute number_of_jobs and job_remaining
  While (job_remaining > 0)
    Send Available signal to P_1 toP_{size-1}
    Receive Ready signal from P_1 toP_{size-1}
    Depending on the availability of the job Send Selected signal to P_1 toP_{size-1}
    Send Slice_Vector[sz + 1] to the Selected processors
    Receive Result_Vector[sz ∗ 2][sz] from P_1 toP_{size-1}
    Compute number_of_jobs and job_remaining
    Send job_remaining to P_1 toP_{size-1}
  End While
  Align the partially aligned results into the final result
End of processor rank = 0

If the Processor rank > 0 *and* $rank < size - 1$
  Receive Available from P_0
  Send Ready to P_0if Ready to execute the job
  Receive Selected from P_0 if Selected for execution
  While (Selected)
    Receive Slice_Vector[sz + 1] from P_0
    Use the BLAST algorithm for sequence alignment and store the result in Result_Vector[sz ∗ 2][sz]
    Send Result_Vector[sz ∗ 2][sz]to P_0
    Receive job_remainingfrom P_0
  End While
  Finalize the processing
End of Processor rank > 0 *and* $rank < size - 1$

Figure 2: Algorithm for Single Master Method.

2.2 Two Master Method for BLAST Algorithm

In the existing Single Master Method for BLAST that Single Master is involved in sending Sliced database sequences to all participating nodes and receiving the Result_Vector[sz ∗ 2][sz]. The Single Master is also involved in sending Available signals and Selected signals to Size − 1 processors, receiving Ready signals from Size − 1 processors, and sending job_remaining to Size − 1 processors. The Single Master $P_0$is also involved in processing the final alignment. As the number of slices, S_c to be computed increases, the Single Master becomes the bottleneck in the computation. When the size of the database sequence m increases, the number of slices, S_c also increases, thus increasing the size of the Result_Vector in the memory of the Single Master. Increase in the size of the Result_Vector and the database sequence would result in page faults eventually delaying the computation process. It can be observed that if the jobs of the Single Master are shared by two masters, the processing speed would increase. In the Two Master Method, two masters are used, The Slice Master and the Collection Master. The Processor $P_0$is used as the Slice Master and the Processor, P_{size-1} is used as the Collection Master. The total number of participating nodes in the system N_p would be N − 2. The Send Master is involved in sending the sliced database sequences to all participating nodes, sending Available and Selected signals to Size − 1 processors, receiving Ready signals from Size − 1 processors, and sending job_remaining to Size − 1 processors. The Receiving Master would be involved in collecting results and processing the final alignment.

Algorithm for BLAST Two Master Method

Copy the Query Sequence residues from Secondary Storage device to the Vector X[n]
Build 64 combination of 3 letter DNA sequence into Index_Vector[64][3]
If the processor rank = 0
  Copy the Database sequence into the Vector Y[m]
  Compute number_of_jobs and job_remaining
  While (job_remaining > 0)
    Send Available signal to P_1 toP_{size-2}
    Receive Ready signal from P_1 toP_{size-2}Depending on the availability of the job Send Selected signal to the P_1 toP_{size-2}
    Send Slice_Vector[sz + 1] to the Selected processors
    Compute number_of_jobs and job_remaining
    Send job_remaining to P_1 toP_{size-2}
  End While
End Processor rank = 0
If the processor rank = size − 1
  Compute number_of_jobs and job_remaining
  While (job_remaining > 0)
    Receive Result_Vector[sz ∗ 2][sz] from P_1 toP_{size-2}
    Compute number_of_jobs and job_remaining

End While
Align the partially aligned results into the final result
End processor rank = size − 1
If the Processor rank > 0 *and* rank < *size* − 2
Receive Available from P_0
Send Ready to P_0if Ready to execute the job
Receive Selected from P_0 if Selected for execution
While (Selected)
Receive Slice_Vector[sz + 1] from P_0
Use the BLAST algorithm for sequence alignment and store the result in Result_Vector[sz ∗ 2][sz]
Send Result_Vector[sz ∗ 2][sz]to P_{size-1}
Receive job_remainingfrom P_0
End while
Finalize the processing
End of Processor rank > 0 *and rank* < *size* − 2

Figure 3: Algorithm for Two Master Method.

3. Comparison of Single Master Method and Two Master Method

Communication Matrix for Single Master Method for Input Set I = (1024,256,32,8) is shown in Figure 4. In the example, total number of database Slices,$S_c = 32$, the number of nodes participating in the computation, $N_p = 7$, Number of database Slice per participating nodes $S_c^{Np} = 4.57$. It is shown in the Communication Matrix that there is communication only between processors P_1 toP_{size-1} and the Master Node, P_0. All the participating nodes in the first four passes are getting equal number of database slices, but in the last pass of computation, only 57% of participating nodes will get the job; that is only four out of the seven processors have been assigned jobs in the last pass. There is a load imbalance in the last pass of computation. The amount of communication between various processors is shown in the Communication Matrix with different color band for size of the communication. It is shown that the last three processors' (shown as blue for sending and dark green while receiving) communication size is less than the first four processors (shown as light green while sending and dark red while receiving).

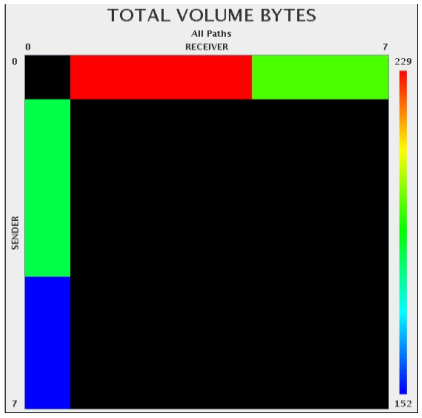

Figure 4: Communication Matrix for Single Master Method I = (1024, 256, 32, 8).

Communication Matrix for Two Master Method for Input Set I = (1024,256,32,8) is shown in Figure 5. In the example, total number of database Slices, $S_c = 32$, the number of nodes participating in the computation, $N_p = 6$, Number of database Slice per participating nodes $S_c^{Np} = 5.33$. It is shown in the Communication Matrix that there is communication between processors P_1 toP_{size-2} and the Master Node, P_0 as well as P_1 toP_{size-2} and the Collection Master P_{size-1}. All the participating nodes in the first five passes are getting equal number of database slices, but in the last pass of computation, only 33% of participating nodes will get the job; that is only two out of the six processors have been assigned job in the last pass. There is a load imbalance in the last pass of the computation. It can be noted from the diagram that there are different sizes of communication between processors. In Two Master Method, the communication between P_1 toP_{size-2} and the Master Node, P_0has been reduced. The difference in size in receiving the communication of the $P_0$and $P_{size-1}$is due to the load imbalance in the system. It is to be noted that the amount of data received by the processor P_0 is minimum, and P_0 is spending time sending the Sliced Vectors; $P_{size-1}$is not participating in sending; it is only receiving data.

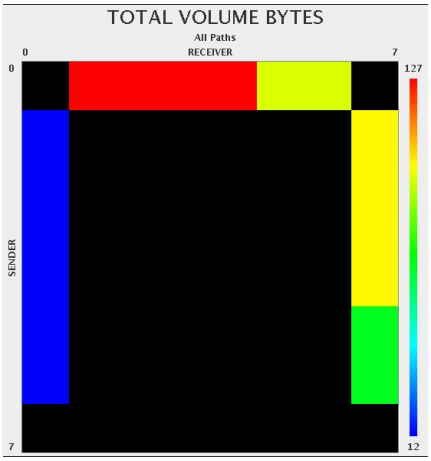

Figure 5: Communication Matrix for Two Master Method for I = (1024, 256, 32, 8).

Experiments on BLAST Single Master and on Two Master Method were conducted varying the database sequence size m from 512 to 1024. The sequence size n was fixed at 256. The System size N was varied from 4 to 16. The reason for keeping the minimum length of database sequence as 512 was that anything less than the size of 512 for large Slice Size sz and large system size N would result in participating processors being assigned very few residues for comparison. A database size of more than 1024 in a single processor would result in an increase of page faults. The size of n was fixed at 256 since it would not take much space in creating the search tree data structure and storing the string in memory. The system size of N was varied from 4 to 16 to get all possible results. Experiments were conducted on 18 different inputs for both One Master Method and Two Master Method and results which help in analysis are shown in the tables.

Equations (1), (2), and (3) are used for computing the Computational TimeT_c, Communication Time T_m and Initialization and Finalization Time T_f. The total time T_t is calculated.

Results for the Single Master Method is shown in Table 2. For Input Set, I = (512,256,8,4), the total time taken T_t = 4378 milli sec . It was observed that keeping the processor size the same, if the Slice_Size sz is increased, the computational time would decrease. For input Set, I = (512,256,16,4), T_t = 3433 milli sec. This means a

reduction of time by 22%. If sz is further increased to 32, $T_t = 2209$ milli sec, which is a reduction of time by 36%. The trend can be observed for the database sequence size $m = 1024$, and query sequence of size $n = 256$. The reduction in time was less than the theoretical assumptions because of the time spent in communication; initialization has not decreased much and the Single Master bottleneck could be the reason for this phenomenon.

Table 2: Experimental Results for Single Master Method

m	n	sz	N	T_c	T_m	T_f	T_t
512	256	8	4	1059.00	663.80	2655.20	4378.00
512	256	16	4	887.36	509.20	2036.80	3433.36
512	256	32	4	647.48	312.40	1249.60	2209.48
1024	256	8	4	2671.28	2023.64	8094.56	12789.48
1024	256	16	4	1960.00	1378.40	5513.60	8852.00
1024	256	32	4	1233.64	813.32	3253.28	5300.24

Table 3 shows total computational Slice S_c, Number of Participating Nodes, N_p and the Slice per Participating nodes S_c^{Np} for Single Master Method. It was evident that the slice per participating node is a fraction in all the cases, an indication that there was load imbalance in the final pass of the computation. For $I = (512,256,8,4)$, the $S_c^{Np} = 21.33$ and only 33% of the participating nodes are given computational tasks in the last pass. Only one node out of the three nodes is given a task. The load imbalance is observed for $I = (512,256,16,4)$ as well. $S_c^{Np} = 10.67$ for the input sent and only 67% of the nodes are given tasks in the final pass. The load imbalance could have also contributed to the total computational time. Results for the Two Master Method are shown in Table 4. For Input Set, $I = (512,256,8,4)$, the total time taken $T_t = 1847$ milli sec. It was observed that keeping the processor size the same and increasing the Slice_Size sz, the computational time would decrease. For input Set, $I = (512,256,16,4)$, $T_t = 1471$ milli sec.

Table 4: Results for Two Master Method.

m	n	sz	N	T_c	T_m	T_f	T_t
512	256	8	4	964.18	737.11	145.97	1847.26
512	256	16	4	757.24	357.81	355.96	1471.01
512	256	32	4	585.00	265.17	270.84	1121.01
1024	256	8	4	2366.52	1761.74	278.47	4406.73
1024	256	16	4	1480.79	1047.54	247.97	2776.30
1024	256	32	4	1106.73	663.82	343.31	2113.86

This means a reduction of time by 20%. If sz is further increased to 32, $T_t = 1121$ milli sec, a reduction of time by 24%. The trend can be observed for the database sequence size $m = 1024$, and query sequence of size $n = 256$. For all the Input Sets we had used for Single Master Method, there was a good performance increase in the Two Master Method. The assumption of increase in the performance if worked with two Master Method has thud been proved. Table 5 shows total computational Slice S_c, Number of Participating Nodes, N_p and the Slice per Participating nodes S_c^{Np} for Two Master Method. It was evident that the slice per participating node is a whole number in all the cases, an indication that there was load balance in the computation.

For $I = (512,256,8,4)$, $S_c^{Np} = 32$, and all participating nodes are given computational tasks in the last pass. The load balance can be observed for $I = (512,256,16,4)$ as well. For the input set, $S_c^{Np} = 16$, all nodes are given tasks in the final pass. The load balance would have helped the total computational time to decrease.

Table 3: Results for Single Master Method.

$I = (m, n, sz, N)$	T_t	S_c	N_p	S_c^{Np}
$I = (512,256,8,4)$	4378.00	64	3	21.33
$I = (512,256,16,4)$	3433.36	32	3	10.67
$I = (512,256,32,4)$	2209.48	16	3	5.33
$I = (1024,256,8,4)$	12789.48	128	3	42.67
$I = (1024,256,16,4)$	8852.00	64	3	21.33
$I = (1024,256,32,4)$	5300.24	32	3	10.67

Figure 6 shows the comparison of BLAST Single Master Method and BLAST Two Master Method. The graph is for $m = 512$ and $n = 256$. The graph is plotted with Total Time, T_t on x-axis and the Slice_Size sz on y-axis. sz was increased from 8 to 32. In the graph, SM stands for Single Master Method, TM stands for Two Master Method. It was observed that for all Input Size I, the performance of Two Master Method was better than that of Single Master Method.

Table 5: Results for Two Master Method.

$I = (m, n, sz, N)$	T_t	S_c	N_p	S_c^{Np}
I = (512,256,8,4)	1847.26	64	2	32
I = (512,256,16,4)	1471.01	32	2	16
I = (512,256,32,4)	1121.01	16	2	8
I = (1024,256,8,4)	4406.73	128	2	64
I = (1024,256,16,4)	2776.30	64	2	32
I = (1024,256,32,4)	2113.86	32	2	16

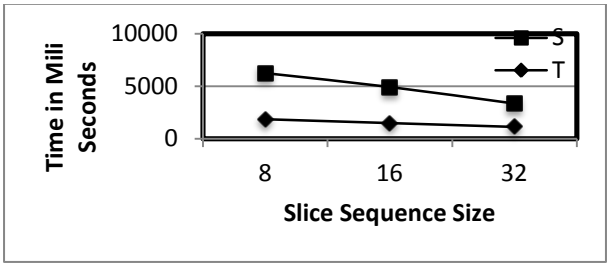

Figure 6: Comparison of the two methods for m=512.

It was observed that for m = 1024 also the performance of the Two Master Method was better than the Single Master Method.

4. CONCLUSION

To conclude, the BLAST parallel algorithms were experimented upon. It was observed that between the database segmentation and query segmentation method, the database segmentation method performs better since the method would reduce the page fault. It was also observed that the existing Single Master Method for database segmentation suffers from Single Master choke. Single Master Method was improved by using Two Master Method. In the Two Master Method, as experimented with, the tasks of the Single Master were split and the tasks in both Masters were executed in parallel. The load balance in the system improved. The performance of the Two Master Method was better than the performance of the Single Master Method.

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 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.

[2] Altschul F Stephen, Ralf Bundschuh, Rolf Olsen and Terence Hwa. 2001. "The estimation of statistical parameters for local alignment score distributions". Nucleic Acids Research, Vol. 29, No.2, pp.351-361.

[3] 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.

[4] Bjornson R.D, A.H. Sherman, S.B. Weston, N. Willard, J. Wing. 2002. "TurboBLAST: A Parallel Implementation of BLAST Built on the TruboHub". Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02), IEEE Computer Society.

[5] Braun R.C, K.T. Pedretti, T.L. Casavant, T.E. Scheetz, C.L. Birkett, C.A. Roberts. 2001. "Parallelization of local BLAST service on workstation clusters". Future Generation Computer Systems, Vol. 17, pp.745-754.

[6] Darling E Aaron, Lucan Carey, Wu-dhun Feng. 2003. "The Design, Implementation, and Evaluation of mpiBLAST". ClusterWorld Conference & Expo and the 4th International Conference on Linux Clusters: The HPC Revolution 2003. LA-UR 03-2862.

[7] 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.

[8] 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.

[9] Feng, W. "Green Destiny + mpiBLAST = Bioinformagic". 2003. 10th International Conference on Parallel Computing: Bioinformatics Symposium, PARCO, pp.653-660.

[10] Grant J.D, R.L. Dunbrack, F.J. Manion and M.F. Ochs. 2002. "BeoBLAST: distributed BLAST and PSI-BLAST on a Beowulf cluster". Bioinformatics. Vol. 18 no.5, pp. 765-766.

[11] Gropp William, Ewing Lusk, Anthony skjellum. 1999. Using MPI – Portable Parallel Programming with message-passing Interface, Second edition. London, England, Cambridge, Massachusetts: The MIT Press.

[12] Julich Anne. 2002. "Implementations of BLAST for Parallel Computers". CABIOS. Vol.11, no.1, pp.3-6.

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

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

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

[16] Mathog. R David. 2003. "Parallel BLAST on split databases". Bioinformatics, Vol.19, no.14, 1865-1866.

[17] 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.

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

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