Parallelizing and Analyzing the Behavior of Sequence Alignment Algorithm on a Cluster of Workstations for Large Datasets

Sathe. S.R and Shrimankar. D.D
Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology
Nagpur, Maharashtra, India

ABSTRACT

An MPI based parallelization technique for improving the scalability of the global sequence alignment algorithm on clusters of workstation is presented. We propose the parallel implementation of the Wavefront algorithm based on a chunk size transformation to handle large dataset with message passing model. Molecular biologists frequently align DNA sequences of entire genomes to detect important matched and mismatched regions. Even though efficient dynamic programming algorithms exist for this problem, the required computing time is still very high due to the size of these sequences. Because the number of sequenced organisms is increasing rapidly, fast and accurate solutions are of highest importance to research in this area. We show that an appropriate choice of the number of processes and chunk size has great impact on the overall system performance on cluster system. We have conducted the experiments on real-life DNA samples of house mouse mitochondrion and the DNA of rabbit mitochondrion obtained from the public database GenBank [GenBank, http://www.ncbi.nih.gov] in our experiment to measure the algorithm behavior appropriately. The results obtained from performed experiments, demonstrate that developed parallel Wavefront algorithm exposes high speedup and scales linearly with the increasing number of processes. Also the communication among processes and memory requirements are kept at minimum to achieve high efficiency. The experiments were performed on cluster which consists of two workstations of 12 core each with multithreading environment.

Keywords: Index Terms--- Sequence Alignment, MPI, Cluster Computing, Parallel Wavefront algorithm.

1. INTRODUCTION

In the last decade, we have observed an exceptional development in molecular biology [1,2]. An extremely high number of organisms have been sequenced in genome projects and included in genomic databases, for further analysis. These databases present an exponential growth rate and they are intensively accessed daily. The biological sequence comparison is one of the most important problems in computational biology [3]. It is in fact a problem of finding an approximate pattern matching between two sequences, possibly introducing spaces (gaps) into them. The most important types of sequence comparison problems are global and local. To solve a global alignment problem is to find the best match between the entire sequences. Local alignment algorithms must find the best match (or matches) between parts of the sequences. In this article, we will treat mainly global alignments. Needleman and Wunsch in 1970 [4] proposed an algorithm (NW) based on dynamic programming to solve the global alignment problem. It is an exact algorithm that finds the best global alignments between two genomic sequences of size n in quadratic time complexity and space complexity $O(n^2)$. In genome projects, the size of the sequences to be compared are constantly increasing, thus an $O(n^2)$ solution is still expensive. For this reason, heuristics were proposed to reduce time complexity to $O(n)$. NW is the most sensitive method but also the slowest one for similarity searches between sequences. One obvious improvement is the use of parallel processing to speedup the NW computations. Even in this case, the quadratic space complexity remains a problem and techniques must be used to reduce it. Biologists are thus faced with the problem of dealing with large datasets, in the search of meaningful similarities among biological sequences [5]. In order to do that, high computing power and large memory space are necessary. Moreover, sophisticated algorithms must be used that realistically model the relationships among the organisms. BLAST [6] is the most widely used heuristic tool that searches for similarities between biological sequences. Unlike the exact methods [4, 7] that present quadratic time and space complexity, BLAST is often executed quickly for pair wise sequence comparison, in a small memory space. In order to accelerate the production of sequence alignment algorithms, many parallel strategies have been proposed in the literature [8, 9, 10, 11, 12, 13]. Most of these strategies were implemented in homogeneous clusters and used message passing interface (MPI). However, nearly all recent clusters use multicore machines as their building blocks. Dynamic programming based algorithms can compute the optimal alignment of a pair of sequences [7]. However, since their complexities are quadratic with respect to the length of the two sequences this approach leads to a high computing time. One frequently used approach to speed up this time consuming operation is to introduce heuristics to the alignment algorithm [14, 15]. The main drawback of this solution is that the more time efficient the heuristics, the worse is the quality of the result [16]. Another approach to get high quality results in a short time is to use parallel processing. The system presented in [17] parallelizes the dynamic programming calculation and is able to achieve a speedup of 41 on a 64-node PC cluster for aligning two DNA sequences of length 816,394 and 580,074. In this study, we propose and evaluate our developed parallel Wavefront algorithm based on message passing model to implement NW algorithm on a cluster of workstation that uses commodity hardware and operating system. Our algorithm has a time complexity of $O(n)$. This means that the algorithm scales linearly with the increasing size of the dataset. One may conclude that it will be possible to handle very large datasets with our algorithm, depending on the available hardware, without having restrictions such as dataset size and other relevant criteria. Several test studies are performed and obtained results show that the algorithm possesses high and linear speedup with minimum communication requirements. We show that this approach generates high quality alignments

and leads to significant runtime savings on clusters of workstation. By aggregating the computing power of several cores in a cluster, this runtime can be reduced even further. To the best of our knowledge, there has been no previous report about the development and application of parallel Wavefront algorithm in the literature. We implemented the global sequence alignment (GSA) algorithm using C and OpenMPI library to align sequences of variable data sets on a platform consisting of two homogeneous workstation of 12 cores each. In our experimental results, we were able to reduce the execution time from 3 hour and 30 minutes (single fastest core) to 9 minutes and 30 seconds (24 homogeneous-core platform).

The remainder of this paper is structured as follows. In Section 2, we introduce the basic sequence alignment algorithm and the extension to compute simple Wavefront pattern and parallel Wavefront alignment on cluster of workstation. Section 3 provides description of the cluster architecture and mapping of application onto the parallel and distributed cluster architecture. The main results in terms of performance measurements of our parallel algorithm are presented in Section 4. Section 5 provides the concluding remarks.

2. Biological sequence comparison

The first algorithm for comparing biological sequences using the dynamic programming technique was proposed by Needleman and Wunsch in 1970 [4]. The algorithm consists of two parts: the calculation of the total score indicating the similarity between the two given sequences, and the identification of the alignment(s) that lead to the score. This algorithm had a great impact on later sequence alignment algorithms, such as the well-known Smith-Waterman method [7] and others [18] [19]. Therefore, speeding up dynamic programming algorithms for finding optimal solutions to sequence comparisons is an important problem in computational biology and bioinformatics. In this paper we will concentrate on the calculation of the score, since this is the most computationally expensive part.

Biological sequence comparison is in fact a problem of approximate pattern matching [20] that consists of finding which parts of the sequences are alike. To compare two sequences, we need to find the best alignment between them that is to place one sequence above the other making clear the correspondence between similar characters [21]. In an alignment, spaces can be inserted in arbitrary locations along the sequences so that they end up with the same size. Given

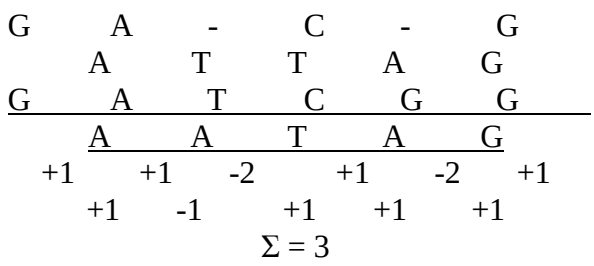

Figure 1. Global alignment between s = GACGATTAG and t = GATCGAATAG

an alignment between two biological sequences s and t, a score can be assigned for it as follows. For each column, we assign, for instance, +1 if the two characters are identical, -1 if the characters are different and -2 if one of them is a space. The score is the sum of the values computed for each column. The maximal score is the similarity between both sequences. Figure 1 shows the alignment of sequences s and t, with the score for each column. An exact algorithm (NW) based on dynamic programming that obtains the best global alignment between two sequences was proposed by Needleman and Wunsh [4]. To compute exact local sequence alignments, Smith and Waterman [7] proposed an algorithm (SW), also based on dynamic programming, with quadratic time and space complexity. Hirschberg [22] proposed an exact algorithm that calculates a local alignment between two sequences s and t in quadratic time but in linear space. This approach splits sequence s in the middle, generating subsequences $s_1$and s{}_{2}, and calculates the corresponding place to cut sequence t, generating subsequences t{}_{1} and t{}_{2}, in such a way that the alignment problem can be solved in a split and merge recursive manner. Usually, one given biological sequence is compared against thousands or even millions of sequences that compose genetic data banks. One of the most important gene repositories is the one that is part of a collaboration that involves GenBank at the National Center for Biotechnology Information (NCBI), the EMBL at the European Molecular Biology Laboratory and DDBJ at the DNA Data Bank of Japan. These organizations exchange data daily and a new release is generated every two months. By now, there are millions of entries composed of billions of nucleotides. In this scenario, the use of exact methods such as NW and SW is prohibitive. For this reason, faster heuristic methods are proposed which do not guarantee that the best alignment will be produced. Usually, these heuristic methods are evaluated using the concepts of sensitivity and sensibility. Sensitivity is the ability to recognize as many significant alignments as possible, including distantly related sequences. Sensibility is the ability to narrow the search in order to discard false positives [23]. Typically, there is a tradeoff between sensitivity and sensibility. Usually, heuristic methods use scoring matrices to calculate the mismatch penalty between two different proteins. In figure. 1, we assigned a unique value for a mismatch (-1 in the example) regardless of the parts involved. This works well with nucleotides but not for proteins. For instance, some mismatches are more likely to occur than others and can indicate evolutionary aspects [21]. For this reason, the alignment methods for proteins use score matrices which associate distinct values with distinct mismatches and reflect the likelihood of a certain change.

2.1 A simple Wave-front Pattern - Our Method to Implement the Global Algorithm

The Wavefront algorithm is a very important method used in a variety of scientific applications. The computing procedure is similar to a frontier of a wave to fill a matrix, where each block's value in the matrix is calculated based on the values of the previously-calculated blocks. In our implementation, the term wavefront has been used to describe our parallelizing strategy. The left part of figure. 2 shows the traditional wavefront structure for parallelizing the global algorithm, where the value of each block in the matrix is dependent on the left, upper, and upper-left blocks. Here the wavefront rounds which will be executing in parallel are put in same gray shade. The right part of figure. 3 shows our method to implement this wavefront pattern. Here in this paper we have used the term chunk in place of block. As shown in figure.

3(a) - (b) our wavefront structure executes these chunks in parallel. Thus core c_1 will execute the chunks 1,5,9... , core c_2 will excute the chunks 2,6,10... . That is, in time step 1 core 1 will execute chunk 1, in time step 2 core 1 will execute chunk 5 and core 2 will execute chunk 2 parallely, similarly in next time step core 1,2,3 will parallely execute chunk 9,6,3 respectively and so on. This parallel execution by corresponding cores are shown with the same gray shade in figure 3(b).

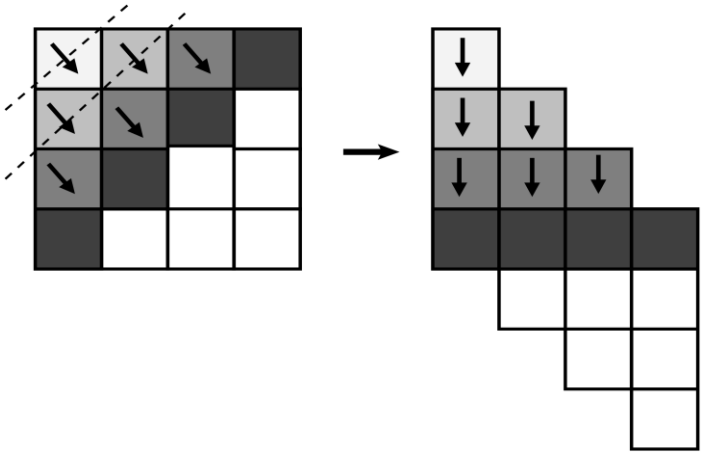

Figure 2. Wavefront structure of Global Algorithm for Biological Sequence Alignment. The left-hand side of the figure shows the traditional wavefront process with 4 rows and 4 column, and right-hand side shows our method of executing wavefront process with 7 rows and 4 column. For the detail of this method, please refer to [24].

Thus in our implementation, we employ a new chunk-based mechanism into the "Wavefront" method [24] to accelerate the global algorithm on cluster architecture. In general, our contribution is, we provide the design, implementation, and experimental study, of a new chunk-based mechanism based on the wave-front method to enable the alignment of very long sequences. For the sake of simplicity instead of using traditional rows and columns, we used diagonal rows during computation as shown in figure 2 left part. Each core before starting computation of a chunk, reads the above row and left column, and north-west entry, on the corresponding diagonal row of the matrix. Work is assigned in a column basis, i.e., each core calculates only a set of columns on the same diagonal. When a core finishes calculating a chunk, it writes its bottom row, and right column, and south-east entry in buffer space, which will be needed for next chunk calculation of the next diagonal of a matrix. In figure 2, left part has 4 rows and 4 columns which is a traditional wave front method. In our algorithm, as shown in right part of figure 2, we considered our new rows as 7 i.e row+column-1 and 4 columns. The idea behind this technique is that it is possible to simulate the filling of the original matrix by just using buffer in memory instead of complete chunk of data, since to compute chunk [i][j], we just need the finalRow value, finalColumn value, and entry of north-west corner of calculated blocks [i-1][j],[i][j-1], and [i-1][j-1] respectively.

2.2. Parallel Wave-front algorithm

In this section, we present our parallel implementation of Wavefront algorithm. The master–slave model and SPMD (Single Process, Multiple Data) technique on distributed-memory multiprocessor system [25] is applied to the algorithm as the parallelization technique. In this approach, each copy of the single program (each process) runs on cores independently and communication is provided by the manner of sending and receiving messages among cores. The terms of processor, core and process will be used interchangeably in the text. In our cluster system, each workstation is connected with an underlying Ethernet network. Datasets are stored in I/O (Input/Output) server via network file system, so that each node in the virtual system can have an access to datasets in a shared manner. Communication requirements between master and slave processes are managed by using Message Passing Interface (MPI) [26]. MPI is a specification rather than an implementation. OpenMPI [27] implementation is employed because of its high performance and it is one of the most widely used MPI implementation. We followed new chunk based algorithm with diagonal rows in parallel implementation of Wavefront algorithm. In this approach, each process works on a specific partition of the dataset and executes nearly identical code segments of the algorithm.

In the employed algorithm (see Algorithm 1), dataset is partitioned evenly among cores in a chunk manner as shown in figure. 2. Each subset of the dataset has a size of N/CS, where n is the total size of sequence X and CS is the chunk size, which gives tileXcount. Where tileXcount is the number of chunks in x direction. Similarly, tileYcount is the number of chunks in y direction of the matrix.

Algorithm 1 shows the parallel Wavefront algorithm, where the overhead of communication is little. The algorithm uses a master-slave parallelization method; the communication between the master process and the slave process only comprises chunk sizes of sequences pairs. Typically, the lengths and count of sequences are in master and slave buffer space. They are accessed from secondary memory during actual computation using offset value. Thus, the message in the communication is small. However, when the number of tileXcount and tileYcount is large, one can expect that frequency of communcation between the master and slave processes become high. So it is reasonable that a MPI implementation may show superior performance with large dataset size and large chunk size. In the parallel implementation the master process schedules the chunks of

sequences pairs on all other slaves. Each core performs the parallel alignment among chunks of sequence pairs. We implemented the parallel part using OpenMPI because the alignment for each pair of sequences for any dataset is the dynamic programming algorithm, where time complexity is O(mn), where m and n is the length of two sequences, respectively. Each dataset alignment takes varying amount of time. Thus algorithm reduces the memory requirement using split and merge strategy during execution of chunks in parallel.

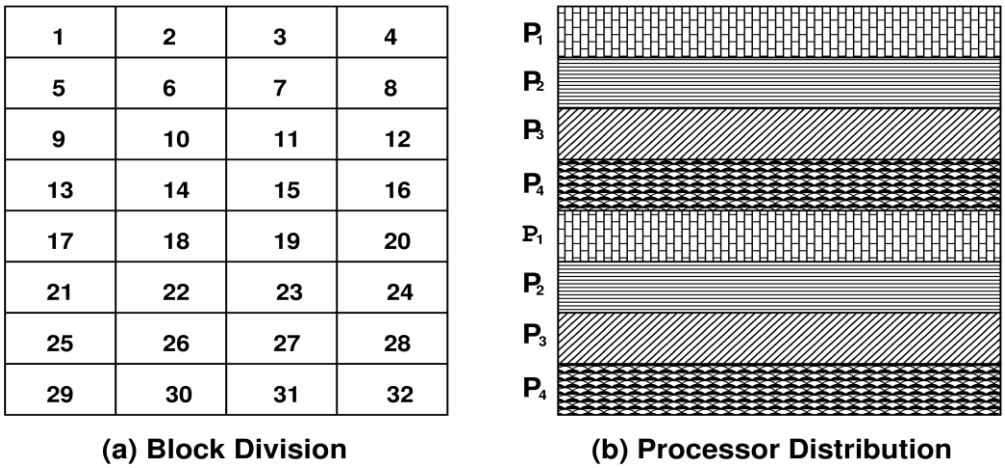

Figure 3. Partition of the similarity matrix. Figure 2(a) shows the processor number on corresponding chunks. Figure 2(b) executes these chunks in parallel as shown in same color.

Agorithm 1: MPI based Parallel Wavefront algorithm

1. for i = 1 to ndr do parallel /* ndr is the number of diagonal rows as shown in figure.3 right part*/
2. master node m_0 dynamically maps chunks of sequences to slave nodes, s_i, $1 \leq i \leq (r+c-1)$;
3. slave node s_i receives its own chunk data through Master_Buffer;
4. Each node computes alignment scores (as shown in figure 1), then send the scores to master m_0;
5. master m_0 stores the result from each slave si and schedules dynamically for next execution of chunk for next diagonal row
6. endfor
7. The final Score is received when last slave of last row (figure. 3 right part) finishes computation (through parameters finalScoreRowIndex & finalScoreColIndex)

3. Our Computational Architecture

The computational cluster architecture used in this paper consists of a cluster of workstations. The driving force and motivation behind this approach is the price/performance

ratio. Using workstation clusters is currently one of the most efficient and simple ways to gain supercomputer power for a reasonable price. Combining several of such clusters in a computational grid can improve the cost/performance ratio even further. The architecture of our system is shown in figure Therefore, executing parallel applications that have been designed for uniform speed interconnects can lead to severe performance degradations. However, many of these applications can increase their efficiency by reducing the inter-core data transfer. This can be achieved by fitting communication patterns to the interconnection structure of the architecture. In the following we describe a parallel programming environment consisting of two levels of MPI programs that is used to map the application described in Section 2 efficiently onto a cluster. The software architecture can be divided into two layers. The upper layer is an MPICH-G2 [29, 30] program that runs on the control node of each cluster. This allows (slow) inter-node communication. The

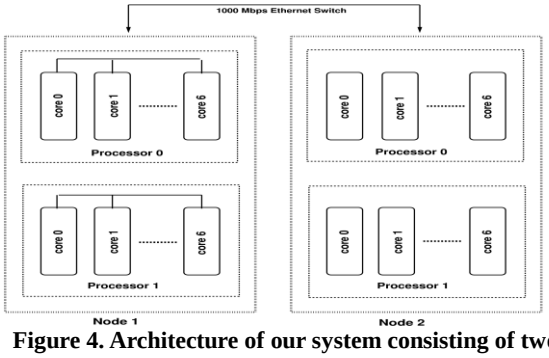

Figure 4. Architecture of our system consisting of two Linux based workstation of 12 core each with multithreading environment (Intel 5650 processor 2.66 GHz). A 1 Gbit Ethernet switch connects each workstation

lower layer consists of an MPICH [31] program that runs on all cores within a cluster. This allows (fast) intra-node communication. Data can be exchanged between the MPICH-G2 program and the MPICH program by reading and writing to a memory block in the control node (process) of each cluster that is shared by both programs (see figure 5).

3.1 Mapping the application onto the cluster architecture

The mapping of long DNA sequence alignment onto the cluster architecture consists of parallelization of the split and merge technique to calculate the actual alignments.

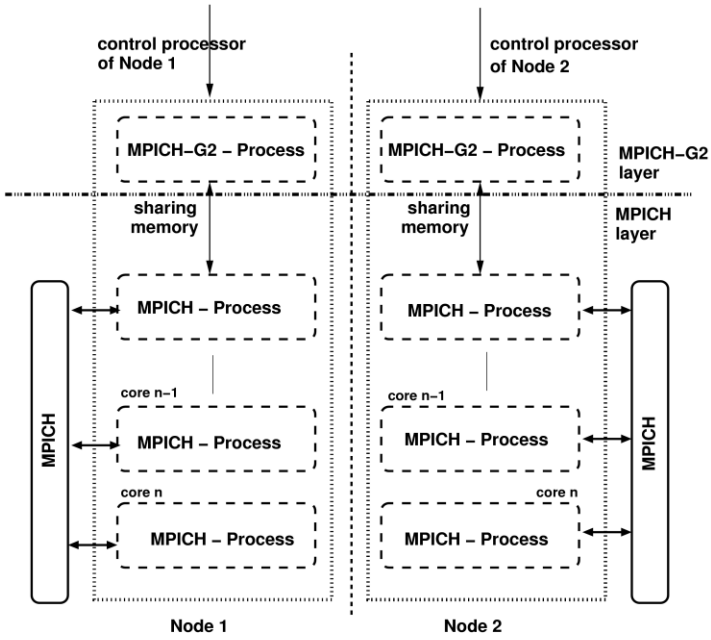

Figure 5. The parallel programming environment consisting of MPICH-G2 and MPICH programs. Both programs can exchange data in the control core of each node by means of a shared memory block.d

4. Results and Discussion

The experiments were performed on a cluster system each with a 2.66 GHz clock speed workstations where each workstation or node has 24 cores with Gbit Ethernet (1 Gbit/s) cards as communication infrastructure as shown in figure. 4. Two different real life DNA samples of house mouse mitochondrion (0.53MB) and rabbit mitochondrion (1.1MB) were used in the experiments [GenBank, http://www.ncbi.nlm.nih.gov]. The aim of selecting two dataset is to measure the performance of our algorithm on all types of datasets. The proposed algorithm gives better performance for very large size database with more number of autonomous nodes over MPI model. Processing times shown cores from disk during initialization stage and writing the results back to the disk at the end of the execution .

Parallelization of the similarity matrix calculation is based on the Wavefront communication pattern. Figure 3 displays the dependency relationship: each entry (i,j) of the matrix is computed from the entries (i−1,j), (i,j−1), (i−1,j−1). The Wavefront moves in anti-diagonals as shown in figure 3.

The shift direction is from north-west to south-east. Parallelization of the Wavefront computation has been done in different ways depending on the particular parallel architecture being used. On fine-grained architectures, the computation of each cell within an anti-diagonal is parallelized [32,33]. However, this technique is only efficient on architectures such as systolic arrays, which have an extremely fast inter-processor communication. On our computational architecture like homogeneous clusters of workstation, it is more efficient to assign an equal number of adjacent columns to each processor as shown in figure 3. Figure 2(a) shows an example of the computation for 4 processors, 4 columns and a clunk size of 1×1. The mapping on our architecture has two levels of partitioning. First the matrix is divided into parts of adjacent columns equal to the numbers of nodes. Afterwards the part within each node is partitioned. The computation is then performed in the same way as shown in figure 3. This reduces the inter-node data transfer to a single column per iteration step. In order to avoid bottlenecks on the homogeneous cluster architecture, the number of columns assigned to each node depends on its computational capabilities. Figure 6 displays the partitioning on our computational architecture.

in our results consider only the core times. The behavior and performance of the developed parallel algorithm are investigated and obtained results are presented in terms of execution times, speedup and efficiency for each source datasets i.e dataset of mouse (DSM) and dataset of rabbit (DSR) with respect to varying chunk sizes CS-2^{10} (1024x1024), CS-2^{11} (2048x2048), CS-2^{12} (4096x4096), CS-2^{13} (8192x8192), CS-2^{14} (16384x16384)). The results for datasets DSM, DSR were obtained with 2, 4, 8, 12, 16, 24, 32, 40 and 48 core architecture using multithreading environment. Execution times are obtained by using MPI routine MPI_Wtime. A shortcoming of dealing with large datasets in Wavefront approach is that the required I/O operations may dominate the execution time. This situation can be seen from table 1. The time spent for I/O operations includes the required time for loading the large dataset to local memories of compute

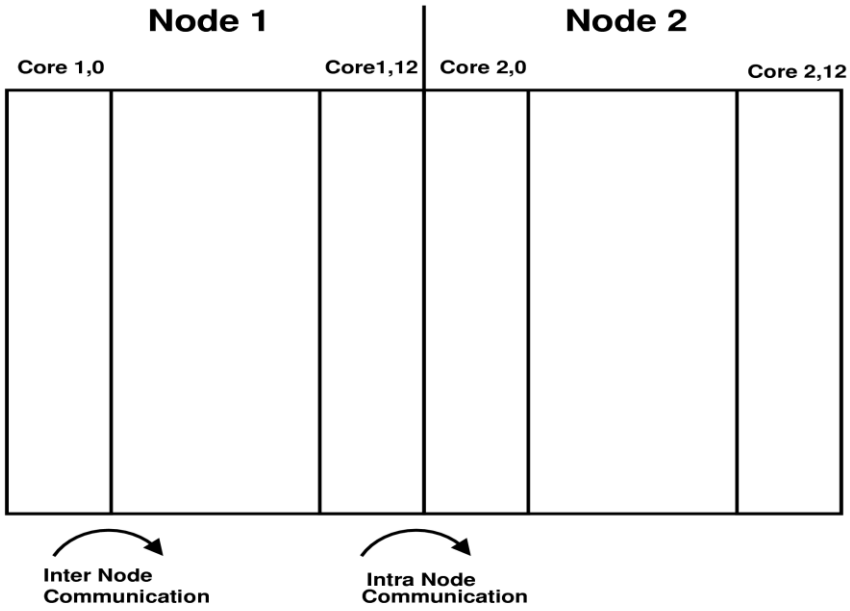

Figure 6. Partitioning of the wavefront computation on the cluster architecture to execute chunks in parallel using MPI.

Table 1
Execution times (in seconds) with (upper) and without (lower) consideration of I/O times for DSM with varying chunk sizes (2^{10}, 2^{11},2^{12},2^{13}, 2^{14}) and number of processors (np)

Dataset/np	2	4	8	12	16	24	32
DSM_2^{10}(I/O)	12023	4319	2783	1207	902	781	646
DSM_2^{12}(I/O)	11698	4010	1781	1150	865	693	609
DSM_2^{14}(I/O)	10747	3857	1746				
DSM_2^{10}	277	79.45	53.32	18.06	9.7	4.8	3.5
DSR_2^{12}	262	70.23	39.44	15.16	7.6	2.0	1.2
DSR_2^{14}	241	67.11	25.03				

It could be possible to argue that the improvement with parallelization may not be beneficial as much as expected due to the high I/O requirements during the initialization stage. However, this limitation could be turned into a gain since one may utilize two aspects that the parallelization may bring benefits for data processing.

The first benefit is obtained by applying a parallelization scheme directly to CPU-bound operations and the second one handling larger sizes of datasets that it could not be possible to process sequentially due to memory limitations. As seen from table 1, the values for I/O times decreases with increasing number of cores.

This split and merge type data distribution is achieved in our algorithm just by sending corresponding global data pointers to compute nodes/workstations. Each compute node loads necessary data to local memory from file server via network file system. Consequently, the time spent for I/O requirements is that distributing the data for IO-bound operations and processing in the context of distributed computing. The benefit obtained from the former one in our study seems to be relatively small due to the IO-bound nature of the Wavefront approach. But, it should be mentioned that there is a steady decrease in execution time without I/O times by increasing number of cores. The later one brings the capability of

is divided among processors. This expected benefit is obtained from gigabit Ethernet. Also, the execution times without consideration of I/O operations are decreased considerably because of the fact that communication time decreases in the merge phase. Execution times with I/O for different chunk size for DSM is calculated and depicted in figure. 7 for varying number of processes. Also, it is observed that the time spent for reading and processing the data is independent of dataset size and also indicates good load balancing feature of the developed algorithm on dataset of mouse and rabbit

.

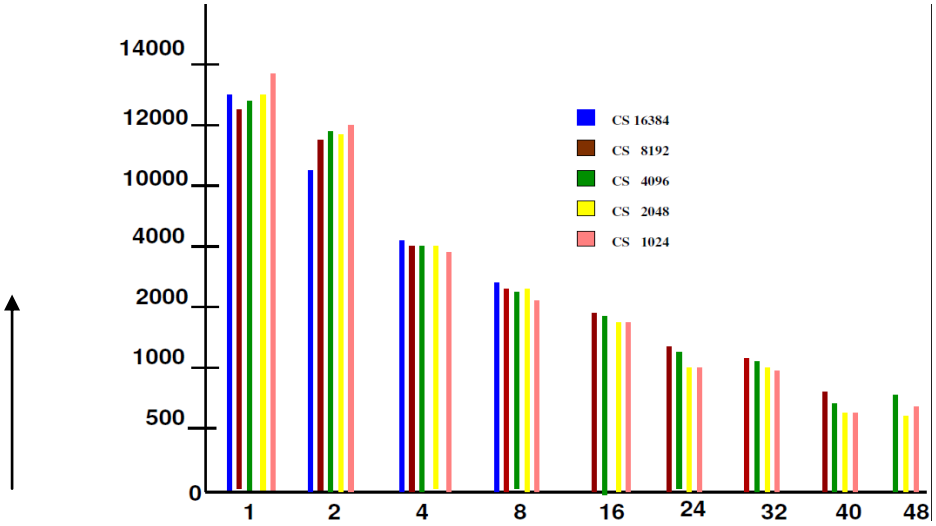

Fig.7.a

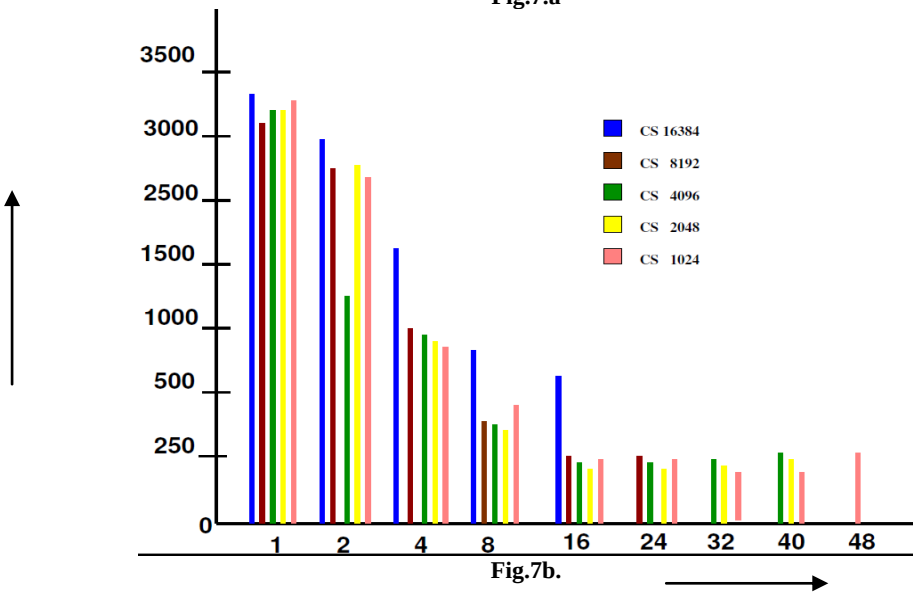

Fig.7b.

Figure 7. Execution time with I/O on (a) DSM and (b) DSR

Overall, our conclusion is that the total execution time is decreased considerably by parallelization of CPU-bound and/or IO bound operations and applying a parallel algorithm using Wavefront approach is beneficial. Since I/O time highly depends on the speed of network infrastructure, this conclusion may be more applicable in the case of Gbit Ethernet. As our parallel Wavefront algorithm is executed on cluster of workstation using faster network infrastructure such as 1/10 Gigabit Ethernet and/or using faster storage networks, it is expected that this sort of shortcoming (I/O time limitation) could be surmounted to some extent.

Although a considerable improvement with parallelization is obtained by distribution of IO-bound operations, those I/O times are not considered for the presented figures in the rest of the text. The reason for this is that the time spent for reading of dataset and writing of result from/to the disk is highly dependent on numerous factors including read–write speed of the disk, speed of the network and format of the input/result file. Hence, we have only focused on the parallelization behavior of the developed parallel Wavefront algorithm.

The behavior of execution times by increasing number of cores for datasets DSM and DSR with different number of processes and chunk sizes are plotted in figure. 8. The expectation of decrease in execution time by increasing number of processes is satisfied for almost all the cases except some specific cases for both datasets. In figure 8(c) (CS 2^{12}), for DSM and DSR execution time for more than 24 cores and 40 cores respectively are found to be greater. Some more cases are found, like in figure. 8(d)&(e) (CS 2^{11} and CS 2^{10}), for DSM and DSR execution time for more than 40 processes and 32 processes respectively are found to be greater. This

case can be explained due to the additional communication time per cluster of sending master/slave buffer data space, chunk space and time required for storing scores from all cores by master to update its own data. But, updating master/slave data buffer space for each chunk size is the main reason for this performance degradation. As the number of cores increases, less time is required in the operation of updating data in buffer of reduced chunk space per cores. Accordingly, we obtained very good timing values and linear speedup in performance. Updating is implemented as scanning transformed chunk space and replacing all occurrences of old buffer data with new ones. This procedure is performed in a fast manner via pointer which directly points to memory of all associated units.

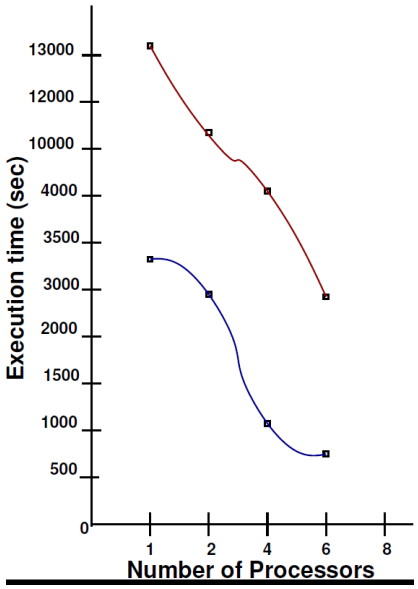

Fig.8.a.(for chunk size 2^{14}x 2^{14})

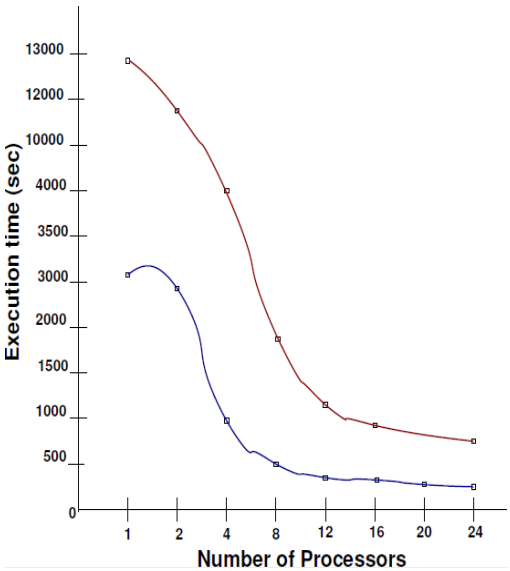

Fig.8.b.(for chunk size 2^{13}x 2^{13})

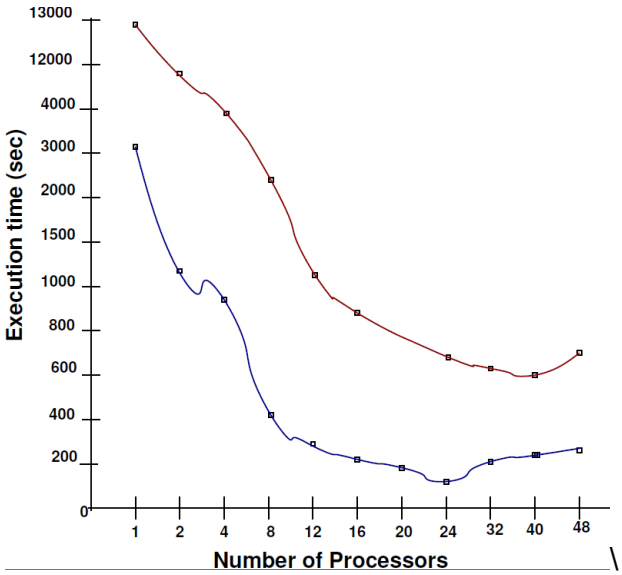

Fig.8.c.(for chunk size 2^{12}x 2^{12})

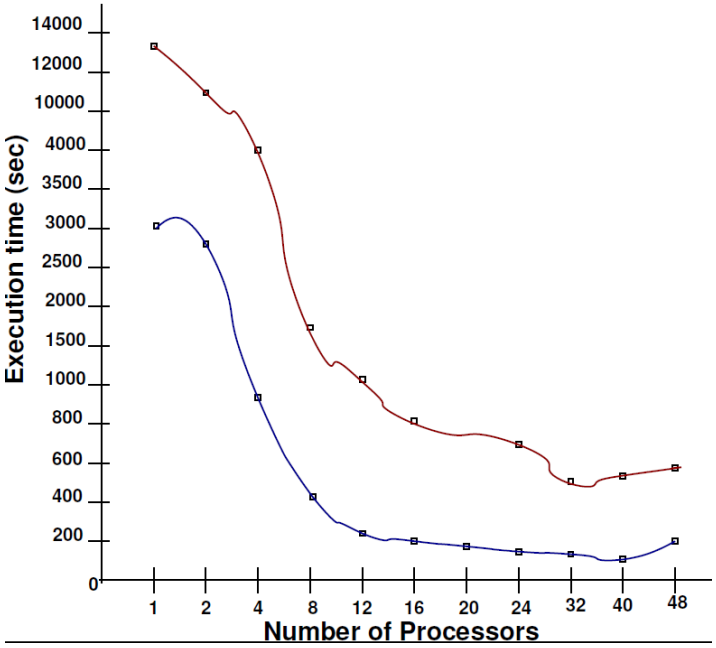

Fig.8.d.(for chunk size 2^{11}x 2^{11})

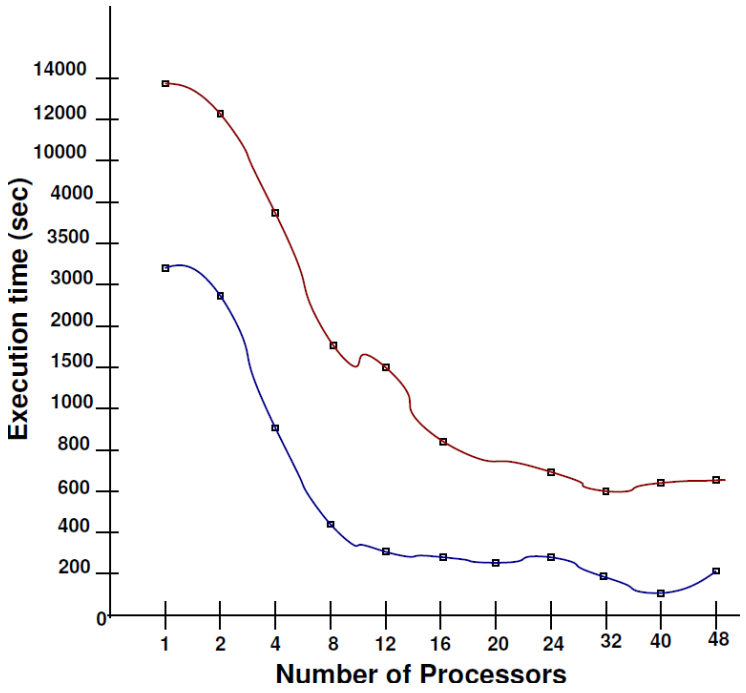

Fig.8.e.(for chunk size 2^{10}x 2^{10})

Figure 8. Execution times for both dataset with varying chunk sizes (2^{14},2^{13},2^{12},2^{11},2^{10}) and number of processes (2, 4, 8, 16, 24, 32, 40, 48)

Speedup (Sp) is defined as the ratio of computation time on a single core to the computation time on C cores. The obtained speedup ratios for DSM, DSR with number of cores and different chunk sizes are presented in figure. 9(a)–(e), respectively. It is clearly seen from these figures that our developed parallel Wavefront algorithm scales almost linearly. In figure. 9(a), the speedup line for DSR is higher than the lines for DSM. The number of communications from chunk size 2^{14} to 2^{10} (figure 9.b to 9.e)is decreasing and hence time required for internal computation is increasing. This elevation of the line for both datasets at most time consuming chunk size value confirms the suitability of our parallel Wavefront algorithm for huge datasets with large chunk size. The speedup values for both datasets figure. 9(a) - (e) are almost same up to 8 processors. The separation of the lines at the processor size 16 can be explained as the unsustainable balance between the computed local data and communicated data. There are comparatively less amount of communication for 8 cores while less amount of calculation for 16 cores. This situation changes for cores more than 16 for all chunk sizes where lines at each processor sizes are not matching. The small deviations at that processor size are simply due to the increasing communication requirements. This effect is more apparent for chunk size 2^{13} on 16 processes. Due to comparatively less amount of computations at that size, the time spent for the communication becomes considerable.

For all chunk size values, the slope for speedup values is reduced from 16 to 32 processors regardless of the dataset size. It is seen from the figure. 9(a) - (e), the linear behavior of the algorithm is getting improved as chunk size and dataset sizes increases. Table 2 shows the obtained efficiency values for both datasets with different number of cores. The reason of presenting efficiency results in this table for all chunk size is that it represents the behavior of our algorithm most conveniently. Obtained efficiency values for DSM are considerably high for np = 8 and np = 12, then exhibits a steady decrease with the increasing number of cores. This behavior is due to increasing communication requirements. Updating local buffer from memory by each core increases hence communication cost per processors increases.

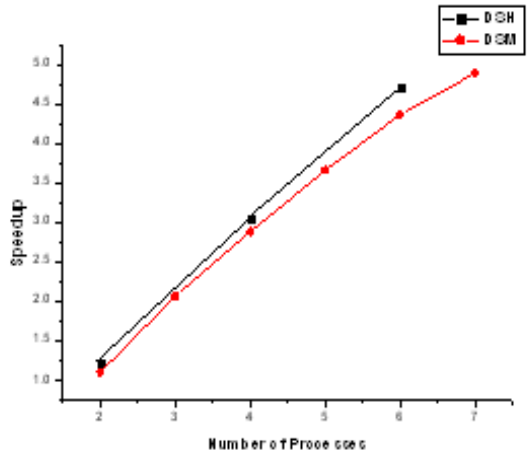

Fig 9(a)

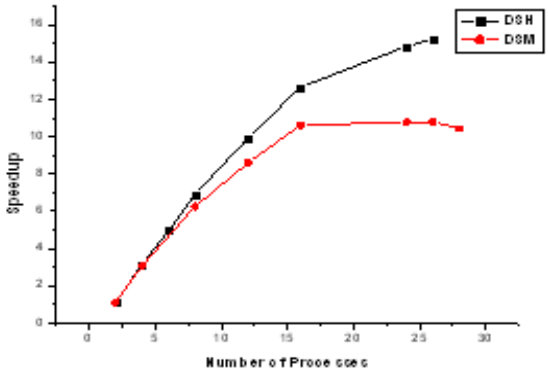

Fig 9(b)

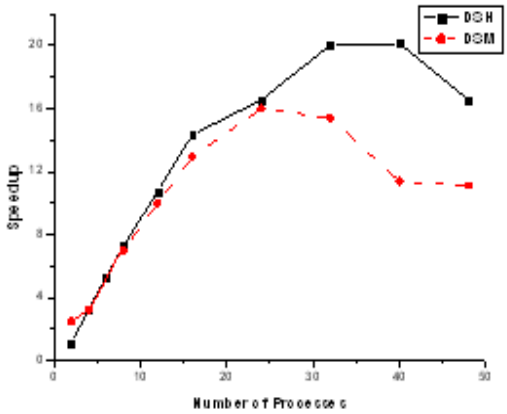

Fig 9(c)

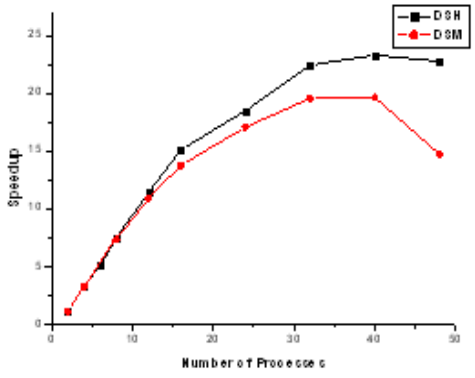

Fig 9(d)

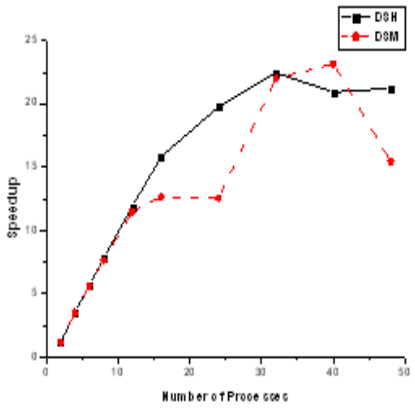

Fig 9(e)
Figure 9. Speedup vs Number of Processes for all chunk sizes on both data set

Table 2
Efficiency with (upper) and without (lower) consideration of I/O times for DSM with varying chunk sizes and number of processes

Dataset/np	2	4	8	12	16	24	32	40	48
DSM (1024)	0.59	0.88	0.96	0.96	0.79	0.52	0.69	0.58	0.32
DSM (2048)	0.56	0.83	0.92	0.91	0.86	0.71	0.61	0.49	0.31
DSM (4096)	1.23	0.80	0.87	0.83	0.81	0.67	0.48	0.28	0.23
DSM (8192)	0.55	0.77	0.78	0.72	0.66	0.45			
DSM (16384)	0.55	0.72							
DSR (1024)	0.57	0.89	0.98	0.99	0.99	0.82	0.70	0.52	0.44
DSR (2048)	0.56	0.82	0.94	0.95	0.94	0.77	0.70	0.58	0.47
DSR (4096)	0.55	0.81	0.91	0.89	0.90	0.69	0.63	0.50	0.35
DSR (8192)	0.55	0.79	0.87	0.83	0.82	0.62			
DSR(16384)	0.61	0.76							

We have obtained better efficiency values as chunk size increases. The reason is that the execution time of Wavefront transform neither depends on the cluster size complexity nor dataset sizes but depends on the performance of local component of updating master/slave buffer space which directly depends upon chunk size. For this reason, as chunk size increases, finding the local components on that transformed values is faster compared to small chunk size local values instead of implementing master–slave model and thus each processor can decide its start time of new computation.

values. This implies a relation between chunk size and memory size related to efficiency for a given dataset. Our parallel implementation can be further improved such that the efficiency trend might be better than the present situation. The present minor restriction is due to the fact that master core waits all cores before starting new execution of new diagonal row. A possible solution is broadcasting masters'

Speedup values with respect to the number of processors for both datasets are also plotted to understand the algorithmic behavior as another point of view, see figure. 10(a) & figure. 10(b) each symbols on the data line correspond to number of processors in datasets DSM & DSR respectively. Figure. 10(a) shows that speedup values are linearly increasing for all chunk sizes up to number of cores 16. After that of chunk size 2^{11} speedup decreases from np=8 onwards. This is due to increasing communication of processors with respect to increasing chunk sizes.

When DSR is used to show speedup analysis as shown in figure. 10(b), the speedup values remain almost same for all chunk sizes up to np = 24. Then same as DSM there is a subtle decrease for CS 2^{10} and CS 2^{12}. The relatively high speedup is obtained for DSR dataset than DSM for 32, 40 cores that means system supports the suitability of the algorithm for larger datasets.

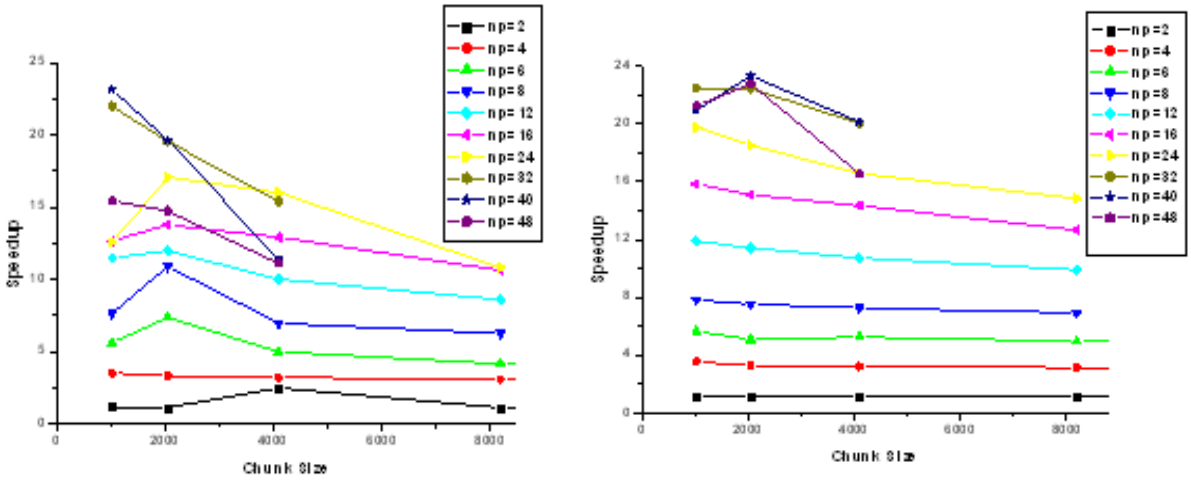

Fig 10(a) **Fig 10(b)**

Figure 10. Speedup with varying number 0 chunk sizes and number of processors (2,4,6….. 48) for (a) DSM (b) DSR

6. CONCLUSION

In this study, an efficient parallel implementation of Wavefront algorithm based on the message passing model for distributed-memory multiprocessor system is developed. We have presented obtained execution time, speedup and efficiency results for varied number of chunk size on different datasets for different number of processors, to reveal the performance of developed algorithm. Experiments are performed on a PC cluster of 2 workstations with 48 cores (using multithreaded environment) and having gigabit Ethernets as underlying communication hardware. Results have shown that our parallel Wavefront algorithm exposes superior speedup and can be employed to overcome time complexity, space complexity constraint as well due to low memory consumption. As a parallel implementation strategy, we adopted master/slave model and followed our new chunk based method with diagonal rows in Wavefront approach to increase the performance and to reduce the amount of memory consumed at each processor for holding local datasets. The communications among processors are kept at minimum to achieve high efficiency, such that each processor accesses its subset of large dataset directly in a shared manner. A minor restriction is due to the fact that parent processor waits all processors before starting new computation to new processor. When this minor restriction is addressed, the presented parallel implementation can be further improved. An another way of improving the parallelization of the presented Wavefront technique is to feasibly apply the algorithm on a heterogeneous cluster architecture using thread programming technique on OpenMP/MPI hybrid model

.

7. REFERENCES

[1] A. Boukerche, Alba Melo, “Bioinformatics applications in grid computing environments,” *Grid Computing for Bioinformatics and Computational Biology*, pp. 301-325, 2007.

[2] A. Boukerche, A.C. de Melo, “Computational molecular biology,” *Parallel Computing for Bioinformatics and Computational Biology: Models Enabling Technologies, and Case Studies*, pp. 149-166, 2006.

[3] J. C. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*. Brooks/Cole Publishing Company, 1997.

[4]. Saul B. Needleman and Christian D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two sequences,” *Journal of Molecular Biology*, pp. 443-453, 1970.

[5] A. Zomaya, *Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, 2006.

[6] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, David J. Lipman, "A basic local alignment search tool," *Journal of Molecular Biology*, pp 403-410, 1990.

[7] T.F. Smith, M.S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol*. Pp. 195-197, 1981.

[8] Chaichoompu K, Kittitornkun S, Tongsima S. "MT-clustalW: multithreading multiple sequence alignment," *IPDPS*,

[9] Li KB, "ClustalW-MPI: ClustalW analysis using distributed and parallel computing," *Bioinformatics*, pp. 1585–1586, 2006.

[10] Schmollinger M, Nieselt K, Kaufmann M, Morgenstern B, "DIALIGN P: fast pair-wise and multiple sequence alignment using parallel processors," *BMC Bioinform*, 2004

[11] Tan G, Feng S, Sun N, "Parallel multiple sequences alignment in SMP clusters," *Highperformance computing*, 2005.

[12] Zola J, Yang X, Rospondek A, Aluru S, "T-Coffee: a parallel multiple sequence aligner," *PDCS*, pp. 248–253, 2007.

[13] Boukerche A, Correa JM, Melo ACMA, Jacobi RP, "A hardware accelerator for the fast retrieval of DIALIGN biological sequence alignments in linear space," *IEEE Trans. Comput*, pp. 808–821, 2010.

[14] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., "Basic local alignment search tool," *Journal of Molecular Biology*, pp. 403-410, 1990.

[15] Chao, K.M., Zhang, J., Ostell, J., Miller, W., "A local alignment tool for long DNA sequences," *Computer Applications in the Biosciences*, pp. 147-153, 1994.

[16] Pearson, W.R., "Comparison of methods for searching protein sequence databases," *Protein Science*, pp. 1145-1160, 1995.

[17] Martins, W.S., del Cuvillo, J.B., Cui, W., Gao, G.R., "Whole Genome Alignment using a Multithreaded Parallel Implementation," *Proceedings 13th Symposium on Computer Architecture and High Performance Computing*, September 2001.

[18] Osamu Gotoh. "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, pp. 705-708, 1982.

[19] E.W. Myers and W. Miller, "Optimal alignments in linear space," *Computer Applications in the Biosciences*, pp. 11-17, 1988.

[20] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, 2001.

[21] J.C. Setubal, J. Meidanis, "*Introduction to Computational Molecular Biology*," Brooks/Cole Publishing Company, 1997.

[22] D.S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, pp. 341-343, 1975.

[23] David W. Mount, *Bioinformatics: Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, 2004.

[24] S. Aji, F. Blagojevic, W. Feng, D.S. Nikolopoulos. "Cell-SWat: Modeling and Scheduling Wavefront Computations on the Cell Broadband Engine" *Proceedings of the ACM nternational Conference on Computing Frontiers*, pp. 13-22, 2008

[25] C. Quammen, *Introduction to programming shared-memory and distributed memory parallel computers*, Crossroads 12, 2005.

[26] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, USA, 1994.

[27] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham,T.S. Woodall, "Open MPI: goals, concept, and design of a next generation MPI implementation," *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, pp. 97–104.

[28] Plaat, A., Bal, H.E., Hofman, R.H.F., "Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects," *Proceedings 5th IEEE HPCA'99,* pp. 244-253, 1999.

[29] Foster, I., Geisler, J., Gropp, W., Karonis, N., Lusk, E., Thiruvathukal, G., Tuecke S., "Wide-Area Implementation of the Message Passing Interface," *Parallel Computing*, pp. 1735-1749, 1998.

[30] http://www.niu.edu/mpi

[31] http://www-unix.mcs.anl.gov/mpi/mpich/

[32] Schmidt, B., Schr¨oder, H., Schimmler, M., "Massively Parallel Solutions for Molecular Sequence Analysis," *Proc. IPDPS'02,* Ft. Lauderdale, Florida, 2002.

[33] Schmidt, B., Schr¨oder, H., Schimmler, M., "A hybrid architecture for bioinformatics," *Future Generations Computer Systems*, pp. 855-862, 2002.