Parallel Algorithm for Multiple Genome Alignment on the Grid Environment

Nova Ahmed[1], Yi Pan[1] and Art Vandenberg[2]
[1]*Department of Computer Science,* [2]*Information Systems & Technology*
Georgia State University, Atlanta, GA 30303, USA
Email: pan@cs.gsu.edu

Yuzhong Sun
Institute of Computer Technology, Chinese Academy of Science, Beijing, China

Abstract

The multiple genome sequence alignment problem falls in the domain of problems that can be parallelized to address large sequence lengths. Although there is communication required for the computation of the aligned sequences, the proper distribution can reduce the overall problem to a set of tasks to be solved independently and then merged. A parallel algorithm for the alignment of multiple genome sequences is described. The algorithm is experimentally evaluated in a distributed Grid environment that provides very scalable and low cost computation performance. The Grid environment is evaluated with respect to a traditional cluster environment and results are compared to evaluate the effectiveness of a Grid environment for large computational biology.

1. Introduction

Biological sequence alignment is very important for the analysis of the genome and protein structures [15]. Multiple genome sequence alignment is particularly important as it shows the relationships among the structures being aligned. This problem draws the attention of the researchers as it becomes extremely computationally intensive for sequences of very large size.

Many different approaches have been used to align genome sequences. There are methods using the Needleman and Wunsch [10] algorithm to find the optimal alignment of sequences and then to search for the global alignment among all the sequences. But aligning all the sequences can take a significant amount of time and thus there are methods that forego optimal results, reducing computation time in exchange for finding only a near optimal [18] alignment. There is also a need to find a local alignment among multiple sequences when the sequences have some common regions to share but they are different otherwise. There are many different approaches to find the local multiple alignments [7].

For multiple sequence alignment, many algorithms are used that rely on the comparison of pairwise sequence alignment [13]. The double dynamic method can be used for multiple sequence alignments [16] reducing some complexity of other algorithms. There are some algorithms that consider the structure of the sequences for alignment [14]. Multiple sequence alignment can be done globally or locally but the optimal algorithm to align multiple sequences becomes very computationally intensive with more than three sequences [1]. The multiple sequence alignment algorithms often forego optimal algorithms for less computationally intensive solutions that can get near optimal results. In this paper, a simple approach to find out the multiple sequence alignments among different sequences is described which uses the basic pairwise algorithm [11] [3]. The computation time is reduced by introducing parallelism to distribute work evenly among processors while minimizing communication latency between the processors.

Taylor [13] uses a very basic algorithm for global alignment among sequences, comparing the pairwise alignment among the sequences using the basic dynamic programming algorithm and then finding global alignment among the sequences depending on the score of the pairwise alignments. Although suboptimal alignment algorithms are used by many applications to avoid the excessive computation time required by optimal global alignment, the algorithm in this paper is designed to handle computation intensive tasks.

The Grid environment has been chosen for an experimental environment as it shows promise in terms of computationally intensive jobs. It provides the user with a service-oriented view [5] that accesses the power of the distributed resources with minimum concern to the user about the structure of the architecture. Tests of a grid-enabled cluster with its grid middleware layer show

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

comparable performance to the performance of a non-grid-enabled cluster. Moreover, the computational grid environment allows for larger sequences to be aligned and compared than can be accommodated by the shared memory environment that was tested. The studies show that the grid is very promising for computation intensive problems like the computational biology problems.

The paper is organized in the following manner: section 2 describes the grid environment in general and the specific environment used; section 3 describes the multiple sequence alignment problem which covers the basic algorithm and the parallel algorithm as well; section 4 covers the experimental results; and the paper is concluded in section 5.

2. The grid environment

The Grid environment is a very promising one in distributed computing because of its enormous computation and resource sharing capability and for the user level abstraction in accessing resources without having any actual knowledge of those resources once the user is connected to the grid. Also the Grid environment has a major advantage of being robust and scalable. According to the definition of Ian Foster and Carl Kesselman [4] *"A Computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities"*

The Grid protocol architecture defines protocols of how Virtual Organization (VO) users negotiate, establish and manage sharing relationships [3]. Developers of applications for VOs have certain requirements for the Grid environment. VO interoperability across organizational unit boundaries is a major aspect to consider as it indicates how a service would be defined across distributed resources and the set of protocols to invoke that particular service. The Grid service oriented view has many advantages like allowing a standard interface definition mechanism and simplified virtualization [5]. Although there is the overhead of the Grid protocols in the distributed environment, from the user's point of view the grid environment is very convenient for large applications that require a greater amount of resources than may be available in the local organization.

The grid described here uses the Globus Toolkit, built on the Open Grid Services Architecture (OGSA) which is open source software that allows users to share computational and other resources without sacrificing local autonomy. OGSA includes software services and libraries for resource monitoring, discovery and management and takes care of the security issues. This toolkit gives users an abstraction for accessing the different grid-enabled resources.

Grid communication is complicated as it has to address the heterogeneity of the environment and a potentially large variety of protocols. A special communication library called the Nexus is used which allows multi-method communication with a single API for a wide range of protocols. Using Nexus [3], a grid enabled Message Passing Interface, MPICH-G2 [17] has been used in the experiments described here.

Resource management in Globus uses a higher level resource management services layer on top of the local resource management services. There are three main components of the resource management system – the extensible resource specification language (RSL), the interface to the local resource specification tools, and a co-allocator.

The RSL provides a common interchange language to describe various resources. It provides the skeletal syntax used to define complex resource descriptions. It is used to specify which resource to access when a job is submitted to the grid. There are different built-in functions of the toolkit to simplify the job submission and execution (*globus-job-submit*, *globus-job-run*) and to check the status (*globus-job-status*). These functions have been used to launch the specific grid-enabled applications described in this paper.

3. Multiple sequence alignment

Multiple genome sequence alignment is important in biological sequence computations where several genome sequences are aligned rather than only two based on some limited criterion. Multiple sequence alignment can be done globally or locally but an optimal algorithm to align multiple sequences can become very computation intensive when there are more than three sequences. That is why most multiple genome sequence alignment solutions work with heuristic algorithms, trading off improved speed for somewhat reduced optimality of the result. Local and global sequence alignment algorithms are discussed here, the basic method being an optimal algorithm using comparison of a sequence with every other sequence. This computation intensive task is distributed among different processors in a shared memory environment or a grid environment so that the computation speed is retained while achieving more optimal results.

3.1. Basic pairwise sequence alignment algorithm

The method described here for multiple genome sequence alignment is based on pairwise sequence alignment algorithm [1]. The sequence alignment of genome structures is different from usual string sequence matching as genomes may allow a mismatch, a change of

symbol, or a gap to occur between two sequences, and yet still be considered a sequence match. A match or a mismatch is taken into account by a score assigned to each pair of sequences to be matched [9].

Using this algorithm, the optimal result can be easily found from the previously computed partial results. A two dimensional array SM is required to store the partial results. SM has height and width according to the length of the sequences to be matched. Every match in the sequence is given an individual score of 1 and a mismatch is given –1. The total score of the two sequences is considered. An element in SM[x,y] can be generated as shown by the following recurrence relation where *ss* indicates the substitution score indicating a match or mismatch and *gp* is the gap penalization.

$$SM[x, y] = \max \begin{cases} SM[x, y-1] + gp \\ SM[x-1, y-1] + ss \\ SM[x-1, y] + ss \\ 0 \end{cases}$$

The Similarity Matrix SM is built following the stated recurrence equation where every element needs its neighbor elements to be computed prior to it. The maximum score is considered as the best score when creating the matrix.

3.2. Basic multiple sequence alignment algorithm

The significance matrix is a two dimensional matrix consisting of the pairwise sequence comparison scores of the genome sequences [13]. The pairwise alignment can be computed using any of the existing sequence alignment algorithms. This significance matrix is used to find the best alignment among multiple sequences.

In this method, the best pairwise alignment score is chosen from the significance matrix as the starting point of the algorithm. Then the next best pair of alignments is found which includes one sequence of the previous pair. This process continues, extending the connection in both directions from the original best pair, until all the sequences are linked. There will be a unique ordered list among the genome sequences unless there is a tie in the pairwise alignment score.

It should be mentioned that if the sequences A, B, C are compared for multiple genome sequence alignment, there may be alignment between sequences A, B and between B, C having the two best scores, creating the chain of A, B and C. However the relation resulting from the alignment process does not have the transitive property in regards A, C alignment.

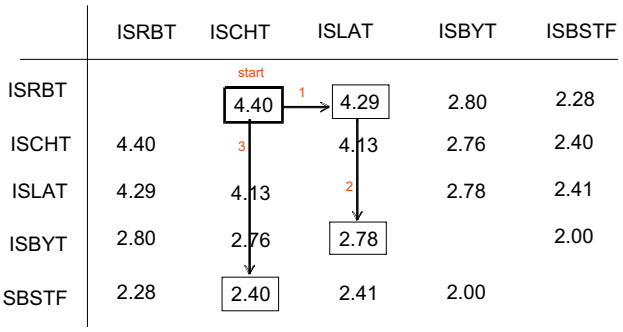

Figure 1. Basic algorithm for multiple sequence alignment [13]

Figure 1 shows a significance matrix with five separate sequences to be compared for alignment. The highest matching pair of the sequences is ISCHT with ISRBT (4.40) and so ISCHT and ISRBT are considered as the starting points of the alignment. Then the next best alignments are found, ISCHT with ISLAT (4.13) and ISRBT with ISLAT (4.29) and the latter is chosen due to its higher score. The two new ends of the list (ISCHT, ISLAT) are then considered and the next two best scores are chosen in the same manner, treating each of the ends as new starting points. ISCHT with ISBYT (2.76) and ISLAT with ISBYT (2.78) are considered, and the sequence ISBYT is chosen as the next best pair for its alignment score with ISLAT (2.78). Again the two ends (ISCHT, ISBYT) are considered and finally ISBSTF is chosen as the next best alignment among those remaining. Although, there are better alignments among pairs of sequences, the chain of sequence needs to consider a pair that has at least one member at the end. For example, when considering the chain having two ends (ISCHT, ISLAT), it is seen that ISBYT aligns best with the sequence ISRBT having a score of 2.80, but ISRBT is not one of the ends. ISBYT's alignment (2.78) with ISLAT determines the connectivity of the chain.

3.3. Parallel algorithm for multiple sequence alignment

For the multiple sequence alignment algorithm the basic improvement is achieved by introducing parallelism to divide the workload among several processors. The individual processors are responsible for pairwise alignment of several sequences and this computation does not have any inter-dependency between processors. Thus by carefully dividing the workload, the individual processor load is reduced and the overall performance of the algorithm is improved.

The searching for the best score is also improved as individual processors need to search the best score only

from the data it computed. In the basic multiple sequence alignment program each sequence must be compared with all the other sequences in order to find out the best alignments. For a single processor implementation, as the number of sequences increases, the searching time increases. Parallelization of the searching reduces the number of sequences to be searched by a single processor as the load is divided among different processors.

The parallel algorithm divides the sequences among the processors with respect to the corresponding significance matrix. Each processor searches for its local best alignment pair from the starting point that includes a sequence from that pair. Then the processors communicate to find a global maximum before searching for the next best pair. The aligned sequence is generated in the same way as it was done in the basic algorithm. Although there is a communication overhead to determine the global maximum, the searching time improves by operating in parallel on different processors.

There are three major steps in the parallel algorithm for multiple sequence alignment. The first step finds out the starting point from which the sequence will be aligned. An index is set with each sequence which indicates the number of comparisons of a particular sequence. This index can have the maximum value of 2 so that a particular sequence can be compared in two directions with other sequences but not more than that. In the second step each processor finds out the local maximum score of pairwise alignment and the highest value along with the matching sequences that has at least one of the sequence previously selected sequences. Then the global maximum of the local maximum scores is determined and all the processors are notified about the new pairwise sequences. Steps two and three repeat making sure that a chain has been created including all the sequences that are unique (unless there is a tie for the score values of the sequences).

3.4. Data structure and program formulation

The underlying data structure used here is very simple and similar to the sequential program for multiple genome alignment. The sequences to be aligned are distributed among the processors and every processor is responsible for pairwise alignment of its own particular genome sequences. To distribute the workload evenly, the matrix of sequences (columns and rows of sequences) is distributed among different processors. Sequences are divided among the processors in the grid like manner shown in **Figure 2**.

The parallelism is increased as every processor is responsible for its individual pairwise alignment task without any dependencies on the task of another processor. The processors can perform the pairwise alignment of sequences in parallel. Every processor keeps the list of sequences to be aligned and another array *used*, which indicates whether or not a particular sequence has been chosen for the multiple genome sequence alignment. There is another important data structure *list* that keeps the alignment of multiple genome sequences.

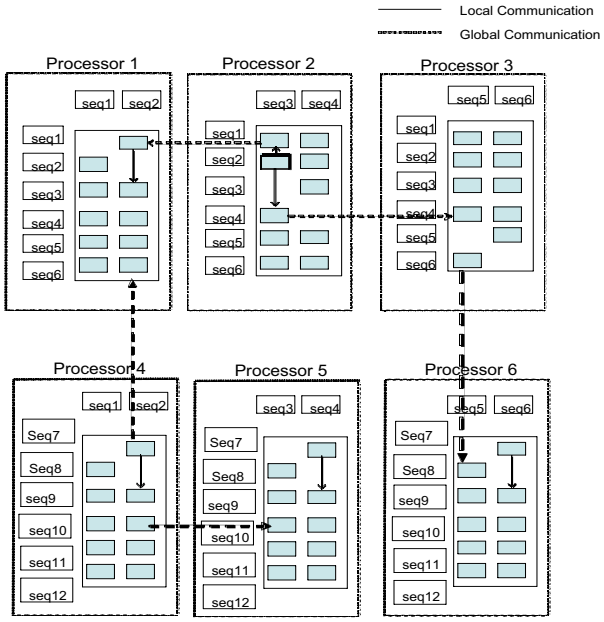

Figure 2. Parallel load distribution among processors for multiple sequence alignment

After the processors finish the pairwise sequence alignments, the multiple genome sequence alignment process starts. For the first iteration, each processor looks for its best pairwise alignment score. The master processor selects the best score and broadcasts this best pair to all other processors. The list and the used lists are updated accordingly. Then the consecutive sequences are chosen from among the sequences that are in the same row or column of the previous sequence. Every processor that has sequences that are in the same row or column or both of the previous sequences, searches for its best alignment and lets the master processor know. This procedure continues until all the sequences are included in the list.

To implement this, a parallel program has been written using the Message Passing Interface (MPI). *MPI_Reduce* reduces the values of different processors to a final one depending on the desired operation needed. It obtains the global maximum values from the local maximums among the processors comparing the pairwise scores. To reduce the communication overhead, the indices of the sequences compared are also reduced in the same message – the indices are contained in the score of the significance matrix as a negligible weight (kept as decimal points.) This does not affect the reduce instruction unless there is

a tie. When there is a tie, the order is not important so the global maximum is obtained correctly.

The master processor, considered to be the processor with id 0 (*processor0*), controls the global issues. After receiving the local maximums from individual nodes, *processor0* updates the multiple genome alignment list and informs the other processors about the final decision.

3.5. Multiple genome alignment in grid

The multiple genome alignment problem can be a suitable application for the grid environment. The algorithm described divides the workload independently among the processors and requires minimal communication between processors. The grid environment is a distributed environment where the resources may be loosely coupled. Minimal communication among the processing elements is necessary in order to get good performance, especially when there are additional overhead for communication among the grid middleware components.

The grid-enabled application works in the same way as it does in a shared memory or non-grid cluster environment except for the fact that the Globus Toolkit handles the job submission process and communication details. The Message Passing Interface (MPI) version compatible with the grid environment is called MPICH-G2. It enables a user to use the services from Globus Toolkit, like startup and security, to couple machines of different locations and architectures [17]. MPICH-G2 supports multi-protocol communication selecting TCP for communication related to inter-machine messaging and vendor supplied MPI for intra-machine messaging.

4. Experimental results

The multiple sequence alignment algorithm described above distributes a large amount of computation across the processors and requires minimal communication at the end of each iteration. This algorithm was studied in three different distributed environments including a single cluster environment, a single cluster grid environment (that used the same cluster but with the grid software layer), and a multi cluster grid environment that includes multiple grid-enabled clusters.

In the single cluster case, homogenous machines were used: a Beowulf cluster of eight nodes, each having four 550MHz Pentium III processors with 512 MB of RAM. The single cluster grid was the same environment as the single cluster environment except it had the grid software Globus Toolkit 3.0 installed. The multi cluster grid environment included only the head nodes of three individual grid-enabled clusters, each having four 550MHz Pentium III processors with 512 MB of RAM. The program results running under the Grid or the Cluster environment include the communication time used along with the computation time. The genome sequences used were selected from the NCBI website [19] from which a dataset of 150 sequences was used with sequences varying from 5 thousand to 15 thousand in length.

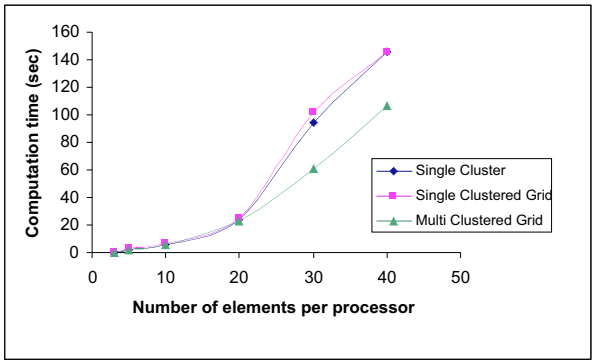

Figure 3. Computation time with varying number of elements per processor in different environments

The experiment in **Figure 3** was run using 9 processors in each environment with computation time measured as varying loads were given to the processors in the different environments. An interesting result is observed: for a larger job, multiple clusters show better performance despite the added communication overhead between the clusters. This result may signify that there is intra-cluster bus communication overhead in a single cluster that is favorably offset when the work is distributed between clusters. It is noted that the multiple clusters were in two different buildings and connected by 100 Mbit/second network.

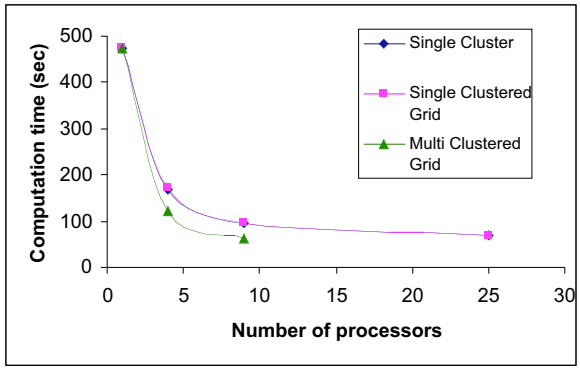

Figure 4. Computation time using different number of processors in different environments

Figure 4 shows the computation time using different numbers of processors in the three different environments. The results indicate that the when the number of processors were increased, the multi cluster

grid environment retained good performance. Note that there were only 9 processors in the multi cluster environment compared with to up to 32 processors on the single cluster and single cluster grid environment. This 9 processor limit in the multi cluster environment was due to the particular cluster configurations studied – other configurations are possible that would remove this limit.

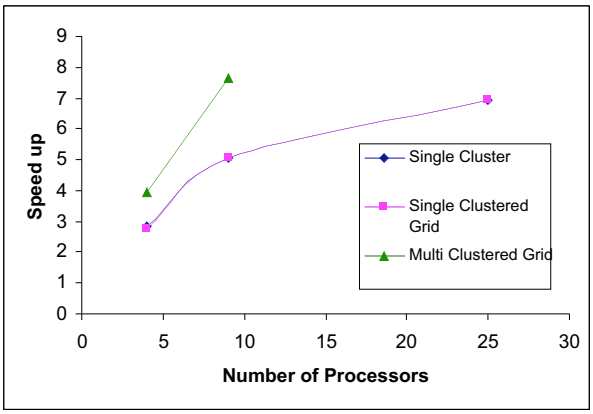

Figure 5. Speed up for different number of processors in different environments

Figure 5 shows the speed up that corresponds to the computation time of Figure 4. Speed-up is measured as the time on a single processor divided by the time on multi processor for the same application. A much better speed up is shown for increasing number of processors in multi cluster grid environment although the numbers of processors used was limited to a total of 9 due to configurations studied.

A possible reason for the results obtained from the comparison of the multi cluster grid environment with the single cluster and the single cluster grid environments may be the architecture of the multi cluster grid. It may be the case that the multi cluster environment provides additional local buses. Therefore if communication is mostly on the local buses, the single cluster environment (with a single bus) incurs delays from contentions. The experiment results point out interesting research ideas for launching computation intensive applications in multi cluster environments, and investigating performance factors such as having suitable high speed network connections among the clusters, or latency issues in distributed algorithm performance.

5. Conclusions

The multiple genome alignment algorithm revealed interesting phenomenon when it was run in different distributed environments. The multi cluster grid environment shows better performance compared to the single cluster and single cluster grid environment. From this result it appears that the grid environment may play an important role for computation intensive applications while still supporting all the protocols (and overheads) for communication. The environment is further useful for its scalability – resources can be added according to the need of the application without the intervention of the user. Where the memory space is limited in a shared memory environment and the scalability is therefore constrained, the grid environment can provide both memory and scalability, and do so by taking care of lower level details.

Acknowledgment: *This material is based in part upon work supported by the National Science Foundation Middleware Initiative Cooperative Agreement No. ANI-0123937. Any opinions, findings, conclusions or recommendations expressed herein are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.*

Special thanks to University of Alabama at Birmingham's Jill Gemmill, Zach Garner, Joshi Pravin, John-Paul Robinson and others who provided machine environments for testing and assistance with using them.

10. References

[1] Alphey, L., "*DNA Sequencing from experimental methods to bioinformatics,*" BIOS Scientific Publishers, 1997.

[2] Ahmed, N., Y. Pan and A. Vandenberg, "*Memory Efficient Pair-wise Genome Alignment Algorithm – A Small-Scale Application with Grid Potential*," GCC 2004, LNCS 3251, H. Jin, Y. Pan, N. Xiao, and J. Sun (Eds.), Springer-Verlag Berlin Heidelberg, pp. 777–782, 2004.

[3] Foster, I., N. T. Karonis, C. Kesselman and S. Tuecke, "*Managing Security in High-Performance Distributed Computing,"* Cluster Computing*, 1(1):95-107, 1998.*

[4] Foster, I., and C. Kesselman, "*The Grid: Blueprint for a New Computing Infrastructure,*" Elsevier, 2004.

[5] Foster, I., C. Kesselman, J. M. Nick and S. Tuecke, "*The physiology of grid – An Open Grid Services Architecture for Distributed Systems Integration,*" Open Grid Service Infrastructure WG, Global Grid Forum, June22, 2002.

[6] Foster, I., C. Kesselman and S. Tuecke, "*The anatomy of Grid: Enabling Scalable Virtual Organizations,"* International J. Supercomputer Applications*,* 15(3), 2001.

[7] Gibas, C. and P. Jambeck, *"Bioinformatics Computer Skills,"* O'REILLY, 2001.

[8] Karonis, N., B. Toonen and I. Foster," *MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface,*" Proceedings of the IEEE/ACM SC2000 Conference*, November 4-10, 2000.*

[9] Martins, W.S., J. del Cuvillo, W. Cui and G. Gao, "*Whole Genome Alignment using a Multithreaded Parallel Implementation,"* Symposium on Computer Architecture and High Performance Computing, Pages 1- 8, Pirenopolis, Brazil, September 10-12, 2001.

[10] Needleman, S. B. and C. D. Wunsch, *"A general method applicable to the search for similarities in the amino acid sequence of two proteins,"* Journal of Molecular Biology, 48: 443-453, 1970.

[11] Sankoff, D., "*The early introduction of dynamic programming into computational biology*," Bioinformatics, vol. 16 no 1, 41-47, 2000.

[12] Smith, T. F. and M. S. Waterman, "*Identification of common molecular subsequences,"* Journal of Molecular Biology, 147(1): 195-197, 1981.

[13] Taylor, W. R., "*Multiple sequence alignment by a pair wise alignment,*" Comp. App. Bio. Sci. 3, 1858-1870,1987.

[14] Taylor, W. R., "*Multiple sequence threading: an analysis of alignment quality and stability,"* J. Molec. Biol. 269, 902-943, 1997.

[15] Taylor, W. R. and C.A. Orengo, "*Protein Structure Alignment,*" J. Mol. Biol. 208: 1-22, 1989.

[16] Taylor, W. R., G. Salensminde and I. Eidhammer, "*Multiple protein sequence alignment using double-dynamic programming,*" Comput Chem., 24(1):3-12, 2000.

[17] The official website for MPICH_G2, http://www.hpclab.niu.edu/mpi/

[18] *"Tools for Multiple Sequence Alignment,*" Link: http://pbil.univ-lyon1.fr/alignment.html

[19] The NCBI homepage http://www.ncbi.nlm.nih.gov/