Aligning Multiple Biosequences Progressively on Cluster Grids using ALiCE

Yong Meng TEO, Yew Kwong NG and Kuo-Bin LI[†]
Department of Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543
Email: teoym@comp.nus.edu.sg

Abstract

In the absence of powerful supercomputer hardware, grid computing offers an alternative avenue by providing a heterogeneous, scalable and reliable high performance processing environment to address problems involving large computational granularities and enormous datasets. The physical and life sciences typically include numerous classes of sophisticated problems and the retrieval of information from large volumes of formatted databases. This paper reports the development and deployment of a bioinformatics problem, Progressive Multiple Sequence Alignment (PMSA), on cluster grids using ALiCE, a grid computing middleware. PMSA comprises of three consecutive stages: pairwise sequence comparison, guide tree construction and sequence profiles alignment. Our implementation of PMSA involves parallelizing the first and third stages of the algorithm. Experiments on homogeneous and heterogeneous cluster grids demonstrate how performance scales with problem size and computational power, illustrating that grid computing is a feasible means to approach several categories of problems in the life sciences by integrating pooled resources to produce supercomputing capabilities.

1. Introduction

The popularity and scalability of the Internet has revolutionized traditional computing concepts and practices. With the availability of powerful computers and high-speed networking technologies as low-cost commodity components, it is possible to cluster or couple a wide variety of heterogeneous resources including supercomputers, storage systems, data sources and special classes of devices distributed geographically and use them as a single, unified resource [3]. This methodology is known as *grid computing*, and it enables the development of virtual organizations [13] sharing core competencies, resources and skills, to respond to business opportunities and large-scale application processing requirements more

[†] Bioinformatics Institute, 30 Medical Drive, Level 1, IMCB Building, Singapore 117690

effectively [1]. A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [10]. The pioneer grid efforts spun off as experimental attempts to link supercomputing sites in the US, and grid computing has since then evolved to adopt a service-oriented [12] approach, emphasizing on capability rather than connectivity.

Research in the physical and life sciences characteristically involves the manipulation of astronomical volumes of raw scientific data produced in experiments. In the biological and biomedical fields, the data takes the form of biosequences flat files, three-dimensional structures, motifs and three-dimensional microscopic image files [30]. However, while genome projects and DNA arrays technology are constantly and exponentially increasing the amount of data available, the capability of a sequential computer can allow only a small portion of the huge dataset to be processed in a reasonable response time. Furthermore, several well-known bioinformatics problems such as protein folding simulation [28] and phylogenetic tree construction [22] encompass numerous sophisticated algorithms that may not be readily handled by conventional computers. The use of supercomputer systems may not be economically viable. Therefore, the opportunities provided by grid computing offers an avenue to address computational problems in the sciences by leveraging available and affordable resources to create a virtual scientific environment.

Grid development from scratch is typically tedious, involving numerous issues such as resource management and scheduling, efficient and reliable data transfer technologies and security in a distributed environment. An increasing number of research groups worldwide have implemented middleware, libraries and tools [5] to facilitate the construction of grids. The Globus [11] project is a well-established example of a metacomputing abstract machine, setting out to provide a collection of fundamental low-level services that can be used to develop and integrate higher-level services and capabilities. The key Globus components have been successfully deployed in the I-WAY [6] experiment.

ALiCE [27] is a platform independent, Java-based user-oriented Grid computing middleware developed by the School of Computing at the National University of Singapore. It comprises of three types of entities: *resource broker*, *producers* and *consumers*. The resource broker is a logical point of control for all ALiCE grid resources and applications running on these resources. It provides system services such as resource management and monitoring. Through the producer client, user gives access to share their compute resources to execute ALiCE tasks. Submission of ALiCE applications is through the consumer client. The

underlying communications mechanism between the entities is implemented on Sun Microsystems' Jini and JavaSpaces [15] service, which adopts a distributed-shared memory programming model [16]. ALiCE also offers an alternative communications protocol using GigaSpaces™ [14], an industrial implementation of JavaSpaces. Figure 1 shows the ALiCE three-layered architecture.

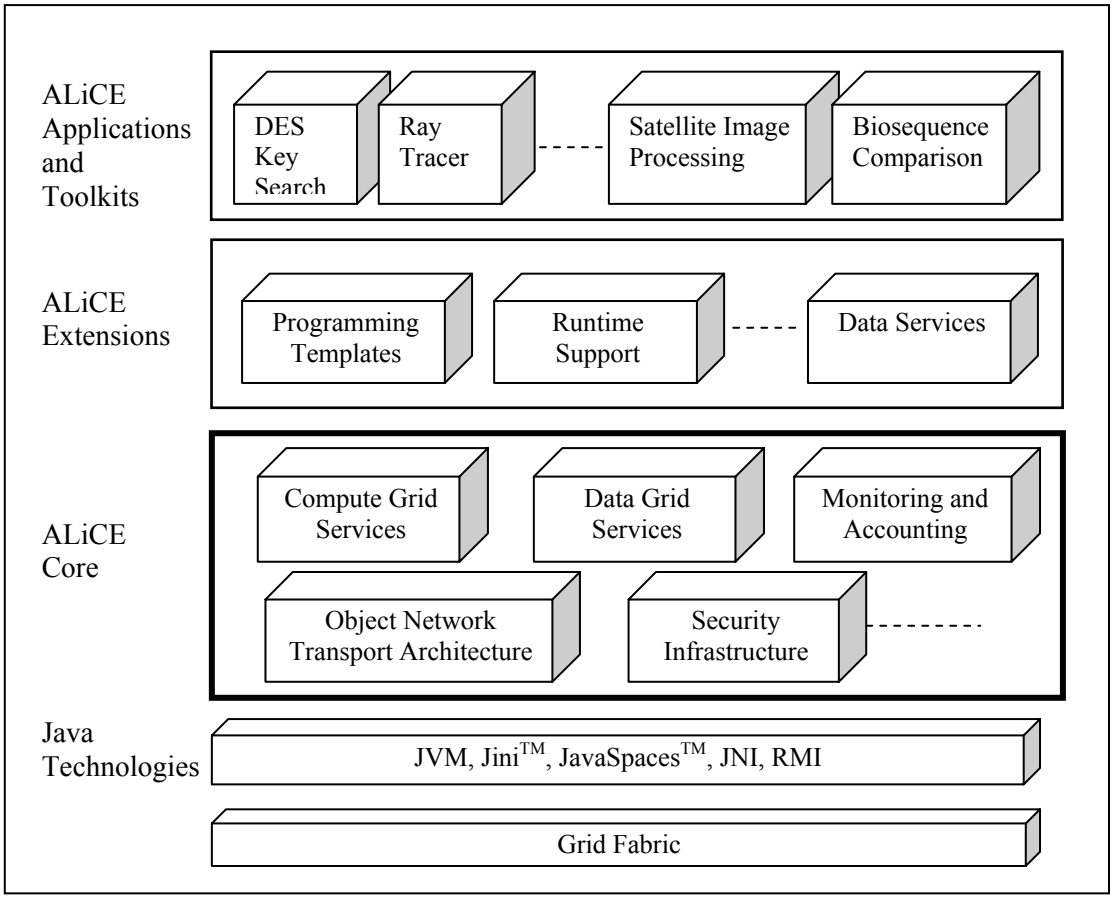

Figure 1: ALiCE Three-Layered Architecture

The remainder of this paper discusses the development and deployment of the PMSA problem on the ALiCE grid. Section 2 presents an overview of PMSA. Section 3 describes the parallelization of the PMSA problem and its mapping onto ALiCE. Section 4 discusses the performance results on two cluster grids of different natures. Section 5 presents our concluding remarks.

2. PMSA Overview

The comparison of a pair of protein or DNA sequences is the most fundamental problem in bioinformatics [25]. Frequently, scientists are interested in studying the existence of homologies between tens, or hundreds, of sequences, and not just between one specific gene

and a collection of existing genes [8]. The simultaneous alignment of multiple sequences is now an essential methodology in molecular biology. PMSA is often used to find diagnostic patterns to characterise protein families, to detect or demonstrate homology between new sequences and existing families of sequences, to help predict the secondary and tertiary structures of newly sequenced genes, and to suggest special primers for Polymerase Chain Reaction (PCR) [29]. Evolutionary scientists adopt PMSA techniques to conduct evolutionary analysis on newly discovered species of living organisms.

The PMSA problem has been addressed in a commercial biomedical research toolkit, CLUSTALW [29]. PMSA generally comprises of three successive stages, which will be discussed in the subsequent subsections. For the rest of this paper, we assume the following notations:

s, t	: arbitrary database sequences
i	: ith residue of s
j	: jth residue of t
D	: set of database sequences to be aligned
n	: number of sequences in D

2.1 Stage 1: Pairwise Comparison

This is the preparatory stage of PMSA, and its chief objective is to identify the *similarity* between every pair of biosequences (s, t) to be aligned, so that sequences that are more closely related aesthetically can first be aligned together. With this information, we can determine the order in which alignments are to take place later. The similarity between s and t can be mathematically computed with the function shown in Figure 2.

$$sim(s[1..i], t[1..j]) = \max \begin{cases} sim(s[1..i], t[1..j\text{-}1]) - 2 \\ sim(s[1..i\text{-}1], t[1..j\text{-}1]) + p(i, j) \\ sim(s[1..i\text{-}1], t[1..j]) - 2 \end{cases}$$

Figure 2: Similarity Score Function

This similarity score function basically obtains the similarity score of the sequence pair using a dynamic programming mechanism developed by Smith and Waterman [18] to fill up a *substitution matrix* [25]. $p(i, j)$ is an alignment column score function that returns the maximum value when $i = j$, the minimum value when $i \neq j$, and a given gap penalty value when either i or j is a gap in the corresponding sequence. Similarity scores can alternatively be derived using some other more complicated techniques, such as the Linear Space similarity algorithm [20], which consumes less memory, and the BLAST [2] algorithm.

The reciprocals of the corresponding similarity scores for all sequence pairs are used to fill up a *distance matrix*. Figure 3 shows an example of such a matrix describing the pairwise distances of five globin sequences. It suggests that the species `Hbb_Human` is likely to be most closely related to `Hbb_Horse`, and that `Hba_Human` has a significant level of similarity with `Hba_Horse`, because of their respective minimum distance values.

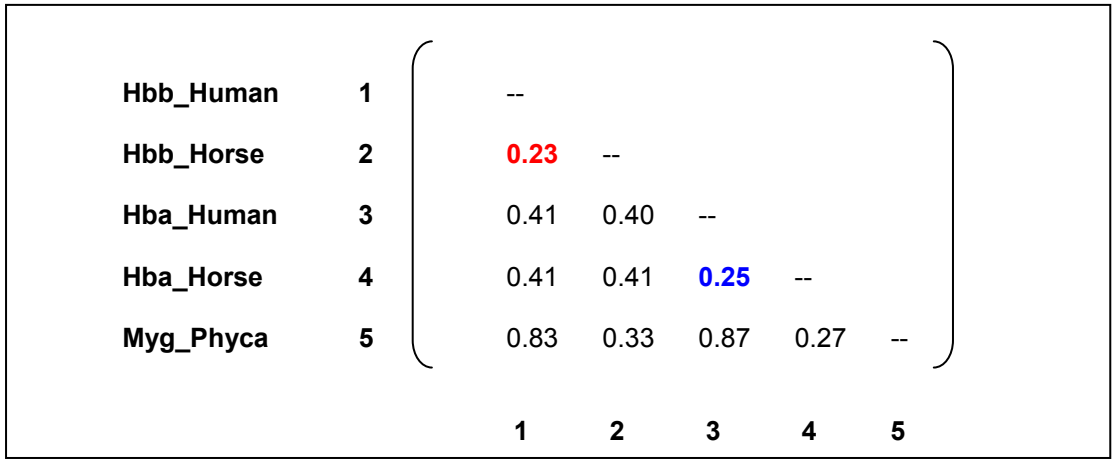

Figure 3: Distance Matrix for Five Globin Sequences

2.2 Stage 2: Guide Tree Construction

The distance matrix produced during pairwise comparison is used to build a phylogenetic tree mapping out the alignment order of the sequences. The tree can be produced efficiently with distance matrix algorithms such as *Unweighted Pair Group Method using Arithmetic Mean* (UPGMA) [26], the FITCH Least-Squares Distance method [9], the Wagner method [7], the neighbour-joining method [24] and the minimum evolution method [21, 23].

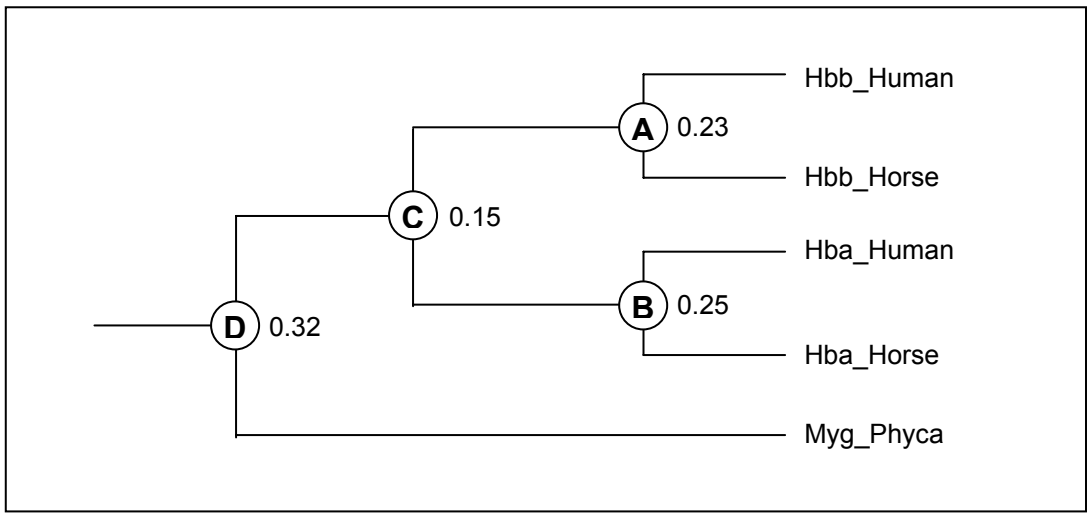

Figure 4: Guide Tree Produced from Distance Matrix

The general idea of these tree building techniques is to compute the minimum distance value in the matrix, group the relevant sequence pair (*s*, *t*) to form a new taxonomical unit, and repeat this process with the newly formed taxonomical unit and the remaining sequences represented in the matrix. The taxonomical units produced in the grouping process correspond to the internal nodes in the tree. Complex algorithms such as the Wagner method and the neighbour-joining method involve the use of statistical and weight functions to compute the tree. Figure 4 illustrates the guide tree constructed for the matrix in Figure 3.

2.3 Stage 3: Profiles Alignment

The goal of this stage is to produce the complete multiple alignment of the sequences, with the help of the guide tree manifested in the previous stage. In general, alignment begins, pairwise, from the leaves and progresses towards the root of the tree. In the process, *alignment profiles* will be formed, which corresponds to the internal nodes of the tree.

Let us consider the example in Figure 4 above. According to the pattern of the guide tree, the alignment of `Hbb_Human` and `Hbb_Horse` to form alignment profile *A* can proceed immediately. Likewise, `Hba_Human` and `Hba_Horse` can be aligned to form profile *B* at once. The order in which these two alignments are carried out does not matter, since they are mutually independent alignments. Next, profiles *A* and *B* will be aligned together to form profile *C*. Finally, *C* will be aligned with the sequence representing the species `Myg_Phyca` to produce *D* - the complete multiple alignment of the five sequences.

The major problem with PMSA stems from the assumption that the sequences/profiles align well at every node in the guide tree. However, if the initial alignments yield low similarity score values, then the anormalies will be disseminated throughout the remaining alignments. The common approach adopted is to skip an alignment if it results in a similarity score value that is below a specified threshold, and carry it out later.

Figure 5 outlines the entire PMSA algorithm for a given set of database sequences *D*. Lines 3-5 performs pairwise comparison (PC) over all sequences in *D*. Lines 6-16 constructs the guide tree (GT) from the distance matrix, *M*, thus produced. Lines 17-24 perform progressive profiles alignment (PA) depthwise bottom-up in the tree. The complete multiple alignments of all the sequences in *D* is returned at the end of the algorithm.

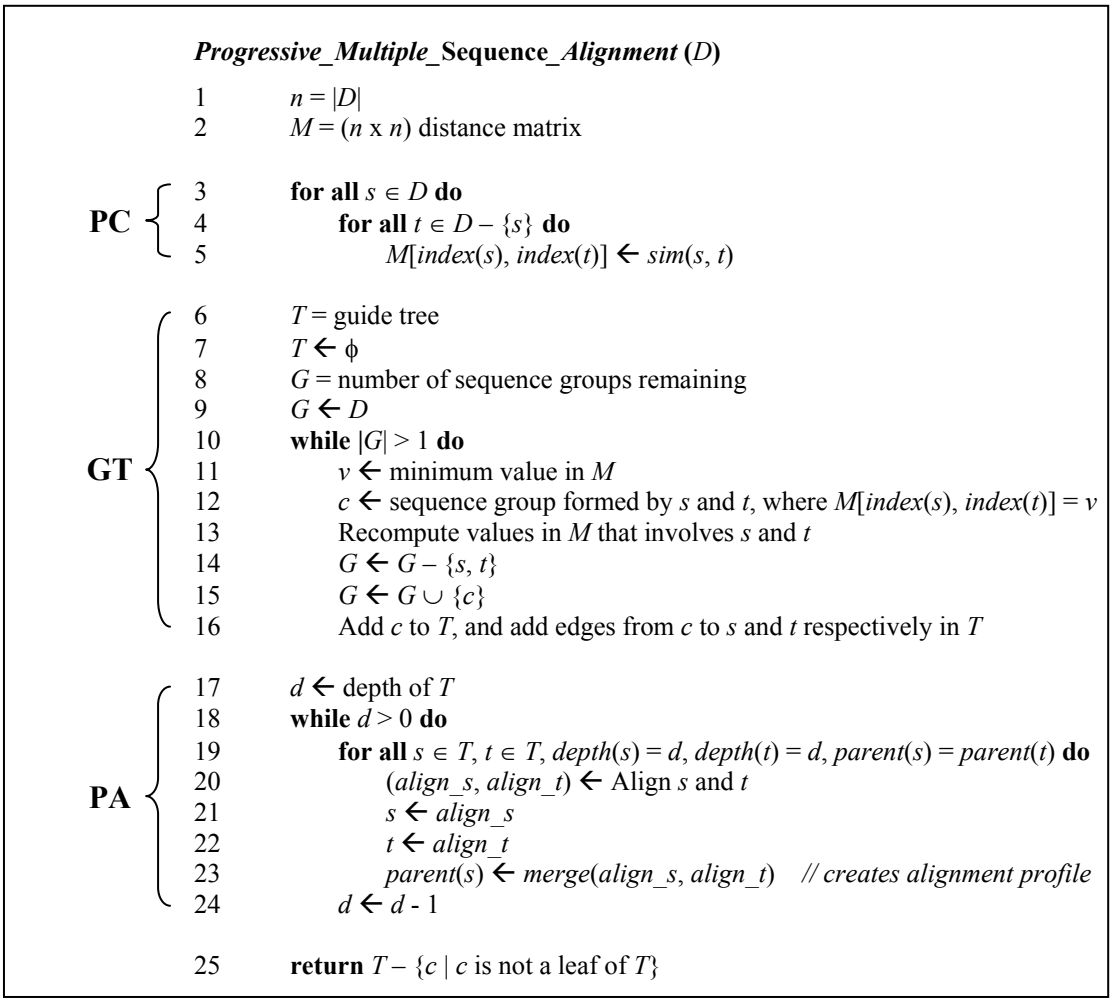

Figure 5: PMSA Algorithm

With reference to the algorithm, the double loop in PC yields a time complexity of $O(n^2)$, but overall, the PC stage incurs a complexity of at least $O(n^3)$ since the best performing similarity algorithm is known to have a linear time complexity. It can even be $O(n^4)$ or higher, depending on the complexity of $sim(s, t)$. GT involves searching for the minimum value in the distance matrix prior to each grouping of sequence pairs, and thus yields a total complexity of $O(n^3)$. PA incurs a linear time complexity of $O(n)$, since there are $(n - 1)$ internal nodes in the guide tree and therefore, $(n - 1)$ profiles alignment. As illustrated, the complexity of the whole PMSA algorithm is bounded by the PC stage, which has a minimum possible complexity of $O(n^3)$. Under such circumstances, the overall execution time of a sequential PMSA program will increase exponentially with an increase in the number of database sequences. This is undesirable for real-time lab research scenarios, which may involve hundreds or even thousands of sequenced proteins.

3. Distributed PMSA using ALiCE Grid

SGI has attempted to parallelize the CLUSTALW algorithm in an associated application, known as Parallel CLUSTALW [17]. However, Parallel CLUSTALW is problem-specific and is unable to handle grid environmental features, such as task scheduling and sequence data retrieval from remote databases. In a typical scenario, a molecular biologist may wish to align numerous genes that are sequenced in laboratories worldwide and thus stored in geographically distributed databases. This will be a hassle in the absence of an appropriate data handling engine. Here, we describe the parallelization of PMSA using ALiCE, thereby enabling the ease of development and deployment of our application on the grid.

3.1 Task Partitioning

ALiCE applications are written using the ALiCE object-based programming template [27] that hides the specific details of the distributed implementation and dynamic code linking mechanisms required for execution on the grid. ALiCE API consists of three Java classes: *TaskGenerator* generates, initializes and submit computational tasks, *Task* implements the Task objects of the application, and *ResultCollector* implements methods for collecting the application's results. The Java-based template classes can be extended to cater to the development of different applications, thereby exploiting task-level parallelism and greatly reducing development time.

PMSA involves a *semi-regular* [30] computation pattern, and we can parallelize the pairwise comparison and profiles alignment stages separately. Guide tree construction involves little independent computations, and the only parallelizable portion is the search for the minimum value in the distance matrix before every grouping of sequence pairs. The granularities of these search tasks are too small for parallelization to be feasible. Pairwise comparison can be parallelized in a straightforward manner since each comparison is independent of others. However, the alignments of profiles are not mutually independent tasks because some alignments cannot proceed until the results of other alignments deeper in the guide tree are obtained.

Given n sequences, there will be a total of $n(n - 1)/2$ pairwise comparisons to be made. Hence, suppose $n = 500$, partitioning the comparisons uniformly into independent tasks, where each task handles, say, 60 comparisons, there will be a total of $\lceil 500(500 - 1)/(2 \times 60) \rceil = 2080$ tasks to be executed in this stage. Assuming that the guide tree built for these 500 sequences is well-balanced, then the tree depth will be $\lceil \log_2 500 \rceil = 9$. The alignments of sequences or

profiles at a given depth are mutually independent, and they can be done simultaneously. Therefore, all the sequence pairs at the lowest depth (i.e. the leaves) can be aligned in parallel, followed by all the profile pairs at depth = 8, 7, 6, and so on, in successive sweeps. Figure 6 illustrates this parallelization concept with a portion of a guide tree. The *k*th sweep comprises of eight parallel tasks, the (*k*+1)th sweep has four tasks and the (*k*+2)th sweep involves two tasks.

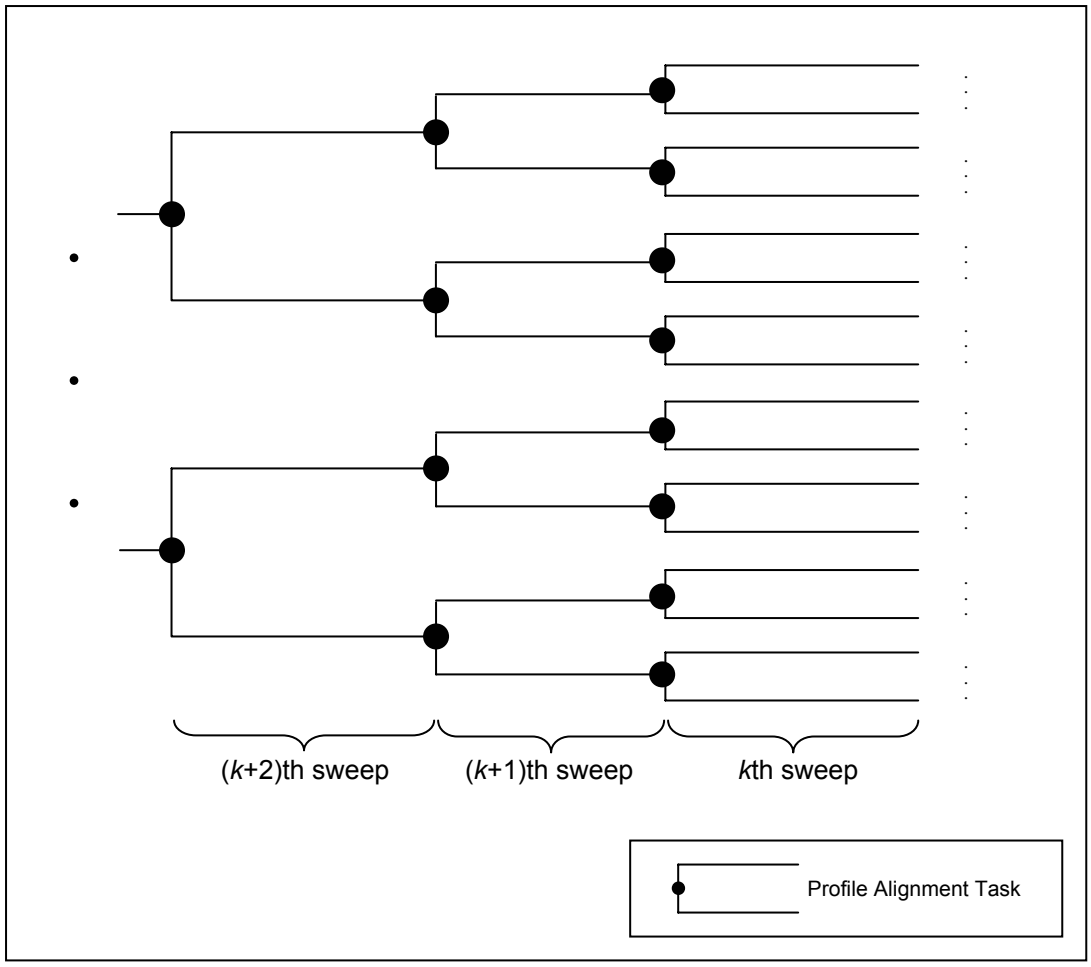

Figure 6: Grid-based Parallelization of Profiles Alignment Stage

The flow diagram in Figure 7 highlights the PMSA procedure using ALiCE, with respect to the example above.

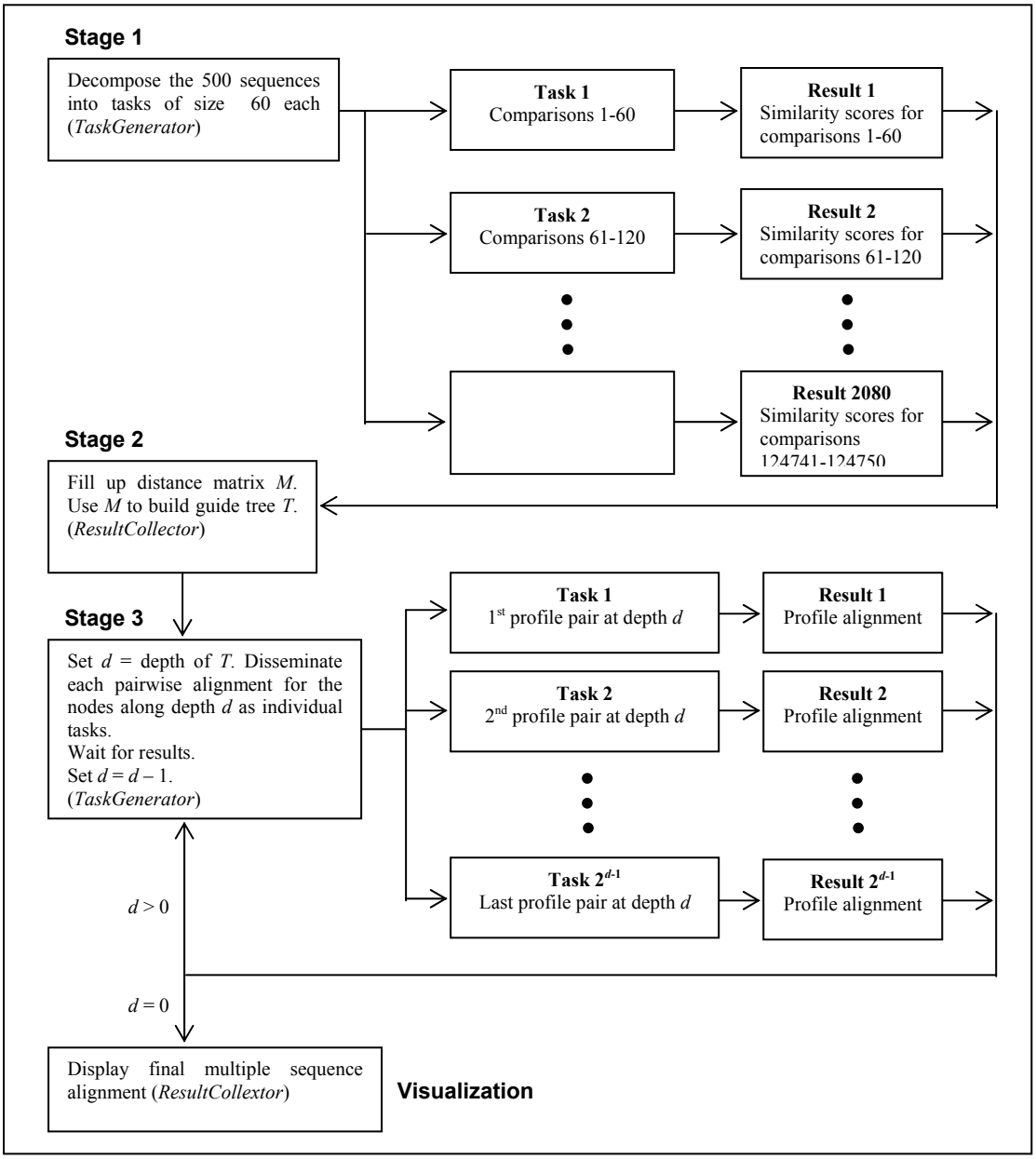

Figure 7: Grid-based PMSA using ALiCE

The task granularity in the pairwise comparison stage can affect the communications-to-computations ratio [4] of the distributed execution, and in order to achieve respectable performance, there is a need to determine a balance between *task size* and *number of tasks* for a given amount of processing power and number of ALiCE producers. Another performance factor is the shape of the guide tree, which inadvertently determines the number of alignments at each depth, and hence, the average *sweep size* (i.e. the number of parallel tasks generated at each depth of the tree). If the tree is heavily skewed, then the sweep sizes at all depths will be small, and this results in load imbalance throughout the profiles alignment stage. In general,

however, skewed trees are seldom encountered if the sequences to be aligned are somewhat related.

3.2 PMSA Implementation in ALiCE

Figure 8 presents the main Java classes of the ALiCE distributed PMSA application.

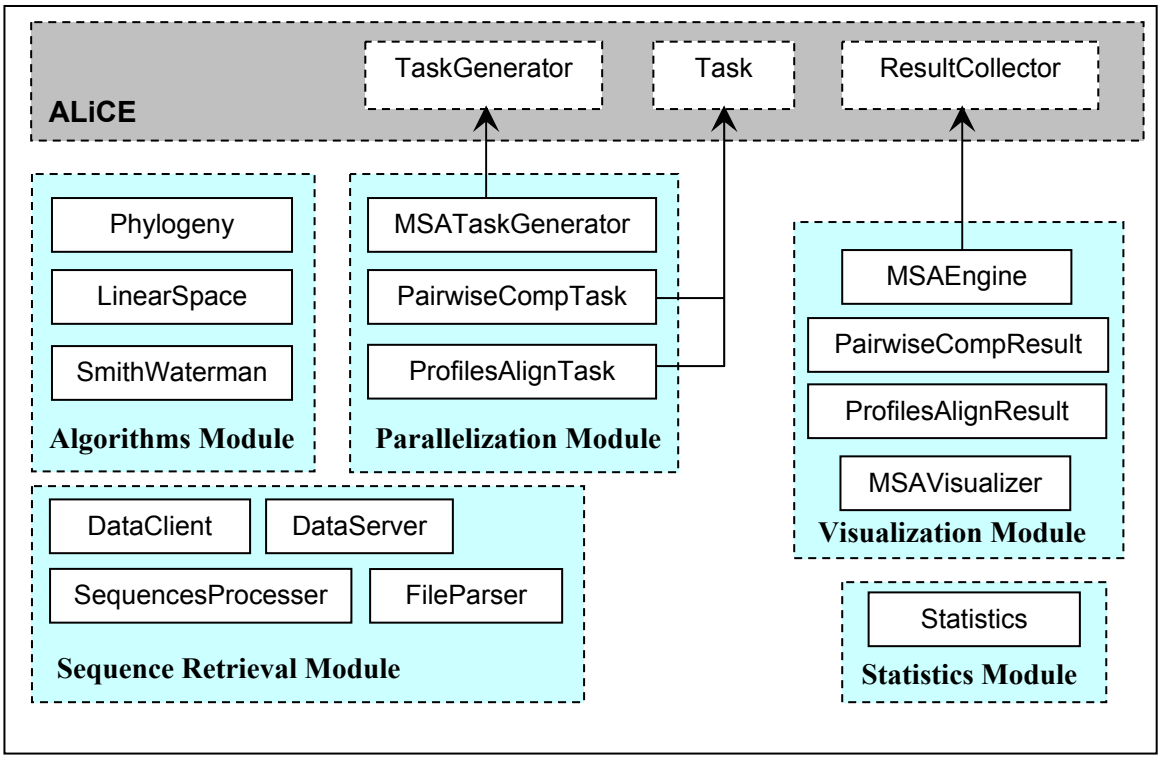

Figure 8: Class Components of Distributed PMSA Using ALiCE

The application is developed using the ALiCE programming template and is composed of five modules. Table 1 summarizes the functionalities of each of the modules.

Module	Functionality
Algorithms	Encompasses the bioinformatics algorithms used in PMSA, including the similarity computation mechanisms, alignment techniques and phylogenetic tree-building methods.
Parallelization	Handles the distribution and execution of the tasks in the pairwise comparison and profiles alignment stages.
Sequence Retrieval	Retrieves the necessary sequences for computation. If the required sequences are located on a remote machine, they will be fetched via a TCP connection with the data server at the remote computer.
Statistics	Compiles meta-information, such as the sequence count in the databases, mean sequence length, task processing times, and the overall response time of a given run of the application.
Visualization	Collects and presents the execution results to the user via a GUI.

Table 1: Functionalities of the ALiCE PMSA Modules

Figure 9 presents the skeletal outline of the PMSA application's main Java classes that are associated with the ALiCE programming template.

PMSA Task Generator

```
import alice.consumer.*;
// other miscellaneous import statements.

public class MSATaskGenerator extends TaskGenerator {

   /* ALiCE methods: */

   public void init() {
      // Create the statistical module to keep track of task processing time.
   }

   public void main(String args[]) {
      // Loop:
            // Call GetUserInputs().
            // Call MSAKernel().
   }

   /* Application-specific methods: */

   private void GetUserInputs() {
      // Obtain the user-specified PMSA parameters from the result collector.
   }

   private void MSAKernel() {
      // Generate and disseminate tasks for pairwise comparison stage.
      // Wait for call to start profiles alignment stage.
      // Generate and disseminate tasks for profiles alignment stage.
  }
}
```

PMSA ResultCollector

```
import alice.result.*;
// other miscellaneous import statements.

public class MSAEngine extends ResultCollector {

  /* ALiCE methods: */

   public void collect() {
      // Setup the result collector.
      // Loop:
            // Send the runtime parameters to the task generator.
            // Call CollectPairwiseComparisonResults().
            // Call BuildGuideTree().
            // Call CollectProfilesAlignmentResults().
            // Call DisplayResults().
   }

   /* Application-specific methods: */

   private void CollectPairwiseComparisonResults() {
      // Loop:
            // Call ALiCE primitive collectResult() to collect pairwise comparison results.
            // Exit if there are no more pairwise comparison results.
   }

   private void BuildGuideTree() {
      // Constructs the guide tree to map out the order of profiles alignment.
      // Uses either of UPGMA, neighbour-joining, or minimum evolution methods.
   }

   private void CollectProfilesAlignmentResults() {
      // Loop:
            // For each depth of the guide tree,
            // call ALiCE primitive collectResult() to collect profiles alignment results.
            // Exit when the tree root is reached.
   }

   private void DisplayResults() {
      // Displays the PMSA results for the current run on the visualizer.
   }
}
```

Pairwise Comparison / Profiles Alignment Task

```
import alice.consumer.*;
// other miscellaneous import statements.

public class PairwiseCompTask / ProfilesAlignTask extends Task {

  public PairwiseCompTask / ProfilesAlignTask(….) {
  }

  // ALiCE task execution routine.
  public Object execute () {
     // If this is a PairwiseCompTask task, then fetch the required database
     // sequences from the relevant machine and perform similarity computation.
     // Otherwise, if this is a ProfilesAlignTask, then align the profile pair
     // assigned to this task.
  }
}
```

Pairwise Comparison / Profiles Alignment Result

```
import java.io.*;

public class PairwiseCompResult / ProfilesAlignResult implements Serializable {
   // data structures to store the results.

   public PairwiseCompResult / ProfilesAlignResult(….) {
      // create the result object with the respective arguments. This updates the data
      // structures.
   }
}
```

Figure 9: PMSA Code Skeleton Using ALiCE Programming Template

Figure 10 shows the user interface and visualization during the alignment of 700 sequences of DNA extracted from various mammal species. The interface on the left provides statistical information about these sequences, and enables the user to specify the desired task size for the alignment. The one on the right displays the intermediate results obtained in each stage and the complete multiple alignments of the DNA sequences.

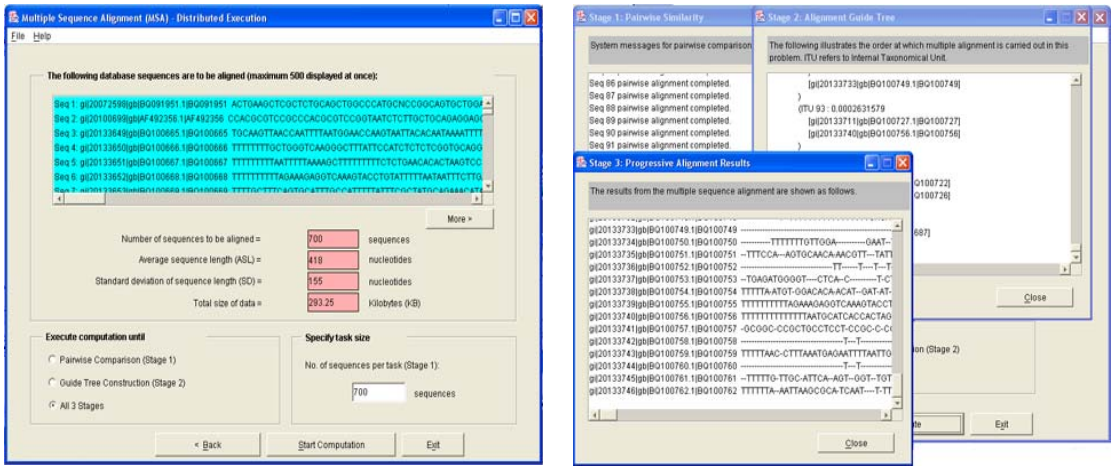

Figure 10: Multiple Alignments of 700 DNA Sequences

4. Performance Evaluation

We study performance in two approaches, using the ALiCE PMSA application that we have developed. Firstly, we compare with ClustalW and investigate how the application execution time scales with the number of sequences involved in the multiple alignments. Then, given a fixed problem type and size, we determine the effect of varying task size on overhead. Finally, we proceed to analyze the effect of varying the number of producers on execution time.

The test environment comprises of a *homogeneous cluster grid* and a *heterogeneous cluster grid* with all nodes running RedHat Linux OS. The 64-node homogeneous cluster grid (*Grid I*) comprises of dual processors Intel Xeon 1.4GHz processors with 1GB of memory, connected by a Myrinet network. The 50-node heterogeneous cluster grid (*Grid II*) consists of nodes connected in two network segments and spread over two physical buildings. The first segment composes of twenty-six dual processor nodes Intel Xeon 1.4GHz processors with 1GB of memory connected by a Myrinet network. The second segment is made up of sixteen nodes Pentium II 400MHz with 256MB of RAM, and eight nodes Pentium III 866MHz with 256MB of RAM, all connected via a 100Mbps Ethernet switch. The two segments communicate via a fast Ethernet LAN. Both grids run GigaSpaces for ALiCE. A 10MB FASTA [19] database composed of 25,000 GPCR (*G-Protein Coupled Receptor*) [31] protein sequences provides the dataset used in the experiments. The execution times reported in this section are averages for *three* replications. All experiments involving distributed computations on Grid II are conducted with a fair mix of producers from the two segments, even for the cases where small numbers of producers are involved.

4.1 Problem Scalability

We study the scalability of the problem to obtain an idea of the execution times required for different problem sizes on a uniprocessor, and to enable us to select a reasonable problem size for the second part of our experiments. Here, we carry out sequential executions on partitions of the GPCR protein database using the CLUSTALW software and a sequential PMSA that we implemented in Java. These sequential runs are conducted on one Intel Xeon 1.4GHz node selected from our test environment, and the results are presented in Table 2. The breakdown of the execution times for the Java sequential PMSA application, for the three stages of the algorithm, is shown in

Figure 11. The numbers above each bar in the chart refers to the corresponding total execution times.

n	Size of *D* (MB)	Mean Seq Length (bytes)	Execution Time (hours)	
			ClustalW	Java Sequential PMSA
500	0.2	390 ± 14	2.4	2.2
1,000	0.4	385 ± 21	17.3	16.9
2,000	0.8	389 ± 18	40.1	38.2
4,000	1.5	384 ± 23	103.6	95.3
8,000	3.2	397 ± 29	267.9	237.6

Table 2: Sequential Performance – Varying Problem Size

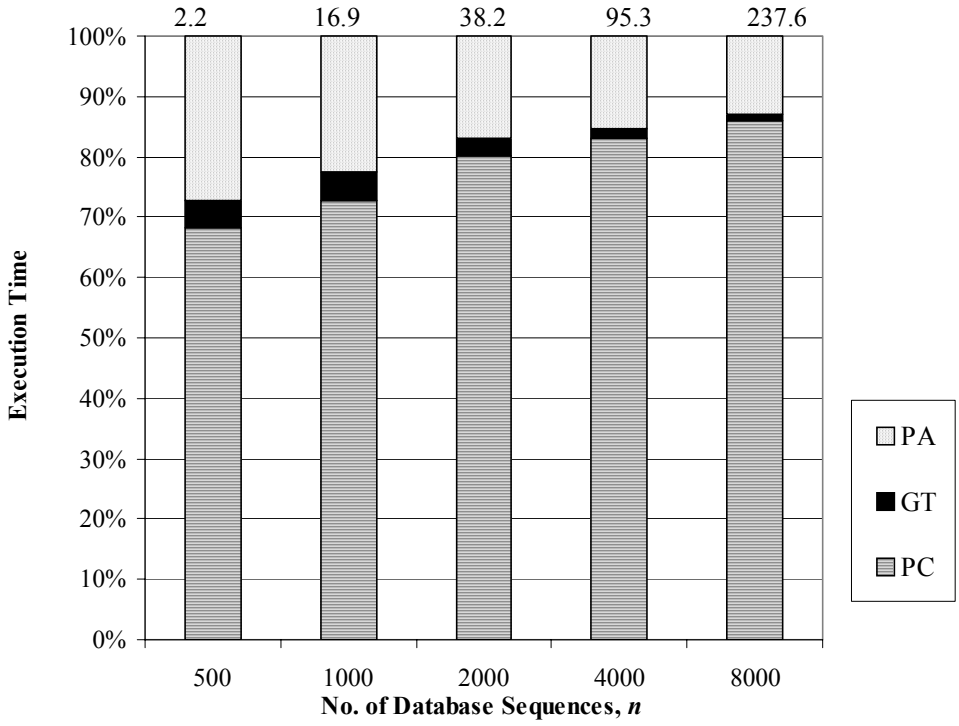

Figure 11: Sequential Performance for the Three PMSA Stages

The results indicate that doubling n (number of sequences) causes between two- to three-fold increased in the sequential execution times for both implementations where n is reasonably large. For the case where n = 8,000, it takes around 10 to 11 days to complete the multiple alignment.

4.2 Effect of Task Size Variations

We now investigate how the performance of distributed PMSA is affected by task size, using the GPCR protein dataset with n = 8,000. ALiCE is setup with *eager scheduling* of tasks, and the dataset is centralized on an NFS. We vary the task size for the pairwise comparison stage only, because the tasks generated in the profiles alignment stage are not all mutually independent. Table 3 presents the execution times for eight and sixteen producer nodes respectively.

Pairwise Comparison Task Size			#tasks $= \left[\frac{n(n-1)}{2(\#seqs)}\right]$	Execution Time (hrs)			
				Grid I		Grid II	
#seqs	(MB)	Time/task (mins)		8 prod	16 prod	8 prod	16 prod
1,000	0.4	0.5	31,996	162.5	113.9	179.8	127.0
2,000	0.8	0.9	15,998	117.8	80.9	126.2	87.3
4,000	1.6	1.8	7,999	71.4	47.9	72.9	49.1
8,000	3.2	3.6	4,000	73.9	48.6	76.6	53.6
16,000	6.4	7.1	2,000	79.0	50.3	82.4	56.9

Table 3: Varying Task Size

As reported, the task computational time is relatively short for smaller task sizes, but the corresponding total execution time is long. This is because of the greater number of tasks that will be executed in the case of small task sizes, resulting in greater communication overhead overwhelming the grid. From the above results, the overall performance is optimal when the pairwise comparison task size is 4,000 sequences, because of a fair balance between communication and computation overheads.

4.3 Computational Scalability

We now study how the application execution time scales with the number of computational producers used. We run the experiments for up to 48 producer nodes, in the case of both grids, with a pairwise comparison task size of 4,000 sequences. The results are presented in Table 4, where PC, GT and PA record the execution times for each of the three PMSA stages respectively.

No. of Producers	Execution Time (hrs)							
	Grid I				Grid II			
	PC	GT	PA	Total	PC	GT	PA	Total
2	148.1	2.6	23.5	174.2	151.9	2.6	25.1	179.6
4	98.6	2.6	19.7	120.9	106.1	2.6	22.8	131.5
8	52.2	2.6	16.6	71.4	61.1	2.6	19.5	83.2
16	32.4	2.6	12.9	47.9	35.6	2.6	17.1	55.3
32	17.8	2.6	9.8	30.2	22.1	2.6	13.8	38.5
48	10.2	2.6	8.0	20.8	14.7	2.6	11.7	29.0

Table 4: Performance on the Cluster Grids

Figure 12 illustrates how the overall execution time scales with the number of producers for both cluster grids.

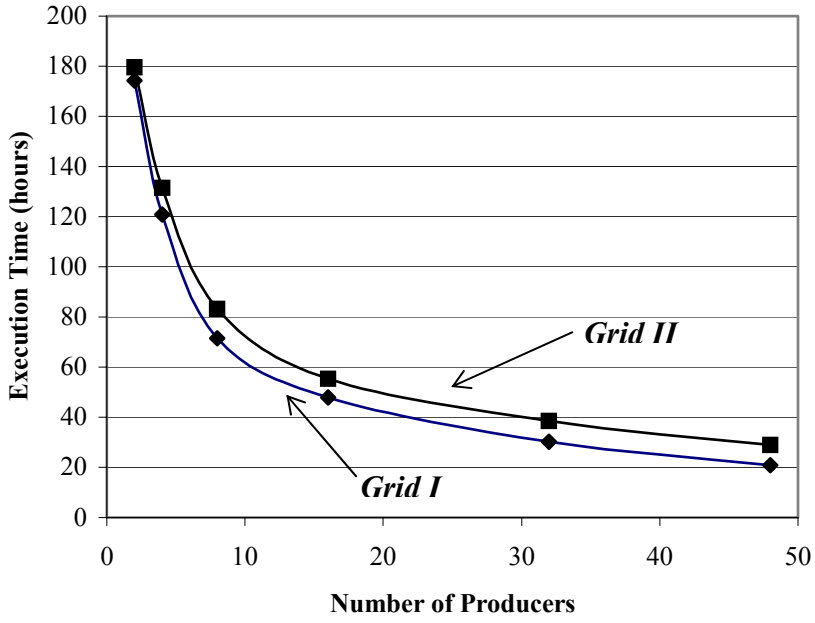

Figure 12: Computational Scalability

Grid I generally performs better than Grid II, and the difference between the performances of both grids becomes more significant as the number of producers used increases. There are two major reasons for this behavior. Firstly, Grid II comprises of a mixture of three different types of processor platforms, the second segment of the grid consisting of resources that are less powerful than the ones in the first segment (Intel Xeon 1.4GHz). Therefore, tasks that are being assigned to execute on the producers in the second segment will take a longer time to complete. Conversely, Grid I comprises of all Intel Xeon 1.4GHz nodes, and therefore is able to derive greater computational power and better performance. Secondly, Grid I is connected solely by a Myrinet switch, which is fast and powerful. Conversely, Grid II exists on a LAN consisting of a combination of Myrinet and Ethernet connections, resulting in a lower

bandwidth. Furthermore, the two cluster segments of Grid II are physically located apart from each other, and this means a longer transmission time for tasks, data and computation results between the consumer, resource broker and producers.

We also discovered that, in both grids, as we increase the number of producers, the execution time for the pairwise comparison stage makes up a smaller fraction of the total execution time, but the execution time fraction for the profiles alignment stage is significantly increased (Figure 13). This is due to the difference in the parallelization nature of these two stages. The pairwise comparison stage involves totally independent tasks which can be executed immediately upon generation, as long as there are free producers available. Hence, volunteering more producers to the grid will improve the stage's performance. The profiles alignment stage, on the other hand, involves tasks that are dependent on results from the lower depths of the guide tree. As we ascend the guide tree, there is lesser room for parallelization since the number of nodes at each depth decreases as the tree root is approached. Therefore, increasing the number of producers does not have a significant effect on the performance of profiles alignment.

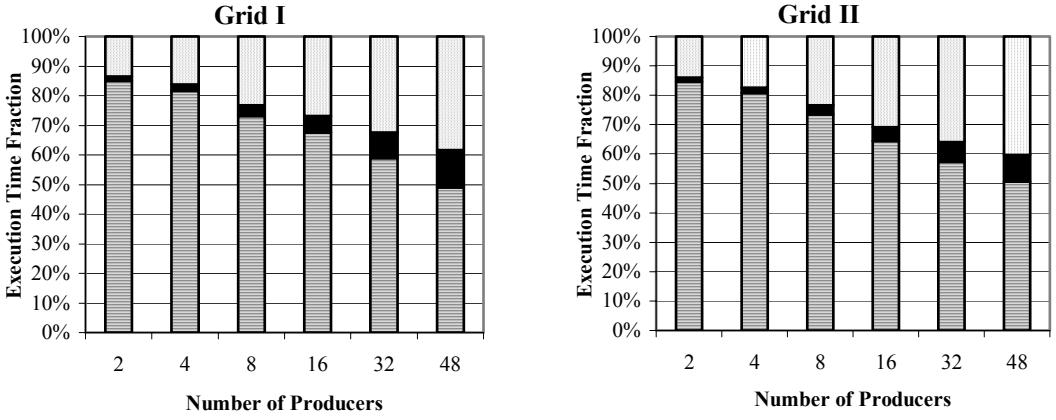

Figure 13: Grid Performance for the Three PMSA Stages

5. Conclusions and Further Works

We have successfully developed and deployed the bioinformatics PMSA problem on the grid, using the ALiCE grid computing middleware as the implementation tool. We have demonstrated how bioinformatics problems with a semi-regular computational pattern can be readily parallelized for distributed grid execution. To study how the performance of grid-based PMSA scales with amount of computational power, we conducted experiments on two grids (homogeneous and heterogeneous), and we observed that given a fixed problem and task

size, the overall execution time decreases considerably with an increase in the number of producer machines used. The heterogeneous cluster grid, mimicking a real-life grid environment with variable types of resources and networking speeds, does not perform as well as the homogeneous grid, which is more tightly coupled and comprises of uniform resources with similar computational abilities.

With the success of the human genome project, the life sciences industry looks set to be a promising area of development for the next few decades. Grid computing provides a timely catalyst to bridge the gulf between molecular biology and bioinformatics. The next approach is to develop a grid programming model for life sciences applications. This potentially facilitates the deployment of complicated bioinformatics problems on the grid without re-implementing commonly used component algorithms that can be very tedious to parallelize. Work in this aspect is currently ongoing in the ALiCE research team.

Acknowledgements

This work has been supported by the Singapore-MIT Alliance program. Special thanks to the Bioinformatics Institute of Singapore for providing the application domain knowledge.

References

1. Baker, M., Buyya, R. and Laforenza, D., Grids and Grid Technologies for Wide-Area Distributed Computing, *International Journal of Software: Practice and Experience (SPE)*, 32(15), Wiley Press, USA, November 2002.
2. Bottu, G., Algorithms for Multiple Sequence Alignments, http://www.hgmp.mrc.ac.uk/embnet.news/vol2_1/align.html.
3. Buyya R., Abramson D., and Giddy J., Nimrod, G., An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *Proceedings of the 4th International Conference and Exhibition on High-Performance computing in the Asia-Pacific Region (HPCASIA'2000)*, China, IEEE CS Press, USA, 2000.
4. Culler, D. E., Singh, J. P. and Gupta, A., Parallel Computer Architecture: A Hardware/Software Approach, 3, Morgan Kaufmann Publishers, San Francisco, California, August 1998.
5. De Roure, D., Baker, M. A., Jennings, N. R. and Shadbolt, N. R., The Evolution of the Grid, Research Agenda, UK National eScience Center, 2002.
6. DeFanti, T., Foster, I., Papka, M., Stevens, R. and Kuhfuss, T., Overview of the I-WAY: Wide Area Visual Supercomputing, *International Journal of Supercomputer Applications*, 10, pp. 123-130, 1996.
7. Farris, J. S., Methods for Computing Wagner Trees, *Systematic Zoology*, 38(4), pp. 83-92, 1970.
8. Feng, D. F. and Doolittle, R. F., Progressive Alignment of Amino Acid Sequences and Construction of Phylogenetic Trees from Them, Methods in Enzymology, 266(21), pp. 368-382, 1996.
9. Fitch, W. M. and Margoliash, E., Construction of Phylogenetic Trees, *Science*, 155, pp. 279-284, January 1967.

10. Foster, I. and Kesselman, C., Computational Grids, Morgan Kaufmann Publishers, 1998.
11. Foster, I. and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11(2), pp. 115-128, 1997.
12. Foster, I., Kesselman, C., Nick, J. M. and Tuecke, S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, *Proceedings of CGF4*, February 2002, http://www.globus.org/research/papers/ogsa.pdf.
13. Foster, I., Kesselman, C. and Tuecke, S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputer Applications*, 2001.
14. GigaSpaces Platform White Paper, GigaSpaces Technologies, Ltd., February 2002.
15. Hupfer, S., The Nuts and Bolts of Compiling and Running JavaSpaces Programs, Java Developer Connection, Sun Microsystems, Inc., 2000.
16. Itzkovitz, A. and Schuster, A., Distributed Shared Memory: Bridging the Granularity Gap, *Proceedings of the First ACM Workshop on Software Distributed Shared Memory (WSDSM)*, Greece, June 1999.
17. Mikhailov, D., Cofer, H. and Gomperts, R., Performance Optimization of Clustal W: Parallel Clustal W, HT Clustal, and MULTICLUSTAL, white paper, SGI Life and Chemical Sciences, 2001.
18. Monge, A. and Elkan, C., The Field-matching Problem: Algorithm and Applications, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 267-270, August 1996.
19. National Center for Biotechnology Information (NCBI) Homepage, National Library of Medicine, Maryland, United States of America, http://ncbi.nlm.nih.gov.
20. Pearson, W. R. and Miller, W., Dynamic Programming Algorithms for Biological Sequence Comparison, Methods in Enzymology, 183, pp. 575—601, 1992.
21. Rzhetsky, A. and Nei, M., A Simple Method for Estimating and Testing Minimum-Evolution Trees, *Molecular Biology and Evolution*, 9(13), pp. 945-967, 1992.
22. Saitou, N., Reconstruction of Gene Trees from Sequence Data, Methods in Enzymology, 266(25), pp. 427-449, 1996.
23. Saitou N. and Imanishi T., Relative Efficiencies of the Fitch-Margoliash, Maximum-Parsimony, Maximum-Likelihood, Minimum-Evolution, and Neighbor-Joining Methods of Phylogenetic Tree Construction in Obtaining the Correct Tree, *Molecular Biology and Evolution*, 6(5), pp. 514-525, 1989.
24. Saitou N. and Nei, M., The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees, *Molecular Biology and Evolution*, 4(4), pp. 406-425, 1987.
25. Setubal, J. and Meldanis, J., Introduction to Computational Molecular Biology, PWS Publishing Company, 3, pp. 47-104, 2000.
26. Sneath, P. H. P. and Sokal, R., Numerical Taxonomy, W. H. Freeman, San Francisco, 1977.
27. Teo, Y. M., Tay, S. C. and Gozali, J. P., Geo-rectification of Satellite Images using Grid Computing, Proceedings of the International Parallel & Distributed Processing Symposium, IEEE Computer Society Press, Nice, France, April 2003.
28. Thomasson, W. A., Unraveling the Mystery of Protein Folding, Breakthroughs in Bioscience, FASEB Journal, January 2002.
29. Thompson, J. D., Higgins, D. G. and Gibson, T. J., CLUSTALW: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice, Nucleic Acids Res., 222(4673), 1994.
30. Trelles, O., On the Parallelization of Bioinformatics Applications, Briefing in Bioinformatics, Henry Stewart Publications, 2(2), pp. 181-194, May 2001.
31. Wess, J., Structure/Function Analysis of G-Protein Coupled Receptors, John Wiley, New York, 1999.