



CONTRACT NUMBER 508830

DEISA
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR
 SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
 Integrated Infrastructure Initiative

JRA4 – Midterm report on activity status

Deliverable ID: D-JRA4-6

Due date: October 30, 2006

Actual delivery date: November 30, 2006

Lead contractor for this deliverable: IDRIS – CNRS, France

Project start date : May 1st, 2004

Duration: 4 years

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	X
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Content

Table of Content.....	2
1. Introduction.....	3
3. Implementation, optimisation of codes for protein docking and scoring	4
4. MPIGrid.....	7
5. Life Sciences portal applications.....	10

1. Introduction

The current JRA4 work program involves three different lines of activity:

- Support to research activities in the protein dynamics domain
- Deployment of specific middleware for relatively simple parallel applications,
- Selection and deployment of the three leading applications that will be operated by the forthcoming Life Sciences portal.

One of the priorities of the DEISA Computational Biology program has been the development of computational tools helping users to access in a more efficient way to DEISA computational facilities, like the MareNostrum system in BSC or the IBM and SGI systems in several other DEISA sites.

Analysis of the needs of scientists working in the area of life sciences has shown two main problems, which can benefit from supercomputing support:

- Sequence related problems. Here the computational problem is simple and the need to use a supercomputing system is due to the need to repeatedly perform the same data treatment in a very efficient way, and with very efficient access to shared databases.
- Structure related problems. These applications involve more expensive computations. A correct optimisation of the code and the selection of the correct compilation directives can reduce dramatically the cost of the calculation. The effective support required for these applications by users communities which are not traditional supercomputing experts requires:
 - Optimization of executable codes to improve their performance on a given platform
 - Benchmarking of available programs to determine the best suited code for a given biological problem
 - Help the user in the generation of input files in the management of output files as well as in the flow of data between programs.

The protein dynamics work has been guided by the analysis above. It belongs to the second class of applications (structure related problems). The most important results have been: I) the developed of a general parallelization utility, ii) the installation, optimisation and benchmarking of docking programs, and v) the development of a front-end for the analysis of trajectories obtained from different platforms and the input \leftrightarrow output and output \leftrightarrow output translation.

2. Implementation, optimisation and refinement of codes for protein-protein docking and scoring (BSC)

Determining the protein interaction network of whole organisms has become a major theme of functional genomics and proteomics efforts, and we can expect that vast amount of supercomputer resources will be focused in helping to the determination of the “interactome”. Protein docking is an emergent technique in this field, in which the objective is to predict the complex formed when proteins physically interact using the atomic coordinates of two individual proteins. Due to the large number of degrees of freedom involved in the calculation, protein docking is one of the most computationally demanding fields in bioinformatics.

2.1 - Generation and test of a multi-program docking platform.

The purpose of this activity is to analyze and benchmark the most representative and the (a priori) better known programs in order to determine which one provides the best results from a purely biological point of view. The objective is not only to determine the average quality of each method, but also to determine the performance of the methods to deal with problems of different difficulty (this mostly depends on the conformational changes they suffer, the size of the interface area or the kind of relation that proteins establish such as enzyme/inhibitor or antigen/antibody). The final objective will be to assist the community in the choice of the best methods for each possible requirement. We can even think in expert programs able to manage and re-score the different docking solutions provided by the different programs based on their expected quality to solve the specific docking problem.

2.2 - Programs selected

The selected programs fulfilled three conditions:

- Good performance from a biological point of view. That means providing good docking predictions for the input proteins
- Implementation of different basic algorithms.
- Source code or PowerPC Linux binary available (for operation in Mare Nostrum).

The responsible of this part of the project, Juan Fernandez-Recio, has strong expertise in development and benchmarking of protein-protein docking methods, which is essential for the selection, setting up, and running of the docking programs. In addition, Alfonso Valencia, a well known scientist with large experience in protein modelling, Bioinformatics, and assessment of structure prediction methods (e.g. he is actively involved in the management of CASP; <http://predictioncenter.org/>), has acted as advisor of the Computational Biology Program at the BSC in this project. The programs retained are the ones in the following list (of course, installation of new programs that satisfied the above criteria is expected):

- PyDock
- 3D-Dock

- Hex
- RosettaDock

2.3 - Installation of the programs in Mare Nostrum

These were the steps taken for each of the installations in Mare Nostrum (supercomputer with PowerPC processors and a Linux 2.6 operating system installed):

- 3D-Dock
 - o Installation of the Fast Fourier Transform library (FFTW 2.1.5), needed by FT-Dock, using the available source code. Different versions were tested: 32 and 64 bits, XLC and GCC binaries, serial and parallel versions, double and single precision float numbers.
 - o Installation of FT-Dock, the shape complementary algorithm
 - o Installation of RPScore, the Residue Level Pair Potential Scoring algorithm to rescore the solutions obtained from FT-Dock
 - o Multidock, the last step of 3D-Dock (the refinement algorithm) had to be discarded because the source code was not available
 - o
- Hex
 - o There was no binary for PowerPC and the source code was not available neither
 - o In collaboration with the creator of the algorithm (David Ritchie, dritchier@csd.abdn.ac.uk) we obtained an optimized binary for PowerPC that was working in Mare Nostrum
 - o
- PyDock
 - o Installation of Python 2.3
 - o Installation of several mathematical & bioinformatics Python libraries
 - o Required FT-Dock, but it was already installed
 - o Impossibility to use Z-Dock due to the unavailability of the source code or a binary for PowerPC
- RosettaDock
 - o Installation of the program

IMassive execution in Mare Nostrum required the parallelization of these codes. In all the cases but one, the MPIGrid library by Javier Vera was used, a tool which allows a really easy parallelization of the code without using any MPI instruction (see the next setion). This tool simply enables the “embarrassingly parallel” option of running different executions at the same time, and we obtained of course a linear speed-up.

The only exception on the use of MPIGrid was PyDock, for which a standard MPI internal parallelization was developed. Thanks to the repetitive code, a linear speed-up could be achieved. This is the only method “internally” parallelized, given that one of the members of the team, Juan Fernández Recio, is the author of the method.

2.4 - Benchmarking

The second part of the project has been the execution all the methods in Weng's benchmark (a selection of 80 protein-protein complexes for which the structures of both the complex and unbound forms are available). Calculations have been already done for 3D-Dock, Hex and pyDock. Given the high computational costs for running RosettaDock in the optimal conditions indicated by its authors, the benchmark of this method is not yet finished for all the cases in Weng's test set. We hope to have finished the benchmarking and conditional-ranking of this program by the end of 2006.

3 - MpiGrid. A simple way to implement parallel execution (BSC)

3.1 - Introduction

The efficient usage of supercomputing environments in Bioinformatics projects often requires doing the same task many times or a group of tasks in parallel. Although bioinformatics applications are in most cases already prepared for supercomputing environments, this is not necessary the case for in-house software or ad-hoc implementations made by bioinformatics researchers. Additionally, the data that is handled in modern bioinformatics (complete genomes, whole set of protein families, etc.) would make difficult the use of classic, well known, software that was originally written for limited amounts of data and become highly inefficient against such type of problems.

When this need appears there are a lot of issues to consider:

- choosing a technology to parallelize the program
- implementing the parallel code
- getting a good performance (doing a well-balanced application)
- validating the program
- ensuring that the execution is correct and the results are right
- monitoring the execution
- what to do when there are problems (i.e. the system falls)

MpiGrid is an easy way to implement parallelism in certain programs that do not have data dependencies. The use of MpiGrid will reduce the total time in development, maintain and upgrade these programs. MpiGrid also allows the parallelization of programs with dependencies but this feature is less powerful than other technologies, and cannot compete with the classic parallelization approaches.

Although DEISA focuses mainly on capability computing which involve in general very complex parallel applications having string data dependencies, computational biology provides in several cases embarrassingly parallel computational contexts that justify the usage of efficient “quick and dirty” parallelization techniques. This is the case of MPIGrid.

3.2 - Overview of the Program

MpiGrid is a library implemented in C with underlying MPI support. This programming environment is designed to offer a friendly and simple interface to the end user, allowing him to parallelize programs in fast and simple way. Parallelization with MpiGrid requires a minimum number of changes in the original code, and allows monitoring of failed executions and balancing

Typical bioinformatics cases where MpiGrid can help include:

- Need to perform a lot of executions of the same program with different inputs.
- Need to execute a lot of different programs at same time
- Need of multiple executions of workflows (consecutive executions of programs)

In the MpiGrid framework, the “Program of interest” is defined as a “function” inside the same program. Traditional MPI calls can be done inside MpiGrid environment, and the total execution can be monitored with internal MpiGrid functions. After the execution, the user can look at the *master_log* file to ensure that all executions were successful or repeat those that failed.

3.3 Structure of MpiGrid

The general structure of a MpiGrid INPUT is as follow

```
Example:
include "mipgrid.h"
(...)
bd = iniEnvironment(``the_data.txt");
while (( ut = nextTask(``bd" )!= NULL)
{
    function_that_does_something(ut);
}
endEnvironment();
```

The user of the program has to do the following steps:

- ✓ Include a file `` mpigrid.h "
- ✓ Put a call “iniEnvironment”
- ✓ Define a loop within nextTask.
- ✓ Inside this loop the programmer has to write all code he want execute, (i.e. system(returned_value_of_nextTask))
- ✓ Make a call endEnvironment after the loop

Only three functions are needed for handle MpiGrid implementations:

iniEnvironment: This function initializes the environment of MpiGrid. “file_data” is read out by all processes and it's allocated enough memory to support all the work-units (Units of execution code, *wu*).

nextTask, when this function is called there are two behaviours:

1. The process which is calling the function will receive the next *wu* to execute
2. The Thread Communication Manager (TCM) will receive the result of the last *wu* executed by the process which is calling the function. When there aren't anymore *wu* to exec, nextTask will return NULL. It is controlled by TCM and a scheduling algorithm.

endEnvironment: Exits the MPI environment and the main execution

Monitoring can be done using an additional function: HowManyDone which returns two values: *Total tasks* and *how many tasks* are already done.

A “*master_log*” file is written by all functions to keep track of the execution process.

3.4 Scheduling

A simple scheduling algorithm is used. All executions are managed by TCM. This thread is the responsible of calling the work-units and of the redistribution of the execution among the other threads. Once a thread finishes the assigned task, TCM picks up the process with a maximum number of pending wu's and splits it in two halves, the second being reallocated to the free thread. This assures a well balanced redistribution of processes. To avoid excessive reallocation of processes, the scheduling algorithm is used only if the remaining *wu's* in a given thread would last more than a specified time value.

The MpiGrid environment has been tested at the BSC by the computation node of the National Institute of Bioinformatics, with classical bioinformatics applications as Blast and HMMer.

This part of the JRA4 work program – deployment of MPIGrid - is considered as completed.

4 – Life Sciences Portal applications (IDRIS, BSC)

The eDEISA project has substantial funding for the system and middleware activities required for the deployment and operation of a Life Sciences Portal. An ad-hoc middleware task force for the deployment of the portal is today in operation, working in close collaboration with the IDEA, the technology provider company. The DEISA JRA4 Life Sciences activity is responsible for the deployment and operation of selected applications.

4.1 – Application selection:

The objective of the portal strategy is not just decorating the access of existing applications by existing users with Web interfaces. The purpose is to hide supercomputing resources from non traditional users that would never access a supercomputing infrastructure in the standard way. Portals are a way of widening the outreach of leading European supercomputing resources, by allowing new users to benefit from leading technologies without investing too much effort in a long learning process.

For this reason, two basic criteria were applied for application selection for the Life Sciences Portal:

- Relevance and necessity of access to high performance computing resources. It is not DEISA intention divert its leading supercomputing resources to deal with subjects that can be solved in with lightweight computational resources. However, some areas in computational biology are reaching a point where supercomputing resources are required for leading edge projects. Such is the case of the three subjects retained (sequencing, phylogeny, protein docking).
- Existence of potential users communities that would be attracted to supercomputing class problems by the portal availability. Indeed, the purpose of the portal is to expand the supercomputing user community. Again, this criterion is fulfilled by the selected scientific sectors.

During the selection process our concern was to find diverse applications widely used by the community or even better potentially important in the future. Those applications should be already parallelized or an excellent candidate for parallelization, in order to avoid delays the first phase of the project. With our expertise those applications will be easily made accessible and adapted to the portal Web interfaces..

The following software packages have been retained :

- mpiBLAST-1.4.0 (<http://mpiblast.lanl.gov/>) (similarity search)
- RAxML-VI-HPC (<http://icwww.epfl.ch/~stamatak/>) (Phylogeny)
- NAMD (molecular dynamics)

The BLAST suite from NCBI is an obvious choice since it is the most used application in the similarity search/Alignment domain, maybe even in the whole bioinformatics field. A parallel MPI version (mpiBLAST) is available. Blast is essentially used to search into a database all the biologic sequences similar to a query sequence (with a closeness criterion). Previous experiences with parallel versions of MPI in our facilities are very encouraging. There are ongoing efforts made by the developers to also parallelize the I/O.

RAXML is a smaller project but very dynamic and promising. In the sea of phylogenic applications it has the merit to be parallelized in an MPI version. An OpenMP version is also available, iRAXML is used to infer and build phylogenic trees from biologic sequences multi-alignments, comparing each sequence two by two and building a distance matrix.

NAMD is a molecular dynamics software package that is used in several disciplines. It benefits from strong support at BSC for protein dynamics applications (see previous JRA4 deliverable).

4.2 - Ongoing JRA4 activity:

After selection of the three initial applications for the Life Sciences Portal, the current work plan in this activity is organized as follows:

- Assuring that all three applications are installed and running all three DEISA scalar architectures and operating systems (PPC in MareNostrum, IBM AIX and Linux SGI Altix) facilities. This would imply installing MPI-Blast at BSC and on one SGI site, and NAMD at IDRIS and another SGI site. Either institution would provide their knowledge to help the other if problems occur.
- Defining test problems for all applications of the appropriate size to take profit of the supercomputing architectures. Typically this would mean genome-sized Blast queries or 50-100000 atoms molecular dynamics simulations. BSC or IDRIS associated researchers would help to define biologically relevant cases.
- Both BSC and IDRIS would check that test calculations can be performed with comparable results. At this point efficiency benchmarks would be performed.
- BSC and IDRIS as part of the DEISA initiative, have to built and efficient communication protocol between both institutions. Data transfer would be tested. Typically Blast would imply 1Gb input and small output and NAMD, small input and Gb sized output, possibly fragmented in several smaller files. Eventually, data compression protocols will be tested.
- Finally, tests will be performed comparing local and remote execution for all applications, and a stable framework for remote execution will be established.