Towards High-Throughput, Multi-Criteria Protein Structure Comparison

Azhar Ali Shah, Gianluigi Folino, and Natalio Krasnogor

Manuscript received XXX, 2009; revised XXX, 2009.

G. Folino is with the CNR-ICAR, Institute of High Performance Computing and Networking, Italy. E-mail: folino@icar.cnr.it

A.A Shah and N. Krasnogor are with the School of Computer Science, University of Nottingham, UK, NG8 1BB. E-mail: {psxaass,Natalio.Krasnogor}@nottingham.ac.uk

Abstract

Protein Structure Comparison (PSC) is an essential component of biomedical research as it impacts on, e.g., drug design, molecular docking, protein folding and structure prediction algorithms as well as being essential to the assessment of these predictions. Each of these applications, as well as many others where molecular comparison plays an important role, requires a different notion of similarity that naturally lead to the Multi-Criteria Protein Structure Comparison (MC-PSC) problem. ProCKSI (www.procksi.org), provides algorithmic solutions for the MC-PSC problem by means of an enhanced structural comparison that relies on the principled application of information fusion to similarity assessments derived from multiple comparison methods. Current MC-PSC works well for moderately sized data sets and it is time consuming as it provides public service to multiple users. Many of the structural bioinformatics applications mentioned above would benefit from the ability to perform, for a dedicated user, thousands or tens of thousands of comparisons through multiple methods in real-time, a capacity beyond our current techonology. In this paper we take a key step into that direction by means of a high-throughput distributed re-implementation of ProCKSI for very large data sets. The core of the proposed framework lies in the design of an innovative distributed algorithm that runs on each compute node in a cluster/grid environment to perform structure comparison of a given subset of input structures using some of the most popular PSC methods (e.g. USM, MaxCMO, Fast, DaliLite, CE and TMalign). We follow this with a procedure of distributed consensus building. Thus the new algorithms proposed here achieve ProCKSI's similarity assessment quality but with a fraction of the time required by it. Our results show that the proposed distributed method can be used efficiently to compare *a)* a particular protein against a very large protein structures data set (target-against-all comparison), *b)* a particular very large scale dataset against itself or against another very large scale dataset (all-against-all comparison). We conclude the paper by enumerating some of the outstanding challenges for real-time MC-PSC.

Index Terms

Protein structure, Comparison, Alignment, Multi-Criteria, Real-Time, Very Large Scale Data Sets, MPI, GRID.

I. Introduction

The comparison of protein structures is an essential activity of biomedical research as it impacts on structure-based drug design [1], protein structure prediction/modeling [2]–[4], classification [5], [6], molecular docking algorithms [7] and other structural bioinformatics applications. The specific ability of protein 3D structure comparison to reveal more significant evolutionary interrelations between proteins that share very little common sequence (primary structure) has given rise to various world wide structural

genomic and proteomics initiatives such as the Structural Genomics Consortium (SGC) [8], the Protein Structure Initiative (PSI) [9] , and the Human Proteome Organization (HUPO) [10] amongst others. These initiatives are targeted at lowering the cost and enhancing the efficiency for the experimental determination or computational prediction of novel protein 3D structures, leading for instance to the identification of new structure-based medicine or therapeutics for treating genetic and infectious diseases. As a consequence, there is a vast growing number of protein 3D structures available in the Protein Data Bank (PDB) [11] demanding more efficient and reliable software analysis tools and services, especially for determining their structural similarities and classifying them into families according to similarity relationships.

Several methods and tools have been developed to investigate the (dis)similarities among protein structures [12]. Not surprisingly, there is no agreement on how to optimally *define* what similarity/distance means as different definitions focus on different biological criterion such as sequence or structural relatedness, evolutionary relationships, chemical functions or biological roles etc and these are highly dependent on the task at hand. This observation calls for an explicit identification and understating of the various stages involved in the assessment of proteins' similarities.

As illustrated in Figure 1, the first four stages, which have dominated the research in protein structure comparison so far, are: similarity conception, model building, mathematical definition and method implementation. Interestingly, the fifth stage, where one would seek to leverage the strength of a variety of methods by using appropriate consensus and ensemble mechanisms has barely been investigated. One such approach has recently been introduced by means of the *Protein (Structure) Comparison, Knowledge, Similarity and Information* (ProCKSI) web server [13]. Using a set of modern decision making techniques, ProCKSI automatically integrates the operation of a number of the most popular comparison methods (as listed in Table I) and provides an integrated consensus that can be used to obtain more reliable assessment of similarities for protein datasets. The consensus-based results obtained from ProCKSI take advantage of the *'collective wisdom'* of all the individual methods (i.e, the biases and variances of a given method are compensated by the other methods biases and variances) and minimizes the chances of falsely attributing similarity to (sets of) proteins. That is, false positives are more frequent at individual method level because usually most of globally different proteins still share some common substructures.

In this paper we describe a high-throughput implementation of the entire protocol shown in Figure 1 whereby very large protein structure dataset comparisons are done in parallel using several methods and exploiting the intrinsic MIMD (*Multiple Instructions Multiple Data*) structure of the problem. Thus this work takes a step forward towards the ultimate goal of real-time multi-criteria similarity assessment of very large protein datasets. Section II presents the review of the related literature, specially focusing

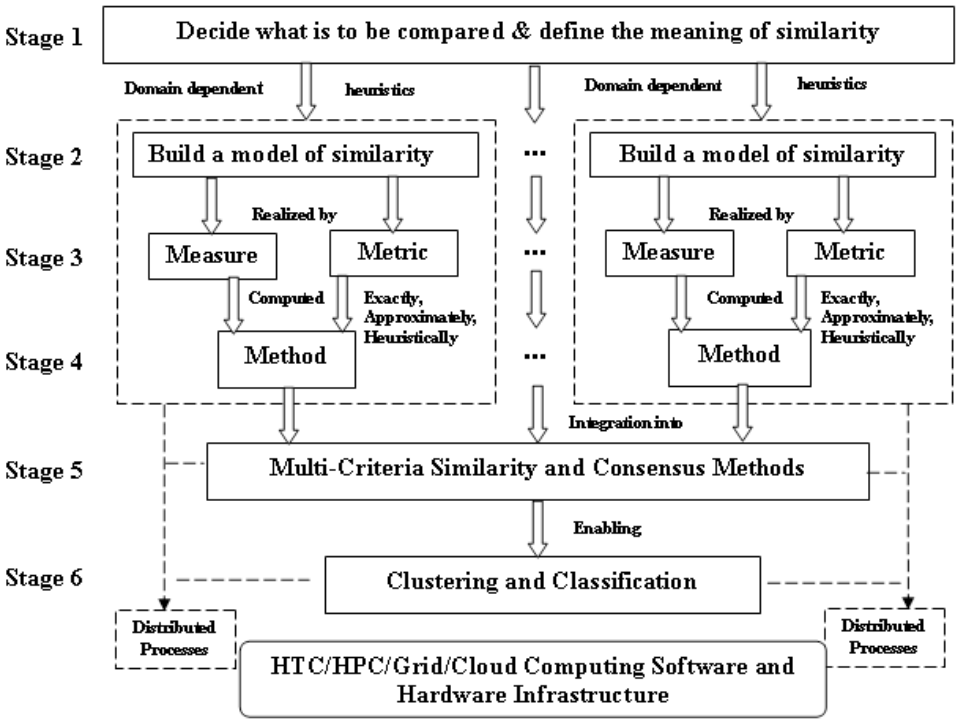


Fig. 1

STAGES IN THE DERIVATION OF A PROTEIN'S CLASSIFICATION: (1) DECIDE WHAT "SIMILARITY" MEANS, WHICH IS A DECLARATIVE AND PROBLEM-DEPENDENT STEP. (2) HEURISTICALLY BUILD A MODEL OF SIMILARITY BASED ON *1*. THIS NEW SIMILARITY/DISTANCE CONCEPTION WILL HAVE ITS OWN BIAS, VARIANCE AND OUTLIERS. (3) DECIDE WHETHER THIS IDEALIZED MODEL WILL BE INSTANTIATED AS A DISTANCE/SIMILARITY MEASURE OR METRIC. (4) ONE OR MORE ALGORITHMS ARE IMPLEMENTED IN ORDER TO CALCULATE *3*, WHICH CAN BE SOLVED EXACTLY AND IN POLYNOMIAL TIME ONLY IN THE SIMPLEST OF CASES. THE MORE INTERESTING SIMILARITY DEFINITIONS, HOWEVER, GIVE RISE TO COMPLEX PROBLEMS REQUIRING HEURISTICS/APPROXIMATE ALGORITHMS FOR THEIR SOLUTION. (5) COMBINING MANY DIFFERENT METHODS WITH DIFFERENT VIEWS OF SIMILARITY PRODUCES A MULTI-COMPETENCE PARETO-FRONT, FROM WHICH A CONSENSUS PICTURE MIGHT BE DERIVED. IN TURN, THIS ALLOWS THE STRUCTURAL BIOLOGIST TO (6) CLUSTER AND CLASSIFY PROTEINS RELIABLY. FURTHERMORE, IN ORDER TO PROVIDE MOST EFFICIENT (REAL-TIME) RESULTS BASED ON THE PHILOSOPHY OF (5), THE NEED FOR THE DATA AND COMPUTATION TO BE DISTRIBUTED AND EXECUTED IN A HIGH-THROUGHPUT ENVIRONMENT BECOMES INDISPENSABLE.

on the use of parallel and distributed computing for protein sequence/structure alignment. A succinct description of multi-criteria protein structure comparison is provided through an overview of ProCKSI in section III. Also in this section we provide an in-depth description of the computational challenge at the core of real-time *MC-PSC*. Section IV provides the architectural design and analysis of our newly proposed framework. Experimental results and their analysis are presented and discussed in section V.

TABLE I

BUILDING BLOCKS FOR MULTI-CRITERIA PROTEIN STRUCTURE COMPARISON. THE NAME AND REFERENCES FOR EACH OF THE METHOD IS SHOWN IN COLUMN "METHOD", FOLLOWED BY THE COLUMN "FEATURES" WHERE THE SPECIFIC FEATURE(S) THAT EACH METHOD USES TO DETERMINE (DI)SIMILARITY ARE MENTIONED. COLUMNS "ALGORITHMS" AND "MEASURES" SUMMARIZE HOW EACH SIMILARITY FEATURE IS COMPUTED AND IN WHAT FORM IT IS RETURNED. THE LAST COLUMN GIVES AN INDICATION OF RELATIVE COMPUTATIONAL REQUIREMENTS (TIME) FOR THE DIFFERENT METHODS. **Key**: *AL* = NUMBER OF ALIGNMENTS;*OL* = NUMBER OF OVERLAPS;*Z* = Z-SCORE; *TMS* = TM-ALIGN SCORE; *SN* = NORMALIZED SCORE. * CPU TIME FOR A SINGLE PAIR OF PROTEIN STRUCTURES (389 RESIDUES PER STRUCTURE) ON A STANDARD P4 (1.86 GHZ, 2GB RAM) DUAL-CORE MACHINE. THUS THE TOTAL EXECUTION TIME TAKEN BY ALL SIX METHODS (WITH A TOTAL OF 15 DIFFERENT SIMILARITY MEASURES/METRICS) FOR THE COMPARISON OF A SINGLE PAIR OF PROTEIN STRUCTURES IS 31.79 SECS PLUS SOME ADDITIONAL TIME FOR PERFORMING I/O.

Method	**Features**	**Algorithms**	**Measures**	**Time* [sec]**
MaxCMO [14]	contact map	Variable Neighborhood Search (VNS)	AL, OL	24.66
DaliLite [15]	inter-atomic distance	distance matrices combinatorial simulated annealing	AL,Z, RMSD	2.95
CE [16]	inter-residue distances rigid body superposition	heuristics dynamic programming	AL, Z, RMSD	2.80
TM-align [17]	inter-atomic distance	rotation matrix dynamic programming	AL, RMSD,TMS	0.71
USM [18]	contact map	Kolmogorov complexity	USM-distance	0.62
FAST [19]	inter-residue distances	heuristics dynamic programming	RMSD, AL, SN	0.05

II. RELATED WORK

Recent advances in high-throughput techniques have led to a data deluge in terms of the availability of biological and biomedical data such as 1D sequences (flat files), 3D structures, microscopic images,

videos and motifs, etc. [20]. This has put considerable strain in the computational resources that are routinely use to store, manage, process and analyze the vast amount of data being generated. As to cope with the increase in computational demands instigated by very large data sets, three routes are usually followed [20]: (a) the development of new algorithms or the redesign/modification of existing ones based on faster *heuristic* techniques [21], [22]; (b) development of special purpose ROM based hardware chips [23], [24]; and (c) the use of parallel and distributed computing. Routes (a) and (b) can only be applied in very specific cases as they require considerable in-depth knowledge of a problem or substantial economic resources respectively. The third alternative, the utilization of distributed and parallel computation is becoming a more ubiquitous approach as in some cases distributed/parallel solutions in one problem can be reused (with slight modifications) in other problems. Moreover, due to ongoing advances in processor and networking technologies, the scope of parallel computing is also extending from traditional supercomputers to massively parallel computers, clusters of workstations (COW) and even crossing the boundaries in the form of clusters of clusters i.e *grid computing* [25]. This paradigm shift in the provision of parallel computing facilities afford scalability at very low cost. Furthermore, in terms of code maintenance and code portability, as compared to traditional super computers, distributed computing fares better [26], [27]. Several successful applications to nanobiosciences are discussed in [28]–[34].

Notwithstanding the above successes, parallel and distributed computing have no magic applicability formula and many different parallelization solutions might exists for a given problem. Which one of these strategies would be the best one to use will depend to a large extent not only on the specific problem structure but also on factors such as available hardware, interconnection types, security protocols and human resources. For example, the BLAST (Basic Local Alignment Search Tool [35]) algorithm has been parallelized/distributed through a variety of ways [28], [32], [36]–[42]. Some of these approaches use combinations of MPI (Message Passing Interface) [43], Grid and Public Computing based architectures to distribute either the query sequence (which could be as long as 80 billions of base pairs [44]) or the target dataset/database(which could have up to 76 million records [44]) or both. All these approaches use a simple master/slave task scheduling strategy with coarse-grained level task distribution for minimizing communication overheads [20]. Coarse-grained approaches are not always suitable: given the variable length of the sequences to be compared and the different processing power of individual nodes in a heterogeneous cluster/grid environment, deciding the actual unit of work to be assigned to a particular node is a non-trivial matter for which efficient dynamic load-balancing strategies are needed. Martino et al. [45], describe a simple, inexpensive and effective strategy that divides the target dataset/database in n

buckets of fixed size (where n represents the number of available processors). The load-balancing in this case is achieved by ordering the sequences by their length (number of bases or residues) and assigning them to each bucket in a way that the longest sequence is assigned to the segment having smallest sum of sequence lengths and continuing this process in a round-robin fashion until all sequences are assigned to buckets. This type of load-balancing strategy reduces the *percentage of work load imbalance* within homogeneous computing architectures but does not take into account the heterogeneity of cluster and grid environments. Trelles et al. [46] present another load-balancing approach based on variable size of blocks (buckets). This strategy initially distributes blocks with small sizes so as to reduce the latency time for each node to receive its first unit of work. It then increases the size of blocks (in the same way as classical *Self Guided Scheduling* (SGS) reduces their size) until the first half of dataset/database is processed and then again starts decreasing their size. The smallest size of final blocks guarantees that all *n* processors will terminate either at a same time (ideal case) or with a maximum time difference that depends on the size of the final block (i.e its execution time). This strategy has been tested on a cluster of 15 nodes with significant enhancement in the performance. Proteins 3D structure comparison algorithms (e.g. those listed in Table I) present a similar structure to algorithms for sequence comparison (e.g BLAST, FASTA and ClustalW etc) and hence sometimes similar parallel/distributed strategies can be used [20]. However, as compared to their sequence counterpart, there are very few instances of the application of parallel computing for 3D structure comparison methods. Ferrari et al. [47] describes the distributed implementation of a geometric indexing based PSC algorithm. In a first step this algorithm uses a pre-computed hash table (that stores angular properties of secondary structure elements (SSEs) for all 3D structures from PDB) as a quick look-up for finding the hypothetical similar structures for a given query. It then performs more refined matching of the query with the subset of structures obtained from the look-up using atomic representation of each structure. This algorithm was adapted to grid environment by distributing the target dataset/database on each node with a simple load balancing strategy that divides the data into smaller subsets of fixed sizes and assigns them to each node repeatedly. The test experiments were conducted on a grid environment (Globus with MPICH) consisting of only 4 nodes (standard workstations) all at a single location. The authors do not provide speedup information or detailed time analysis for their distributed implementation but they claim to have successfully compared a target protein structure (*1TIM*) against a database of 19,500 proteins in a minimum time of 119 seconds. It should be noted that algorithms like this do not perform an exhaustive pairwise comparison and hence one can expect many false negatives. Park et al. describes another distributed environment to speed-up the performance of a genetic algorithm based PSC method named FROG (*Fitted Rotation and Orientation*

of protein structure by means of real-coded Genetic algorithms) [48]. This algorithm uses the generation alternation model with a simple master/slave task scheduling approach to distribute the new parents to each slave repeatedly. The distributed system uses Ninf-G based RPC on a Linux cluster of 16 nodes and achieves a speed-up of 15.61. Other examples of distributed protein structure comparison include the use of distributed and semantics web technologies for PSC at individual method level (e.g. [49] which is actually an extension of [47] as described above), etc. None of these methods, however, deal with the much more complex issue of efficient and scalable distributed implementations for Multi-Criteria Protein Structure Comparison methods. In what follows we discussed the computational issues arising from MC-PSC by focusing on distributed strategies for the ProCKSI server.

III. Computational Challenges in Multi-Criteria Protein Structure Comparison

ProCKSI is an online automated system that implements a protocol for MC-PSC. In particular, it allows the user to submit a set of protein structures and perform either all-against-all or target-against-all protein comparisons with the methods listed in Table I. ProCKSI combines the results of pairwise comparisons delivered by the various available methods, normalizes them and presents a consensus form of the results through an intuitive web-based visual interface. Furthermore, it gathers information about the proteins being compared through hyper links to external sources of information e.g. *Information Hyperlinked Over Protein* (IHOP) [50], *Structural Classification of Proteins* (SCOP) [51], and *Class Architecture Topology and Hierarchy* (CATH) [52]. ProCKSI executes, on a given pair of proteins, several comparison methods that, in turn, can result in one or more similarity measures. In order to be able to compute a consensus similarity assessment, ProCKSI normalizes all of the resulting similarity measures derived from the methods it uses to values in the $[0, 1]$ interval. The normalization process proceeds as follows. First, the origin of the matrix containing the similarity/dissimilarity values is shifted to zero by subtracting the matrix' minimum value from each matrix element. Then, the intermediate matrix is scaled by dividing all matrix elements by the matrix' maximum value. Although this makes sure that the values of all elements are within the correct range, it does not satisfy an additional requirement for *self-similarity* (SS) values, namely, that self-similarity should be 0 for *Distance Matrix* (DM) and 1 for *Similarity Matrix* (SM). Furthermore, similarity values also depend on the size (length) of the proteins. Therefore, in order to normalize a similarity value S_{ij} resulting from the comparison of proteins P_i and P_j, it is divided by the highest SS value of both proteins [53]:

$$S_{ij,norm} = \frac{S_{ij}}{max\{S_{ii}, S_{jj}\}} \tag{1}$$

When applying Equation 1 to SS values S_{ii}, one obtains normalized values $S_{ii,norm} = 1$ as $max\{S_{ii}, S_{ii}\} = S_{ii}$. For the purpose of clustering, the above similarity can be easily converted to distances. As demonstrated in [13], and previously suggested in [54] and [55], the ensemble and consensus based approach adopted by ProCKSI yields more reliable results of biological significance as compared to the results obtained with any single structure comparison method developed so far. However, the integration of multiple methods for protein structure comparison, on the one hand, coupled with a rapidly growing number of 3D structures in the Protein Data Bank (PDB), on the other hand, gives rise to a computational challenge that is far beyond the capabilities of a single standard workstation or a group of workstations, specially if one would like to perform a multi-criteria comparison for very large datasets in real-time. That is, as the number of protein structures being compared increases, the corresponding number of pairwise comparison jobs, I/O files and directories, computational time and memory required for each comparison method and associated pre-processing (e.g. data extraction and contact map preparation) and post-processing (e.g. consensus generation, clustering and result visualization) methods also increases. An estimate of some of these complexities is presented in the following sections.

A. Job complexity

Job complexity for protein structure comparison depends on the size (i.e number of structures) of the dataset/database in hand as well as the mode of comparison. As of April 28, 2009 there are *52,905* protein structures in the PDB and this number grows steadily. If we compare a particular protein against all the proteins in a given dataset (e.g. PDB), this is referred to as *target-against-all* mode of comparison. While being the simplest mode, it is usually used to compare a protein of unknown function but known structure with those whose structures and functions are known. The results of comparison would provide clues regarding the function of the query protein. The number of pairwise comparison jobs in this mode is directly related to the number of structures in the target dataset. For example, given the current holdings of PDB, there will be *52,905* comparison jobs while using target-against-all mode of comparison. However, in the case of multi-criteria comparison the actual number of jobs will be the number of target structures $\times$ *the number of methods* being used for multi-comparison.

Another mode of comparison is the one in which we compare all the elements of a particular dataset among itself or with all the elements of another dataset. This mode is referred as *all-against-all* comparison and is mostly used to cluster/classify a group of structures. The resulting clustering/classification is aimed to reveal the functional and evolutionary similarities among the proteins. The number of pairwise comparison jobs in this mode is proportional to the square of the number of protein structures involved

in the comparison[1] $\times$ *the number of methods*. For example, the comparison jobs for current holdings of PDB using all-against-all mode with only one method will be:

$$N_j = n^2 = 52905^2 = 2,798,939,025$$

Where, N_j represents the number of pairwise comparison jobs, while *n* being the current number of protein structures available in the PDB.

As mentioned above the actual number of jobs will be *2,798,939,025* $\times$ *the number of methods* being used. Therefore, it will require an optimal way to distribute all these jobs in the form of some smaller subsets (working packages) that could be submitted for parallel/distributed execution. Needless to say, this complexity calls for a high performance computing solution. Please note that protein structure prediction methods, e.g. Robetta [56] and I-TASSER [57], often sample thousands of "decoys" that must be compared and clustered together at each iteration of the algorithm as to obtain a centroid structure. Thus comparing thousands or ten of thousands of protein structures is not limited to assessing the PDB only but actually occurs as a subproblem in many other structural bioinformatics activities.

B. Time complexity

Different protein structure comparison algorithms have different time complexities and run time profiles. Table I provides an indicative comparison between the times taken by the algorithms we used in our experiments for a typical protein pair. Arguably, depending on the length of the members of a protein pair, the times mentioned in the table would change. However, these can be use to give a rough estimate[2] of the run time profile that can be expected from these algorithms:

target-against-all: for a given protein structure compared against the *52,905* structures in the PDB (assuming only one chain per PDB file), a Multi-Criteria comparison with the methods available in Table I consuming the time mentioned in the fifth column, would take, on a P4 (1.86GHz, 2GB RAM) dual-core workstation *19.5* days.

all-against-all: if one were to execute this type of comparison for the entire PDB, this will result on *2,798,939,025* pairwise comparison jobs (assuming again one chain per PDB file) and it would take about *85.4* years for all jobs to finish on a single machine.

[1]Please note that some methods return different similarities for the comparison of P_i with P_j and the reverse comparison

[2]In later sections we provide a more detailed analysis of run times.

C. Space complexity

Executing potentially millions of pairwise protein structure comparison has strict requirements in terms of memory and bandwidth allocation. MC-PSC jobs generate a very large number of output data files that need to be parsed and summarized in a way that enables the execution of the normalization and consensus steps but also that falls within the constraints of the available computational infrastructure. With the current number of proteins structures in PDB, and the total number of comparison measures/metrics for all six methods (Table I) there may be as many data items in the resultant di(similarity) matrix as:

$n^2 \times (N_{mt} + 2)$ =52,905^2 $\times$ 17= 47,581,963,425.

Where n again represents the current number of protein structures in PDB, N_{mt} represents the total number of measures/metrics (see Table I) and the additional *2* accounts for the two protein IDs involved in each comparison. Using a minimum of 5 digits/charachers to hold each data item it may require about $238GB$ to hold the matrix. Given the size of this matrix, it becomes indispensable to compute and hold its values in a distributed environment and use some parallel I/O techniques to assemble each distributed portion directly at an appropriate storage location.

The above back-of-the-envelope calculations point to the need for a high-performance solution to the MC-PSC problem.

IV. A High-throughput Distributed Framework for Protein Structure Multi-comparison

In this section we present the algorithmic framework we use to compute in a distributed environment solutions to the MC-PSC problem. Figure 2 illustrates the overall architecture of the proposed system. The top module performs the distribution (through two different decomposition approaches as explained in the following sections) of pairwise comparisons and allocates them over the available nodes. Then, using the assigned (bag) proteins, each node performs, in parallel and without the need for synchronization, the pairwise comparisons required by its associated protein bag using each of the available PSC methods. That is, each compute node computes a sub-matrix from the all-against-all similarity matrices associated to each method. Afterwards, a phase of normalization and estimation of missing/invalid values is executed. This phase exchanges information among nodes, as it needs the global minimum and maximum similarities for the normalization as well as for the estimation of missing/invalid cells. All the results concerning the current node are stored on a local matrix. Note that no global and centralized matrix is maintained by the system and that all the communication among the nodes are performed using the MPI (Message

Passing Interface) libraries for a cluster of computers and using the MPICH-G2 libraries [58] in the case of a grid-based implementation.

The pseudo-code shown in Algorithm 1 illustrates the main steps performed by each node in the distributed framework. This procedure is further optimized for minimizing communications; indeed, data are exchanged only for the methods in which missing/invalid values are reported. The subsequent phases of normalize_diagonal and normalize_extrema do not require any further communication among nodes.

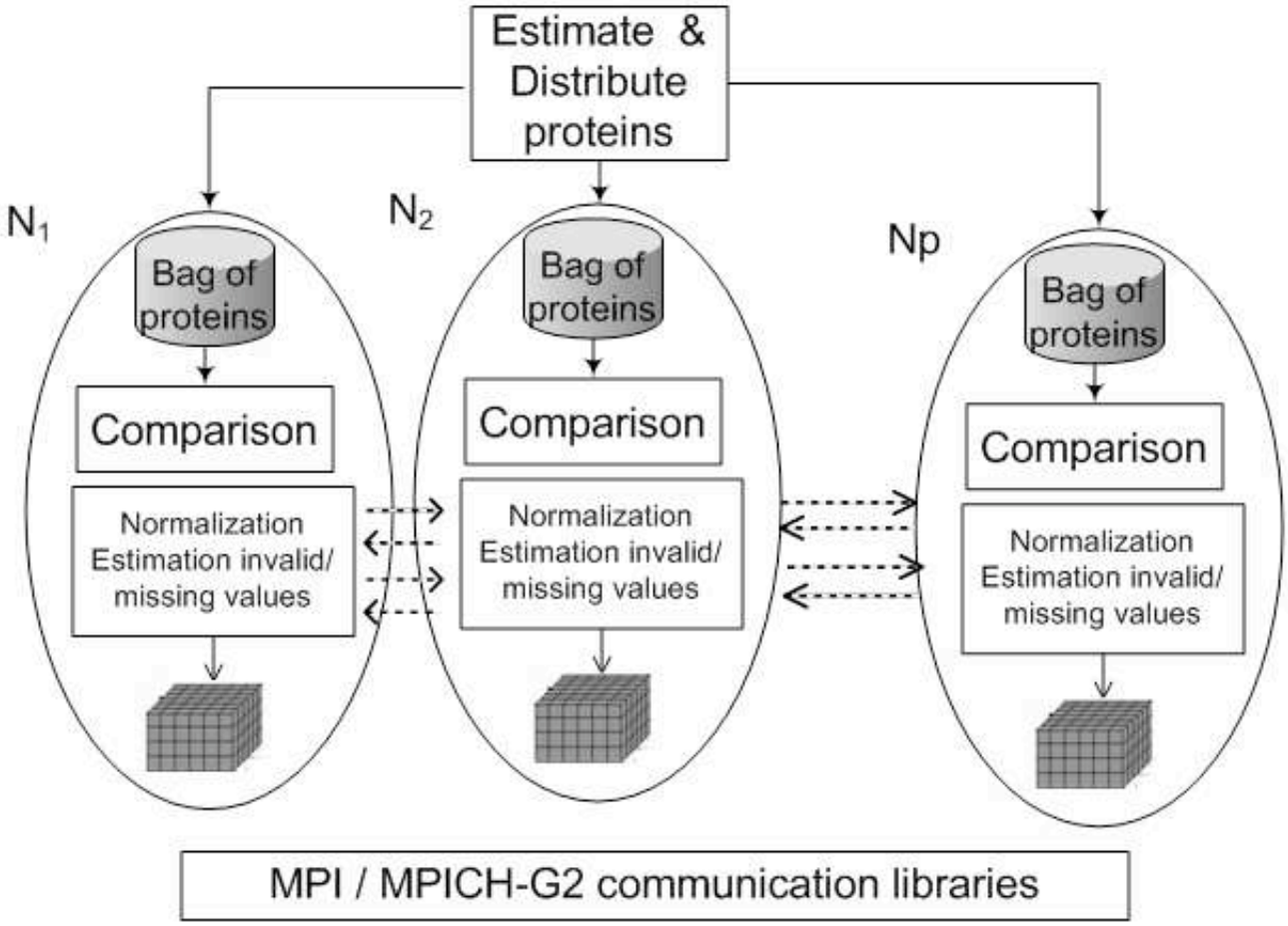


Fig. 2

SOFTWARE ARCHITECTURE OF THE DISTRIBUTED FRAMEWORK.

Algorithm 1 Pseudo-code executed from each node x concerning the multi-comparison part and the normalization/replacing invalid missing values part. Line 1 iterates for each method, with m representing the total number of methods.

```
1: for all method k such that 1 ≤ k ≤ m do
2:    for all protein i in row (x) do
3:       for all protein j in column (x) do
4:          compute_method k on the couple of proteins i and j {on node x}
5:       end for
6:    end for
7: end for
8: for all k such that 1 ≤ k ≤ m do
9:    find_local_min
10:   find_local_max
11: end for
12: exchange and find all global min and max {MPI_Allreduce routine}
13: replace_invalid_missing_values
14: normalize_diagonal
15: normalize_extrema
```

A. Decomposition Strategies

The efficiency of the distributed framework strongly depends on the way in which proteins are assigned to compute nodes.

A good load balancing strategy should considerably reduce the execution time and the memory necessary to store the main matrix and other data structures necessary to the overall computation of MC-PSC.

Consider a set of resources (nodes of the clusters or machines on the grid) $N_1, N_2, \ldots, N_n$ and the main matrix ($proteins \times proteins \times methods$) storing the result of the computation and of the normalization (and estimating invalid/missing values) phases. Let p be the total number of proteins and m the total number of methods computed. Note that, indeed, M indicates the total number of **indices** computed by the different m methods; in fact, $M = \sum_{k=1}^{m} M_k$, where M_k is the number of indices computed by the method k (see Table II for complete nomenclature).

In order to distribute the overall M among the nodes, there may be as many possible partitioning

schemes as:

1) Comparison of *one pair* of proteins with *one method*. This will create $p \times p \times m$ jobs
2) Comparison of *one pair* of proteins with *all methods*. This will create $p \times p$ jobs
3) Comparison of *all pairs* of proteins with *one method*. This will create m jobs
4) Comparison of a *subset of pairs* of proteins with a *set/subset of methods*. This will create an optimal number of jobs based on the availability of nodes as well as the number of proteins in a given dataset.

Partitioning $1\ and\ 2$ will be too *fine-grained*, whereas 3 will be too *course-grained* to be considered for a large cluster/grid environment. Partitioning 4 on the other hand could be devised in an intelligent way to achieve better load-balancing.

We investigate the 4^{th} partitioning scheme by applying two different approaches. The first decomposition adopted is shown in figure 3. The main matrix that stores the results is decomposed among the available nodes along the two proteins axis, so each comparison among two proteins for all the methods is performed on the same node, better balancing the different methods. This decomposition is the more efficient in terms of inter-jobs communication overhead, as it minimizes the number of information exchanges amongst compute nodes. Furthermore, the matrix is perfectly partitioned as each node is responsible for the computation and storage of same number of proteins $\frac{p^2 m}{n}$. In the next subsection these results will be analyzed in more detail. However, initial experiments suggested that execution times for different couples of proteins can largely fluctuate (see table IV), making the load among the different nodes not really balanced.

A second strategy is to balance the total execution time per compute node rather than the number of pairwise comparisons. Thus, this strategy takes into account the inhomogeneities in the size of the proteins being compared and is shown in figure 4. In order to set up a bag of proteins having the same overall number of residues on each node, the following largely used strategy was followed. Consider the case of proteins to be assigned to the $\sqrt{n}$ row processors (but the procedure is analogous for the column processors). First of all, proteins are sorted by the number of residues. Then, they are assigned, from the longest to the shortest one, to the node having the current lowest sum of residues. This procedure is not really time consuming, as it requires $p\ log\ p$ for sorting the proteins and $p(\sqrt{n})^2 = pn$ for assigning them to the correct node. The same distribution obtained for the row is also chosen for the column, so that the order of rows is not different of that of the columns and the operation of normalization and removing invalid/missing values could be performed without other overheads.

Each of the two proposed load balancing approaches result in a different CPU and memory usage.

In what follows we analyze the benefits and drawbacks behind each of them. Unless otherwise stated, a given analysis/argument applies to both of the strategies. Henceforth, the first decomposition will be referred to as *even* and the second one as *uneven*.

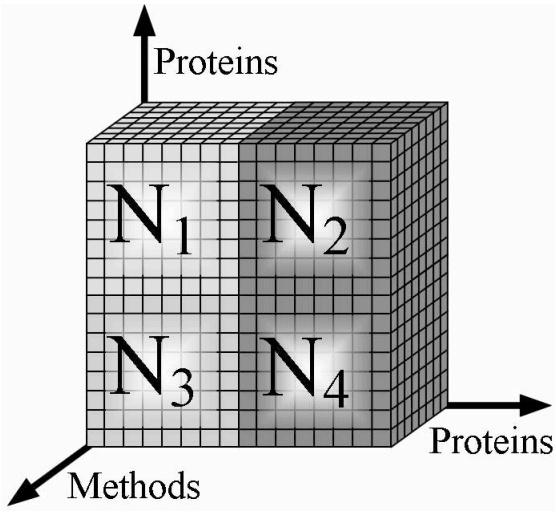


Fig. 3

EVEN DISTRIBUTION OF THE PROBLEM SPACE ($proteins \times proteins \times methods$). EACH NODE IS RESPONSIBLE FOR THE SAME COMPUTATION, I.E. SAME PORTION OF THE MATRIX)

.

B. Cost Analysis

Space analysis

In what follows we do not take into account transient memory requirements by the different methods (e.g. internal data structures) as these have, on the one hand, been already analyzed in the paper where each method was originally introduced and, on the other hand, these transient space requirements are released as soon as a particular pairwise comparison is done. The nomenclature used in our analysis is summarized in Table II.

The entire matrix, storing the comparison/normalization results, is decomposed along each of the two proteins axis among n nodes. So, in the case of even distribution, each node handles a matrix of size $\frac{p^2 m}{n}$ and of size $= max(row\_prot_x \times col\_prot_x) \times m$ for the uneven distribution, where p is the number of proteins, n the number of nodes, m the total number of computed methods and row_prot_x and col_prot_x

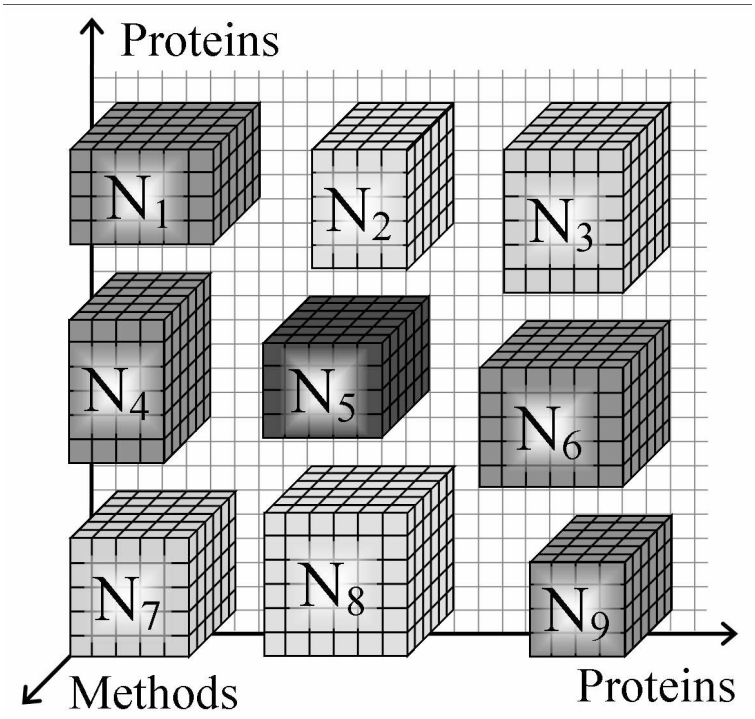


Fig. 4

UNEVEN DISTRIBUTION OF THE PROBLEM SPACE ($proteins \times proteins \times methods$). NOTE THAT THE DIFFERENT SIZES TAKE DIFFERENT PROTEIN SIZES INTO ACCOUNT (E.G. ONE NODE ONLY FOR A FEW BIG PROTEINS, WHICH TAKE QUITE LONG TO CALCULATE; AND ONE NODE FOR MANY SMALLER PROTEINS, WHICH ARE QUICKER TO CALCULATE).

are respectively the proteins stored on the row and on the column of the matrix assigned to the node x. In this case the space $\rightarrow \frac{p^2 m}{n}$ if $max(row\_prot_x \rightarrow \frac{p}{n}$ and $max(col\_prot_x \rightarrow \frac{p}{n}$, i.e. almost the same number of proteins is stored on each node.

The space necessary to store the proteins is $\frac{p^2 m \overline{S}_p}{n}$ for even distribution and $max(row\_prot_x \times col\_prot_x) m \overline{S}_p$ for uneven distribution, where $\overline{S}_p$ is the average size of proteins. This is the worst case as in many cases row proteins and column proteins are overlapped.

Obviously, in the case of the uneven distribution, the memory space is balanced only if the number of proteins stored on a node are not much different from those stored in the others.

Time analysis

Let $\overline{Tm}$ be the average execution time of all the methods over all the couple of proteins and $\overline{Tm}_x$ be average execution time of all the methods over all protein pairs stored on node x.

	Description
p	Number of proteins
n	Number of nodes (processors)
m	Number of methods used (i.e. MaxCMO, FAST, etc..)
M_k	Number of indices computed by the method k
M	Total number of indices computed by all the methods
row_prot_x	Number of row proteins present on node x
col_prot_x	Number of col proteins present on node x
$Eval_x$	Number of evaluation conducted on node x ($row\_prot_x \times col\_prot_x$)
$\overline{S}_p$	average size of proteins
$\overline{Tm_x}$	average execution time of all the methods over all the couples of proteins stored on the node x
$\overline{Tm}$	average execution time of all the methods over all the couples of proteins

TABLE II

SYMBOLS USED FOR THE ANALYSIS.

So, only for the computation part of the algorithm, in a single node execution, the total execution time will be $T_s = p^2 \times \overline{Tm}$. As for the distributed case, formulation is not so simple as, depending on the distribution of the proteins, average execution times could be really different from node to node. In such case, in the even distribution the parallel execution time will be $T_p = \frac{p^2 \overline{Tm_x}}{n}$ and $T_p = max(row\_prot_x \times col\_prot_x \times \overline{Tm_x}) \leq max(row\_prot_x \times col\_prot_x) \times max(\overline{Tm_x})$ for the uneven distribution. So, one has to balance this product as to obtain a fair computation; in the case of even distribution only if $Tm_x \rightarrow Tm$ for each node, a balanced load is achieved.

Communication overhead

In addition to considering different execution times over different nodes, communication overheads must also be taken into consideration. This overhead happens in the first phase, when proteins are distributed over the nodes (using *MPI_Bcast* routine) and in the latter phase, when normalization and invalid/missing value replacement must be conducted (using *MPI_Allreduce* routine).

Moving the proteins to different nodes does not require an excessive time in comparison with the large computation time of the computing phase. Naturally, *even* decomposition needs slightly less overhead than *uneven* one as almost the same number of protein must be send to each node. The amount of data exchanged is, as discussed before, $\frac{p^2 \overline{S}_p}{n}$ for even distribution and $max(row\_prot_x \times col\_prot_x)\overline{S}_p$ for

uneven.

As for the normalization phase, we need to compute the global *minimum* and *maximum* for all the methods for a total of 2 m values exchanged. For the correction of invalid or missing values we need the *minimum* or *maximum* for each row and column and method in which we got an invalid value. Thus, in the worst case, we have to exchange $2n^2m$, but typically invalid values are found only for a few methods and not for many cells.

Although the same amount of information is exchanged by the two decomposition strategies, the communication overhead is higher for the *uneven* strategy. This is mainly due to the worse efficiency for collective communication in an environment in which there are different number of rows and columns for each processor.

C. Discussion

The two decomposition strategies adopted present different pros and cons. Although the even decomposition better utilises memory both in terms of cells of the matrix ($\frac{p^2m}{n}$) and proteins ($\frac{p^2\overline{S}_p}{n}$), it does not balance well the execution time on the different nodes, especially if, as usual, proteins have very different structures (or number of residues). On the contrary, the uneven distribution, paying the cost of a larger memory requirements ($max(row\_prot_x \times col\_prot_x) \times m$ for the matrix and $max(row\_prot_x \times col\_prot_x)\overline{S}_p$ for proteins), is the only approach usable for obtaining appreciable reduction in execution times for small-medium and not well balanced datasets of proteins.

V. Experimental Results and Discussions

Different experiments were conducted to validate the quality of the two decomposition strategies. Two metrics are usually used for testing the computational scalability of a parallel/distributed system: the speedup S and the efficiency E. The speedup of a parallel algorithm is the ratio between the time taken by the best sequential implementation of an application measured on one processor T_s and the execution time taken by the same application T_p running on p processors.

$$S = \frac{T_s}{T_p} \tag{2}$$

The optimal case is given by a linear speedup, i.e. If we run the same application on p processors, then we can expect at best a reduction in time of p, and therefore that the speedup will be at most p. In fact, this is only a theoretical condition because the parallel algorithm introduces an overhead, mainly due to the communication times among different processors. If the problem is not sufficiently complex,

and the communication times are not negligible with respect to computational time, then the speedup might be noticeably smaller. Efficiency is given by the ratio between the speedup S and the number of processors p:

$$E = \frac{S}{p} \tag{3}$$

and it represents an index of the fraction of time usefully spent by each processor. In this case, the highest value of efficiency (equals to 1) is attained when all the processors are utilized to the maximum (communication times and other overheads equal to zero).

A. Datasets and Test Suite

All the experiments were performed on a Linux cluster, named $spaci$ and placed at ICAR-CNR institute in Italy, with 64 dual-processors Itanium2 1.4GHz nodes each having 4GB of main memory and being connected by a Qsnet high performance network. In addition to this, some experiments (for the large dataset) were also conducted on the eScience infrastructure provided by the National Grid Service (NGS), UK. In this case we used MPIg [58] (grid-based implementation of MPI) to spawn the jobs across two NGS sites; one at Leeds and the other at Manchester. Each of these sites have 256 cores (AMD Opteron) with 2.6GHz and 8GB of main memory.

In our experiments, we used the first chain of the first model both for the Rost and Sander dataset (RS119) and for the Chew-Kedem (CK34) data set (see Table III for the characteristics of these datasets). As an example of a large dataset, we used the one proposed by Kinjo et al. [59]. This dataset has been prepared by using PDB-REPRDB [59] algorithm to select 1012 non-redundant protein chains. The length of each chain in this dataset is greater than 50 with a sequence identity less than 30%. Furthermore, the dataset does not contain any chain with non-standard residues or chain breaks and all of its chains have resolution better than 2 Å and R factor better than 20%.

B. Scalability of the Even Decomposition

To evaluate the quality of the even decomposition, the previously introduced metrics of scalability and efficiency were used, together with the execution time on different numbers of processors. The speed-up values obtained for the two medium datasets CK34 and RS119 are shown in figure 5.

For both datasets, the speed-up remains good using up to 16 processors, but using more processors does not help to speed up the total execution time to the same degree. This is due to the structural

Dataset	# Chains per Datasets	# Comparisons per Datasets	# Residues per Datasets
CK34 [60]	34	1,156	6,102
RS119 [61]	119	14,161	23,053
Kinjo et al. [59]	1012	1,024,144	252,569

TABLE III

OVERVIEW OF THE DATASETS USED IN THE EXPERIMENTS. THE HASH SYMBOL (#) IS AN ABBREVIATION FOR *Number of*

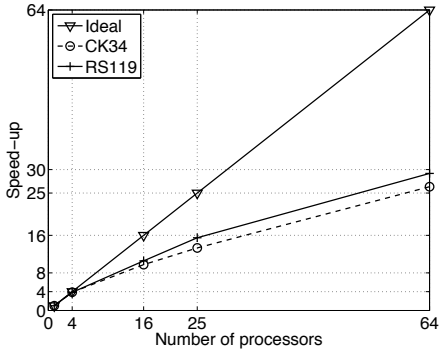


Fig. 5

SPEEDUP OF THE EVEN DECOMPOSITION USING THE CK34 AND RS119 DATASETS ON *spaci* CLUSTER.

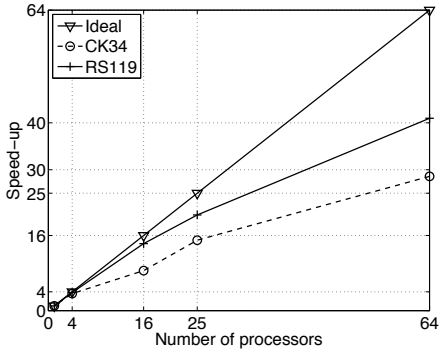


Fig. 6

SPEEDUP OF THE UNEVEN DECOMPOSITION USING THE CK34 AND RS119 DATASETS ON *spaci* CLUSTER.

differences of the proteins, as each protein is composed by a different number of residues. Indeed, in spite of having the same number of proteins on each node, some proteins could have a large number of residues on a node and a few on another one. This consideration is confirmed by the large variance in the execution times of the different methods (Table IV). As for the execution time, for the RS119 and the CK34 dataset, the entire execution time was reduced respectively from about 6 days and 6.2 hours, using the sequential implementation on one machine, to 4.8 hours and 14.15 minutes on 64 processors. However, on 64 processors, the efficiency degrades to the values of 41% and 46% respectively for CK34 and RS119.

TABLE IV

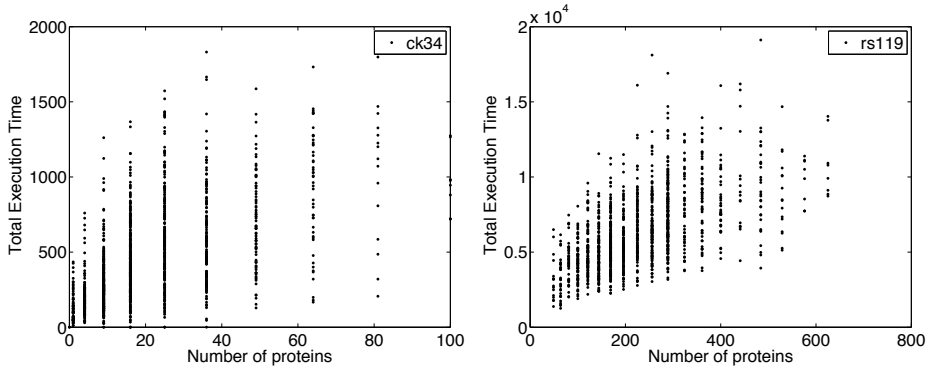
TOTAL NUMBER OF RESIDUES AND AVERAGE NUMBER OF CHAINS PER DATASET AND AVERAGE EXECUTION TIMES AND STANDARD DEVIATION (MINUTES) OF THE DIFFERENT METHODS FOR THE CK34 AND RS119 DATASETS AVERAGED OVER 60 TRIES.

Dataset	# Res. per Datasets	# Res. per Chain	USM	FAST	TM-ALign	Dali	CE	MaxCMO
CK34	6102	179	0.52 ± 0.28	0.14 ± 0.07	0.28 ± 0.11	3.49 ± 1.53	3.20 ± 0.66	0.99 ± 0.34
RS119	23053	197	3.68 ± 0.31	2.16 ± 1.05	5.78 ± 2.86	44.59 ± 20.51	41.05 ± 20.41	20.13 ± 9.69

C. Empirical Analysis of the Load Balancing Factors

It is important to understand whether the execution times of the different methods described in the previous sections depends on the number of proteins, on the numbers of residues, or on both of them. To this end we randomly divided the proteins composing the two datasets CK34 and RS119 among 64 nodes (a 8x8 grid of processors) and we run all the available methods and measured the execution time, the overall number of residues and of proteins present on each node. This procedure was repeated for 20 times for a total of 1280 different measures of time.

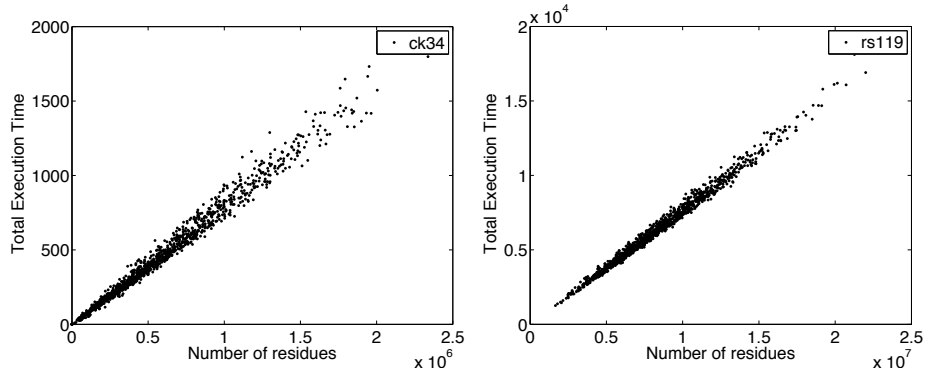
Then, we plotted the execution time vs the number of proteins (figures 7 a and b) and the execution time vs the overall number of residues (figures 8 a and b). Observing the figures, it is clear that the execution time depends mainly on the overall number of residues present on a node, i.e. the dependence of time as a function of residues number is nearly linear, while it does not exhibit a linear dependence on the number of proteins. The largely used *Pearson product-moment correlation coefficient* (PMCC) was computed to better assess the dependency between time and residues versus the time and proteins. In the first case, we obtained a coefficient of 0.992 and 0.995 respectively for the *CK34* and *RS119* dataset, while in the latter case we obtained only 0.582 and 0.585 for the same two datasets.



(a) (b)

Fig. 7

EXECUTION TIME VS NUMBER OF PROTEINS PRESENT ON THE NODE FOR THE (A) CK34 AND (B) RS119 DATASET



(a) (b)

Fig. 8

EXECUTION TIME VS NUMBER OF RESIDUES PRESENT ON THE NODE FOR THE (A) CK34 AND (B) RS119 DATASET

Further analysis aimed to explore whether this linear dependence was influenced by one or more slowest methods or is verified for all the methods. Figures 9 show the execution time vs the number of residues for each method for CK34 (a and b) and RS119 (c and d). Although FAST, USM and MacCMO perform faster as compared to Dali, CE, TM-Align, however the dependency is quite evident for each of them. Also considering PMCCs, they are always higher than 0.95 for the residue case and lower than 0.60 for the protein case. The only exception is the USM method that obtained a value of 0.914 and 0.888 (respectively for CH34 and RS119) for residues and a value of 0.667 and 0.6958 for the protein

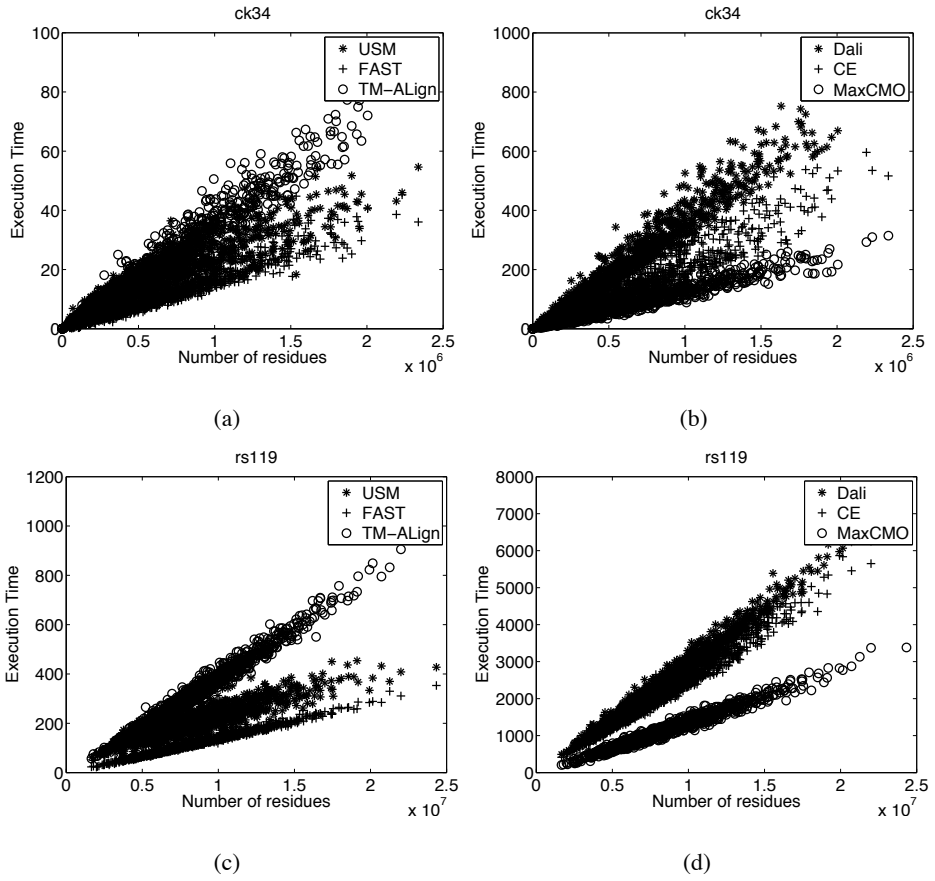


Fig. 9

EXECUTION TIME VS NUMBER OF RESIDUES PRESENT ON THE NODE FOR THE DIFFERENT METHODS FOR THE (A) CK34 AND (B) RS119 DATASET

case.

D. Scalability of the Uneven Decomposition

From the previous section, it is clear that the execution time strongly depends on the number of residues per node. Thus, scalability experiments for the same dataset as the *even* distribution were also conducted with *uneven* decomposition and results are reported in figure 6. For the RS119 (CK34) dataset, the entire execution time was reduced from about 6 days (6.2 hours), using the sequential implementation on one machine, to 3.4 hours (11.65 min.) on 64 processors. In comparison with the even strategy, we obtained an improvement on 64 processors of about 18% for the CK34 dataset and of about 29% for the RS119 dataset. Furthermore, on 64 processors, the efficiency is maintained at a quite good value of 64% for

RS119. For the CHK34, we obtained a value of 50% that is not a bad result, given the small grain of the dataset.

E. A further experiment on a large dataset

The last experiment was performed using the the uneven strategy running on 4, 16, 25 and 64 processors applied to the Kinjo dataset, comprising of 1012 non-redundant protein chains. Using this dataset, the algorithm performed about 1 million of comparisons for all the methods.

As the execution time on a single processor is extremely large, this case was not considered, instead, scalability was measured based on an estimated base line on 4 processors runing the faster of all the methods, namely, the FAST algorithm. For reference note that FAST takes approximatedly 11 days to execute on a single processor for such a large number of proteins.

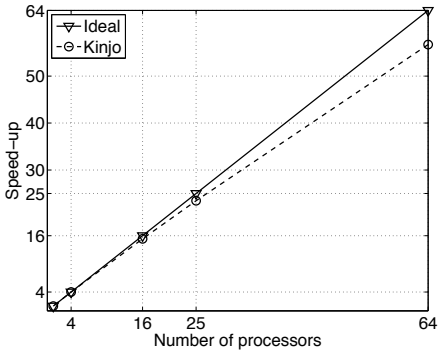


Fig. 10

SPEEDUP OF THE UNEVEN DECOMPOSITION USING THE KINJO DATASET ON *spaci* CLUSTER.

The execution time of the algorithm applied to this huge dataset was reduced from 152 days on 4 processors, to 39.7 days on 16 and finally to 10.7 days on 64 processors. Obviously the scalability obtained is very close to the ideal case, as you can see in figure 10. In fact, on 64 processor, respectively a scalability value of 57 and an efficiency value of 89% were measured.

In order to investigate further scalability of this dataset, some more experiments were conducted on the NGS infrastructure and the results are ... [**TO BE WRITTEN**]

VI. CONCLUSIONS

A high-throughput/grid-aware distributed Protein structure comparison framework for very large datasets is proposed, based on an innovative distributed algorithm running both in a cluster and grid environment.

This framework is able to perform structure comparison using all or a selection of the available methods. The design of this algorithm have been analyzed in terms of space, time, and communication overhead. Based on this analysis two different load balancing approaches have been used to improve the overall performance: *even* and *uneven* strategies. The former permits to obtain the best distribution in terms of memory, while the latter performs better in terms of execution time and scalability on cluster computers. Experiments conducted on medium and large real datasets prove that the algorithm permits to reduce execution time (i.e. for the RS119 dataset it was reduced from 6 days on a single processor to about 5 hours on 64 processors) and to cope with problems otherwise not tractable on a single machine as the Kinjo dataset, which took about 11 days on a 64-processors cluster. ~~[NGS RESULTS TO BE WRITTEN HERE].~~ In the future, we intend to investigate in more depth the use of grid computing environments in order to cope with very large proteomics datasets.

VII. Acknowledgments

The authors would like to thank the ICAR-CNR institute and particularly Gennaro Oliva for providing assistance and useful information in using the SPACI cluster. We would also like to acknowledge the use of the UK National Grid Service ~~in further carrying out this work.~~

References

[1] T.-S. Mayuko, T. Daisuke, C. Chieko, T. Hirokazu, and U. Hideaki, "Protein structure prediction in structure based drug design," *Current Medicinal Chemistry*, vol. 11, pp. 551–558, 2004.

[2] A. Zemla, C. Venclovas, J. Moult, and K. Fidelis, "Processing and analysis of casp3 protein structure predictions," *Proteins Struct Funct Genet*, vol. Suppl 3, pp. 22–29, 1999.

[3] D. Kihara, Y. Zhang, H. Lu, A. Kolinski, and J. Skolnick, "Ab initio protein structure prediction on a genomic scale: Application to the mycoplasma genitalium genome," *PNAS*, vol. 99, pp. 5993–5998, 2002.

[4] Y. Zhang and J. Skolnick, "Automated structure prediction of weakly homologous proteins on a genomic scale," *PNAS*, vol. 101, p. 75947599, 2004.

[5] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia, "Scop: a structural classification of proteins database for the investigation of sequences and structures," *J Mol Biol*, vol. 247, pp. 536–540, 1995.

[6] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton, "Cath - a hierarchic classification of protein domain structures," *Structure*, vol. 5, pp. 1093–1108, 1997.

[7] S.-Y. Huang and X. Zou, "Efficient molecular docking of nmr structures: Application to hiv-1 protease," *Protein Sci.*, vol. 16, pp. 43–51, 2007.

[8] A. Williamson, "Creating a structural genomics consortium," *Nat Struct Biol*, vol. 7, 2000.

[9] B. Matthews, "Protein structure initiative: getting into gear," *Nat Struct Mol Biol*, vol. 14, pp. 459–60, 2007.

[10] "Proteomics' new order," *Nature*, vol. 437, pp. 169–70, 2005.

[11] H. Berman, K. Henrick, and H. Nakamura, "Announcing the worldwide protein data bank," *Nat Struct Biol*, vol. 10, p. 980, 2003.

[12] R. Kolodny, P. Koehl, and M. Levitt, "Comprehensive evaluation of protein structure alignment methods: Scoreing by geometric measures," *J Mol Biol*, vol. 346, pp. 1173–1188, 2005.

[13] D. Barthel, J. Hirst, J. Blazewicz, E. K. Burke, and N. Krasnogor, "The ProCKSI server: a decision support system for protein (structure) comparison, knowledge, similarity and information," *BMC Bioinformatics*, vol. 8, p. 416, 2007.

[14] D. A. Pelta, J. R. Gonzalez, and M. V. M., "A simple and fast heuristic for protein structure comparison," *BMC Bioinformatics*, vol. 9, p. 161, 2008.

[15] L. Holm and J. Park, "Dalilite workbench for protein structure comparison," *Bioinformatics*, vol. 16, pp. 566–567, 2000.

[16] I. Shindyalov and P. Bourne, "Protein structure alignment by incremental combinatorial extension (ce) of the optimal path," *Protein Eng*, vol. 11, pp. 739–747, 1998.

[17] Y. Zhang and J. Skolnick, "Tm-align: A protein structure alignment algorithm based on tm-score," *Nucleic Acids Res*, vol. 33, pp. 2302–2309, 2005.

[18] N. Krasnogor and D. A. Pelta, "Measuring the similarity of protein structures by means of the universal similarity metric," *Bioinformatics*, vol. 20, pp. 1015–1021, 2004.

[19] J. Zhu and Z. Weng, "Fast: A novel protein structure alignment algorithm," *Proteins Struct Funct Bioinf*, vol. 58, pp. 618–627, 2005.

[20] T. O, "On the parallelisation of bioinformatics applications," *Briefings in Bioinformatics*, vol. 2, pp. 181–194, 2001.

[21] A. SF and S. A. Madden TL, "Gapped blast and psi-blast: A new generation of protein db search programs," *Nucleic Acids Res.*, vol. 25, pp. 3389–3402, 1997.

[22] P. W. R. and L. D. J., "Improved tools for biological sequence comparison," *Proc. Natl Acad. Sci.*, vol. 85, pp. 2444–2448, 1988.

[23] "The bio-accelerator." [Online]. Available: http://sgbcd//weizmann.ac.il/

[24] R. K. Singh, W. D. Dettlo, V. L. Chi, D. L. Homan, S. G. Tell, C. T. White, S. F. Altschul, and B. W. Erickson, "Bioscan: A dynamically recongurable systolic array for biosequence analysis," in *Proc. of CERCS96, National Science Foundation*, 1996.

[25] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing*, ser. LNCS 3779, 2005, pp. 2–13.

[26] A. Shah, D. Barthel1, P. Lukasiak, J. Blacewicz, and N. Krasnogor, "Web and grid technologies in bioinformatics, computational biology and systems biology: A review," *Current Bioinformatics*, vol. 3, no. 1, pp. 10–31, 2008.

[27] A. Shah, D. Barthel, and N. Krasnogor, "Grid and distributed public computing schemes for structural proteomics," in *Frontiers of High Performance Computing and Networking ISPA 2007 Workshops*, ser. LNCS 4743. Berlin:Springer, 2007, pp. 424–434.

[28] S. Pellicer, G. Chen, K. C. C. Chan, and Y. Pan, "Distributed sequence alignment applications for the public computing architecture," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 7, pp. 35–43, 2008.

[29] W.-L. Chang, "Fast parallel dna-based algorithms for molecular computation: The set-partition problem," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 6, pp. 346–353, 2007.

[30] I. Merelli, G. Morra, and L. Milanesi, "Evaluation of a grid based molecular dynamics approach for polypeptide simulations," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 6, pp. 229–234, 2007.

[31] M. Mirto, M. Cafaro, S. L. Fiore, D. Tartarini, and G. Aloisio, "Evaluation of a grid based molecular dynamics approach for polypeptide simulations," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 6, pp. 124–130, 2007.

[32] A. Arbona, S. Benkner, G. Engelbrecht, J. Fingberg, M. Hofmann, K. Kumpf, G. Lonsdale, and A. Woehrer, "A service-oriented grid infrastructure for biomedical data and compute services," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 6, pp. 136–141, 2007.

[33] M. Cannataro, A. Barla, R. Flor, G. Jurman, S. Merler, S. Paoli, G. Tradigo, P. Veltri, and C. Furlanello, "A grid environment for high-throughput proteomics," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 6, pp. 117–123, 2007.

[34] A. Boccia, G. Busiello, L. Milanesi, and G. Paolella, "A fast job scheduling system for a wide range of bioinformatic applications," *IEEE TRANSACTIONS ON NANOBIOSCIENCE*, vol. 6, pp. 149–154, 2007.

[35] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, p. 403410, 1990.

[36] H. Lin, P. Balaji, R. Poole, C. Sosa, X. Ma, and W. chun Feng, "Massively parallel genomic sequence search on the blue gene/p architecture," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–11.

[37] A. J., W. chun Feng, and T. E., "A pluggable framework for parallel pairwise sequence search," in *29th Annual International Conference of the IEEE*, 2007, pp. 127–130.

[38] O. C. and N. J., "Scalablast: A scalable implementation of blast for high-performance data-intensive bioinformatics analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, pp. 740– 749, 2006.

[39] A. Krishnan, "Gridblast: a globus-based high-throughput implementation of blast in a grid computing framework," *Concurrency and Computation: Practice and Experience*, vol. 17, pp. 1607–1623, 2005.

[40] A. E. Darling, L. Carey, and W. chun Feng, "The design, implementation, and evaluation of mpiblast," in *In Proceedings of ClusterWorld 2003*, 2003.

[41] R. L. D. C. Costa and S. Lifschitz, "Database allocation strategies for parallel blast evaluation on clusters," *Distrib. Parallel Databases*, vol. 13, no. 1, pp. 99–127, 2003.

[42] R. Braun, K. Pedretti, T. Casavant, T. Scheetz, C. Birkett, and C. Roberts, "Parallelization of local blast service on workstation clusters," *Future Generation Computer Systems*, vol. 17, p. 745754, 2001.

[43] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, Sep. 1996.

[44] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "Genbank," *Nucleic Acids Res.*, vol. 17, p. D25D30, 2008.

[45] R. Martino, C. Johnson, E. Suh, B. Trus, and T. Yap, "Parallel computing in biomedical research," *Science*, vol. 265, pp. 902–908, 1994.

[46] O. Trelles-Salazar, E. Zapata, and J. Carazo, "On an efficient parallelization of exhaustive sequence comparison algorithms on message passing architectures," *Bioinformatics*, vol. 10, pp. 509–511, 1994.

[47] C. Ferrari, C. Guerra, and G. Zanottib, "A grid-aware approach to protein structure comparison," *J. Parallel Distrib. Comput.*, vol. 63, pp. 728–737, 2003.

[48] S.-J. Park and M. Yamamura, "Frog (fitted rotation and orientation of protein structure by means of real-coded genetic algorithm) : Asynchronous parallelizing for protein structure-based comparison on the basis of geometrical similarity," *Genome Informatics*, vol. 13, pp. 344–345, 2002.

[49] M. Cannataro, M. Comin, C. Ferrari, C. Guerra, A. Guzzo, and P. Veltri, "Modeling a protein structure comparison

application on the grid using proteus," in *Scientific Applications of Grid computing (SAG2004)*, ser. LNCS 3458. Berlin:Springer, 2004, pp. 75–85.

[50] "ihop: Information hyperlinked over proteins." [Online]. Available: http://www.ihop-net.org

[51] "Scop: Structural classification of proteins." [Online]. Available: http://scop.mrc-lmb.cam.ac.uk/scop

[52] F. M. G. Pearl, C. F. Bennett, J. E. Bray, A. P. Harrison, N. Martin, A. Shepherd, I. Sillitoe, J. Thornton, and C. A. Orengo, "The cath database: an extended protein family resource for structural and functional genomics," *Nucleic Acids Res*, vol. 31, pp. 452–455, 2003.

[53] D. A. Pelta, N. Krasnogor, C. Bousono-Calzon, J. L. Verdagay, J. D. Hirst, and E. Burke, "A fuzzy sets based generalization of contact maps for the overlap of protein structures," *Fuzzy Sets and Systems*, vol. 152, pp. 102–123, 2005.

[54] O. Camoglu, T. Can, and A. Singh, "Integrating multi-attribute similarity networks for robust representation of the protein space," *Bioinformatics*, vol. 22, pp. 1585–1592, 2006.

[55] F. V and S. S, "Heterogeneous data integration with the consensus clustering formalism," in *1st International Workshop on Data Integration in the Life Science (DILS)*, ser. LNCS 2994. Berlin:Springer, 2004, pp. 110–123.

[56] C. A. Rohl, C. E. M. Strauss, K. M. S. Misura, and D. Baker, "Protein Structure Prediction Using Rosetta," in *Numerical Computer Methods, Part D*, ser. Methods in Enzymology, L. Brand and M. L. Johnson, Eds. Academic Press, Jan. 2004, vol. Volume 383, pp. 66–93.

[57] S. Wu, J. Skolnick, and Y. Zhang, "Ab initio modeling of small proteins by iterative TASSER simulations." *BMC Biol*, vol. 5, no. 1, p. 17, May 2007.

[58] N. T. Karonis, B. Toonen, and I. Foster, "Mpich-g2: a grid-enabled implementation of the message passing interface," *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 551–563, 2003.

[59] A. R. Kinjo, K. Horimoto, and K. Nishikawa, "Predicting absolute contact numbers of native protein structure from amino acid sequence," *Proteins Struct Funct Bioinf*, vol. 58, pp. 158–165, 2005.

[60] L. P. Chew and K. Kedem, "Finding the consensus shape for a protein family," in *Proceedings of the 18th Annual Symposium on Computational Geometry (SCG)*. New York: Springer, 2002, pp. 64–73.

[61] B. Rost and C. Sander, "Prediction of protein secondary structure at better than 70% accuracy," *J Mol Biol*, vol. 232, pp. 584–599, 1993.