Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 391–397
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Algorithmic re-structuring and data replication for protein structure comparison on a GRID☆

G. Ciriello[a], M. Comin[a,*], C. Guerra[a,b]

[a] *Department of Information Engineering, University of Padova, Italy*
[b] *College of Computing, Georgia Tech, USA*

Received 30 January 2006; accepted 26 March 2006
Available online 4 August 2006

Abstract

This paper describes a major restructuring of PROuST, a method for protein structure comparison, for an efficient porting to the Grid. PROuST consists of different components: an index-based search that produces a list of proteins that are good candidates for similarity, and a dynamic programming algorithm that aligns the target protein with each candidate protein. Both components use the same geometric properties of secondary structure elements of proteins. Thus, an important issue arises when porting the application to the Grid, i.e. the tradeoff between data transfer and data recomputation. Our restructured application avoids recomputation by re-using the data as much as possible, once they are accessed. The algorithmic changes to PROuST allow to reduce the number of data accesses to storage elements and consequently the execution time. This paper also discusses data replication policies on a Grid environment to optimize the data transfer time.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Protein similarity; Dynamic programming; Data management; Distributed applications; Grid computing

1. Introduction

Comparing protein structures is important for protein classification and for understanding protein functions. It is a very active research area in computational biology and bioinformatics [12,9]. We have developed a method PROuST [6] that allows efficient retrieval of similarity information from a database containing all protein structures of the Protein Data Bank PDB [1]. This paper discusses issues arising when porting the application to the Grid and presents a major restructuring of PROuST for an efficient porting. PROuST consists of two main components: an index-based search component that generates a list of candidates proteins for a target protein, and a subsequent refinement component that performs pairwise comparisons and alignments by dynamic programming (DP).

The index-based search considers geometric properties of the secondary structure elements (SSE) of the proteins, i.e. α-helices and β-strands. It computes for all triplets of SSE the angles formed by the three pairs of linear segments associated to the SSE and uses them as indexes to a three-dimensional (3D) hash table. The hash table allows to retrieve a list of proteins that are good candidates for similarity with the target protein, without the need to examine separately every single protein of the PDB. A pairwise structure comparison aligns the target protein with each protein of the candidate list. By combining these two approaches, PROuST achieves good results in terms of robustness and agreement with existing classifications of protein structures. In addition its time performance compares well with that of other existing approaches. However, the amount of computation and data involved is quite high, given the current size of the PDB of more than 33,000 structures. For an exhaustive analysis of classification accuracy and time performance we refer the reader to [6].

The new version of PROuST integrates the index-based search and the dynamic programming algorithm, and requires some major changes to the algorithms and data structures resulting in a more efficient solution. It exploits the fact

☆ An extended abstract related to this work is included in the IEEE Proceedings of High-performance Computational Biology HICOMB 2006.

* Corresponding author. Tel.: +39 049 8277928.
E-mail addresses: ciriello@dei.unipd.it (G. Ciriello), ciompin@dei.unipd.it (M. Comin), guerra@dei.unipd.it (C. Guerra).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.03.029

that the same geometric properties of secondary structure elements of proteins, angles and distances, are used by the two main components of PROuST. Obviously, an important issue of the distributed implementation is data transfer vs. data recomputation tradeoffs. Our solution avoids recomputation by re-using the hash table data as much as possible, once they are accessed.

The emerging Grid technology [8] is becoming an important aspect for the solution of compute intensive problems. The computational Grid enables the use of a large number of different machines acting as a single one, by sharing both storage capacity and computing power. The importance of sharing data and resources in a secure manner is proved by the increasing interest of scientists towards this technology, especially in the biomedical community, that includes bioinformatics and medicine [11]. The Grid.it project [2], within which this work has been done, is aimed at developing a middleware layer enabling science in areas such as High Energy Physics, Earth Observation and Biomedicine. Biomedical applications have very challenging requirements in terms of computational power and amount of data. This paper focuses on a structural bioinformatics application on a Grid and deals with the issues of data management and algorithmic enhancement for an efficient porting of the application to the Grid. In addition, it discusses data replication policies to optimize the data transfer time. Replica management is a crucial aspect of a gridifying strategy [3,10], because the availability of data close to the Grid nodes where the job requesting the data runs enables latency reduction and efficient access.

The paper is organized as follows. In the next section we review the PROuST method. In Section 3 we describe a restructuring of the application that reduces the number of data accesses and avoids recomputation and in Section 4 we present its distributed implementation. In Section 5 the porting of the application to the Grid is discussed. Time performance is analyzed in Section 6. This work ends with conclusions in Section 7.

2. PROuST: An all-to-all protein structure comparison method

PROuST [6] combines different techniques that allow fast retrieval of similarity information from a database containing all the protein structures of the PDB. Proteins from the PDB are represented by linear segments associated to their SSE, i.e. α-helices and β-strands. In our internal representation, the proteins from PDB are encoded by a set of files, each consisting of the list of the SSE segments of a protein. Starting from these files, all triplets of SSE are computed, and the angles formed by the three pairs of segments of a triplet are used as indexes to a 3D hash table, where the triplets are stored. Once the hash table is built, an entry or cell contains the triplets of all proteins characterized by similar dihedral angles. The hash tables are accessed to retrieve very efficiently the proteins most similar to a given target protein. As a result, a list of candidate proteins is generated. The candidate list varies in length, but, typically, for a protein with an average number of SSE (i.e. 12-13 SSE) it consists of more than 5000 proteins.

The second component of PROuST is a pairwise protein structure comparison which aligns the target protein with each candidate protein of the list. It uses as input the internal representation of the proteins, and returns a similarity score according to which the candidate proteins are re-ranked into the final similarity list. The alignment is obtained by DP. If P is the target and Q a candidate protein, with SSE segments $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$, respectively, DP finds the associations (p_i, q_j) that maximize a given similarity score and also satisfy the continuity constraints, i.e. if (p_i, q_j) and (p_h, q_k) are two pairs of associated SSE and $i < h$ then $j < k$. The DP determines the optimal non-decreasing path in a 2D score matrix M. The main characteristic of our algorithm, that distinguishes it from other DP algorithms applied to SSE alignment, is that the score matrix is built from geometric properties of triplets of secondary structures, the same properties used by the indexing procedure. The score of the pair (p_i, q_j), stored at $M(i, j)$, is given by the number of times the pair (p_i, q_j) occurs in any two equivalent triplets of SSE segments of P and Q. Two triplets (p_u, p_v, p_z) and (q_r, q_s, q_t) are *equivalent* if they have similar angle and distance values. Two equivalent triplets determine three candidate pairs of corresponding segments: (p_u, q_r), (p_v, q_s), and (p_z, q_t), that contribute to the entries $M(u, r)$, $M(v, s)$, $M(z, t)$ of score matrix M, whose values are incremented by one. The DP uses as input the internal vectorial representation of the two proteins to generate the score matrix, based on which the similarity score is determined. As we will see in the next section, the construction of the score matrix M is the step most affected by the restructuring of PROuST.

Even though the DP computation for a pair of proteins is quite fast, and the list of proteins has been reduced from the 33,000 proteins of the PDB to a few thousand candidates, the amount of time required by all DP alignments may still be quite high. We remark here that often only a set of representative proteins is selected from the PDB for structure comparison, in such cases obviously the time requirements go down considerably. We choose to search the entire PDB because we want to discover non-trivial similarities, not only related to the standard family classification.

3. Re-structuring PROuST

Here we describe a major restructuring of our application for an efficient porting to the Grid. As already mentioned, PROuST uses the same geometric properties of SSE, angles and distances, in different components. An important issue arises in the distributed implementation of the pairwise structural alignments by DP, that is whether to exploit and reuse the information that has been already computed and stored in the hash table or recompute it from the internal representation of the candidate proteins.

The recomputation of the angles and distances of segments for all candidate proteins requires access to many small files each containing the vectorial representation (list of SSE segments) of a protein. In a local scenario, recomputation may be more cost-effective, due to the large size of the hash table

and large number of elements in each table cell. With more than 30,000 proteins, the number of triplets in the hash table is above 40,000,000. Consequently, the access time for the table is relatively large. Moving from a local to a distributed system data transfers become a crucial aspect. We propose a solution that avoids recomputation by re-using the hash table data as much as possible, once they are accessed. It combines indexing and structural alignment: the index-based search in addition to generating a candidate list of proteins also builds the score matrices to be used later by the pairwise structural comparisons. The internal vectorial representation of the proteins is no longer needed because the angles and distances of segments are not recomputed. This results in a computational advantage.

The new version of PROuST consists of the following steps:

(1) **Pre-processing phase:**
- Hash-table construction and updates.

(2) **Index-based structural alignment:**
- Search for similarities of the target protein with all proteins stored in the hash tables and build the score matrices.
- Use the DP algorithm to obtain the optimal path in each score matrix and the corresponding optimal structural alignment.

(3) **Atomic pairwise protein superimposition:**
- on user's demand.

The following is a sketch of the unified procedure that, given the target protein P, returns the list $\mathcal{C}$ of candidate proteins and the list $\mathcal{L}$ of score matrices. We denote by M_Q the score matrix for the pair of proteins P and Q. M_Q is created only if protein Q has at least one triplet of SSE equivalent to a triplet of P.

Unified Procedure

Step 1.

Initialize the list $\mathcal{L}$ and $\mathcal{C}$ to empty.

Step 2.

Consider all triplets of SSE of P and for each such triplet (p_u, p_v, p_z), with $u < v < z$, do the following:

(i) Compute the angles $\alpha_{uv}, \alpha_{vz}, \alpha_{uz}$ and the distances d_{uv}, d_{vz}, d_{uz} of the three pairs of segments. Access the hash table at the cell indexed by the three quantized angles.

(ii) If the cell is not empty, scan all triplets of SSE stored in the cell.

Let (q_r, q_s, q_t) be one such triplet, $r < s < t$, with distances h_{rs}, h_{st}, h_{rt}, and Q the protein containing q_r, q_s, and q_t.

if the distances of the two triplets are within a given threshold TD, i.e.

$|d_{uv} - h_{rs}| < TD$, $|d_{vz} - h_{st}| < TD$ and $|d_{uz} - h_{rt}| < TD$ **then**

(ii.a) **Index-based Search**

Cast a vote to protein Q and insert it into list $\mathcal{C}$ if not present.

(ii.b) **Score matrix building**

If $\mathcal{L}$ does not contain an entry for protein Q, create a new score matrix M_Q, initialized to 0, and insert it into $\mathcal{L}$.

Update M_Q as follows:

$\{M_Q(u, r) = M_Q(u, r) + 1;\ M_Q(v, s) = M_Q(v, s) + 1;\ M_Q(z, t) = M_Q(z, t) + 1;\}$

End.

The details of the voting process are omitted here (see [7]). The above procedure builds all score matrices at the same time; with a protein of average size, this may imply the creation and update of more than 5000 data structures. To guarantee reasonable storage requirements, we use an ad hoc dynamic data structure for storing a matrix, called *Dynamic Matrix* (DyM). A score matrix is represented as a linked list of column vectors each of length n (n being the number of SSE of P). A vector is associated to an SSE q of the candidate protein Q and its ith element is the score of q with the ith SSE of P. A column vector is inserted into the column vector list only if there is at least a triplet of SSE containing q that is equivalent to one triplet of P. The order of the columns in the linked list representation of the matrix follows the sequential order of the SSE along the backbone chain. This is a requirement of the DP algorithm.

4. A distributed implementation

The first and last step of modified PROuST, i.e. the hash table construction and the atomic superposition, are executed off-line, locally on a cluster of machines. The index-based structural alignment is executed on-line on the nodes of a computational Grid, as we will describe later. The construction of the hash tables is compute intensive, due to the large number of proteins in the PDB inserted in the hash table with a storage requirement for the hash table of more than 5 GB. The pre-processing phase is also triggered by updates in PDB, or by timeout expiration (each month), or by user intervention.

A distributed implementation of the hash table construction was presented in [5,7] on a cluster of machines; it is based on a partition of the hash table into subtables, each containing a subset of the proteins of the PDB. Each machine of the cluster constructs a subtable of the hash table. The input proteins are inserted into the subtables by a greedy procedure that randomly partitions the proteins into groups of fixed size k, with k larger than the number of subtables, and assigns each group to the first available computer of the cluster for insertion into its associated subtable. As shown by the experiments, this simple procedure generates subtables of approximately the same size; furthermore it distributes the set of all proteins almost uniformly across the subtables.

We now describe a distributed implementation of the index-based structural alignment. It follows a master/slave paradigm. For a given target protein, the master broadcasts information about the target protein (PDB file) to all slaves. The index-based structural alignment is carried out independently by all slaves each operating on a different subtable. A slave generates a list of candidate proteins selected among those stored in its hash subtable. It also determines the score matrices associated to the target protein and each candidate protein. All the data involved in the computation are stored in the hash subtable. Then, the slave performs all DP computations for the candidate proteins,

either the entire list of candidates or a reduced list of top-ranked candidates. According to the similarity measure determined by the structural alignment, the candidate proteins are sorted into a new list, that is sent back to the master. The master collects from the slaves all such lists and then merges them to obtain the overall ranking.

The degree of parallelism and the number of slaves depend on the partition of hash table into subtables. We experimented with partitions of the hash table into 3, 9 and 30 subtables, approximately of sizes 1.5 GB, 500 MB, and 150 MB, respectively.

5. Gridifying PROuST

Our implementation was developed for the Italian Grid.it, more specifically for the INFNGrid production grid that is also part of the project Grid.it [2]. The INFNGrid counts more than 20 sites among Italian institutions. Each site has several Computing Elements (CE), i.e. clusters of machines, or parallel machines. In addition, some sites have an associated Storage Element (SE). The main applications of Grid.it were originally in physics, however it has become also open to other fields such as bioinformatics and biomedicine.

Here we explain how to exploit the Grid capabilities to optimize the execution time and data storage of PROuST. Job scheduling is handled by the Globus default scheduling strategy even if in the future we plan to explore other possibilities.

5.1. Data distribution and replica

Replica optimization is a crucial aspect of a gridifying strategy, because the availability of data in the SE close to the CE where the job requesting the data runs enables latency reduction and efficient access [3,4]. Replica management in the European DataGrid and Grid.it is handled by independent services interacting via the Replica Manager (RM) [3]. In a Grid, a file is first registered with an identifier, the Grid Unique ID (GUID), then it is replicated and distributed. The main advantage of using replicas is that one can refer to a file simply using its GUID, then the RM through its *Replica Catalog* (RC), links the GUID to all the replicas of the file. Referring to all replica files using a unique GUID allows to ignore how many replicas are in the Grid and where they are.

The *Replica Optimization Service* (ROS) selects the best available replica of the data files a job needs. Using these services we experimented with two main policies for replica management, that we called *on-line replica* and *off-line replica*.

On-line replica means that every time a job needs data, the CE where the job runs first looks for them at the local SE. If the data are not available locally, the ROS optimization service chooses whether to access them remotely or replicate them on the local SE [3]. A replica is made when the estimate of the time spent accessing the data from a remote site is greater than the overall time needed to replicate the data at the local site and access them locally. The on-line replica policy requires that the user has high control over the storage resources. In fact, replicating data files on-line requires a permission to insert new

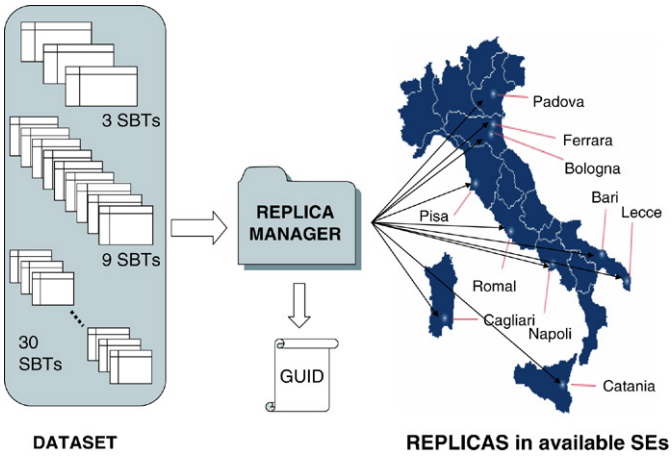

Fig. 1. Data replication.

files into an SE of the Virtual Organization (VO) and to delete replicas of other files if insufficient space is available. This permission is not always granted because, typically, a single storage resource is shared by many users and even by many VO.

The *off-line replica* policy refers to the case when all replica operations are made once and for all, before the user submits any job and the resource broker assigns the jobs to the CE. In this scenario, data are accessed either from the local SE, if a replica of the data is available there, or directly from the remote SE selected by the ROS based on geographical location and network latency.

We adopt an *off-line replica* policy, by replicating the hash subtables in all available SE of the INFNGrid production grid (see Fig. 1). With only about 8 SE available to our VO and a large number of jobs requesting data, the on-line policy would end up replicating all data in all SE exactly as the off-line policy. However, since the replicas for the on-line policy are made at run time, this strategy would be less efficient.

In this situation, it may still happen that a job running on a CE needs to access data from a remote SE. This is because not all CE of Grid.it have an on site local SE, furthermore permission to write is limited to some SE of the Grid. The different steps of our application running on a Grid environment are summarized in Fig. 2.

For the analysis of the replica policies, an important performance metric is the *effective network usage* r_{ENU} introduced in [3] and defined as:

$$r_{\mathrm{ENU}} = \frac{N_{\mathrm{remotefileaccesses}} + N_{\mathrm{filereplications}}}{N_{\mathrm{localaccesses}}}$$

where $N_{\mathrm{filereplications}}$ is the number of times the replica optimization service decides to replicate the files; while $N_{\mathrm{remotefileaccesses}}$ and $N_{\mathrm{localaccesses}}$ are the numbers of files accessed remotely and locally, respectively. The performance metric for the off-line replica policy becomes:

$$r_{\mathrm{ENU}} = \frac{N_{\mathrm{remotefileaccesses}}}{N_{\mathrm{localaccesses}}}$$

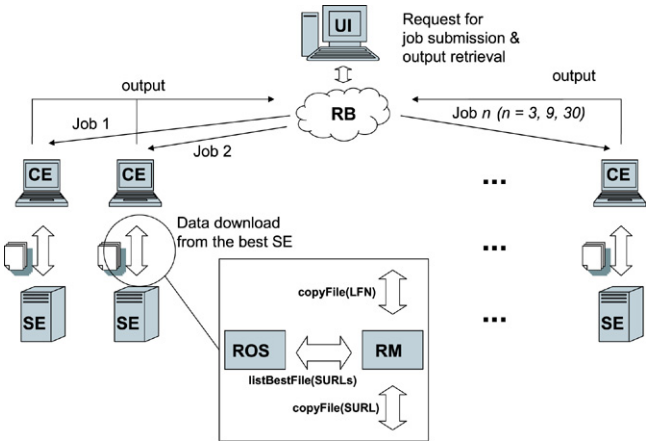

Fig. 2. Application execution steps in a Grid environment.

because file replications are not allowed during the execution of the jobs, thus $N_{\text{filereplications}} = 0$. Performance measures, using the above metric, will be given in the following section.

6. Time analysis

The time analysis is divided in two parts: the analysis of the time spent to download the subtables and of the overall execution time. We have to make some preliminary remarks: the analysis of the time needed to download the data and perform the computation involves many variables that are hardly predictable. Among them, the most crucial is the network traffic that can slow down the speed of downloads significantly. Also important is the load of the Grid's nodes that can severely affect the time performance. In a real Grid environment, we observed variations of the execution times for different runs of the same task even by one order of magnitude. Therefore, we run the application several times over a period of about two months, from August to October 2005, under different conditions of network traffic and performance of the nodes. The software consists of C programs for comparing an input protein against all proteins of the PDB, and of a pool of BASH scripts to manage job description, submission and status control over the Grid.

6.1. Data transfer analysis

We determined the time to download an hash subtable for the three different partitions of the original table into 3, 9, and 30 subtables. For each partition we consider two data distributions:

(1) a single SE stores all subtables,
(2) all subtables are replicated in all available SE.

Table 1 summarizes the results. The times reported in the table are the mean values over all the runs over a two month period. As expected, the increased granularity in the hash table's partition and the data replica throughout the storage elements positively affect the data transfer time.

In Fig. 3 we consider samples of transfer times for a single subtable of the 9 subtable partition. All values are obtained

Table 1
Data transfer time

Time (s)	3 subtables	9 subtables	30 subtables
Single SE	748	279	90
Many SE	492	160	49

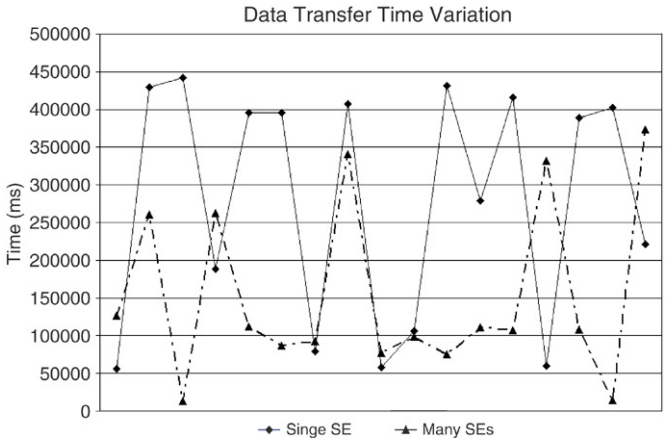

Fig. 3. Data transfer time variation: one vs. many replicas.

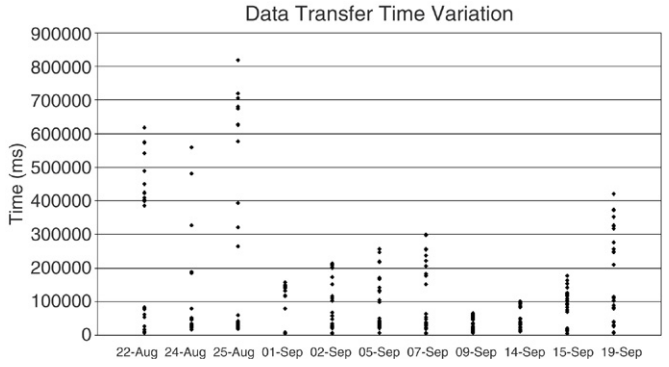

Fig. 4. Typical data transfer time variation using many replicas during a month.

by runs in the same day. The differences in transfer time are mostly due to the different ways the data are accessed: remotely or locally. Typically, higher values are related to remote accesses. The continuous line in the figure shows data transfer time variations when a single fixed SE is used. As expected, large variations are observed in this case, since local accesses occur rarely and remote access times vary according to the geographical location of the CE. The dotted line shows transfer times measured when all replicas are available in all SE. Even if some high values can still be observed, indicating remote accesses, the number of local accesses increases, thus reducing the average time significantly.

Fig. 4 represents typical transfer time variations over a period of one month. Here we consider the case when the 9 subtables are replicated in all SE. Each day several runs were made and the corresponding times are reported. The variations observed in the figure are due to the network traffic and Grid conditions. For this set of experiments we also evaluated the effective network usage $r_{\text{ENU}} = \frac{24}{96} = 0.25$.

Table 2
VersionB (before re-organization)

Time (s)	3 subtables	9 subtables	30 subtables
t_1: index-based search	19	5	1
t_2: DP + score matrices building	1016	370	136

Execution time breakdown for protein 1a2z chain C on a single CE.

Table 3
VersionA (after re-organization)

Time (s)	3 subtables	9 subtables	30 subtables
t_3: index-based search + score matrices building	980	84	19
t_4: DP	2	<0.5	<0.5

Execution time breakdown for protein 1a2z chain C on a single CE.

6.2. Execution time analysis

Before running the application on a Grid, we conducted experiments to evaluate the effect on time performance of the algorithmic changes to PROuST discussed in Section 3 when it is executed on a single computing element, a standard PC. In the following, versionB and versionA are the two versions of the algorithm, before and after the re-organization of its components, respectively. Execution times are determined for three partitions of the hash table, into 3, 9 and 30 subtables. The execution time breakdown for versionB are reported in Table 2: t_1 refers to the average time to obtain from an hash subtable the list of candidate proteins. This time, for a given partition of the hash table, is averaged over all subtables. It has to be noted, however, that there is little variance across the subtables. t_2 is the total execution times of all DP computations to align the target protein with the candidate proteins extracted from a subtable; this time is averaged over all subtables of the hash table partition. Recall that, in versionB, the DP algorithm builds the score matrix and re-computes the angles and distances of all triplets of secondary structures of the target and of the candidate protein. The execution times in Table 2 were obtained with the input protein 1a2z (chain C), a peptidase protein, that contains 14 SSE. Only 12 SSE were included in our analysis since SSE with less than 3 residues were discarded. Table 3 shows the execution times of the re-organized components when executed with the same target protein 1a2z as input. In versionA, the hash table search includes the building of the score matrices (t_3), thus the DP (t_4) does not have to re-compute the geometric properties of SSE.

As can be seen from these results, versionA is consistently better than versionB and the gain in performance increases with the number of subtables. Furthermore, results on proteins of different sizes, i.e. different number of SSE, show that this improvement is more relevant for proteins of average/small sizes. For very large proteins, relatively infrequent in the PDB database, some improvement can still be observed but it is negligible. These experiments, repeated over a large set of proteins, led us to the conclusion that the algorithmic changes introduced in this paper result in an enhancement even when the application was executed on a single CE. Thus we can only expect a further improvement when porting the application to the Grid.

We show below the execution times of PROuST for sixteen input proteins, with sizes ranging from 5 to 99 secondary structures. These times are for the overall computation: the index-based search and all DP computations to align the target protein with all candidate proteins. The execution times for each protein chain reported in Table 4 and Fig. 5 are the mean values over several runs of the same computation performed during a two month period. In Fig. 5 times are presented on a *log* scale. We observe that by increasing the number of subtables, up to 30, and consequently the parallelism degree we obtain a constant ratio of the execution times. For partitions with more than 30 subtables we could see a degradation in time performance.

Table 4
PROuST execution times

Protein	#SSs	3 subtables	9 subtables	30 subtables
4hck.	5	17 (s)	4 (s)	1 (s)
110m.	8	263	32	7
111m.	8	228	31	8
112m.	8	221	31	8
1etc.	8	199	21	4
1fgz A	8	180	20	4
2gva A	9	206	26	6
2gva B	9	228	26	5
1a2z A	12	1098	99	17
1a2z B	12	1046	99	18
1a2z C	12	980	88	16
1a2z D	12	1009	90	16
9xia.	20	4238	476	87
8icm A	22	4874	561	102
1ea0 A	95	N/A	4986	676
1ea0 B	99	N/A	5279	707

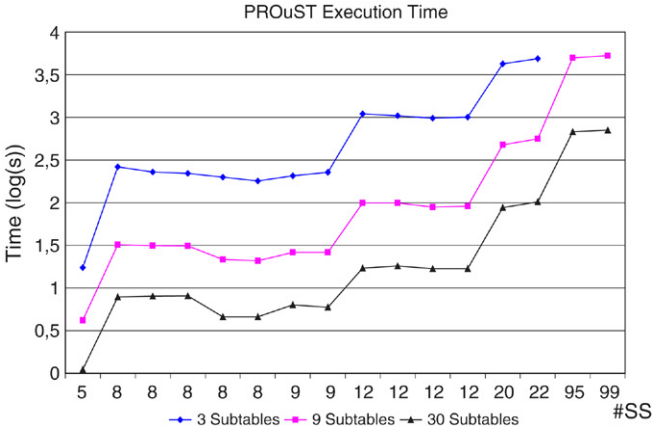

Fig. 5. PROuST execution time.

7. Conclusion

We have presented a distributed implementation on Grid.it of a software tool for protein structure comparison. A major restructuring of the application has been developed for an efficient porting to a computational Grid. Without restructuring

the DP phase of the application makes many small requests of data that involve frequent and costly accesses to SE. The re-organized version of the software uses the data available from the hash tables for more than one operation once they are accessed; i.e. for the determination of the list of candidate proteins and also for the computation of the score matrices used by DP. Furthermore, by re-using the hash table data we avoid the access to the many small files containing the protein vectorial representations that would be needed by DP if the geometric properties were to be recomputed. We have also experimented with different ways of exploiting data replica on the Grid, and with different partitions of the hash table into subtables. The algorithmic changes and the gridification strategies employed allow for a significant reduction in communication time and overall execution time.

References

[1] The Protein Data Bank, Research Collaboratory for Structural Bioinformatics (RCSB), http://www.rcsb.org/pdb.

[2] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, C. Zoccolo, Components for high performance programming in the Grid.it project, in: Proc. of Intl. Workshop on Component Models and Systems for Grid Applications, ACM ICS, 2004.

[3] D. Cameron, J. Casey, L. Guy, P. Kunszt, S. Lematre, G. McCance, H. Stockinger, K. Stockinger, G. Andronico, W. Bell, I. Ben-Akiva, D. Bosio, R. Chytracek, A. Domenici, F. Donno, W. Hoschek, E. Laure, L. Lucio, P. Millar, L. Salconi, B. Segal, M. Silander, Replica management in the European DataGrid project, Journal of Grid Computing 2 (4) (2004) 341–351.

[4] D.G. Cameron, A.Fr. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, F. Zini, Analysis of scheduling and replica optimization strategies for data grids using OptorSim, Journal of Grid Computing 2 (1) (2004) 57–69.

[5] M. Comin, C. Ferrari, C. Guerra, Grid deployment of bioinformatics applications: A case study in protein similarity determination, Parallel Processing Letters 14 (2) (2004) 163–176.

[6] M. Comin, C. Guerra, G. Zanotti, PROuST: A comparison method of three-dimensional structure of proteins using indexing techniques, Journal of Computational Biology 11 (6) (2004) 1061–1072.

[7] C. Ferrari, C. Guerra, G. Zanotti, A grid-aware approach to protein structure comparison, Journal of Parallel and Distributed Computing 63 (7–8) (2003) 728–737.

[8] I. Foster, C. Kesselman, The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1999.

[9] L. Holm, C. Sander, Mapping the protein universe, Science (273) (1996) 595–602.

[10] J.G. Jensen, T. Shah, O. Synge, J. Gordon, G. Johnson, R. Tam, Enabling grid access to mass storage: Architecture and design of the EDG storage elements, Journal of Grid Computing 3 (1–2) (2005) 101–112.

[11] J. Montagnat, F. Bellet, H. Benoit-Cattin, V. Breton, L. Brunie, H. Duque, Y. Legré, I.E. Magnin, L. Maigne, S. Miguet, J.M. Pierson, L. Seitz, T. Tweed, Medical image simulation, storage, and processing on the european datagrid testbed, Journal of Grid Computing 2 (4) (2004) 387–400.

[12] I.N. Shindyalov, P.E. Bourne, Protein structure alignment by incremental combinatorial extension (CE) of the optimal path, Protein Engineering 11 (9) (1998) 739–747.

G. Ciriello received the laurea degree in computer science from the University of Padova in April 2005. From May to December 2005 he was a scientific collaborator at the Bioinformatics and Computational Biology Group of University of Padova, working on Grid implementation strategies of bioinformatics applications. Currently he is a Ph.D. student in computer science at the University of Padova. His research interests include distributed computing, bioinformatics and machine learning.

M. Comin received the laurea degree in computer science in 2003 from the University of Padova (Italy). In 2003 he was visiting student at Purdue University and in 2004 research intern at the IBM T.J. Watson Research Center. Since 2004 he is a Ph.D. student in computer science at the University of Padova. His interests are in the design of algorithms and applications for pattern discovery, data compression, geometric pattern matching, structural proteomics and grid computing.

C. Guerra works in the areas of Computational Biology and Computer Vision. Her recent interests fall in the domains of protein classification, recognition and docking. Formerly an Associate Professor at University of Rome, Italy, she joined the Department of Information Engineering of the University of Padova, Italy, where she became a Professor in the Faculty of Engineering. She has visited extensively with US Institutions, including Rensseleaer Polytechnic and CMU, and has been on the CS faculty of Purdue University for over a decade. Now she is professor at the College of Computing at the Georgia Institute of Technology.