Parallel-ProBiS: Fast Parallel Algorithm for Local Structural Comparison of Protein Structures and Binding Sites

Janez Konc,[1] Matjaž Depolli,[2] Roman Trobec,[2] Kati Rozman,[1] Dušanka Janežič[1,*]

[1]Laboratory for Molecular Modeling, National Institute of Chemistry,
Hajdrihova 19, SI-1000, Ljubljana
[2]Department of Communication Systems, Jožef Stefan Institute,
Jamova cesta 39, SI-1000, Ljubljana,

* Author to whom correspondence should be addressed
email: dusa@cmm.ki.si

Keywords: protein structures, binding sites, pairwise alignment, parallel algorithms, computer clusters

Abstract

The ProBiS algorithm performs a local structural comparison of the query protein surface against the non-redundant database of protein structures. It finds proteins that have binding sites in common with the query protein. Here, we present a new parallelized algorithm, Parallel-ProBiS, for detecting similar binding sites on clusters of computers. The obtained speedups of the parallel ProBiS scale almost ideally with the number of computing cores up to about 64 computing cores. Scaling is better for larger than for smaller query proteins. For a protein with almost 600 amino acids, the maximum speedup of 180 was achieved on two interconnected clusters with 248 computing cores. Source code of Parallel-ProBiS is available for download free for academic users at http://probis.cmm.ki.si/download.

Introduction

The exponential increase in computer power has made searches in protein structural databases of thousands of protein structures routine. A variety of techniques exist for protein similarity searching, ranging in computational complexity from global structural superposition methods[1] to more complex substructure or fingerprint searches.[2-5] The ProBiS algorithm of Konc and Janežič[6] belongs to the latter class of algorithms and detects pairwise local similarities in proteins by computing the similarity of protein graphs, which are representations of specific proteins. It runs in the ProBiS web server[7] at http://probis.cmm.ki.si and the parallel version was used to create the ProBiS-Database,[8] a repository of over 420 million precalculated binding site similarities and local pairwise alignments of PDB structures at http://probis.cmm.ki.si/database. Despite the complexity of comparing entire protein structures, ProBiS has demonstrated its ability to find intricate similar three-dimensional patterns in sites involved in binding of small molecules, ions, proteins, or nucleic acids;[6] its compelling advantages are speed and the ever-increasing availability of appropriate protein structures.

While ProBiS is one of the fastest and most advanced local similarity search techniques in general usage, emerging problems in structural bioinformatics,[9,10] drug repositioning,[11,12] and prediction of protein function[13,14] require dramatically faster methods for calculating structural similarities. The currently available PDB database holds more than 180 thousand single chain protein structures and is currently being used for a variety of bioinformatics analyses. Exhaustive structural similarity searches on a database containing hundreds of thousands of structures present a challenge to commonly used structural alignment algorithms.[1,5] Even more time consuming are exhaustive database searches such as those in ProBiS, which represents

proteins at the level of functional groups and seeks similar three-dimensional binding site patterns. Since there are more than one hundred thousand protein single chains in the PDB, all-against-all comparisons of protein structures which can easily require billions of comparisons, can also present a challenge to conventional algorithms.

The need for parallelization arises from this rapid growth of the PDB. On the other hand, contemporary computers typically have multiple computing units (cores),[15] however, with naive programming approaches a single program will only exploit a single computing core. While a single comparison of two proteins can be done in linear time, procedures such as all-against-all comparison of PDB database require computing times which are the square of the database size. Consequently, as the size of PDB database increases,[5] faster similarity methods are required to handle the increasing computational load.

The Message Passing Interface (MPI) standard for communication in parallel computing offers a solution to this problem.[16] The CPUs of modern computers have multiple cores that are separately programmable and can, when used proficiently, offer significant increases in computation speed over single CPUs,[17] and Ethernet connections enable fast connectivity between separate computers. Parallelization has been applied effectively to related problems in computational chemistry, including molecular dynamics[18] and 3D similarity comparisons of small molecules.[19]

In this paper we present a parallelized version of ProBiS. This is a new algorithm that calculates the local similarity metric between protein structures, and is especially suited to efficient execution on multiple CPUs and on computers of varying power interconnected in a network, which are available in contemporary computing platforms or computing clouds.[18] When calculating local structural similarities in a database of over 29,000 protein structures, this parallel version of the ProBiS algorithm is 180 times faster on a cluster of 49 computer nodes than the existing ProBiS algorithm implemented with script-based concurrent runs. We explain the ProBiS algorithm, provide a description of our new Parallel-ProBiS algorithm, and present performance benchmarks and comparison of the two algorithms. The code is freely available for academic users at http://probis.cmm.ki.si/download.

Methods

Benchmark Database of PDB Protein Structures

The non-redundant PDB database used in this article was built by clustering 181,882 protein single chains in the PDB with >95% sequence identical structures.[5,6] A representative of each cluster is chosen and surface residues of the selected representative proteins are identified and converted to protein graph representations, which are saved into 29,266 'surface files' which enable fast pairwise comparisons by ProBiS.

Overview of the ProBiS Algorithm

ProBiS performs a local structural comparison of the query protein surface against the non-redundant PDB database.[6] It finds proteins that have binding sites in common with the query protein. Similar binding sites can be found even in proteins of different folds. It also detects structurally conserved regions on the query protein structure, and provides structural superimpositions of the query protein and the similar proteins. The algorithm exploits the fact that binding sites share similar patterns of interactions in proteins which perform similar functions.

ProBiS first defines the solvent accessible surface by rolling a probe, an atom of 1.4 Å radius over the protein atoms represented as van der Waals spheres. Residues that are up to 4 Å below this surface are considered for comparison. The surface is represented as a protein graph, i.e., structure of vertices and edges.[2,6] This representation, which is on the level of functional groups, considers both geometrical and physicochemical properties of the surface. Possible interactions of a protein with ligands are thus taken into account. A structural similarity search algorithm, which employs a fast maximum clique algorithm[20] and operates independently of fold and sequence, performs a local, surface-oriented comparison of the proteins represented as graphs. All possible similar regions between two compared proteins are identified. Each maximum clique, i.e., its rotational–translational variation, represents a rigid, local similarity, which is then used to locally superimpose the two compared protein structures. Finally, the two superimposed structures are subject to local alignment of their backbones, to find similarities that were missed using maximum clique approach. This final alignment finds similar sites that adopt different conformations in the two compared proteins. Degrees of structural conservation are calculated for all amino acid residues of the query protein and reveal the extent to which a particular residue appears in the local structural alignments that were found within the protein database.

Parallel-ProBiS Implementation

Parallelization of the time consuming protein comparison process should be accomplished in such a way that it is equally efficient on single or multicore computing systems, and on homogeneous or heterogeneous computing clusters or in computing clouds. This implies that the parallelization methodology must incorporate automatic balancing of computation. The pairwise comparisons by ProBiS are computationally intensive and repetitive and each comparison is independent, and thus the algorithm is appropriate for parallelization. Parallelization on the level of a single computing node can be implemented using batch scripts[6,7] but such an approach is not effective on parallel and distributed computing platforms with a lot of interconnected computing nodes. Because the ProBiS comparison implies a high ratio between the computation time and communication time, it can be better parallelized on a task level with an approach known as *multi-experimental*.[17]

A parallel platform can be represented as a set of slave nodes with a single master node and in this context, the experiments are individual pairwise comparisons, *i.e.,* processes that run on slave nodes. Bookkeeping is implemented as a separate process that runs on the master node. Usually, the master bookkeeping process is much simpler than the comparison processes and the master node can run concurrently with the slave processes.

The program that activates the ProBiS algorithm is the same for all computing slave nodes and has two parts, first for the supervising master process with a process ID of 0, and second for the remaining slave processes. The program is presented as a flowchart in Figure 1. The communication between parallel processes is implemented using a standard MPI library,[17] and consists of the master node sending protein names to slave nodes that will return comparison results to the master node. Communications are short and infrequent: for every pairwise protein comparison, only a few bytes long message with the protein name that is to be compared with the query, and a few kilobytes long return message with the results of the pairwise comparison, are sent. In addition, queues of protein names on the slave nodes serve as buffers, providing slave nodes with work, while the master node is busy. The computation of each pairwise protein comparison takes between hundreds of milliseconds and a few seconds to complete, and thus the time lost on communication between processes is several orders of magnitude shorter than the computation time. Consequently, restrictions posed by the communication channel

bandwidth and message latency are very low. We experienced that the standard 100 Mb/s Ethernet suffices to connect several hundreds of slaves with the master node.

Since the MPI is available in a standard form for most of the existing platforms,[21] our approach is highly portable. The parallel code can also run on heterogeneous systems, with different hardware architecture and different operating systems. Because of the asynchronous design of the proposed solution, the communication requirements are minimal and the computational load is automatically balanced, supporting our expectations that the speedup of the proposed parallel program will be close to ideal.

Benchmarking Methodology

We measured the speedup of the proposed parallelization on a number of benchmarking problems based on the calculation of similarity of a query protein against all the proteins in a non-redundant benchmark PDB database - a common task in many structural alignment applications. We have selected a small set of query proteins, 1phr.A (177 amino acids), 1acb.E (245 a.a.), 1iov.A (306 a.a.), 1uzf.A (589 a.a.), to measure the impact of protein size on the performance of the Parallel-ProBiS algorithm.

The benchmark tests were performed on the cluster at the National Institute of Chemistry (NIC) in Ljubljana, Slovenia. This cluster is composed of 18 state-of-the-art computing nodes, from which, as many as 14, each with two quad-core Intel Xeon 5520 2.26 GHz processors, were available for the test. All nodes ran under Ubuntu 10.04 LTS operating system. Further tests were run on a heterogeneous system obtained by connecting to an additional cluster at the Jozef Stefan Institute (IJS) in Ljubljana, Slovenia, composed of 37 computing nodes, each with a single quad-core Intel Xeon 5520 2.26 GHz processor. These nodes ran under the Ubuntu 11.04 operating system. The topology of the NIC and IJS clusters is shown in Figure 2. One of the nodes in each cluster acts as a gateway to the Internet. The nodes of both clusters are interconnected with Gigabit Ethernet links, however both gateway nodes can communicate with a maximum bandwidth of 100 Mb/s. The MPI library requires that all computing nodes belong to the same network address space, therefore a Virtual Private Network (VPN) was established for the tests involving both IJS and NIC clusters. The VPN server resided on the gateway computer of the IJS cluster. Computers of the IJS cluster were connected to it directly, whereas computers of the NIC cluster communicate via the NIC gateway node. The Mpich2 1.2.1 library was used as an implementation of the MPI standard[21] on both clusters.

We first ran the ProBiS algorithm on single cores to find out the shortest sequential execution time T_1 of the non-parallel version of the algorithm. Then we ran the Parallel-ProBiS algorithm on 1, 2, 4, 8, and 14 nodes of the NIC cluster. These were in two configurations - hyperthreading was either active or inactive, which resulted in 16 or 8 processes per node, and the total number of processes p = {16, 32, 64, 128, 224} or p = {8, 16, 32, 64, 112}, respectively. Finally, we ran the performance tests on the heterogeneous system of 13 NIC cluster nodes with 8 and 16 processes per node, and 36 IJS cluster nodes with 4 and 8 processes per node, for the total of p = 298 and p = 496, respectively. We measured the execution time of the Parallel-ProBiS T_p and calculated the speedup[17] as $S = T_1/T_p$. The maximum theoretic speedup, also termed the ideal speedup, equals p. All tests were run ten times to show the statistical properties of the measured results.

Results

To assess the performance of the Parallel-ProBiS algorithm, we carried out a series of database searches with different query proteins. The performance on a single node, a cluster of nodes, and two clusters located at two institutions and connected through the Internet, was evaluated. Each measurement was repeated ten times and the speedup of the parallel versus the non-parallel ProBiS algorithm was calculated. Details are in Tables SI1 and SI2 in the Supplementary Information.

The Parallel-ProBiS using MPI-based parallelization achieves on a single node slightly better speedups than the script-based naive parallelization of the non-parallel ProBiS algorithm (Figure 3). The speedup is almost ideal with up to 8 processes in the case of inactive hyperthreading; with active hyperthreading, the speedup still increases, but with a slower rate. Hyperthreading contributes up to 30% to the final speedup on all 16 processes, which indicates that the Parallel-ProBiS algorithm is able to concurrently exploit floating point calculation, integer processing, and memory data transfer.

The speedup of Parallel-ProBiS tested on the NIC cluster depends on the size of a query protein (Figure 4). For smaller query proteins of ~200 amino acids, i.e., 1phr.A and 1acb.E, the speedup virtually does not increase when using more than 8 nodes. Therefore small proteins should not be calculated on more than 8 nodes. The parallel scaling efficiency, which is defined as the ratio between the measured speedup and the ideal speedup, is between 44% and 95%,

with inactive hyperthreading, and between 25% and 62%, with active hyperthreading. It decreases with the number of computing nodes and increases with the protein size. Hyperthreading effectively increases the speedup between 15% and 40% for the tested query proteins on up to 14 nodes of the NIC cluster, and should be active, if the number of available nodes is limiting.

We then tested Parallel-ProBiS on NIC and IJS clusters, consisting of 13 and 36 nodes, respectively (Table 1). As before, the speedups are better for larger proteins. The best speedup so far, 161, was obtained for 1uzf.A, the largest protein in the test set. Contrary to our previous results on a single cluster, hyperthreading decreases the speedups for all query proteins. We think that the communication is probably the cause of this worsened performance. If this is true, the speedups for smaller query proteins should decrease more than the speedups for larger proteins with active hyperthreading, because the nodes computing smaller query proteins need to communicate more frequently. This can be seen and is confirmed by the data in Table 1.

Parallel-ProBiS guarantees a balanced computational load at the granularity of a single pairwise comparison, but lack of finer granularity can produce some wait times at the last few comparisons. In the worst case, all processes except a single one finish their assigned comparisons and must wait for the last process that just started its last assigned comparison. We addressed this by sorting the comparison database proteins, i.e., the non-redundant PDB, by their decreasing size (Figure 1), so that small proteins are computed last.

We do not see any significant degradation in speedup stemming from the slower communication link between NIC and IJS clusters. For example, the speedup per process for protein 1uzf.A on NIC cluster with 112 processes is ~0.75; on both clusters with 248 processes ~0.73. The speedup per process is expected to decrease with the number of nodes, and the slower link between the two clusters does not seem to affect this. This confirms that the proposed methodology is very appropriate for the network and cluster computing. The speedup is not as high as expected with active hyperthreading, probably because of the increased communication overhead of the MPI library. Also, with the increasing number of nodes, the number of parallel processes approaches the number of pairwise comparisons, which results in unbalanced load and idle cores. This is a limitation of the current methodology, which becomes apparent, when the algorithm is run on more than 8 nodes and the query protein's size is less than 245 amino acids (Figure 4).

Further increase of the speedup is possible by the parallelization on the level of protein comparison itself, which will result in a finer granularity of the problem and easier load-balancing of processors. Particularly, we plan to parallelize the maximum clique algorithm,[20] which is the essential building block of the proposed protein structural comparisons. We will explore the efficiency of many-core and GPU platforms in further parallelization approaches.

The asset of the proposed Parallel-ProBiS is its ease of use and better performance, compared to naive parallelization using batch script. The only requirement is that the Parallel-ProBiS program is properly installed on all nodes and that the master node is able to communicate to all slave nodes through the MPI library.

Conclusions

The growing size of the PDB requires the development of faster algorithms for structural comparisons of protein structures. We have described a new parallel algorithm Parallel-ProBiS, which enables efficient searches against the entire PDB database. This algorithm is well-suited to implementation on clusters or clouds of heterogeneous computers. It demonstrates an 161-fold speedup on 248 computing cores compared to the non-parallel version that runs on a single core. The proposed parallel algorithm scales well with the number of computers it is running on, enabling high performance on large computer clusters. By providing two orders of magnitude in speedup, the Parallel-ProBiS algorithm enables dramatically larger calculations than previously possible. We anticipate that these local similarity search capabilities will enable a new class of bioinformatics applications from drug repositioning to off-target prediction. The Parallel-ProBiS algorithm is available for download at http://probis.cmm.ki.si/download.

Acknowledgement

The financial support through grants P1-0002, P2-0095 and Z1-3666 of the Slovenian Research Agency is acknowledged.

References

[1] H. Hasegawa, L. Holm, *Curr. Opin. Struct. Biol.* **2009**, *19*, 381.

[2] D. Kuhn, N. Weskamp, S. Schmitt, E. Hullermeier, G. Klebe, *J. Mol. Biol*. ***2006****, 359*, 1023.

[3] D. Shirvanyants, A. N. Alexandrova, N. V. Dokholyan, Bioinformatics 2011, 27, 1327.

[4] M. Jambon, O. Andrieu, C. Combet, G. Deléage, F. Delfaud, C. Geourjon, *Bioinformatics **2005**, 21,* 3929.

[5] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, P. E. Bourne, *Nucleic Acids Res.* **2000***, 28,* 235.

[6] J. Konc, D. Janezic, *Bioinformatics* **2010**, *26,* 1160.

[7] J. Konc, D. Janezic, *Nucleic Acids Res.* **2010**, *38,* W436.

[8] J. Konc, T. Cesnik, J. Trykowska Konc, M. Penca, D. Janezic, *J. Chem. Inf. Model. **2012**, 52,* doi:*10.1021/ci2005687.*

[9] J. Martin, *PLoS Comput. Biol*. **2010**, *6,* e1000821.

[10] V. Skedelj, T. Tomasic, L. Peterlin Masic, A. Zega, *J. Med. Chem. **2011**, 54,* 915.

[11] V. J. Haupt, M. Schroeder, *Brief. Bioinform.* **2011***, 12,* 312.

[12] L. Xie, T. Evangelidis, L. Xie, P. E., Bourne, *PLoS Comput. Biol*. **2011**, *7,* e1002037.

[13] F. Musiani, M. Bellucci, S. Ciurli, *J. Chem. Inf. Model.* **2011**, *51,* 1513.

[14] M. T. Wong, S. B. Choi, C. S. Kuan, S. L. Chua, C. H. Chang, Y. M. Normi, W. C. S. Too, H. A. Wahab, L. L. Few, *Int. J. Mol. Sci.* **2012**, *13,* 901.

[15] D. Watts et al., Tuning IBM System x Servers for Performance; [Online] http://www.redbooks.ibm.com/redbooks/pdfs/sg245287.pdf, **2009**.

[16] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI – The Complete Reference; The MIT Press: Cambridge, **1996**.

[17] S. G. Akl, Parallel Computation: Models and Methods; Prentice Hall: Upper Saddle River, **1997**.

[18] U. Borstnik, B. T. Miller, B. R. Brooks, D. Janezic, *J. Comput. Chem.* **2011**, *32,* 3005.

[19] T. S. Rush, J. A. Grant, L. Mosyak, A. Nicholls, *J. Med. Chem.* ***2005****, 48,* 1489.

[20] J. Konc, D. Janezic, *MATCH Commun. Math. Comput. Chem.* **2007**, *58*, 569.

[21] W. Gropp, E. Lusk, N. Doss, A. Skjellum, *Parallel Computing* **1996**, *22,* 789.

Tables

Table 1. Test results on interconnected NIC and IJS clusters with inactive hyperthreading (248 processes) and active hyperthreading (496 processes).

Number of		Execution time [s]				Speedup			
Processes	Nodes	1phr.A	1acb.E	1iov.A	1uzf.A	1phr.A	1acb.E	1iov.A	1uzf.A
248	13+36	136±48.2	141±63.7	251±19.9	288±4.38	100	82.3	116	180
496	13+36	335±91.0	357±233	359±42.3	399±20.5	40.6	32.6	81.4	130

Figure Captions

Figure 1. Overview of Parallel-ProBiS algorithm. Part of the algorithm executed on master node is in white boxes; part executed on slave nodes is in grey boxes. Input: query protein (protein A); comparison protein database; number of slave nodes (n); slave queue size (q). Output: pairwise local structural alignments of query protein with all database proteins.

Figure 2. Topology of experimental setup; nodes p0 - p17 represent the NIC cluster, and nodes k0 - k36 represent IJS cluster.

Figure 3. Speedup of Parallel-ProBiS on a single computing node of the NIC cluster as a function of the number of processes executed. Hyperthreading was inactive (〄) or active (¤). The speedup of the script-based parallelization (black) is averaged over all tested proteins.

Figure 4. Speedup of Parallel-ProBiS as a function of the number of NIC computing nodes. If hyperthreading is inactive, each computing node executes 8, if active 16 Parallel-ProBiS processes concurrently. Ideal speedup (black), which is the maximum theoretic speedup, equals the number of processes; e.g., for 14 computing nodes, it is calculated as 14x8=112 (8 processes per computing node), if hyperthreading is inactive, and 14x16=224 (16 processes per computing node), if active.