SBLAST: Structural Basic Local Alignment Searching Tools using Geometric Hashing

Tom Milledge[1*], Gaolin Zheng[2*], Tim Mullins[3], Giri Narasimhan[1]
[1]*BioRG, School of Computing and Information Science, Florida International University*
[2]*Department of Mathematics and Computer Science, North Carolina Central University*
[3]*IBM Systems and Technology Group, Rochester, MN 55901,* [*]*Joint first authors*
tmille01@cs.fiu.edu, gzheng@nccu.edu, mullinst@us.ibm.com, giri@cs.fiu.edu

Abstract

While much research has been done on finding similarities between protein sequences, there has not been the same progress on finding similarities between protein structures. Here we report a new algorithm (SBLAST) which discovers the largest common substructures between two proteins using a triangle-based variant of the geometric hashing of protein structures algorithm. The algorithm selects triples (triangles) of selected Cα atoms from all proteins in a protein structure database and creates a hash table using a key based on the three inter-atomic distances. Hash table hits from the triangles of a query protein are extended recursively to determine the largest common substructures less than a threshold deviation level (rmsd). Comparisons between a query protein and a preprocessed protein database can be performed in parallel. Because SBLAST does not rely on protein sequence alignment, common substructures can be detected in the absence of sequence conservation. SBLAST has been tested using the ASTRAL subset of the PDB.

1. Introduction

Determining structural similarity is one of the most important tasks in proteomics. Numerous 3D structure alignment tools have been developed for comparing protein structures, such as CE [1], DALI [2], ProSup [3], and VAST [4]. So far, however, there is no universally accepted algorithm for determining the structural similarity of two proteins. This contrasts with the situation regarding protein sequence comparison where the sequence alignment program BLAST (basic local alignment search tool) [5] is the most frequently used and most widely accepted method for calculating sequence similarity. The BLAST program performs local alignment of sequences and finds short stretches of sequence similarity. While doing local alignment at the sequence level is not an easy task, it is considerably more difficult to perform a similar task at the three-dimensional structure level. And the challenge is further magnified when there are a large number of entries in the database against which good alignments are sought. The number of structures in the Protein Data Bank (www.pdb.org) have increased rapidly with more than 44,000 structures have been deposited by the end of June, 2007. Despite the existence of a number of 3D protein structure alignment tools, there is still a need for the development of new approaches for solving the problem from a slightly different angle. Inspired by the success of BLAST at sequence level, we have developed the program SBLAST (BLAST for structures) to perform BLAST-like search for 3D proteins. In this paper, we will discuss our approach for discovering structurally similar regions in proteins using a variation of the geometric hashing method.

The original geometric hashing concept was introduced in the field of machine vision to solve the object recognition problem [6]. Geometric hashing concept is predicated on the idea that the simplest invariant associated with any set of three points in space under any rigid transformation is the set of the three inter-point distances. These distances can thus be used as a "hash value" of the triple of points. In one implementation of geometric hashing, every choice of a basis of a set of two or three points corresponds to a transformation, and the other points are subjected to the same geometric transformation and indexed into the corresponding hash table bins [7]. Since, in this implementation, all combinations of the bases are selected, the resulting hash table records the locations of all points through all possible basis transformations. For searching with a query image of m points and target image of n points, the time complexity is $O(n^2m)$ if a two-point basis is used, and the time complexity is $O(n^3m)$ if a three-point basis (triangles) is used. While this technique is widely used in computer vision, it is not really suitable to analyze protein structures, which

contain in the order of thousands of atoms. A triangle-based approach will be discussed in this paper to approach local protein structure alignment problem. However the details of the method are considerably different from that of earlier techniques. In particular we discuss how this algorithm is adapted for massively parallel machines.

2. Methods

As mentioned earlier, every triple of points consisting of the 3-D coordinates of selected Cα atoms (henceforth referred to as a "triangle") is hashed to a set of three inter-point distances, giving a 3-dimensional hash value. The algorithm consists of three phases: 1) the preprocessing phase, 2) the hit search phase, and 3) the hit extension phase. The source code was implemented using a mixture of C, C++ and Message Passing Interface (MPI).

2.1 Preprocessing phase

During the preprocessing phase, all proteins in the database of potential database proteins are hashed. In other words, triangle information is extracted from each of the PDB files and is stored in a 3-D hash table with user-defined bin sizes. The hash key is generated using the lengths of each triangle. In practice, this hash function has been found to provide a good balance between hash table size, hash key computation and clustering (collisions). The size of the hash bin determines the granularity of the search. The larger the bin size, the more likely it will be that comparable triangles will hash to the same bin. However, if the bin size is too large, then many unrelated triangles will need to be evaluated during the extension phase. After preprocessing, a list of relevant triangles will be stored in a large hash table that can be used later in the hit search and the hit extension phases.

2.2 Hit Search Phase

Given a query protein structure, we first extract triangles from the query protein and search the matching triangles in the hash table generated at preprocessing phase. To accommodate the situation that a triangle in the border of a neighboring bin might be closer to the query triangle than some triangles in the hash bin that the query triangle is hashed into, we also implemented a neighbor search routine to find matches in neighboring hash bins.

2.3 Hit Extension Phase

As in sequence BLAST, once a hit is found (by matching a triangle from the query structure with a triangle of a database protein), we need to extend the hit to find locally maximal structure segment pairs, one in the query and one in the database protein. A recursive routine is used to extend the triangle hits. The goal is to find a pair of longest common substructures such that when they are structurally aligned, the root mean square deviation (RMSD) is below a user-defined threshold. To facilitate this process, an adjacency list of triangles is constructed and a depth-first search is simulated on it. Note that the triangles are considered to be adjacent if they share an edge.

The extension phase is implemented using a query-driven search that recursively extends the hits until the maximum allowable RMSD is reached. Details of the algorithm are given in Figure 1. A standard procedure is used to find the RMSD between two 3-D protein structures based on a matrix computation approach by Schonemann [8].

3. Parallel Design and Implementation

We used a "Master-Worker" paradigm to implement the parallel algorithm for SBLAST using MPI (message passing interface) routines on the IBM Blue Gene/L massively parallel supercomputer. The Blue Gene/L architecture is designed to scale to 65,536 dual-processor nodes (131,072 processors) with a peak performance of 360 teraflops [9]. The three phases of the algorithm described above were broken into two modules: 1) the preprocessing module and 2) the search module. Both modules were implemented in serial and parallel versions. In the SBLAST preprocessing module, a master processor listens for the requests of idle worker processors and sends a new PDB file for each worker to parse. The parsed results are the sent to an output master processor that outputs the results into four different files: 1) The hash table database stores the hash tables for all the proteins in the database; 2) The hash index file stores the beginning and ending bytes in the hash table for each protein; 3) The attribute database stores the attributes (coordinates, residue type, atom type, etc.) for all the proteins in the database; and 4) The attribute index file stores the beginning and ending bytes in the attribute table for each protein. The SBLAST search module uses the four files generated from preprocessing module. The master processor operates as a job-scheduler servicing requests from worker processors.

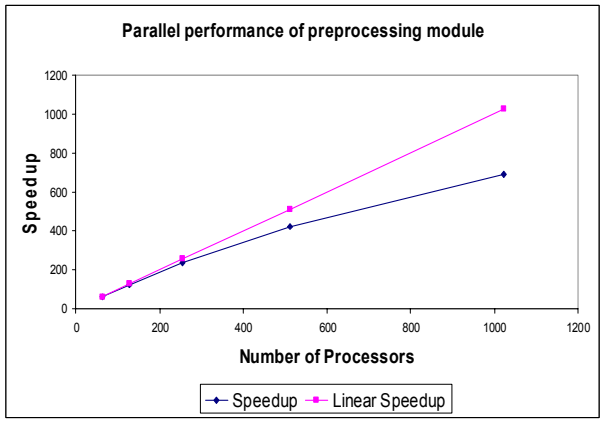
```
Input
  Q1, Q2, … Qi,… //Qi is the i^th triangle in query that has at least one hit
  Qi:{Ti1, Ti2, …, Tia, …} //Associative map for query triangle Qi, Tia is the a^th
matching triangle in target T that matches with Qi
  Q is the union of all query triangles with hits
  T is the union of all target triangles
  maxDev: the maximum root mean square deviation allowed
  minSize: the minimum substructure size allowed

Output:
        A list of matching common substructure pairs (local alignments)

search(Q, T)
   int k;
   VQ = number of triangles in Q
   VT = number of triangles in T
   for (k=0; k<VQ; k++)
      valQ[k] = unseen;
   for(k=0; k<VT; k++)
      valT[k] = unseen;
   Initialize subgraphQ_1 to subgraphQ_n as empty set
   Initialize subgraphT_ij as empty set
   for i=1 to VQ
      Qi is the i^th triangle in query
      for j=1 to number of target triangles that match with Qi
         subgraphT_ij = Tij
          if Qi is not visited
                doubleExtendTriangle(subgraphQ_i, subgraphT_ij, Qi, Tij)
      if maxDev and minSize constraints satisfied
          Output subgraph Q_i and subgraph T_ij as matching common substructure

adjlist(Q)
   for i=1 to VQ
     for j=1 to VQ
          if (Triangle_j shares an edge with Triangle_i)
         adj[i].push_back(triangle_j)

doubleExtendTriangle(subgraphQ, subgraphT, TQ, TT)
   mark triangle TQ as visited
   mark triangle TT as visited
   append unshared point of TQ into subgraphQ
   append unshared point of TT into subgraphT
   if appended subgraphQ is similar to appended subgraphT
      extend subgraphQ and subgraphT and do the following
      for i = 1 to number of neighbors of TQ
         TQ_i is the i^th neighbor of TQ
                for j = 1 to number of target triangles that matches with TQ_i
                TT_ij is the j^th target triangle that matches with TQ_i
                doubleExtendTriangle(subgraphQ, subgraphT, TQ_i, TT_ij)
   else do not extend
```

Figure 1. Pseudo code for the search and extension phase.

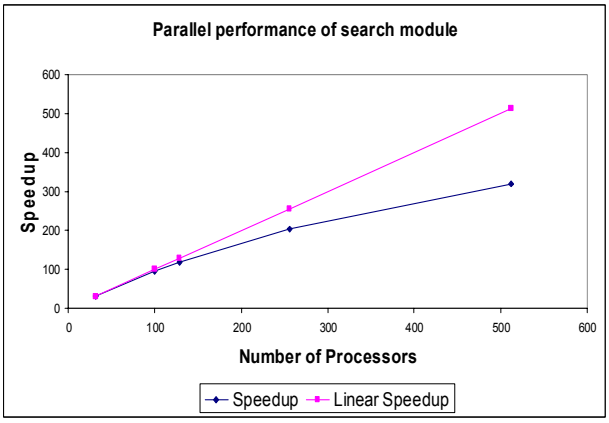
When the SBLAST search is initiated, the names of the query file, hash table database and attribute database are broadcasted to all the worker processors. The master processor then reads the index file for the hash table and the index file attribute table and determines the offsets of hash table and attribute table for each individual worker processor. A pair of offsets (the beginning byte and ending byte) for a particular hash table in the hash table database is sent to each worker processor. A pair of offsets for the corresponding attribute table is sent as well. Every worker will read the chunk of characters determined by the pair of offsets from the respective hash table and attribute table and subsequently parse the tables into data structures. A pair of proteins, consisting of a query and a database protein constitutes a job. A worker processor works on one job at a time and makes a new request to the master processor as soon as it finishes its job. The master works as a server and send the next pair of offsets to the requesting worker processors as soon as it receives the request. Output of the results is done in parallel by the worker processors. A post-processing routine merges the results generated by each individual worker processor. The output files contain matching common substructures and the corresponding RMSD between the superimposed common substructures.

4. Performance Evaluation of SBLAST

We used Message Passing Interface (MPI) to implement the parallel version of our algorithm. Our benchmark database consisted of 2898 proteins from ASTRAL40. The performance for preprocessing module is shown in Figure 2(a) and the performance of search module is shown in Figure 2(b).



(a)



(b)

Figure 2. Parallel performance of SBLAST on Blue Gene. (a) Preprocessing module, (b) Search module. A test database containing 2898 proteins was used. The scaling limit in both cases was the time to process the largest database proteins. Close to linear scaling is expected when preprocessing and searching the PDB as a whole.

5. Experiments

We performed our experiments on a smaller version of the PDB based on the ASTRAL Compendium of protein structures and sequences [10]. The version of the ASTRAL Compendium we used was one in which no two proteins share more than 40% sequence identity (ASTRAL40 in the sequel) (http://astral.berkeley.edu). We first tested if SBLAST program was able to find the common protein domain shared by several proteins. We queried trypsin (ASTRAL ID: d1s83a_) against ASTRAL40 and looked for proteins that shared some common substructure with the trypsin. We chose Trypsin because it is a relatively large protein. Eighty two local alignments with RMSD ranging from 0.098 to 0.669 were identified. We also chose the zinc finger domain for our experiments. A zinc finger is a DNA-binding domain typically consisting of two antiparallel β strands, and an α helix. Many regulatory proteins (e.g., transcription factors) contain zinc fingers. Alkaline phosphatase (ASTRAL ID: d1a1ia_) is a zinc finger protein. We queried d1aliia_ against the ASTRAL40 database to look for common substructures in other proteins in the database. Figure 3 shows a local alignment between d1s83a_ and d1p57b_, and another local alignment between d1a1ia_ and d1p7a_.

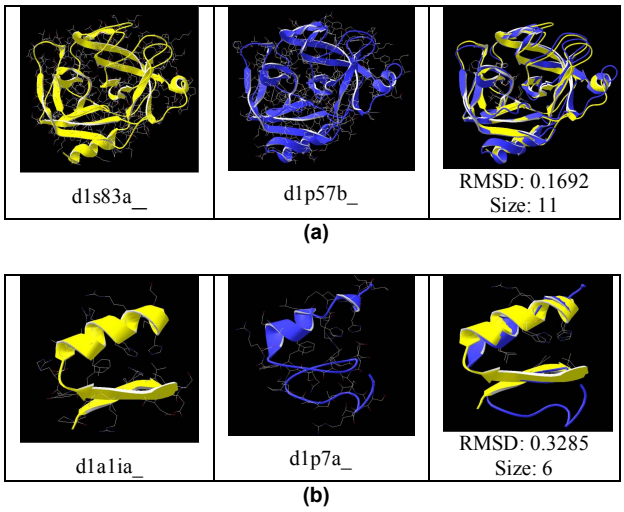


Figure 3. SBLAST search for common substructures in ASTRAL40. In (a), the query protein d1s83a_ is found to share a large common substructure of high similarity with the protein d1p57b_. In (b), the query protein d1a1ia_ is found to share a common substructural region with the protein d1p7a_. Note although d1p7a_ does not have the characteristic secondary structure topology of zinc finger proteins, the zinc binding residues are structurally well conserved.

6. Results and Discussion

Preliminary experiments show that SBLAST was able to identify common substructures in a large set of proteins. In Figure 2, above, the preprocessing module and the search module show effective speedups of 70% and 60% for 1000 and 500 processors respectively. An improvement in parallel efficiency is expected when the ratio between the number of database proteins and the number of processors increases for larger databases. Improvement in the absolute performance of the search function would require that the current parallel algorithm be rewritten to allow more than one processor per database protein. Currently SBLAST provides alignment measures using length and RMSD of the common substructures. Future work will be to explore the distribution of structural alignments so as to better determine the statistical significance of an alignment.

7. Acknowledgements

This work was supported by the IBM Systems Technology Group at Rochester, Minnesota. Zheng and Milledge, who worked as IBM Co-ops, would like to express deep gratitude to all the members involved with Blue Gene/L's software and development team. Special thanks to our manager Carl Obert, our mentors, Carlos Sosa, Brian Smith, Amanda Peters, and Jeffrey McAlister. We would like to express our gratitude to IBM Capacity on Demand Center at Rochester, Minnesota. IBM, Power PC, and Blue Gene are registered trademark of IBM Corporation. The work of G.N. was supported in part by NIH Grants P01 DA15027-01 and NIH/NIGMS S06 GM008205.

8. References

[1] Shindyalov, I.N. and P.E. Bourne, *Protein structure alignment by incremental combinatorial extension (CH) of the optimal path.* Protein Eng., 1998. **11**: p. 739–747.

[2] Holm, L. and C. Sander, *Protein structure comparison by alignment of distance matrices.* J. Mol. Biol., 1993. **233**: p. 123–138.

[3] Lackner, P., W.A. Koppensteiner, M.J. Sippl, and F.S. Domingues, *ProSup: a refined tool for protein structure alignment.* Protein Eng., 2000. **13**: p. 745–752.

[4] Yang, A.-S. and B. Honig, *An integrated approach to the analysis and modeling of protein sequences and structures. I. protein structural alignment and a quantitative measure for protein structural distance.* J. Mol. Biol., 2000. **301**: p. 665–678.

[5] Altschul, S., W. Gish, W. Miller, M. EW, and L. DJ, *Basic local alignment search tool.* Journal of Molecular Biology, 1990. **215**(3): p. 403-410.

[6] Lamdan, Y., J.T. Schwartz, and H.J. Wolfson. *Object Recognition by Affine Invariant Matching*. in *Proceedings of Computer Vision and Pattern Recognition*. 1988.

[7] Lamdan, Y. and H. Wolfson. *Geometric hashing: A general and efficient model-based recognition scheme.* in *Proceedings of the International Conference on Computer Vision*. 1988. Los Alamitos: Computer Society Press.

[8] Schonemann, P., *A generalized solution of the orthogonal procrustes problem.* Psychometrika, 1966. **31**: p. 1-10.

[9] Gara, M.A., D. Blumrich, D. Chen, G.L.-T. Chiu, P. Coteus, M.E. Giampapa, R.A. Haring, P. Heidelberger, D. Hoenicke, G.V. Kopcsay, T.A. Liebsch, M. Ohmacht, B.D. Steinmacher-Burow, T. Takken, and P. Vranas, *Overview of the Blue Gene/L System Architecture.* IBM Journal of Research and Development, 2005. **49**(2/3): p. 195-212.

[10] Brenner, S., P. Koehl, and M. Levitt, *The ASTRAL compendium for sequence and structure analysis*. Nucleic Acids Research, 2000. **28**: p. 254-256.