Sequence Analysis on a 216-Processor Beowulf Cluster

Katerina Michalickova
Dept. of Biochemistry, University of Toronto and Samuel Lunenfeld Research Institute, Toronto, ON, Canada
katerina@mshri.on.ca

Moyez Dharsee
Samuel Lunenfeld Research Institute, Toronto, ON, Canada
dharsee@mshri.on.ca

Christopher W. V. Hogue
Dept. of Biochemistry, University of Toronto and Samuel Lunenfeld Research Institute, Toronto, ON, Canada
hogue@mshri.on.ca

Abstract

In this work we describe the implementation of a 216-processor Beowulf cluster with switched gigabit Ethernet networking. This design includes the use of a 8-CPU high performance midrange computer with 8 gigabit ports as a cluster head, a design that limits I/O contention. We have been developing applications software for bioinformatics research in protein folding, as well as the MoBiDiCK system for managing cluster applications that is extensible to general purpose distributed computing. In addition to the cluster architecture, we present a new cluster application for bioinformatics, a variant of the BLAST family of sequence comparison programs. MOBLAST performs the BLAST algorithm in an exhaustive manner, avoiding its initial heuristic approach to finding hits. This effectively slows BLAST down to approach the speed of other comprehensive search methods such as a Smith-Waterman alignment. MOBLAST requires a sizeable cluster to run. We describe the development of MOBLAST and its use in making an exhaustive M×N database of alignments where M is the set of protein sequences with known 3-D structures, and N is the set of all protein sequences. This M×N database of protein alignments will facilitate further research in protein folding, the ultimate aim of our work with Beowulf cluster technology. Furthermore, we describe a general algorithm for partitioning M×N problems and implement this in the MoBiDiCK computing model.

1. Introduction

1.1. Bioinformatics

We define bioinformatics as the science of testing biological hypotheses using informatics approaches. High-throughput biology has been producing a wealth of new information in the form of DNA and protein sequences, 3-dimensional molecular structures as well as newer data relating to the interconnected networks of molecular interactions that form the molecular machines of life. We are interested in a fundamental problem, the protein folding problem. We have been early adopters of Linux and Beowulf clusters in adapting these computational resources through the creation of new software in order to tackle the compute-intensive problems in bioinformatics.

1.1. M×N sequence comparisons – assigning local structure to sequence

Most of the research in bioinformatics focuses on functional assignment of proteins, the working molecules forming the machinery inside a cell. So far the best results of assigning function to protein sequences come from studies of sequence similarity. A routine operation for a biologist is to query a new protein sequence against a sequence database to obtain a list of similar sequences, then use these results to infer the function of the query sequence. The core algorithms for comparing sequences involve making alignments of sequences and finding regions of sequence that are similar according to a statistical model. Algorithms that find optimal alignments can be time-consuming when run over a large database. The BLAST[1] family of programs are widely used for searching DNA and protein databases for sequence similarities; they employ a heuristic for speeding up the search[2]. Unlike a Smith-Waterman alignment[3] BLAST does not perform an exhaustive optimization of the query against every sequence in the database. Instead it scans the database with a heuristic and only when the condition of the BLAST heuristic is satisfied is the optimal alignment computed, scored and reported.

We are interested in queries of protein sequences with known 3-dimensional structures. With the addition of

3-D information, even a very short alignment can provide significant information to a biologist[4], as it may be inferred that the local 3-D structures of similar sequences are also found in the query sequence. Short alignments are often skipped by the heuristics in BLAST. We have decided to build a cluster-based BLAST application that bypasses the heuristic and performs the exhaustive alignment between the query and each sequence in the database using the core BLAST alignment routines. We present MOBLAST, a cluster application integrated with the MoBiDiCK[5] distributed computing system, developed using the NCBI software development toolkit[6]. MOBLAST performs exhaustive protein-protein BLAST alignments in an M×N manner and reports BLAST statistics and alignments.

To obtain an exhaustive comparison of each protein with 3-D information against the database of known protein sequences, we have implemented software to compare these to a non-redundant protein sequence database (NR) in an M×N fashion. In this case N is the protein sequence database size, and M is the set of sequences from known 3-D structures. We are also aware that other biological comparisons are useful to perform in an M×N fashion[7]. This is because it is often faster to retrieve a precomputed comparison from a database than to compute the result on the fly. Databases of precomputed comparisons are valuable resources for biologists, especially in clustering similar sequences or 3-D structures[7][8]. We report a general method to parameterize an M×N comparison using the MoBiDiCK system that avoids processing symmetric duplicate comparisons in the upper triangle of an overlapping range in a database.

2. Cluster architecture

Our cluster consists of 108 independent 2U rack-mount dual-PIII 450 MHz CPU computers, which contribute a total of 216 CPUs to the system as a Beowulf cluster. These nodes are interconnected with fiber optic cable using Gigabit Ethernet with two 64-port Gigabit Ethernet switches. In addition, the cluster has a "head" computer, a high performance HP N-Class server with 8 64-bit PA-RISC CPUs. The "head" initiates compute jobs and collects the output from the nodes, and it stores results on a high-performance Fiber-Channel RAID system. The head has 8 Gigabit Ethernet ports connecting through 4 fibers to each of the two switches.

The headed cluster was chosen as an architecture because of some severe I/O contention we noticed on our smaller 32-processor cluster as jobs completed on nodes simultaneously with our protein folding software applications, which are described elsewhere[8]. This new architecture enables a large number of simultaneous data exchanges with many nodes in the cluster. This is possible because the head computer has approximately 14 times the bandwidth of a single node, owing to the 8 64-bit PCI Gigabit cards in the head, versus 1 32-bit PCI Gigabit card in each node.

Although the architecture we have selected is certainly not the least expensive route, we selected components and vendors by focusing on the ability to upgrade the cluster in a cost-effective manner. We estimate that doubling the current system's performance will cost ¼ of the initial expenditure. We have identified the following options to upgrade the current architecture.

Upgrade Paths – Nodes. We selected rackmount nodes that employ standard components, and rejected those we considered overly customized. The cases we have selected together with the power supplies and the hard disks can all be conserved on upgrade. Each of the 2U rackmount computers can be upgraded with industry standard ATX or NTX motherboards, RAM and new CPUs. Future upgrades could be selected from Intel, Alpha or even PowerPC architectures, whichever is most cost effective.

Upgrade Paths – Head. The HP N-class was selected based on knowledge that it is one of the latest designs on the market. The head computer can be field-upgraded to faster processors, more RAM and more high speed disk. It is CPU-upgradable to the Intel/HP 64-bit EPIC architecture chips, commonly known as "Merced", (released with the name "Itanium"). Indications are that this will be capable of running 64-bit Linux, however upgrades to faster versions of the PA-RISC architecture CPU and HP-UX are also planned by HP. The "head" is also capable of being upgraded to 64 Gb of RAM.

Upgrade Paths – Networking. The existing network comprising 64-bit PCI network cards, fiber optic cabling, and switches can be conserved. The 64-bit PCI network cards are currently only being used with a 32-bit PCI bus; their speed is not fully realized in the current nodes. We recently learned that Foundry Networks will begin shipping a chassis that is twice as large and supports the same "blades" in our current switch. We can upgrade by consolidating our existing Gigabit "blades" into a single 128-port wire-speed crossbar switch as the cluster backplane. This can allow us to add 10 more nodes from the ports freed up in consolidating the two switches, or perhaps contemplate doubling the size of the cluster.

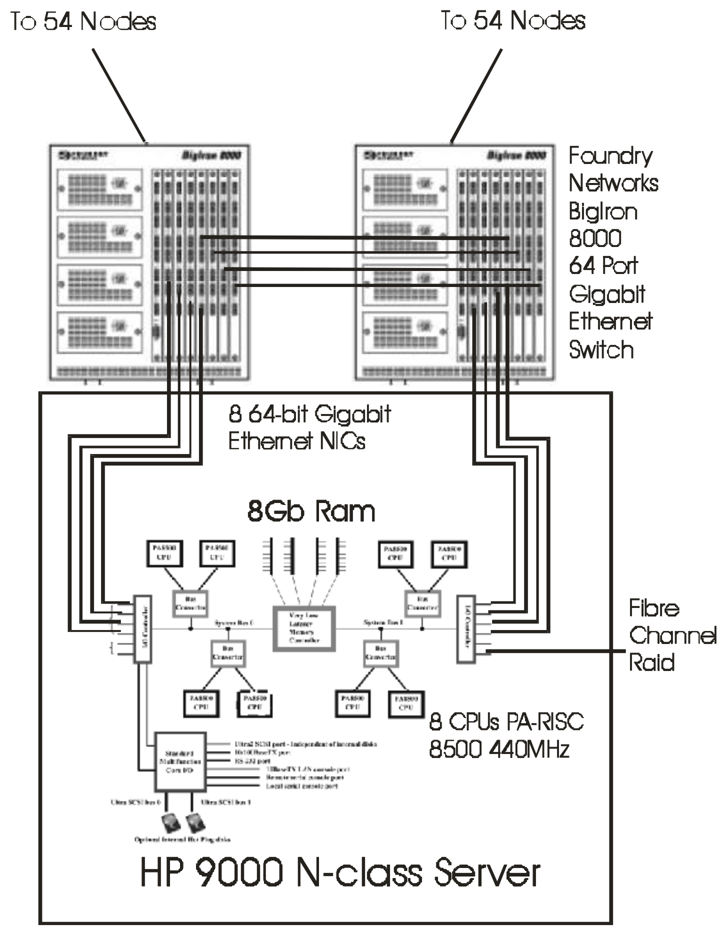

Figure 1. YAC Networking and Head Architecture. In addition to 54 fiber optic cables connecting nodes, each switch has 2 up-links in a ring topology with additional switches on our LAN, accounting for all 128 ports. Photographs of the cluster are at http://bioinfo.mshri.on.ca/yac/.

Table 1. Cluster Specifications

Overall:	
RAM:	64 Gb
HD:	1.6 Tb Disk Storage
CPUs:	224
NIC:	Gigabit Ethernet
OS:	Linux/HP-Unix
PRICE:	$1,300,000 CDN ($884,000 US)
Cluster Head	
Manufacturer:	Hewlett Packard
Model:	9000 N-class Server
RAM:	8Gb
HD:	300 Gb AutoRAID
CPU:	8 440 MHz PA-RISC 8500 64bit
NIC:	8 Gigabit Ethernet, 64 bit PCI cards
OS:	HP-UX 11.0
108 Cluster Nodes	
Manufacturer:	VA Linux (94 nodes + 14 from original cluster)
Model:	FullOn 2x2
RAM:	512K
HD:	14Gb IBM
CPU:	2 450 MHz Intel Pentium III
NIC:	1 Intel Pro/1000 Gigabit Ethernet , 1 integrated Intel 100/BT
OS:	Linux (Custom VALinux SMP kernel)
2 Network Switches	
Manufacturer:	Foundry Networks
Model:	BigIron 8000
NIC:	64 Gigabit Ethernet Ports
Air Conditioner	
Manufacturer:	Carrier
Model:	40RM-014-B600HC
Capacity:	12.5 Ton with Outdoor Roof-Mounted Chiller
6 Racks	
Manufacturer:	DL Custom
Model:	Controller Relay Rack

3. Administration

While the cluster is fully capable of running MPI or PVM based parallel software, we have focused on developing cluster middleware that allows us to perform computations with a minimum of administrative effort. This is important to us for three reasons. First, we are limited in that we do not have operating budgets required to administer a complex system. Second, we need to make our cluster based software accessible to Biologists who may be accustomed to a Web-based interface for running software. Last, we wish to be able to branch out from cluster computing to wider-area distributed computing, and we have sought to develop a mechanism by which we could make use of other computers and clusters over the Internet with MoBiDiCK distributed computing system.

Using MoBiDiCK, a computational task can be partitioned into subtasks and distributed over a set of Web server nodes. On each node a CGI program (called a "TaskApp") is launched with a unique set of parameter values that define a single partition of the overall task. Hence this system implements the single-program, multiple-data (SPMD) parallel processing model. Computations are managed on a kernel server running a set of kernel modules, namely Dispatcher, Status, Statekeeper, Collector, and DataManager. A Web browser interface is used to interact with these modules to administer nodes and computations. MoBiDiCK is back-ended by a database that holds state information about nodes and tasks. The MoBiDiCK kernel runs on the HP N-class server, the cluster head.

A computation is started with the Dispatcher module to partition and dispatch the task. The Collector handles output collection and local node cleanup, while the Status and Statekeeper modules perform task monitoring and fault-tolerance functions. Since there is no need to directly log into a node to run a task, user accounts are not required on the cluster nodes. As with any CGI program, a MoBiDiCK TaskApp is launched by the HTTP server under pre-configured restricted user and group IDs (usually the “nobody” user and group on Linux). We configured the cluster nodes to run the Apache Web Server[9] using a modified “TimeOut” directive to allow TaskApps to execute beyond the 5 minute default value.

4. Cluster applications

The cluster computer is capable of running a variety of software optimized for Beowulf clusters using industry standard systems (PVM, MPI) for controlling parallel applications. This includes software important in our field of research for studying molecular dynamics (CHARMM) and for the production of high-resolution 3-D ray-traced graphics (PVM-POVRAY). We have already described our main application, FOLDTRAJ, which is used for sampling the 3-dimensional conformational space available to a protein structure. Here we present a new application for exhaustive sequence comparisons.

4.1. MOBLAST – parallel M×N BLAST with exhaustive alignments

We report a new cluster application aimed at providing exhaustive protein sequence alignments in an M×N comparison. We approximate using a figure of 500,000 non-redundant sequences, rounded up from the current database size. About 10,000 of these sequences have known 3-D structures. We are interested in obtaining a database of the pre-computed comparison of all sequences against each other, to serve as a resource for a variety of other computations. This is not difficult to compute on a cluster of this size with the currently distributed BLAST application for protein sequences, *blastpgp*, which is supplied in the NCBI toolkit. However we are interested in a more detailed computation for sequences with known 3-D structure, an M×N comparison, and we have written a new application for this purpose, called MOBLAST.

We have undertaken to estimate the time and storage requirements for this new computation as we have been building our cluster. This has guided us to find where there are limiting amounts of time and storage as indicated in Table 2. The most ambitious computation of NxN with an exhaustive BLAST method would require almost a year of time and 30 Tb of space to record the output!

The BLAST programs are a part of the National Center for Biotechnology Information (NCBI) programming toolkit. In MOBLAST we use a core routine which performs a pairwise alignment of two sequences, with a function call known as `BlastTwoSeqs()`. In order to speed up the algorithm, we eliminated some overhead I/O by reading the scoring matrix file into memory at the start of execution instead of it being read at every function call. Additionally, we implemented POSIX threading to run the executable on two CPUs simultaneously which optimized memory use on each node by only loading one copy of the database range being searched.

To estimate how much time one cluster node takes per comparison we averaged executions on different database ranges. Each run compared a subset the database range in NxN fashion (only in lower triangle including the diagonal) using threaded MOBLAST on our dual CPU cluster nodes simultaneously. Timing for MOBLAST was estimated from 26 test runs on a database sample size of 500 sequences (in NxN comparison in lower triangle including diagonal). Sequences were sampled from the database, which at the time of the run contained 408,950 sequences, and the database is present on each node’s hard disk at time of execution. The BLAST parameters were left at defaults, which are the same as the *blastpgp* parameters. The “expect” (E) parameter value was 10.0. The average time to compute one MOBLAST comparison on a single cluster node (threaded on two CPUs) was determined to be 0.027 seconds. Overhead does not add significantly to the computation, as it is “embarassingly parallel” in nature. Only 63% of MOBLAST comparisons return a “seqalign” object, of average length 342 bytes in an optimized format. The rest of the comparisons fall under the cut-off value and are not stored.

We also tested the *blastpgp* program directly from the NCBI toolkit in 30 runs. The parameters were BLAST defaults, which include the E value of 10.0. In each run, one sequence (selected by stepping through the entire database over 15,000 sequences at the time) was compared to the NR database (485,275 sequences). The average query length in this trial was 380 characters (the overall average for the whole database is 313 characters). The average time to complete one query was 48 seconds and average number of hits in one query was 1814. It is important to note that the output can be limited by altering the BLAST parameters to cut off less significant hits.

Table 2. Resource Estimates for M×N BLAST Comparisons

Problem	*No. of Comparisons*	*Time on Cluster*	*Output Storage*
1XN (MOBLAST)	500,000	2 min. (estimated)	108 Mb
1xN (blastpgp - heuristic)	500,000	48 sec. just 1 CPU	171 Kb
M×N (MOBLAST; structures vs. sequences)	500,000 x 10,000 (lower triangle)	14 days (estimated)	1 Tb
NxN (blastpgp - heuristic)	$500{,}000^2$	3 days (estimated)	85 Gb
NxN (MOBLAST)	$\sim 500{,}000^2/2$ (lower triangle)	355 days (estimated)	27 Tb

4.2. A generalization for M×N comparisons with MoBiDiCK

MOBLAST is designed to operate with the Modular Big Distributed Computing Kernel (MoBiDiCK). MOBLAST was integrated with MoBiDiCK with the task applications programming interface (Task API). We recognize that other applications in bioinformatics employ M×N comparisons, and therefore sought a generalized framework to partition tasks in that category under MoBiDiCK.

Given a set of query sequences and a set of target sequences, MOBLAST performs a BLAST comparison of each query sequence with each target sequence. Since a sequence is actually a database record, we can express the query and target sets as database ranges Q* and T* such that $Q^*=[q_L, q_U]$, $T^*=[t_L, t_U]$; q_L and t_L denote lower record boundaries, while q_U and t_U denote upper record boundaries of Q* and T*, respectively.

A database can be viewed as a finite ordered set of discrete points along an axis, each point representing a database record. In Figure 2 we illustrate a database using two axes to represent separately the query and target ranges. Note that both axes correspond to the same database. The shaded region consists of all possible pairs of records between ranges Q* and T*, each pair representing a single exhaustive BLAST comparison. BLAST is a commutative operation, so that given two records *a* and *b*, BLAST(*a*,*b*) is equivalent to BLAST(*b*,*a*). If Q* and T* share an overlap range, V*, then each pair in the upper triangle region of V* has an equivalent pair in the lower triangle; we can thus discard all pairs appearing strictly above the diagonal that divides V*. Figure 3(a-c) shows that if Q* and T* overlap, the overlapping range V* can occur in different ways. In each case the overlap range V* can always be divided into equivalent lower and upper triangles along a diagonal.

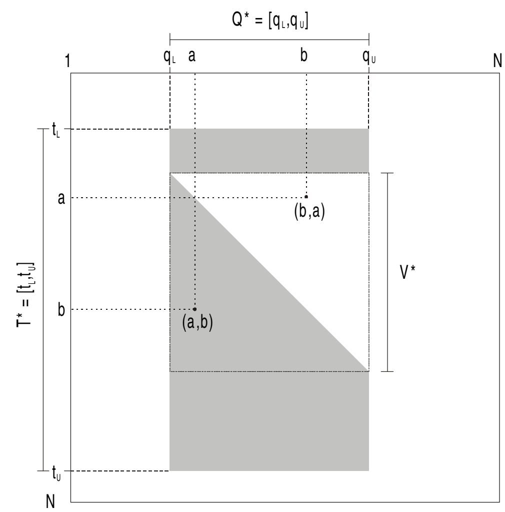

Figure 2. Query and target sets are represented as database ranges. The horizontal axis holds the query range (Q*), while the vertical axis holds the target range (T*). An overlap region (V*) occurs if some records are in both Q* and T*. Record pairs above the diagonal V* are redundant if the operation to be performed is commutative, as is the case for BLAST.

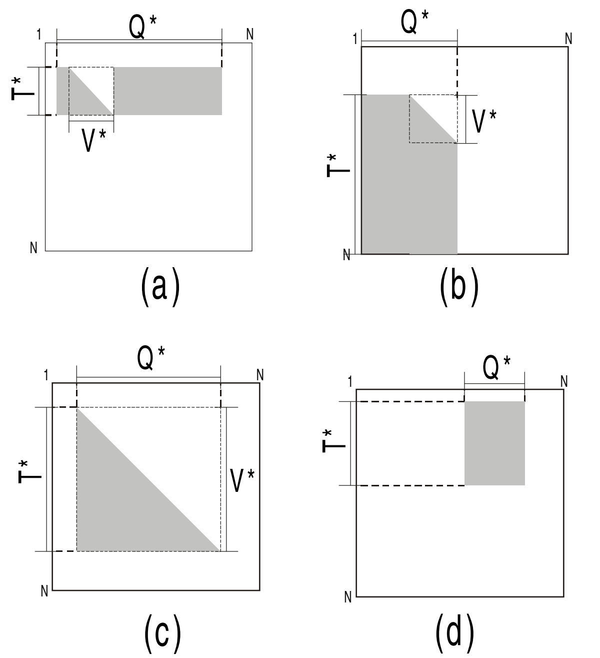

Figure 3. Query (Q*) and target (T*) ranges can overlap partially (a, b), fully (c), or not at all (d). Only record pairs in the lower triangle of the overlap range (V*) are to be compared with BLAST.

We now describe a scheme for partitioning a MOBLAST computation into subtasks that can be distributed using MoBiDiCK. This scheme can serve as an asbstraction that is generalizable to other M×N computations in which a binary operator (such as BLAST) must be applied to record pairs from two ranges of a database. Given a global query range Q* of size M and a global target range T* of size N, we divide both Q* and T* by a partitioning factor *p*, resulting in *local* query ranges Q_1, Q_2, …,Q_p, and *local* target ranges T_1, T_2, …,T_p. as in the example in Figure 4. For simplicity we assume here that both M and N are divisible by *p*. Each cell in the resulting grid represents a distinct MOBLAST subtask that can be performed independently of the others. Partitioning is automatically done by the MoBiDiCK Dispatcher, given values for Q*, T*, and *p*. A total of p^2 subtasks are generated, each subtask being assigned to one node. If the number of subtasks exceeds the number of available nodes, the remaining subtasks that cannot be initially assigned are placed in a queue and are dispatched as nodes become available.

4.3. The M×N algorithm

Each grid cell in Figure 4 represents a subtask of a MOBLAST computation or other pairwise comparison. According to the shape of the shaded region of a cell, we characterize the corresponding subtask as *full* (entire cell is shaded), *partial* (part of the cell is shaded), or *emtpy* (cell is unshaded). All subtasks strictly outside V* are full; subtasks appearing in V* that are strictly below the diagonal are also full. Subtasks completely in V* that are strictly above the diagonal are empty. Finally, all subtasks that intersect V* are partial. Note that if there is no global overlap, all subtasks are full.

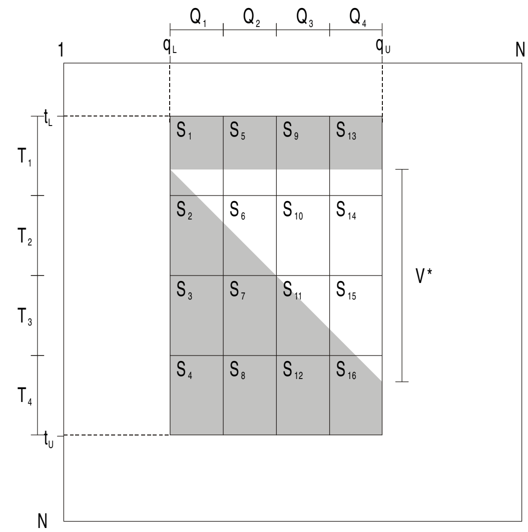

Figure 4. Example of partitioned query and target ranges divided by a partition factor *p* of 4 to produce a grid of 16 subtasks. *Full* subtasks are completely shaded (S_3, S_4, S_7, S_8, S_{12}); *partial* subtasks (S_1, S_2, S_5, S_6, S_9, S_{11}, S_{13}, S_{15}, S_{16}) intersect the diagonal and are only partly shaded; *emtpy* subtasks are unshaded because they appear in the upper triangle of the global overlap region, V*.

We describe the pseudocode for the MOBLAST algorithm given in Table 3. Given global ranges Q* and T* and local ranges Q and T, MOBLAST begins by determining whether the subtask is full, partial, or empty. If neither Q nor T intersect the global overlap range V*, the subtask is full (lines 4, 13). If one of Q and T is in V* (line 4), the subtask is partial only if there is a local overlap between Q and T, that is, if the local overlap range V is non-empty (line 5). If there is no local overlap, the subtask is full if it is in the lower triangle of V*, which occurs only when the lower boundary, T_l, of the target range is greater then the upper boundary, Q_u, of the query range (line 8). If this is not the case, the subtask is in the upper triangle of V* and is thus emtpy, resulting in no comparisons (line 11). BLAST_FULL simply performs a BLAST comparison for every pair between the query and target ranges. BLAST_PARTIAL discards pairs that occur in the upper triangle region of the global overlap range V*.

Table 3. MOBLAST Algorithm

```
MOBLAST(Q*,T*,Q,T)
      V* ← Q* ∩ T*         ⇨ global overlap range
      V  ← Q  ∩ T       ⇨ local overlap range
      if V* ≠ {}                     ⇨ if there is global overlap
         if (Q ∩ V*) OR (T ∩ V*)  ⇨ we're in V*
            if V ≠ {}                    ⇨ there is local overlap
               do BLAST_PARTIAL(Q,T,V*)
            else                               ⇨ no local overlap
               if T_l > Q_u                      ⇨ in lower triangle
                  do BLAST_FULL(Q,T)
               else                              ⇨ in upper triangle
                  return
         else                               ⇨ we're not in V*
            do BLAST_FULL(Q,T)
      else                           ⇨ no global overlap
         do BLAST_FULL(Q,T)


BLAST_FULL(Q,T)      ⇨ compare all pairs
      for i ← Q_l … Q_u
         for j ← T_l … T_u
            BLAST(i,j)


BLAST_PARTIAL(Q,T,V*)      ⇨ only compare pairs outside upper triangle
      for i ← Q_l … Q_u
         for j ← T_l … T_u
            if (i ∈ V*) AND (j ∈ V*)
               if (j ≥ i)
                  BLAST(i,j)
            else
               BLAST(i,j)
```

5. Conclusions and future directions

The M×N comparison of proteins with known structures to other sequences creates a database that will allow us to accumulate plausible local structures that may be rapidly assigned to any given sequence, combined with evolutionary information from similar sequences. The accumulation of information about local structure can effectively seed a conformational search procedure and we anticipate that this combined information will expedite a protein folding computation. This would proceed as follows:

- given a query protein sequence *p*
- *S* similar sequences are found using *blastpgp*
- All 3-D fragments *F* matching any subsequence in *S* are retrieved from the M×N database
- A 3-D trajectory distribution *T* is derived from the assembly of *F* mapped through alignments of *S* onto the sequence of *p*
- *T* and *p* form the input to FOLDTRAJ
- Trajectory directed ensemble sampling attempts to predict the 3-D structure

We will describe in more detail this work at our next opportunity. In this work we devised a partitioning scheme for the MOBLAST computation which divides query and target database ranges into smaller subranges. Each pair of query and target subranges can thus be distributed as an independent subtask to a cluster node using the MoBiDiCK distributed computing system. This scheme, however, is not limited to MOBLAST or to MoBiDiCK, and can be generalized to other operations to be performed between pairs of records in a database.

6. References

[1] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J. Mol. Biol. (1990) 215:403-410.

[2] Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic. Acids. Res. (1997) 25:3389-3402.

[3] Smith, TF and Waterman, MS. Identification of common molecular subsequences. J. Mol. Biol. (1981) 147:195-197.

[4] Simons KT, Kooperberg C, Huang E, Baker D. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. J Mol Biol (1997) 268:209-225.

[5] Dharsee, M. Hogue, C.W.V. MoBiDiCK: A Tool for Distributed Computing on the Internet. In *Proceedings, 9th Heterogeneous Computing Workshop*, May 2000.

[6] Ostell, J., et al. The NCBI Software Development Toolkit. (6.0). ftp://ftp.ncbi.nlm.nih.gov/toolbox/.

[7] Gibrat, J-F., Madej, T., and Bryant, S.H. Surprising similarities in structure comparison. (1996) Curr. Opin, Struct. Biol. 6, 377-385.

[8] Feldman, H.J., Hogue, C.W.V. A Fast Method to Sample Real Protein Coformational Space. Proteins: Structure Function and Genetics, 2000. 39:112-131.

[9] The Apache Software Foundation. Apache HTTP Server Project. http://www.apache.org/httpd.html.