Evolutionary biology and computational grids

Craig A. Stewart (IU), Tan Tin Wee (NUS), Markus Buckhorn (ACsys), David Hart (IU), Donald K. Berry (IU), Louxin Zhang (NUS), Eric Wernert (IU), Meena Sakharkar (NUS), Will Fisher (IU), Donald F. McMullen (IU)

{stewart,dhart,dkberry,palmer,ewernert,wfischer,mcmullen}@indiana.edu
tinwee@irdu.nus.sg,lxzhang@krdl.org.sg,meena@bic.nus.edu.sg
markus@acsys.anu.edu.au

IU: Indiana University, Bloomington Indiana 47408, US;
NUS: National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260;
ACSys: Advanced Computational Systems CRSISE Building, Australian National University, Canberra 0200, Australia.

Please cite as: Stewart, C.A., T.W. Tan, M. Buckhorn, D. Hart, D.K. Berry, L. Zhang, E. Wernert, M. Sakharkar, W. Fischer and D.F. McMullen. Evolutionary biology and computational grids. Paper presented at: *CASCON Workshop on Computational Biology* (Mississauga, Ontario, Canada, 10 Nov, 1999). Available from: http://hdl.handle.net/2022/14008

NB: The proceeding of the Workshop on Computational Biology was originally published online. Through changes in web sites, the online publication of this article became unavailable. It is presented here in document format.

Introduction

The global high performance computing community has seen two overarching changes in the past five years. One of these changes was the consolidation toward SMP clusters as the predominant HPC system architecture. The other change was the emergence of computing grids as an important architecture in high performance computing. Several major national and international projects are now underway to develop grid technologies. Computational grids will increase the resources available to the most advanced computational scientists and encourage the use of advanced techniques by researchers who have not traditionally employed such technologies [1]. In the latter camp are bioinformaticists in general and evolutionary biologists in particular, although this situation is changing rapidly.

The need for advanced computational techniques in biology is being driven by dramatic increases in availability of data. Sequences from 47,000 species, totaling nearly 3 billion base pairs, are already available within Genbank [2]. The sequence data available via Genbank are growing exponentially, doubling roughly every 14 months. When complete, the Human Genome DataBase alone will include data for a total of roughly 3 billion base pairs [3].

The availability of large data sets of genome data make possible the use of statistical techniques for inferring evolutionary relationships among genes, gene products, organelles, and organisms [4]. These statistical techniques include maximum likelihood methods for constructing evolutionary phylogenies. One of the more popular programs for performing maximum likelihood analysis is fastDNAml, a program produced by Olsen et. al. [5], based on Felsenstein's dnaml program [6].With data sets that include hundreds of thousands of nucleotides, such comparisons can clearly involve significant computations. Because of limited availability of computational resources, biologists have often been forced to use only a portion of the data available to address a particular research question. This situation has slowed, without good reason, advances in biological research. Limitations created by lack of computational resources are a particular problem for scientists using maximum likelihood techniques in evolutionary biology, as such techniques are the most compute-intensive of those used for phylogenetic inference [7,8].

We thus face all at once the availability of statistical methods for inference of evolutionary phylogenies, the availability of massive genomic databases, the need for greater computational power to attack significant questions in evolutionary biology, and the emergence of computational grids as a major architecture in high performance computing. The confluence of these events created a natural opportunity to attack problems in evolutionary biology with computational grids. In this report, we first describe the algorithm used by fastDNAml to create maximum likelihood phylogenies. Then we describe the modifications we have made to fastDNAml in order to prepare it to run in a grid-based environment. We then discuss the performance of fastDNAml running on a single SP system and distributed across multiple systems tied together via high-speed international networks. Lastly we touch on highlights of some of the biological results obtained as a result of this work and some future plans for this project. Inasmuch as we discuss some areas that are well known to biologists but not perhaps generally well known to HPC experts, and vice versa, we provide some background in the biological and statistical techniques used in maximum likelihood phylogenetic inference, and some background in the relevant areas of HPC.

Maximum likelihood phylogenetic inference

An evolutionary phylogeny is a description of the evolutionary relationships among genes, gene products, organelles, or organisms. Such phylogenies are usually depicted as a tree diagram (Figure 1).

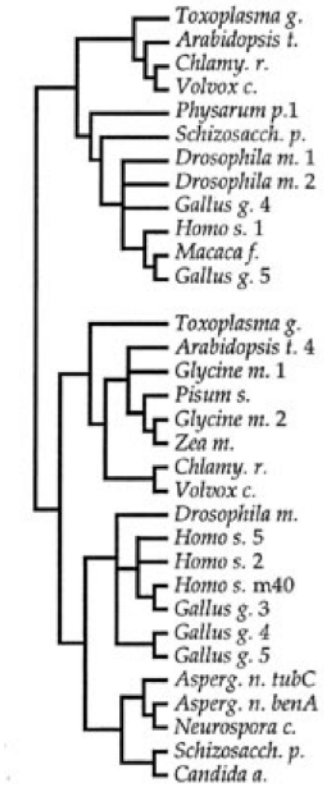

Figure 1. Example of a phylogenetic tree.

A good overview of statistical methods for phylogenetic analysis is provided in [8], and a good general overview of evolution and phylogenetics is available in [9]. Maximum likelihood phylogenies search for an evolutionary model that has the maximal likelihood given the data. In this sense a maximum likelihood phylogeny is different than traditional statistical hypothesis testing, in that the likelihood values can be used only for comparisons among competing models. One of the most important early developments in this area was the development by Felsenstein [4,9] of a program called dnaml. dnaml was the basis for the program fastDNAml, developed by Olsen and his colleagues [5]. The following description of maximum likelihood estimation of phylogenies and the operation of fastDNAml owes greatly to [4] and [5].

Maximum likelihood phylogenetic techniques assume a Markov model of base substitution. That is, they are applicable to evolutionary changes that involve the replacement of one nucleotide by another in the sequence of molecules that comprise the genetic information in a DNA or RNA molecule. These replacement events take place with a probability that may depend on the particular base substitution as well as the particular base locus. However, insertion and deletion events – the addition or removal of a section of genetic information as a unit – are outside the types of evolutionary processes modeled by maximum likelihood analysis. Given correctly aligned sequences of DNA and RNA from any two organisms, similar at some positions but dissimilar at others, and correct probability functions for base substitution events, one can then calculate a

probability value for a transformation from one sequence to the other. It is the ability to calculate such a probability value that makes it possible to create phylogenetic trees such as that shown in Figure 1. Both the tree topology and the branch lengths (which are proportional to the genetic differences between various taxa) are estimated from the data [4,5,8]. Because of this, the maximum likelihood values calculated are not the basis for any sort of hypothesis testing, nor anything like the probability that a particular tree is the correct one. Instead, the likelihood values are used to compare among trees and determine which is best.

All evolutionary phylogenies are bifurcating trees because it is presumed that evolutionary change takes place by a series of bifurcations. The phylogenies produced by maximum likelihood methods are unrooted trees because it may well be the case that no form ancestral to all the others is even included in the data set analyzed. Even if it were, the tree building process itself provides no way to identify it. An ideal brute-force method for finding the maximum-likelihood tree for *n* organisms would be to take all possible topologies, estimate the appropriate branch lengths for the trees, and then take the tree with the highest likelihood value. However, given n taxa, the number of bifurcating unrooted trees is [5]

$$(2n-5)!/2^{n-3}\,(n-3)!$$

which for 50 taxa is $O(10^{74})$. The brute force method is obviously impractical for anything other than small numbers of taxa, whereas the availability of large amounts of sequence data calls for the application of this technique to large data sets.

FastDNAml uses the following basic algorithm to perform incremental searches in the space of all possible trees for n taxa [4,5].

1) Select three taxa from the set to be analyzed. There is only one possible topology for a bifurcating unrooted tree (Figure 2). Calculate the optimal branch length based on the Markov model mentioned earlier.
2) Add in sequence data for another taxon (call this the ith taxon). Add the ith taxon onto the existing tree in every topologically distinct way, creating *2i-5* trees. Optimize the branch lengths for each tree, and keep the tree with the highest likelihood value (Figure 2).

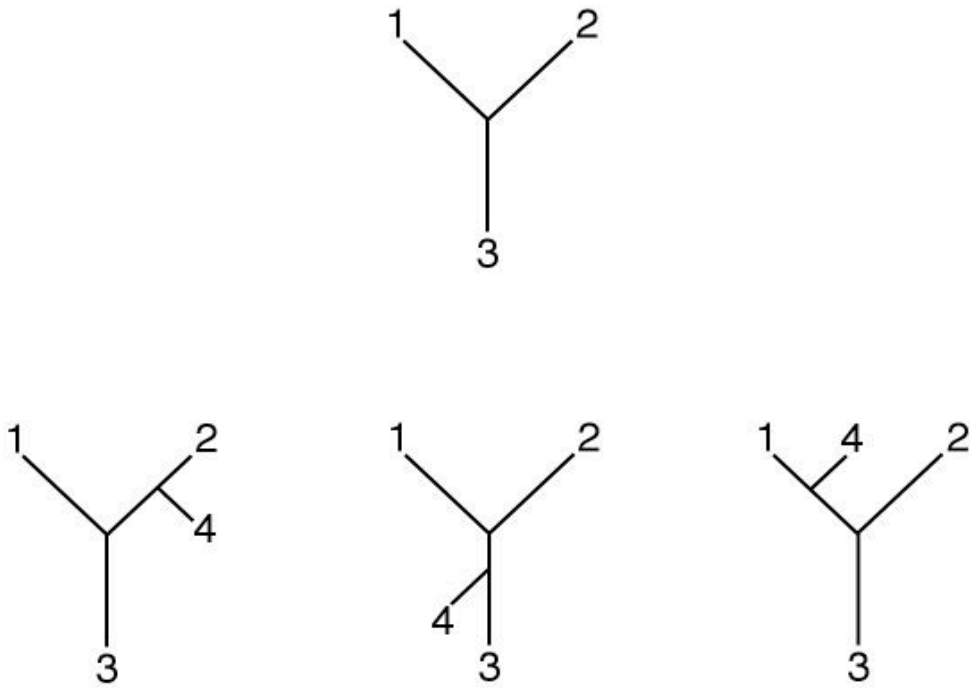

Figure 2. There is only one topology possible for a bifurcating tree containing three taxa. A fourth taxon may be added in one of three different locations.

3) Perform local rearrangements of the tree. That is, swap any subtree to a neighboring branch (Figure 3). If any local rearrangement results in a better tree, then restart the local rearrangement process using this tree. Keep repeating until local rearrangement no longer yields any improvement. If no local rearrangement ever yields an improved tree, there are 2i-6 trees to check.

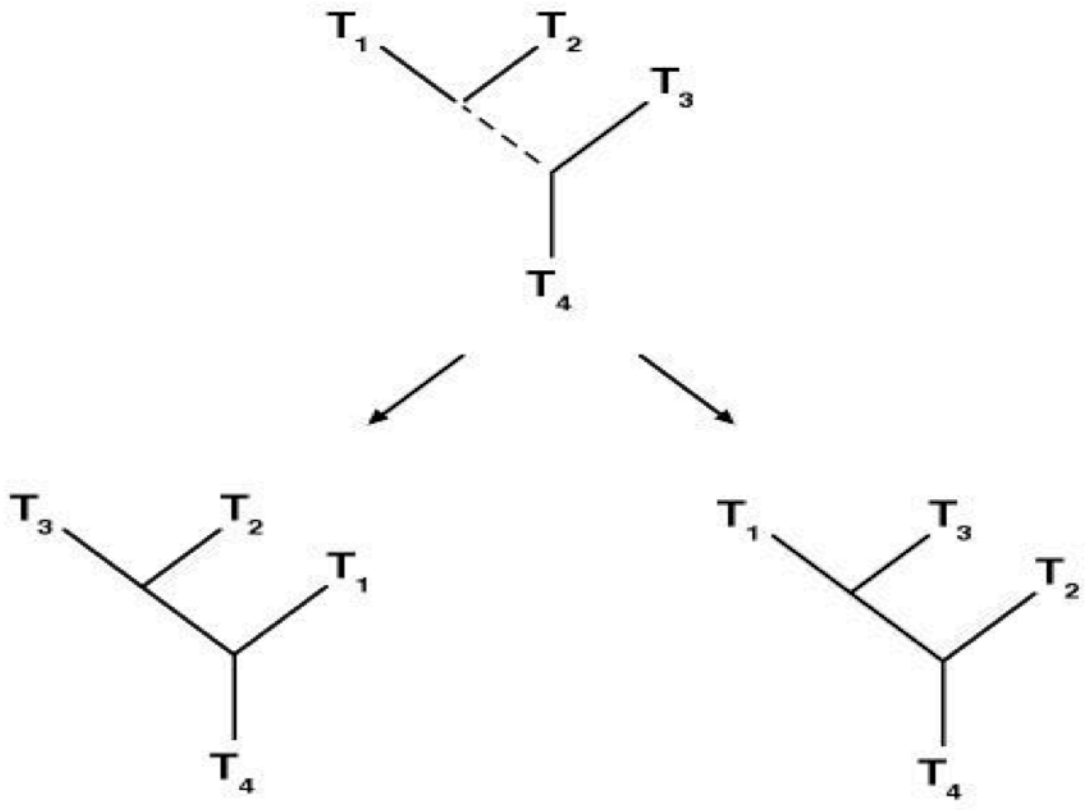

Figure 3. There are two local rearrangements possible when performing local rearrangements of a bifurcating tree with four taxa.

4) Starting with the best tree resulting from Step 3), go back to Step 2). Keep repeating Steps 2) and 3) until all taxa in the data set have been added in.
5) Perform a more thorough rearrangement of subtrees starting with the best tree obtained including all taxa. The number of branch points that can be crossed when moving a subtree is a parameter set by the user of fastDNAml. The final result is the best tree resulting from more extensive rearrangements of the tree.

This process involves an incremental search through the space of all possible trees. As such it is quite possible that in this search the process of starting with three taxa and then incrementally adding the other taxa in a data set being analyzed will place one in a local, rather than global, maximum of the overall likelihood function. Thus the program is run multiple times with each data set, with the order of addition of the taxa (including the selection of the initial three taxa) randomized. By comparing the results of multiple runs one can get some feel for the structure of the tree-space, as well as some information regarding local and global maxima.

fastDNAml is highly computationally intensive. For example, analysis of 50 taxa requires the optimization and evaluation of a minimum of 4,559 trees (this is the minimum under the condition that no local rearrangement ever produces an improved tree). This program is a good candidate for parallelization in the master/worker model because it has a

relatively high computation to communication ratio. The master process constructs trees to be optimized and farms them out to a set of worker processors for analysis. The analysis of each tree involves the calculation of a transition probability using a Markov model for base substitutions at each of hundreds of loci. This process, along with calculating the optimal branch lengths for the tree, is computationally intensive. After initialization of the worker programs, the only communication is the dispatch of a tree to the worker, and the return of a tree and a likelihood value back to the master. The trees are stored in an ASCII string in Newich format [9] that requires roughly 35 bytes per taxon. However, the program does have relatively frequent synchronization barriers. There are in each step of the process (the addition of each taxa) a synchronization barrier implicit in the wait for all slave processes to return optimized trees and associated likelihood values. Figure 4 shows the basic flow of the parallel version of fastDNAml as distributed by Olsen et al. [5,9].

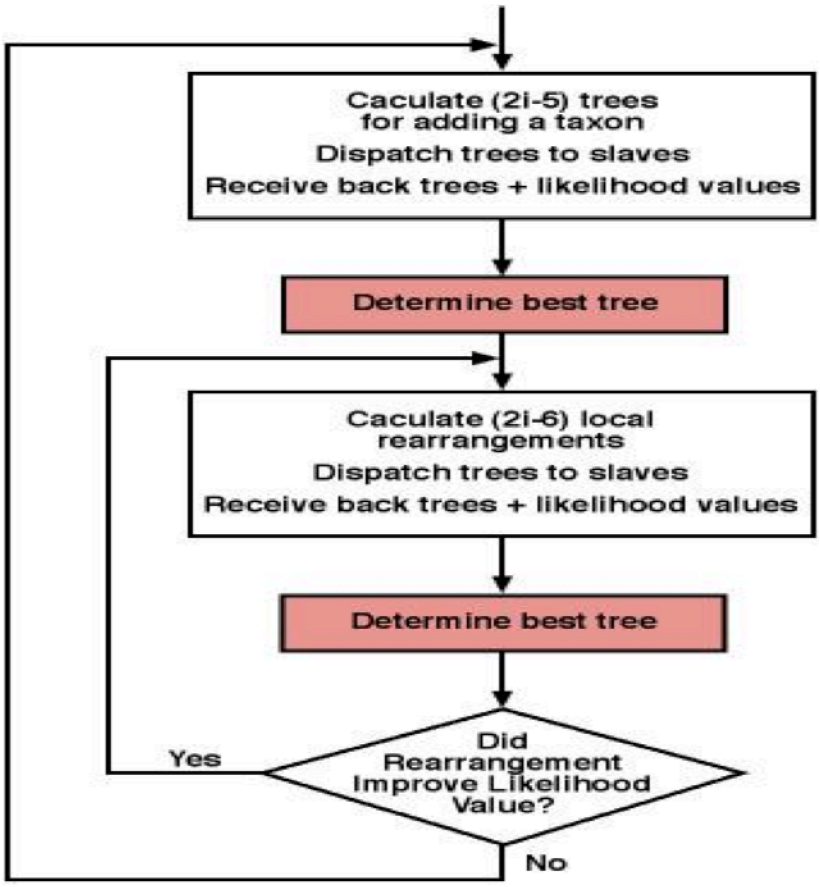

Figure 4. Basic flow diagram of the parallel version of fastDNAml as distributed by Olsen et al.

fastDNAml and grid computing

Indiana University is the lead US institution in the TransPAC network [10], which links the vBNS with the Asia-Pacific Advanced Network. TransPAC network topology is diagrammed in [11]. Indiana University (IU), National University of Singapore (NUS), and Advanced Computational Systems, Australia National University (ACSys) are tied together by a high-speed network connection and the institutional collaborations that made the network possible. Both NUS and IU have significant IBM RS/6000 installations. IU and NUS are home to researchers that make heavy use of fastDNAml [12,13,14]. Organizational and collaborative ties with ACSys made it possible to utilize systems there for a grid project as well. Thus, there were all the raw materials needed for a significant international collaboration in applying grid computing techniques to use of fastDNAml. The iGrid demonstration at SuperComputing98 [15] provided the ideal venue for an experiment using fastDNAml on an international array of high performance computers.

The challenge common to a great many scientists is to properly make the tradeoff between the research questions investigated, the amount of computing resource needed, and a tolerable wall-clock wait. One of the many opportunities in grid computing is to combine geographically distributed resources so as to maximize the size of the problem that can be attacked within a tolerable wait time. The other opportunity in an international collaboration is to take advantage of the time difference among participating institutions. Peak work times in Indiana come roughly during the middle of the night in Singapore and Australia, and vice versa. This makes it possible for researchers working during the day at either IU or NUS to take advantage of processors on the other side of the globe during - what is the middle of the local night and the time of minimum local demand. This helps maximize utilization of resources as well; any cycle that was not put to productive use last night is gone forever! There is, however, a limitation in the types of applications that are suitable for long-distance grids. Light travels halfway around the globe in somewhat more than 50 msec, a much higher latency than the internal communications of a traditional supercomputer.

The parallel version of fastDNAml distributed by Olsen [5,9] is based on the P4 libraries. While easily available, the P4 libraries are certainly no longer the state of the art in parallel computing. Some time ago IU's programmers ported fastDNAml to the PVM (Parallel Virtual Machine [16]) libraries for more efficient execution within a single IBM RS/6000 SP. PVM is, today, probably the simplest mechanism for distributing a parallel program among multiple computing systems. Converting the program to run successfully in a grid environment required several modifications beyond the initial PVM port. The structure of the parallel portion of the code has been simplified somewhat as compared to the P4 version distributed by Olsen *et al*. The program consists of four basic modules: *Host, Master, Foreman,* and *Worker*. *Host, Master, and Foreman* typically run on one system serving as the logical hub of the grid. There are multiple *Worker* processes, and these processes are typically distributed throughout the grid. The functions of these processes are as follows:

Host. The *Host* program is responsible for initialization of PVM on all participating systems and all other initialization matters (including randomization of the order of inclusion of the taxa in the data set).

Master. The *Master* program generates trees for evaluation at each step of the program. That is, it generates the initial tree, which is sent to the *Foreman* for optimization of tree lengths. Foreman returns the optimized tree to the *Master. Master* then creates a new set of trees to be evaluated by adding in the next taxa in every position possible and dispatches them back to *Foreman* for optimization and comparison, and requests that *Foreman* return to it the best of the optimized trees from that step. *Master* then calculates the *2*(number of taxa) – 6* new trees resulting from nearest-neighbor swapping, dispatches them to *Foreman* for analysis, and requests the best of the optimized trees back from *Foreman*. This process continues until all taxa have been added in, at which point the *Master* calculates the trees for the full tree check (as with the original fastDNAml, the number of branch points that can be crossed in making rearrangements is a parameter specified by the user).

Foreman. The *Foreman* program has two primary functions. It tracks the *Worker* processes, dispatching trees to be evaluated to them and monitoring their responses. It is in this process that the greatest changes were required in order to run fastDNAml as a grid program, and these changes will be described in much greater detail below. The *Foreman* process also compares the likelihood values of the optimized trees returned to it by the *Worker* processes, and returns to the *Master* the best tree resulting from each round of optimizations. (The *Foreman* process combines the functions of the Dispatch and Merge modules of Olsen et al.'s P4 version)

Worker. The worker process receives a tree from the *Foreman,* optimizes the lengths of the branches in the tree so as to maximize the likelihood score of that tree (without changing its topology) and returns the new tree, including branch lengths, back to the *Foreman.* This part of the program is essentially unchanged from the version distributed by Olsen *et al.* [4,9].

The above architecture for fastDNAml represents a simplification of the parallel structure of the Olsen *et. al* [5,9] P4 version. However, the greatest changes in the grid-ready version of the program lie in the *Foreman* process. When fastDNAml is run within a single parallel computer it is reasonable to assume that either every processor will operate properly without interruption, or the entire system will experience some sort of problem and come to a halt. Even when running fastDNAml, say, on several systems within one research lab it is quite reasonable to take this all-or-nothing approach to job completion. Thus the Olsen *et al.* distribution – quite reasonably - does not have any mechanism for recovering from the failure of an individual *Worker* process and continuing on to successfully complete a particular run.

Running fastDNAml successfully in a wide-area grid, however, requires the ability to recover from the failure of an individual *Worker* process and go on to successful completion of a particular program run. The reason this is essential is that the program is

being executed across multiple systems communicating via many network links. There is a reasonable chance that during the course of any individual run at least one of the following events will occur: one of the systems participating in the execution of a run will experience some sort of failure; a network link will fail; or a network link will become so slow that the overall program will complete more quickly by dropping the processors connected to the *Foreman* via the slowed link. The common feature of all of these problems is that they are evidenced by a lack of communication by the *Worker* program back to the *Foreman*, as opposed to some explicit error message. We have modified the *Foreman* process to properly deal with the vagaries of a grid environment. To do this, the *Foreman* maintains three queues: a Ready Worker Queue, a Work Pending Queue, and a Blacklist. These queues are diagrammed in Figure 5, and explained below.

Ready worker queue

Work Pending Queue	
Machine #	**Start Time**

Blacklist

Figure 5. The "Ready Worker" and "Blacklist" queues are simply lists of machines (nodes) ready to do work (ready) or taken out of the work queue (Blacklist). The work pending queue shows work underway – the machine number a task was assigned to, and the time the work was started.

The Ready Worker Queue is simply a list of all Worker processes (systems or individual processors within a parallel system) that are waiting to receive new work. Trees are dispatched to the Worker processes in a first in/first out manner. When a Worker process completes work and returns an optimized tree to the Foreman process its ID is re-entered in the Ready Worker Queue. This provides a certain measure of automatic load balancing, as the fastest processors participating in the grid return results most quickly, appear in the ready queue most frequently, and thus get a larger share of the calculations than slower processors.

The Work Queue holds the largest amount of information. It includes one entry for each Worker process currently at work optimizing a tree dispatched to it by the Foreman process. Each entry includes the Worker ID, the start time, and the actual tree dispatched to that worker. The Foreman process periodically checks through the Work Queue looking for jobs that have taken longer than a user-specified timeout period. Whenever a job has gone beyond the timeout limit without being completed by the Worker process, two things happen. The tree that this Worker was optimizing is placed back in the work queue to be dispatched to another Worker, and the Worker ID is placed on the Blacklist so that no additional jobs are sent to it. (The program does have a facility for removing

worker processors from the blacklist and adding them back to the Ready Queue should they give evidence that they are working again).

The modifications to fastDNAml make it possible to run fastDNAml in a grid context effectively. IU developed one additional piece of code to cooperate with fastDNAml. This is a 3-dimensional visualization of the intermediate steps as fastDNAml adds taxa and marches its way through the treespace. Figure 6 shows a snapshot from this visualization. One can, of course, navigate though the images displayed to hone in on any particular feature of interest. This visualization was initially developed for demonstration purposes, so that visitors to the iGrid booth at SC98 would have something to look at as we discussed the evolutionary grid project. The visualization served this purpose well and has additionally turned out to be a useful real-time diagnostic tool. FastDNAml occasionally starts with an early combination of taxa that creates pathological oscillating behavior in the tree as it is developing. That is, the incremental search through the space of all possible trees starts in a place from which it is impossible to get to any reasonable tree, and the structure of the tree simply oscillates back and forth among a variety of unreasonable trees. It has proven to be quite easy to detect this sort of behavior in real time using the 3D visualization, permitting such runs to be terminated and minimizing the time wasted.

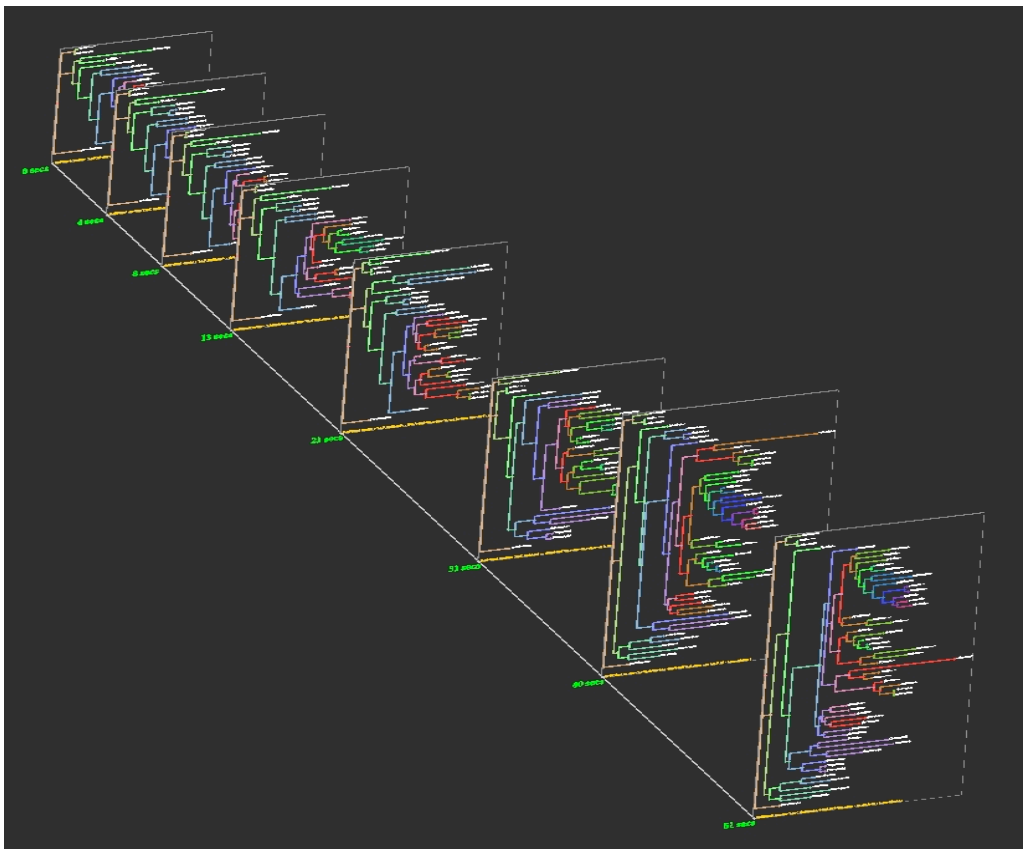

Figure 6. Three-dimensional depiction of the time course of tree development, as taxa are added and trees optimized.

Performance of fastDNAml in an international grid setting

For the iGrid project at SC98 we demonstrated use of fastDNAml distributed across processors located at IU, NUS, and ACSys. IU and NUS each contributed time on 16 processors in an IBM RS/6000 SP (P2SC Thin Nodes, uniprocessor, 165 MHz). ACSys contributed 8 DEC workstations (Digital Alpha 600 workstations, uniprocessor, 266MHz). As part of the demonstration we explored the performance characteristics of fastDNAml. Figure 7 shows performance curves for fastDNAml running within one system located at IU; distributed across two sites – IU and one of the two partners. For these benchmarking runs we used precisely the same order of entry of taxa, but were able to run just one job per combination. The total number of taxa used in these runs was 56**.**

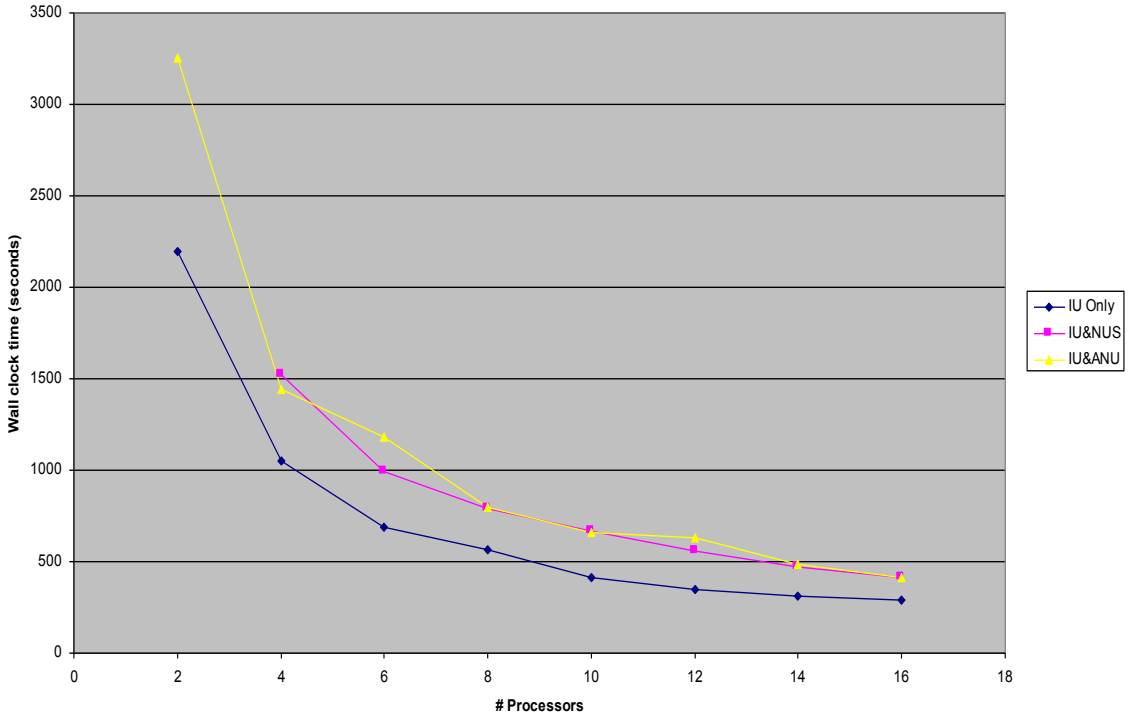

Figure 7. Elapsed wall clock time to complete benchmark runs for fastDNAml.

A few things are quite clear from Figure 7. First, even with a fairly small test case, the time to complete one run of fastDNAml is fairly significant – over half an hour. A parallel job using 16 processors within one SP ran in under 5 minutes – but 16 processors is a fairly significant portion of all but the upper end of the list of top 500 supercomputer sites [17]. Running a parallel job on 16 processors when 8 were at IU and 8 were located at one of the Pacific Rim partner sites took an elapsed time of under 7 minutes. The jobs using all three sites simultaneously took, as one would expect, slightly longer per processor than the jobs split across two locations. These times are for only one random ordering of the taxa to be included. A realistic set of runs for a data set this size would involve 100 random orderings – making any speedup significant in terms of the processing of an entire set of orderings. There is another way to look at the speedup data. If it is possible to increase the total processor count by as few as two processors by distributing your job across multiple computers, the total wall clock time is reduced. In

practical terms, this is the key result in demonstrating the value of grid computing as applied to fastDNAml: picking up a very small number of processors – a seemingly straightforward thing to do – by going to a grid approach reduces the time to solution. (Or, more realistically, increases the size of the problem that can be attacked within the time constraints imposed on the scientist.)

We have managed to accomplish some interesting biology. As part of the iGrid demonstration, researchers investigated the evolutionary relationships among the subcomponents of cytoplasmic coat protein (COP) complexes. Eukaryotic intracellular membrane transport is mediated by vesicles whose formation involves specific COP complexes. Of the seven COP (alpha, beta, beta', gamma, delta, epsilon, zeta) subunits recognized so far, several yeast, fungal, rice, drosophila, mouse, rat, hamster, bovine and/or human have been cloned and sequenced. The tree in Figure 8 shows the phylogenetic relationships of COP subunit genes as determined by fastDNAml [18].

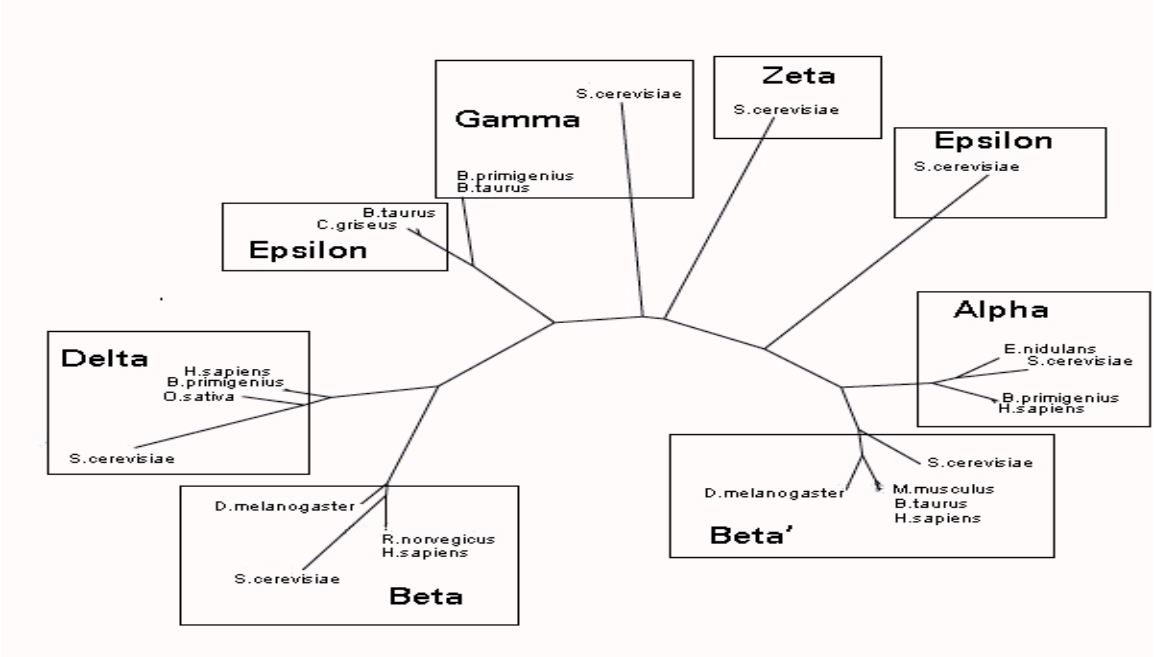

Figure 8. Phylogenetic tree of cytoplasmic coat proteins calculated during demonstration at SC98.

Some thoughts on grid computing and future plans with fastDNAml

There are already two established models for computational grids. One is a grid comprising geographically distributed facilities within one organization, such as NASA [19]. Another model is a grid comprising some sort of formal alliance or consortium. NCSA and NPACI are examples of this sort of grid [20,21]. This project establishes another type of grid: a grid based on pooling and sharing resources within communities

based on commonalities of discipline and use of community codes. This project is different than other community code projects in that the sharing of cycles, designed in particular to utilize processors during times of low local demand, is an explicit part of the collaboration.

Our plans for the future are to continue with the 'evolutionary biology grid' as a periodically available production service. We are also making further enhancements in the grid-enabled version of fastDNAml, including an MPI [22] port that is now running in production mode. Further enhancements in the structure of the code will make it possible to use a single code base and specify the parallel communication library within a set of user-specified parameters. This will significantly improve the longevity of the code in that it will become less vulnerable to changes in the state of the art in parallel libraries. We plan to have our port of the fastDNAml program available for download no later than 9/1/2000. Access to the code will be available from [23]. We also hope to pop up a level of parallelism. As mentioned earlier, there are several implicit barriers created by the need to find the best tree at each step in the search process. Thus when as few as four processors are in use, it is not until the fifth taxa has been added to the search process that it becomes possible to use all of the available processors simultaneously. We hope to develop a version of the fastDNAml code that runs multiple randomizations of the taxon entry order simultaneously. This should make the program easier to use, achieve better scaling, and provide better searches of the peaks and valleys of the likelihood values within the space of all possible trees.

This project arose out of a desire to do more and better biology. The key enabling factors were the commonalities in interests and use of community codes by all of the partners, and the existing high speed intercontinental network links. The fact that IU and NUS both have significant IBM RS/6000 SP installations was a bit of good luck. IU's previous work porting fastDNAml to run under PVM on the IBM RS/6000 SP also 'preadapted' the code to run well in a grid environment. The willingness of system administrators to help us overcome security issues was important. Perhaps more important was the willingness on all sides to suspend accounting rules and put the doing of science first. This enabled us to produce real scientific results and demonstrate the value of a grid-based approach to evolutionary biology prior to attacking the complex problems of fair use and accounting (which remain in "yet to be resolved" status).

In summary, we have demonstrated the practical value of a grid-based approach to the most computationally intensive of the mathematical methods for construction of evolutionary phylogenies, and in the process produced some notable results in evolutionary biology. We have demonstrated what we believe is a largely new type of computational grid – based on sharing of resources born of commonalities within communities of interest – and that this type of grid can feasibly work to the advantage of all participants. And we believe that there is significant work yet to be done in improving the existing parallel versions of fastDNAml. This work will yield improvements in scientists' ability to advance our understanding of evolutionary processes, which has significant practical value in addition to its value in enhancing our understanding of the world around us.

Acknowledgements

This work was greatly facilitated by IBM Shared University Research grants to Indiana University in 1998 and 1999. All graphics other than Figures 1, 6, 7, and 8 were created by W. Leslie Teach, Indiana University. System administrators at NUS and IU deserve special thanks for their help with this project, including Mary Papakhian and Dan Lauer of IU's Research and Technical Services group; Tan Chee Chiang of the NUS Computer Centre's Supercomputing & Visualisation Unit; and Markus Buchhorn, Tim Littlejohn and Tim Potter of ACSys CRC. Penny Studley, IU, did HTML layout of this paper.

References

[1] Smarr, L. 1999. Grids in context. pp1-14 *In:* The Grid: Blueprint for a new computing infrastructure. I. Foster and C. Kesselman, eds. Morgan Kaufmann Publishers Inc., San Francisco.

[2] Genbank: http://www.ncbi.nlm.nih.gov/

[3] GDB(TM) Human Genome Database [database online]. Toronto (Ontario, Canada): The Hospital for Sick Children, Baltimore (Maryland, USA): Johns Hopkins University, 1990-. http://www.gdb.org/

[4] Felsenstein, J. 1981. Phylogenies from DNA sequences: a maximum likelihood approach. J. Mol. Evol. 17:368-376.

[5] Olsen, G. J., Matsuda, H., Hagstrom, R., and Overbeek, R. 1994. fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. Comput. Appl. Biosci. 10: 41-48.

[6] Felsenstein, PHYLIP: http://evolution.genetics.washington.edu/phylip.html.

[7] NERSC Center for Bioinformatics and Computational Genomics, Computational Structural Biology Scientific Simulation Initiative: http://cbcg.lbl.gov/ssi-csb

[8] Hershkovitz, M.A. and D.D. Leipe. 1998. Phylogenetic analysis. pp. 189-230 In A.D. Baxevanis and B.F.F. Ouellette. Bioinformatics: A practical guide to the analysis of genes and proteins. John Wiley & Sons, NY.

[9] Olsen, G. Home page: http://geta.life.uiuc.edu/~gary/

[10] McRobbie, M. K. Adams, D.F. McMullen, D. Pearson, J. Williams. 1998. A high performance network connection for research and education between the vBNS and the Asia-Pacific Advanced Network (APAN) Proceedings of INET 98 annual conference.

[11] STAR TAP: http://www.startap.net

[12] Köhler, S. C.F. Delwich, P.J. Denny, L.G. Tilney, P. W. Denny, L.G. Tilney, P. Webster, R.J.M. Wilson, J.D. Palmer, and D.S. Roos. 1997. A plastid of probable green algal origin in Apicomplexan parasites. Science 275: 1485-1489.

[13] Palmer, J. Home page: http://www.bio.indiana.edu/

[14] Bioinformatics Centre, National University of Sinagapore: http://bic.nus.edu.sg/

[15] M.Brown, T. DeFanti, M. McRobbie, A. Verlo, D. Plepys, D.F. McMullen, K. Adams, J. Leigh, A. Johnson, I. Foster, C. Kesselman, A. Schmidt, S. Goldstein. The International Grid (iGrid): Empowering Global Research Community Networking Using High Performance International Internet Services, INET '99, San Jose, June 22-25, 1998, San Jose, California. http://www.startap.net/PUBLICATIONS/pubs.html#Application Papers

[16] PVM (Parallel Virtual Machine): http://www.epm.ornl.gov/pvm/pvm_home.html

[17] Top 500 Supercomputer Sites: www.top500.org

[18] FastDNAML Phylogenetic Analysis of COP proteins using Supercomputing Grid between National University of Singapore and Indiana University: http://bic.nus.edu.sg/sc98.html

[19] NASA Information Power Grid: www.nas.nasa.gov/IPG

[20] NCSA. http://www.ncsa.uiuc.edu/

[21] NPACI. http://www.npaci.edu/

[22] MPI (Message Passing Interface) Forum: http://www.mpi-forum.org/

[23] High performance computing at Indiana University: http://www.indiana.edu/~rac/hpc/index.shtml