Noname manuscript No.
(will be inserted by the editor)

Building large phylogenetic trees on coarse-grained parallel machines

Keane, T.M.[1], **Page, A.J.**[2], **Naughton, T.J.**[2], **Travers, S.A.A.**[1], **McInerney, J.O.**[1]

[1] Department of Biology, National University of Ireland, Maynooth, Co.Kildare, Ireland.
[2] Department of Computer Science, National University of Ireland, Maynooth, Co.Kildare, Ireland.

The date of receipt and acceptance will be inserted by the editor

Abstract Phylogenetic analysis is an area of computational biology concerned with the reconstruction of evolutionary relationships between organisms, genes, and gene families. Maximum likelihood evaluation has proven to be one of the most reliable methods for constructing phylogenetic trees. The huge computational requirements associated with maximum likelihood analysis means that it is not feasible to produce large phylogenetic trees using a single processor. We have completed a fully cross platform coarse grained distributed application, DPRml, which overcomes many of the limitations imposed by the current set of parallel phylogenetic programs. We have completed a set of efficiency tests that show how to maximise efficiency while using the program to build large phylogenetic trees. The software is publicly available under the terms of the GNU general public licence from the system webpage at http://www.cs.nuim.ie/distributed

1 Introduction

Phylogenetic analysis is a branch of molecular biology that is concerned with the reconstruction of evolutionary relationships between organisms, genes, and gene families. The knowledge gained from the construction of accurate phylogenies can be crucial in understanding such things as the origins of biochemical pathways, regulatory mechanisms in cells as well as the development of complex systems. Given a set of taxa (molecular sequences usually comprising of genes or gene products) and a DNA substitution model, the task is to find the phylogenetic tree that most accurately describes the evolutionary relationships between the taxa. However the decision problem associated with searching for the best tree from a given set of taxa is NP-complete [1]. Many authors have proposed greedy

Correspondence to: tom.naughton@nuim.ie

heuristic solutions in an attempt to reduce the effective search space [2–7]. These algorithms have made the process of producing large phylogenetic trees possible using only a single processor. However these greedy heuristic algorithms often only take the best immediate, or local, solution resulting in a final tree that is far from optimal. The stepwise insertion approach combined with maximum likelihood (ML) evaluation has proved to be especially accurate for building molecular phylogenies. Felsenstein was the first to bring this framework to nucleotide-based phylogenetic inference [8]. Numerous computer studies [9–13] have shown ML programs can recover the correct tree from simulated data sets more frequently than other methods. In a recent study timing the evolution of the HIV-1 virus [14], it was demonstrated that ML techniques can be effective in solving important biological problems. The single factor that is currently limiting the widespread use of ML techniques is the computational requirements [15]. The huge computational requirements associated with ML analysis means that it is not feasible to produce phylogenetic trees for any more than a small number of taxa using a single processor.

A number of specialised parallel ML phylogenetic programs have been developed to address this issue [16–18]. These programs have been very successful in speeding up the process of constructing large phylogenetic trees using ML. However the overriding limitation associated with each of these programs is the specialised hardware and software is often required to run these programs. For most researchers, this can make these programs either prohibitively expensive or simply too complicated to set-up. Furthermore these programs are often implemented in a platform dependent language which imposes a restrictive limit on the numbers and types of machines that can be used in a parallel computation. It should also be noted that some of these earlier parallel programs only allowed the user to choose from a very limited number of DNA substitution models, which often leads to a poor model fit resulting in sub-optimal trees. Therefore, in our opinion, the three most essential requirements of any generally usable parallel tree building program must be: that the program should not require any sort of specialised or expensive parallel hardware, should only require the most basic technical abilities to set-up and use, and should allow the user to choose from an extensive list of molecular evolution models. Currently there is no parallel phylogenetic tree building program that fulfills all of these requirements.

We have developed a fully cross platform distributed application, called Distributed Phylogeny Reconstruction by Maximum Likelihood (DPRml), which we believe to be one of the most general, while powerful, likelihood-based phylogenetic tree building programs currently available. It satisfies each of the three requirements outlined above. The generality of our program is demonstrated by the fact that DPRml, written in Java, can run on virtually any architecture and operating system simultaneously while only using the spare clock cycles of donor machines. No specialised computer hardware is required, and no expense is incurred if idle computing resources are harnessed. This would not be as straightforward for a distributed application written in a native language because the application would have to be compiled for each particular architecture and operating system. We have demonstrated the ease of use and platform heterogeneity of DPRml with

experiments that utilise the spare computing resources of several different architectures and operating systems simultaneously. The user has a very straightforward configuration file with which to tailor the computation and can choose from one of the most extensive ranges of DNA substitution models currently available. Our efficiency analysis shows how a user how to make the most efficiency use of their spare clock cycles to build large phylogenetic trees. DPRml implements an already proven tree building algorithm [16, 19] and uses the popular Phylogenetic Analysis Library (PAL) v1.4 [20] for all its likelihood calculations. A more detailed description of the bioinformatics element of DPRml is available in [21].

The rest of the paper is organized as follows. In Section 2 we introduce the DPRml algorithm in detail. Section 3 describes the Java distributed computing platform used by DPRml. The results of our experimental evaluation are presented in Section 4 and we conclude in Section 5.

2 Coarse Grained Algorithm

One of the first programs to use ML techniques to build phylogenetic trees was DNAml [8]. This was improved and extended in the popular phylogenetic program fastDNAml [19]. A parallel version of this program was recently completed [16]. We have taken the tree building algorithm used by parallel fastDNAml and have implemented a platform independent, distributed, and much more generalised version of the program. Our distributed algorithm is client-server based (see Section 3 for more details of the distributed system design) and operates in a number of stages just like parallel fastDNAml. At every stage a number of new trees are generated and sent to donor machines to have their branch lengths optimised (optional) and likelihood calculated. The DPRml distributed algorithm is described in Figure 1.

The parameters m and v are contained in the parameter file. Step 1 of the algorithm is a localised version of the overall algorithm outlined in steps 2-7. A single donor machine builds an initial tree for m minutes (default value is 30 minutes) and returns this tree to the server. The size of the tree built by this initial step depends on several factors such as the complexity of the substitution model, diversity of the sequences, the number of positions per sequence, and the CPU speed of the donor machine. Due to the network overhead, it is more efficient to build this initial tree on a single donor machine than to distribute this part of the computation. The inputs to the application are a MODELTEST [22] output file, a FASTA sequence file (DNA or RNA), and an input parameter file. MODELTEST is a popular application among molecular biologists that uses hierarchical hypothesis testing to find the DNA substitution model that most accurately fits a given dataset.

The outputs of the program are a New Hampshire format tree file, a PAL tree object file, a human readable tree (text file), and the likelihood of the final tree. We have provided a remote interface to the system that makes it possible to monitor the progress of the application in real-time as it builds the phylogenetic tree. The input parameter file lets the user set various run-time options for the computation such as the maximum number of vertices that rearrangements can span, whether to

```
procedure DPRml( MODELTEST file, FASTA DNA or RNA file,
parameter file )
1. Construct an initial tree for m minutes on a single donor
   machine
2. Take the current best tree and add another taxon to every
   topologically distinct place of the current best tree
   (generating 2i − 5 trees, for i^th taxon being added to the
   tree - see Section 2.1)
3. Send each of the generated trees to donor machines
   to have their branch lengths optimised (optional) and
   likelihoods calculated
4. Take the best tree from step 3 and perform local
   rearrangements crossing up to v vertices (see
   Section 2.2)
5. Send each of the generated trees to donor machines
   to have their branch lengths optimised (optional) and
   likelihoods calculated
6. If the best tree from step 5 has a greater likelihood
   than the best tree from step 3, then rearrange again
   and send each of the generated trees to donor machines
   to have their branch lengths optimised (optional) and
   likelihoods calculated. Repeat this step until there is
   no improvement in the likelihood of the best tree
7. If there are more taxa left to add to the tree then goto
   step 2, else goto step 8
8. If the user chose not to optimise branch lengths
   throughout the computation then send the best tree to a
   donor machine to have its branch lengths optimised, else
   goto step 9
9. Output the best tree
end procedure
```

Fig. 1 A pseudocode description of the DPRml algorithm

keep a copy of the best tree from every stage or just the best tree from the previous stage, whether or not to add the taxa in a randomly generated order, and whether to optimise the branch lengths of every tree that is generated or just optimise the final tree. The input file also gives the user the option to continue building a partially complete tree. Log files included with each set of results make it possible to fully examine and track the entire tree building process.

2.1 Substitution Models

Probabilistic models of nucleotide substitution are central to the accurate reconstruction of complex evolutionary relationships [23]. These models are used in phylogenetic analysis to describe changes in character state, i.e., the rate of change from one nucleotide to another. The evolutionary process is modeled as an evolutionary Markov process [24], also known as a substitution model. Markov models

assume that there is no "memory" in the system, therefore only the instantaneous state of a character is important. The probability of change from state i to state j depends upon the amount of time that has passed and the substitution rate. Well determined time points are not usually available for molecular data so the product of rate and time (equivalent to a genetic distance) is more commonly used. Critical to models of nucleotide evolution is the realization that because there are only four possible character states, it is expected that as genetic distance increases, some sites will undergo multiple superimposed substitutions. Simple measures of distance that do not take multiple substitutions into account are said to be *uncorrected*. *Corrected* distances use one of several models of sequence evolution to estimate the number of sites that have undergone multiple substitutions.

A few of the most popular DNA substitution models currently in use are (listed in ascending complexity) JC69 [25], Kimura-2-Parameter model [26], HKY85 [27], TN93 [28], and the General Time Reversible model [29]. All of the models mentioned assume that each position is evolving independently and identically. However this is rarely the case, therefore each of the models above can incorporate a site to site rate variation also. Two of the most popular rate variation models are the Invariable sites model [30] and the Gamma distribution model [31]. One of the common limitations of the current set of parallel phylogenetic tree building programs is that they only support a very limited number of substitution models and very few support site to site rate variation. As mentioned above, MODELTEST tests one of the most extensive libraries of DNA substitution models (including site to site rate variation) against a dataset in order to find the most suitable substitution model. DPRml supports all of the substitution models that MODELTEST v3.06 provides.

2.2 Add Taxon Stage

The tree building algorithm described above is based on a stepwise insertion approach of adding new taxa to the phylogenetic tree. Initially the algorithm starts by constructing an unrooted bifurcating tree with three taxa (only one topology is possible - see Figure 2). A new taxon is then added to every topologically distinct place of this initial tree. In the general case, this process produces $2i - 5$ trees, for the i^{th} taxon being added to the tree. All of the trees produced are then evaluated using maximum likelihood. The maximum likelihood method computes the likelihood of obtaining the observed sequence with a given tree topology, assigned branch lengths, and a given evolutionary model. Since the likelihood is typically a very small value, the tree with the greatest log likelihood is kept.

2.3 Rearrange Tree Stage

An important part of the algorithm is the internal rearrangements that are made to the best (most likely) tree during each iteration of the algorithm. By repeatedly rearranging the best tree until there is no improvement in the likelihood, the thoroughness of the search of the overall tree space is greatly increased. This is

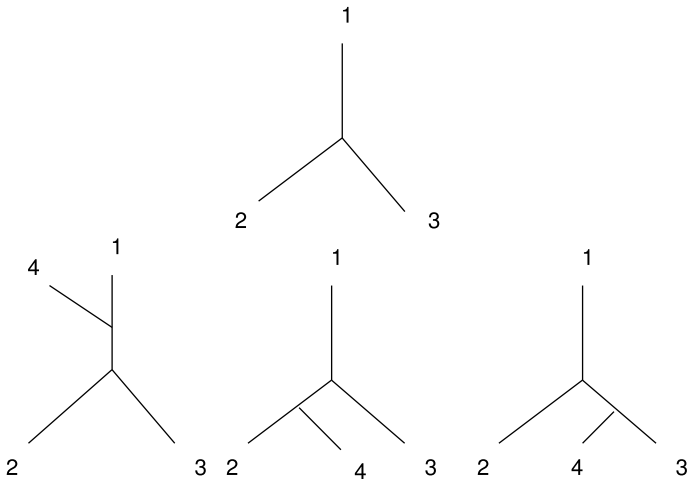

Fig. 2 Adding a fourth taxon to a 3 taxa tree produces 3 new trees.

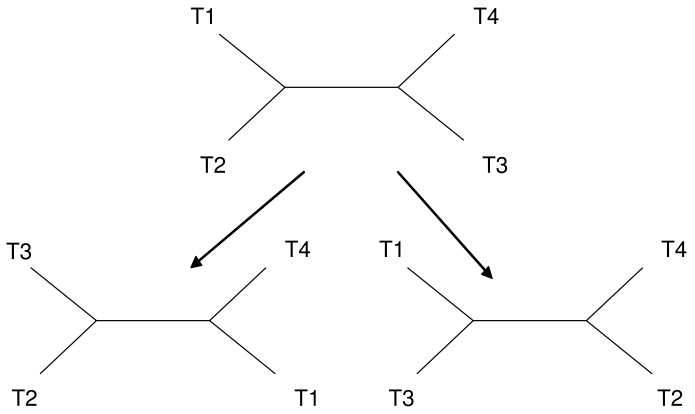

Fig. 3 Rearrangement spanning one internal branch.

done by moving each subtree across one or more internal branches of the tree. The user specifies the maximum number of internal branches that rearrangements can span. Figure 3 is an example of a rearrangement spanning one internal branch of the tree. A higher number of internal branches being spanned by rearrangements will result in more trees being generated and will increase the overall running time of the program. However a lower number will result in a lesser proportion of the tree space being searched and will increase the chance of a sub-optimal tree being produced.

2.4 Algorithm Suitability for Distributed Computing

Our initial work on this application involved performing an investigation on how suitable ML based phylogenetic tree building is to distributed computing. We decided on two main criteria that any problem had to fit in order to be suitable for a distributed implementation. Firstly the problem had to fit the class of "coarse grained" parallel problems. As can be seen from the algorithm above, each stage of the computation produces a set of trees that must have their branch lengths and likelihood calculated. These tasks can be done completely independently on different sites for every tree without interaction required between any two likelihood computations.

The second criterion for evaluating the suitability of phylogenetic analysis to distributed computing was that the problem must display a high "compute-to-data" ratio to make it worthwhile sending the data over a network rather than computing locally. In our efficiency analysis (see Section 4) the dataset that we used consisted of 101 taxa and was less than 200 KB in size. The small size of the data files involved coupled with the long computation times of ML analysis make this problem ideal for distributed computing.

3 Implementation

DPRml is just one of a number of applications [21, 32] that runs on our general purpose distributed computing platform. Our distributed computing platform is loosely based on the design of the Java Distributed Computing Library (JDCL) [33, 34] but offers much greater functionality, flexibility, and usability. The overall design of the system is based on the client-server model [35]. This model describes a system consisting of a single server computer and a number of client computers. The clients can connect over a network to the server. The server controls a resource (such as a database, algorithm, or computer hardware) and the clients initiate requests to the server for access to the resource. Our system is divided into three separate pieces of software: server, client, and remote interface. An overview of the system is illustrated in Figure 4. The server stores the problem (for example, molecular data and an algorithm to process it) and breaks the problem down into smaller problems, called data units. The client software is installed on each donor machine and it connects to the server over the Internet. A client requests a data unit, performs the processing, returns the result to the server, and requests another data unit. Multiple clients can make such requests to the server. The server collates the results of the data units from the clients and constructs the result to the larger, original, problem. The remote interface is used to access all functionality on the server such as adding and removing problems, downloading result files, monitoring progress of computations, changing the priority of problems in the system, and viewing server statistics. Here are a few of the novel features of our system:

- Portability: The entire system is completely network and platform independent
- Scalability: New clients can be added and removed from the system dynamically

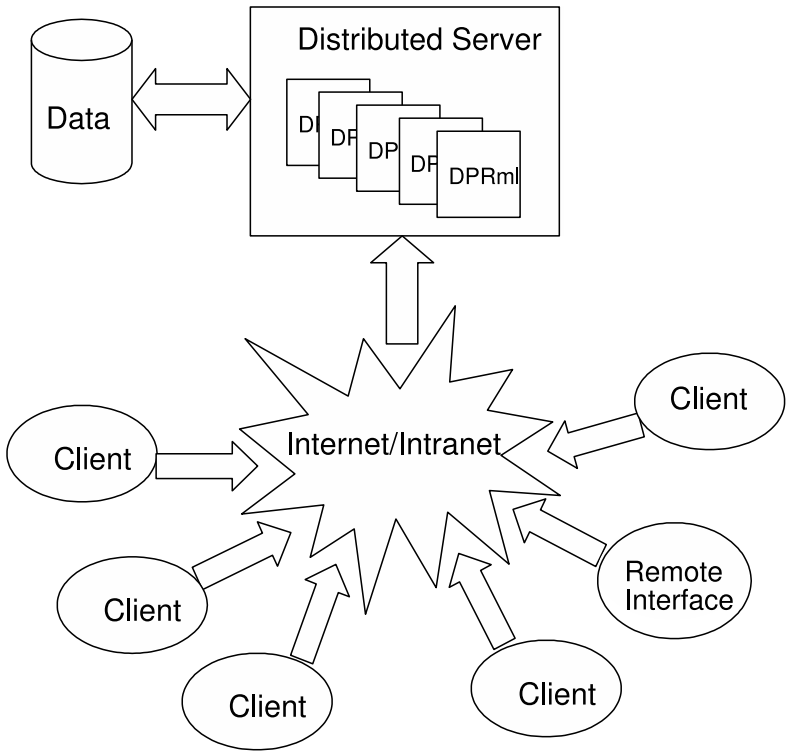

Fig. 4 Diagram of complete system. Although all communication is bi-directional, the arrows indicate the direction of initiation of communication.

- Expandability: The server has the ability to run several different distributed computations simultaneously
- Longevity: Remote updating of client software is supported
- Real Time Updates: Remote real-time problem progress updates are available to users
- Secure: The security of the server and donor machines from subversive distributed applications is guaranteed
- Heterogeneous: An adaptive scheduling algorithm dynamically matches the donor machines with work units that match their computational capacity
- Remote Control: Complete control of all server functionality is possible remotely over the Internet
- Dynamic Job Priorities: Priority of jobs can be changed dynamically to allow problems be allocated greater or lesser fractions of the overall available processing power
- Ease of set-up: The entire system comprises of only three executable Java JAR files
- Modular design: New scheduling algorithms can be implemented without any changes required to the rest of the system or existing distributed applications

The server, client, and remote interface consist of single executable Java JAR files that can be run from the command line. There are a few different ways that the client software can be deployed. To maximise the usage of our semi-idle desktop PC's, we choose to run the client as a low priority background service. This means that even if there is nobody logged on at a donor machine, the client software can run in the background 24 hours a day using the spare clock cycles. In our deployment of the system, we have our client software running on 180 desktop PC's (various hardware specifications from Pentium II's up to Pentium IV's) running multiple operating systems (Windows 98/NT/2000/XP, Linux - Gentoo, Debian, Fedora). To illustrate the portability of our system, we have also installed our client on every node of an IBM Linux cluster (32 Dual P4 1 GHz nodes with 512 MB memory per node) with the desktops and cluster nodes connecting to a single server. We consistently perform approximately 3 Pentium years of processing each week.

3.1 Distributed Programming Model

As part of our general purpose distributed platform, we have designed a general purpose programming interface that allows a user to distribute arbitrary computations among a set of donor machines. To set-up a distributed computation, two Java classes must be extended. These are the `Algorithm` class and the `DataManager` class. Each of these parent classes are part of the system. The developer must overwrite and implement certain methods in order to set-up a distributed computation [36].

To implement the phylogenetic tree building algorithm, we partitioned the algorithm outlined in Section 2 into two sets of functionality, code that runs on the server and code that runs on the clients. The `Algorithm` class, which runs on the donor machines, contains four separate methods. The first method builds an initial tree for m minutes using a localised version of the overall tree building algorithm (default value for m is 30 minutes). The second method takes a set of trees and a substitution model, optimises the branch lengths of each tree (optional), computes the likelihood of each phylogenetic tree, and returns the tree with the greatest likelihood to the server. The third method takes the final tree and optimises the branch lengths of this tree. This method is only executed when the user chooses not to optimise the branch lengths of every tree throughout the computation. The last method takes a tree and performs internal rearrangements on the tree crossing up to a specified number of branches, thus producing a number of new trees. Parameters are sent with each data unit to identify which method to execute on the downloaded data.

The `DataManager` runs on the server and manages the overall computation. The `Constructor` method reads in, parses, checks the validity of the inputs to the program, and sends the data to one donor machine to create the initial tree. We divided the tree building algorithm up into different stages. The `generateDataUnit()` method simply makes a call to the current stage object and requests a data unit for a donor machine to process. If there are no more trees to

be evaluated, the stage object returns `null`. The `processResults()` method is called every time a results set is received. The results are sent to the current stage object and when the stage object returns true, the `DataManager` takes the best tree from the current stage and transitions to the next stage of the tree building algorithm. The `adjustUnitSize()` method is used by the distributed system to dynamically alter the granularity of the data units in an attempt to keep the average processing time within v percent of the optimal unit time t (default values for v and t is 15 percent and 1 hour, respectively). The `getStatus()` method sends useful information about the current state of the computation to the remote interface, allowing the user to get real-time feedback on the progress of their computation. The `closeResources()` method, called when the problem is being removed from the distributed system, ensures that there are no tree generation threads still running.

4 Experimental Evaluation

We performed a full set of performance tests using the dataset that was used to benchmark parallel fastDNAml [16].This dataset consists of three individual sets of taxa consisting of 50 taxa, 101 taxa, and 150 taxa that are 1858 (50 and 101 taxa) and 1269 (150 taxa) nucleotide positions in length. In our tests, we used the HKY [27] DNA substitution model with the same three Ts/Tv ratio parameters as were used by parallel fastDNAml.We examined several trees constructed by DPRml using this dataset and found that there were only minor differences due to the differing randomisation of the taxa addition order. Several of the trees produced are available from the system webpage. For all of our performance tests, we ran a version of the program that adds the taxa to the tree in the same order each time (so that the scaling behaviour of the program could be clearly understood) and the program was configured to optimize the branch lengths of every tree generated. The maximum number of vertices that rearrangements could cross was set to five.

4.1 Single problem speedup analysis

To analyse the speedup that can be gained by running DPRml, we ran a single instance of DPRml on the distributed system with differing numbers of clients and noted the total running time in each case. For these particular tests, the set of clients consisted of two university computing laboratories with a total of 60 desktop PCs (each machine was a Pentium IV 2.4 GHz with 512 MB of RAM running either Windows 2000 or Redhat Linux 7.0). Our server resided on a Pentium III 600 MHz with 256 MB of RAM running Debian Linux with a 10 Mbit/s connection to the laboratories. We had our client installed as a low priority background service and the PC's were in use, and were being rebooted between operating systems during teaching hours. The graphs show the corresponding mean running time decrease (Figure 5) and speedup gained (Figure 6) over two runs for each point on the graphs. The main factor limiting the scalability of the program is the synchronisation barrier created by the staged nature of the algorithm. If any of

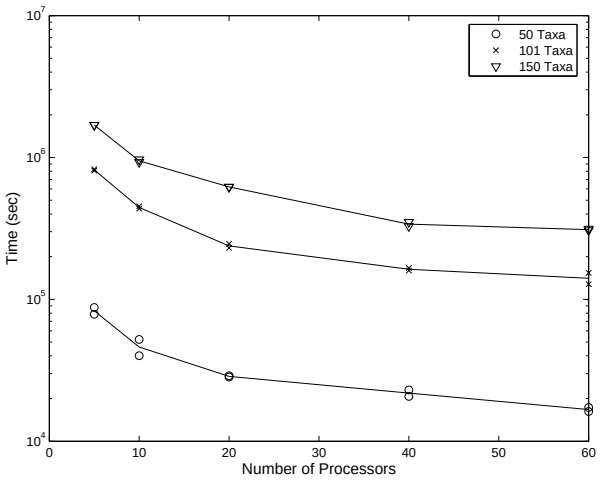

Fig. 5 Decrease in computation time with an increase in the number of processors over each of the three datasets (50, 101, and 150 taxa). The average over two runs for each dataset is shown.

the donor machines are unexpectedly switched off, DPRml must wait for the distributed system to detect this and redistribute the data to another donor machine before it can proceed to the next stage of the algorithm. Figure 6 shows that DPRml scales extremely well, with the speedup increasing with an increase in dataset size. This is consistent with the findings of a special purpose parallel phylogenetic program [16].For this particular dataset, it is expected that the speedup gains should plateau at approximately 150 processors because at this point the number of processors would equal the number of trees being generated at many of the stages. This would also be the case for any other parallel tree building program.

4.2 Multiple problem efficiency analysis

A simple examination of the tree building algorithm outlined in Section 2 reveals a number of clearly identifiable stages within the algorithm. At each stage, a number of phylogenetic trees are generated from the current best tree and each tree must be evaluated by a donor machine before the next stage of the algorithm can commence. Therefore the main factor limiting the parallel efficiency of the program is the synchronisation barrier that is createdby the staged nature of the algorithm. One way of maintaining a high throughput in the distributed system is to run several DPRml computations simultaneously. Researchers generally run the same computation multiple times and take the best overall result, due to the stochastic nature of this heuristic method, so running multiple DPRml computations in parallel mirrors the expected usage of the program. Ideally to maximise efficiency, each computation would be at a different stage in the tree building algorithm and therefore

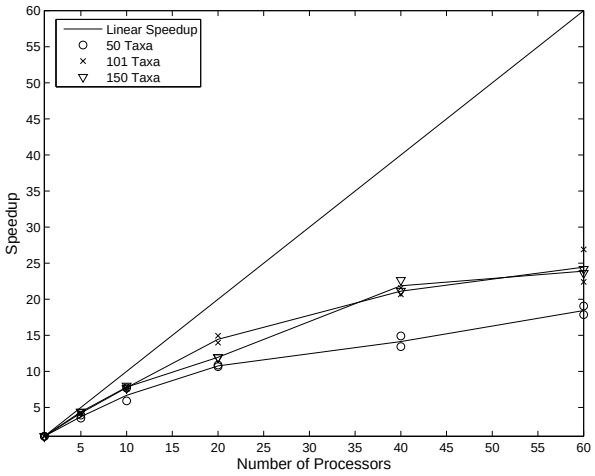

Fig. 6 Speedup achieved over each of the three datasets (50, 101, and 150 taxa). The average over two runs for each dataset is shown. Linear speedup is the theoretical maximum for parallel algorithms.

should result in consistently higher overall throughput. We were also interested in the extent to which an increasing number of DPRml computations would affect the rate at which the phylogenetic trees are built.

To investigate these two related issues, we obtained one of the datasets that was used to test parallel fastDNAml [16], consisting of 101 taxa (1858 nucleotides per taxa), and we ran varying numbers of DPRml computations on the system while keeping the number of donor machines fixed. The program was configured to optimize the branch lengths of every tree generated and the maximum number of vertices that rearrangements could cross was set to five. The set of clients consisted of a university computing laboratory with a total of 44 desktop PC's (each machine was a Pentium IV 2.4 GHz with 512 MB of RAM running Windows 2000). Our server resided on a Pentium III 600 MHz with 256 MB of RAM running Debian Linux with a 10 Mbit/s connection to the laboratory. Using the distributed system log files, we analysed the state of the system when varying numbers of instances of DPRml were run in parallel over a 24 hour period (see Figures 7-12).

Efficiency is the percentage utilization of processing resources, $e(t) = \frac{p(t)}{t \times n}$ where t is the current time and $t = 0$ at the start of the computation, n is the number of clients, and $p(t)$ is the total processing time performed by clients up to time t. Figure 7 shows the efficiency of the system with varying numbers of problems. DPRml is a staged computation, thus when there is only 1 problem in the system, processors are quite often idle (see Figure 8). When 5 problems are processed in parallel the efficiency improves dramatically (as can be seen in Figure 7), and the number of idle processor consistantly low (see Figure 9).

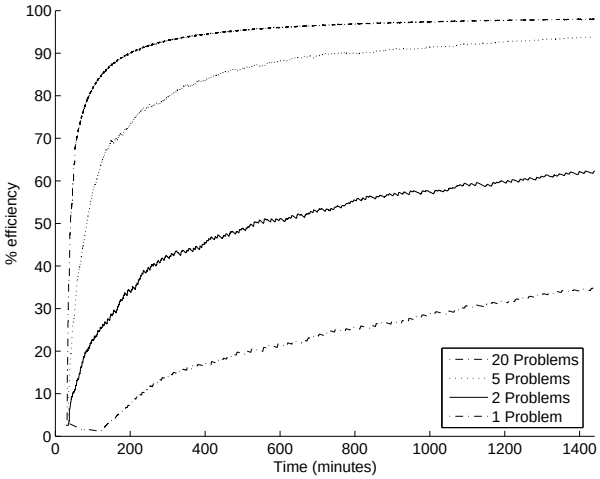

Fig. 7 The percentage efficiency of processor utilization over a 24 hours period with varying numbers of problems.

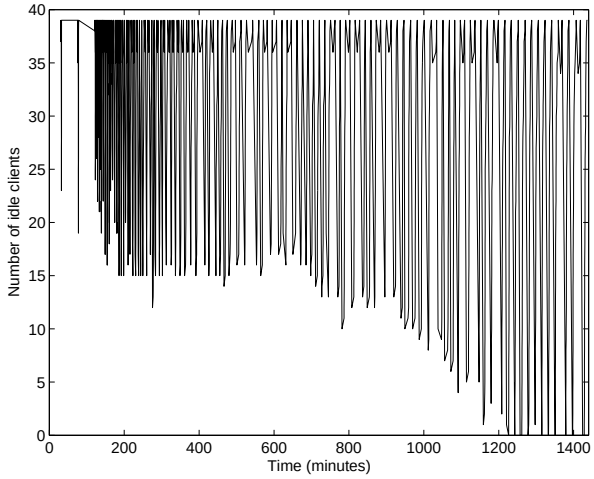

Fig. 8 A trace of the number of idle clients over 24 hours with 1 problem in the system.

The evidence from Figure 7 would suggest that for this particular dataset and set of donor machines, running approximately 5 tree building computations in parallel will utilize the system resources efficiently. After 20 problems there is very little gain in efficiency with the addition of more problems.

Figure 10 shows that as efficiency goes up, the percentage of time processors are idle goes down (as expected). The communication overhead also increases approximately linearly with increasing numbers of problems (see Figure 11). The

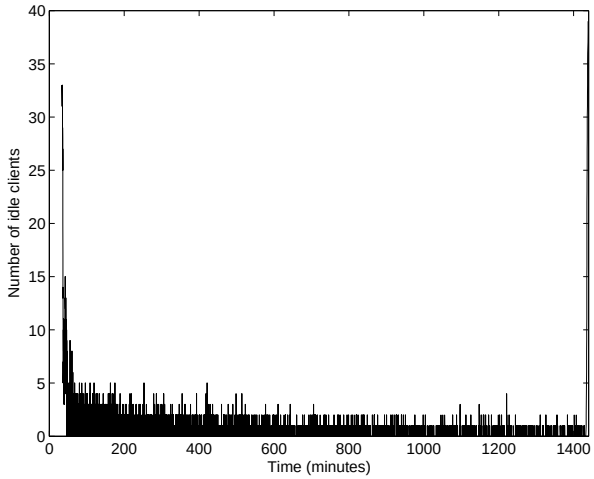

Fig. 9 A trace of the number of idle clients over 24 hours with 5 problems in the system.

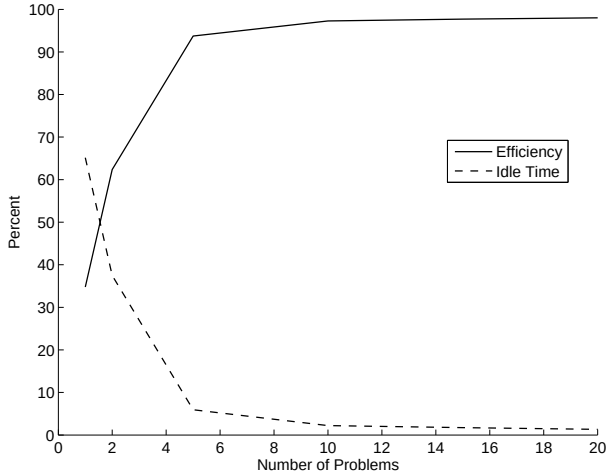

Fig. 10 The percentage efficiency and idle time with varying numbers of problems after 24 hours.

compute-to-data ratio is consistently high, even when 20 problems are being processed parallel, which provides evidence to support scalability of this problem on this distributed system.

Figure 12 shows that increasing the number of tree building computations from 1 to 20 affected the average size of the trees built over 24 hours, but dramatically increased the efficiency of the system (and conversely reducing the amount of time processors were idle).

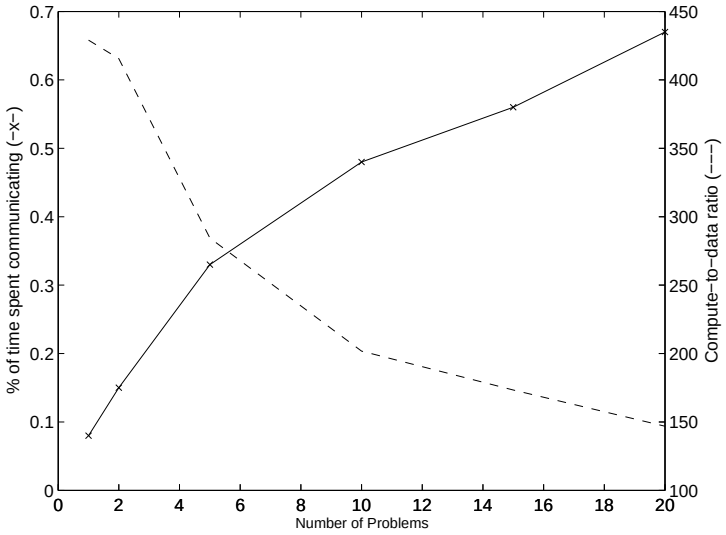

Fig. 11 The percentage communication overhead and the compute-to-data ratio with varying numbers of problems after 24 hours.

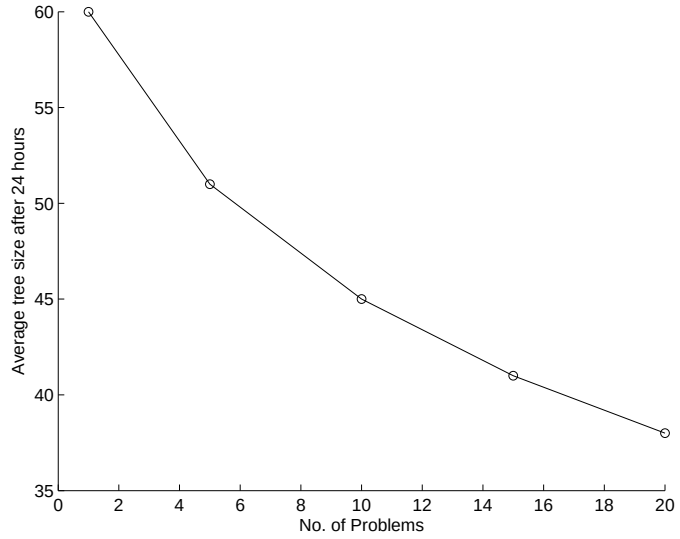

Fig. 12 The average number of leaves in a tree, with varying numbers of problems, after 24 hours.

This result fits well with the expected usage of the program. As the algorithm outlined in Section 2 is stochastic, it is possible to become trapped in a local optimum, rather than a global one. Typically a biologist would repeat the entire tree building process with several different randomizations of the order of taxa and then compare the best of the resulting trees to determine a consensus tree [37].

5 Conclusion

Due to the recent explosion in the size of sequence databases, there is increased interest in producing very large and accurate phylogenetic trees using maximum likelihood. For a large number of taxa, it is not possible to perform an exhaustive search of the tree space. Many authors have proposed heuristic algorithms aimed at speeding up the process of constructing phylogenetic trees. However many of these algorithms do not perform a sufficiently rigorous search of the tree space and often result in sub-optimal trees. Therefore a number of researchers have developed specialised parallel programs in an attempt to perform a more complete search of the tree space and thus produce more accurate phylogenetic trees. We have expressed a number of limitations of these parallel programs that are currently severely limiting the widespread use of parallel computing in phylogenetic analysis.

Distributed computing offers inexpensive access to large amounts of computing power. We have identified the suitability of phylogenetic tree construction to coarse grained distributed computing. We have completed a distributed and fully cross platform phylogenetic tree building program called DPRml. DPRml uses an already proven tree building algorithm and popular phylogenetic analysis library. The usability and generality of our program is demonstrated by the ease of use and platform heterogeneity of DPRml. The user can choose from one of the most extensive ranges of DNA substitution models currently available. We have shown how DPRml can be used to make the most efficient use of available spare clock cycles to build large phylogenetic trees. The final outputs of the program are in standard formats that allow the user to perform further manipulation and analysis of results using other phylogenetic packages.

This is the first release of DPRml and we have identified a number of areas that warrant further research and development. We would like to investigate possible ways of further improving speedup. For example, Ceron has devised a scheme whereby the list of trees to be issued is calculated in advance [38], thus reducing the limiting influence of the synchronisation barrier. As new features and algorithmic improvements appear in later versions of PAL [39], we will release updated versions of DPRml on our webpage to take advantage of the improvements. Finally we would like to extend DPRml to construct a number of trees using a single dataset and then output the overall consensus tree [37]. DPRml is freely available under the terms of the GNU General Public Licence. There is also a user manual available for download from the system webpage.

Acknowledgements This research has been funded by Embark Initiative from the Irish Research Council for Science, Engineering and Technology: funded by the National Development Plan. We would also like to thank Matthew Goode of the PAL project for help and advice on how to use PAL v1.4.

References

1. H. Bodlaender, M. Fellows, and T. Warnow. *Two strikes against perfect phylogeny*, volume 623, pages 273–283. Springer-Verlag, NY, USA, 1992.
2. K. Strimmer and A. von Haeseler. Quartet Puzzling: A Quartet Maximum-Likelihood Method for Reconstructing Tree Topologies. *Molecular Biology and Evolution*, 13(7):964–969, 1996.
3. J. Adachi and M. Hasegawa. MOLPHY version 2.3 programs for molecular phylogenetics based on maximum likelihood. *Computer Science Monographs*, 28:1–150, 1996.
4. J.P. Huelsenbeck and F. Ronquist. MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755, 2001.
5. P.O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–283, 1998.
6. L.A. Salter and D.K. Pearl. Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. *Systematic Biology*, 50(1):7–17, 2001.
7. B.M.E. Moret, S. Wyman, D.A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proceedings of the 6th Pacific Symposium on Biocomputing*, pages 583–594, Hawaii, 2001. World Scientific Publications.
8. J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.
9. J.P. Huelsenbeck and D.M. Hillis. Success of phylogenetic methods in the four-taxon case. *Systematic Biology*, 42:247–264, 1993.
10. M.K. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates [published erratum appears in Molecular Biology and Evolution 1995 May;12(3):525]. *Molecular Biology and Evolution*, 11(3):459–468, 1994.
11. J.P. Huelsenbeck. Performance of phylogenetic methods in simulation. *Systematic Biology*, 44:17–48, 1995.
12. M.S. Rosenberg and S. Kumar. Traditional Phylogenetic Reconstruction Methods Reconstruct Shallow and Deep Evolutionary Relationships Equally Well. *Molecular Biology and Evolution*, 18(9):1823–1827, 2001.
13. V. Ranwez and O. Gascuel. Improvement of Distance-Based Phylogenetic Methods by a Local Maximum Likelihood Approach Using Triplets. *Molecular Biology and Evolution*, 19(11):1952–1963, 2002.
14. B. Korber, M. Muldoon, J. Theiler, F. Gao, R. Gupta, A. Lapedes, B.H. Hahn, S. Wolinsky, and T. Bhattacharya. Timing the ancestor of the HIV-1 pandemic strains. *Science*, 288:1789–1796, 2000.
15. M.A. Hershkovitz and D.D. Leipe. Phylogenetic analysis. In A.D. Baxevanis and B.F.F. Ouelette, editors, *Bioinformatics: a practical guide to the analysis of genes and proteins*, pages 189–230. Wiley-Liss, New York, USA, 1998.
16. C.A. Stewart, D. Hart, D.K. Berry, G.J. Olsen, E.A. Wernert, and W. Fischer. Parallel implementation and performance of fastDNAml: a program for maximum likelihood phylogenetic inference. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, page 20, New York, NY, USA, 2001. ACM Press.

17. A.P. Stamatakis and T. Ludwig. Phylogenetic Tree Inference on PC Architectures with AxML/PAxML. In *Proceedings of IPDPS2003 (High Performance Computational Biology workshop)*, Nice, France, 2003.
18. H.A. Schmidt, K. Strimmer, M. Vingron, and A. von Haeseler. TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18(3):502–504, 2002.
19. G.J. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek. FastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Computer Applications in the Biosciences*, 10:41–48, 1994.
20. A. Drummond and K. Strimmer. PAL: An object-oriented programming library for molecular evolution and phylogenetics. *Bioinformatics*, 17:662–663, 2001.
21. T.M. Keane, T.J. Naughton, S.A.A. Travers, J.O. McInerney, and G.P. McCormack. DPRml: distributed phylogeny reconstruction by maximum likelihood. *Bioinformatics*, 21(7):969–974, 2005.
22. D. Posada and K.A. Crandall. MODELTEST: testing the model of DNA substitution. *Bioinformatics*, 14(9):817–818, 1998.
23. J. Felsenstein. Models of DNA Evolution. In *Inferring Phylogenies*, pages 196–221. Sinauer Associates, Sunderland, MA, 2004.
24. T. Muller and M. Vingron. Modelling amino acid replacement. *Journal of Computational Biology*, 7:761–776, 2000.
25. T.H. Jukes and C.R. Cantor. Evolution of protein molecules. In Munro, editor, *Mammilian Protein Metabolism*, pages 240–252. Academic Press, New York, USA, 1969.
26. M. Kimura. A simple method for estimating evolutionary rate of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16:111–120, 1980.
27. M. Hasegawa, H. Kishino, and T. Yano. Dating the human-age splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22:160–174, 1985.
28. K. Tamura and M. Nei. Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution*, 10:512–526, 1993.
29. F.J. Rodriguez, J.L. Oliver, A. Marin, and J.R. Medina. The general stochastic model of nucleotide substitution. *Journal of Theoretical Biology*, 142:485–501, 1990.
30. M. Steel, D. Hudson, and P.J. Lockhart. Invariable sites models and their use in phylogeny reconstruction. *Systematic Biology*, 49:225–232, 2000.
31. Z. Yang. The among-site rate variation and its impact on phylogenetic analyses. *Trends in Ecology and Evolution*, 11:367–372, 1996.
32. T.M. Keane and T.J. Naughton. DSEARCH: sensitive database searching using distributed computing. *Bioinformatics*, 21(8):1705–1706, 2005.
33. K. Fritsche, J. Power, and J. Waldron. A Java distributed computation library. In *Proceedings of the 2nd International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 236–243, Taipei, Taiwan, July 2001.
34. T. Keane, R. Allen, T. J. Naughton, J. McInerney, and J. Waldron. Distributed Java platform with programmable MIMD capabilities. In N. Guelfi, E. Astesiano, and G. Reggio, editors, *Scientific Engineering for Distributed Java Applications*, volume 2604, pages 122–131, Berlin, February 2003. Springer Lecture Notes in Computer Science.
35. H. Edelstein. Unraveling client/server architecture. *DBMS*, 7(5):34–41, May 1994.
36. T. Keane. Java distributed system: Developer manual. Technical Report NUIM-CS TR-2003-03, Dept. of Computer Science, National University of Ireland, Maynooth, Ireland, 2003.
37. L.S. Jermiin, G.J. Olsen, and S. Easteal. Majority rule consensus of maximum likelihood trees. *Molecular Biology and Evolution*, 14:1296–1302, 1997.

38. C. Ceron, J. Dopazo, E.L. Zapata, J.M. Carazo, and Trelles O. Parallel implementation of DNAml program on message-passing architectures. *Parallel Computing*, 24:701–716, 1998.
39. M. Goode, K. Strimmer, A. Drummond, E. Buckler, and A. Rodrigo. A brief introduction to the Phylogenetic Analysis Library version 1.5. In *CRPIT '29: Proceedings of the second conference on Asia-Pacific bioinformatics*, pages 175–179, Dunedin, New Zealand, 2004. Australian Computer Society, Inc.