Parallel Inference of a 10.000-taxon Phylogeny with Maximum Likelihood

Alexandros Stamatakis[1], Thomas Ludwig[2], and Harald Meier[1]

[1] Technische Universität München, Department of Computer Science, Boltzmannstr. 3 D-85748 Garching b. München Germany
Alexandros.Stamatakis@in.tum.de, Harald.Meier@in.tum.de
[2] Ruprecht-Karls-Universität, Department of Computer Science, Im Neuenheimer Feld 348 D-69120 Heidelberg Germany
thomas.ludwig@informatik.uni-heidelberg.de

Abstract. Inference of large phylogenetic trees with statistical methods is computationally intensive. We recently introduced simple heuristics which yield accurate trees for synthetic as well as real data and are implemented in a sequential program called RAxML. We have demonstrated that RAxML outperforms the currently fastest statistical phylogeny programs (MrBayes, PHYML) in terms of speed and likelihood values on real data. In this paper we present a non-deterministic parallel implementation of our algorithm which in some cases yields super-linear speedups for an analysis of 1.000 organisms on a LINUX cluster. In addition, we use RAxML to infer a 10.000-taxon phylogenetic tree containing representative organisms from the three domains: Eukarya, Bacteria and Archaea. Finally, we compare the sequential speed and accuracy of RAxML and PHYML on 8 synthetic alignments comprising 4.000 sequences.

1 Introduction

Within the ParBaum project at the Technische Universität München, we work on phylogenetic tree inference based on the maximum likelihood method by J. Felsenstein [2]. We intend to develop novel systems and algorithms for computation of huge phylogenetic trees based on sequence data from the ARB [6] ssu rRNA (small subunit ribosomal RiboNucleic Acid) database. In a recent paper [11] we implemented simple heuristics in RAxML (Randomized Axelerated Maximum Likelihood) which accelerate the tree optimization process and yield good results in terms of final likelihood values. In a series of experiments with 9 real data alignments containing 101 up to 1.000 organisms we demonstrate that RAxML is the currently -to the best of our knowledge- fastest and most accurate sequential program for real data under the HKY85 [4] model of nucleotide substitution. In this paper we describe the parallel non-deterministic implementation of RAxML and report speedup values on a LINUX cluster for an alignment containing 1.000 organisms. Finally, we use the sequential and parallel version of RAxML to infer the -to the best of our knowledge- first integral maximum likelihood-based tree containing 10.000 sequences from the three domains: Eukarya, Bacteria and Archaea. This large alignment has been extracted

in cooperation with biologists from ARB. The source code of the sequential and parallel program including all alignment files and final trees is freely available for download at WWWBODE.IN.TUM.DE/~STAMATAK.

Related Work: A comparison of popular phylogeny programs using statistical approaches such as fastDNAml [7], MrBayes [5], treepuzzle [14], and PAUP [8] based on synthetic data may be found in [15]. MrBayes carries out bayesian phylogenetic inference and outperforms all other phylogeny programs in terms of speed and tree quality in this survey. More recently, Guidon et al. published their new maximum likelihood program PHYML [3], which seems to be able to compete with MrBayes. To the best of our knowledge apart from RAxML, MrBayes and PHYML are currently the fastest and most accurate programs for phylogenetic tree inference. In addition, results in [3] and [11] suggest that traditional maximum likelihood methods are still significantly faster than bayesian phylogenetic inference. Thus, maximum likelihood-based programs currently represent the only statistical approach for computation of trees comprising more than 500 sequences. Another important issue is that MrBayes and PHYML have high memory consumption compared to RAxML. For a 1.000 sequence alignment RAxML consumed 199MB, PHYML 880MB, and MrBayes 1.195MB of main memory. Furthermore, both MrBayes and PHYML exited with error messages due to excessive memory requirements for the 10.000 taxon alignment on a processor equipped with 4GB (!) of main memory. Therefore, we made an effort to port MrBayes and PHYML to a 64-bit Itanium2 1.3GHz processor with 8GB of memory. While MrBayes exited for unknown reasons, PHYML finally required 8.8GB of main memory in contrast to RAxML which consumed only 800MB. In what concerns parallel computing, the parallel implementations of bayesian methods are relatively closely coupled such that high performance computers with expensive communication infrastructure are required [1]. For PHYML there exists no parallel implementation yet. There also exists a popular parallel implementation for fastDNAml [13] which is however based on the old sequential algorithm from 1994.

2 Heuristics

Sequential Algorithm: The heuristics of RAxML belong to the class of algorithms, which optimize the likelihood of a starting tree which already comprises all sequences. In contrast to other programs RAxML starts by building an initial tree with the dnapars parsimony program from Felsenstein's PHYLIP package [9] for two reasons: *Firstly*, parsimony is related to maximum likelihood under simple models of evolution such that one can expect to obtain a starting tree with a relatively good likelihood value compared to random or neighbor joining starting trees. *Secondly*, dnapars uses stepwise addition [2] for tree building and is relatively fast. The stepwise addition algorithm enables the construction of distinct starting trees by using a randomized input sequence order. Thus, RAxML can be executed several times with different starting trees and thereby yields a set

of distinct final trees which can be used to build a consensus tree. To expedite computations, some optimization steps have been removed from dnapars.

The tree optimization process represents the second and most important part of the heuristics. RAxML performs standard subtree rearrangements by subsequently removing all possible subtrees from the currently best tree `t_best` and re-inserting them into neighboring branches up to a specified distance of nodes. RAxML inherited this optimization strategy from fastDNAml. One rearrangement step in fastDNAml consists of moving all subtrees within the currently best tree by the minimum up to the maximum distance of nodes specified (lower/upper rearrangement setting). This process is outlined for a single subtree (ST5) and a distance of 1 in Figure 1. In fastDNAml the likelihood of each thereby generated topology is evaluated by exhaustive branch length optimizations. If one of those alternative topologies improves the likelihood `t_best` is updated accordingly and once again all possible subtrees are rearranged within `t_best`. This process of rearrangement steps is repeated until no better topology is found. The rearrangement process of RAxML differs in two major points: In fastDNAml after each insertion of a subtree into an alternative branch the branch lengths of the entire tree are optimized. As depicted in Figure 1 with bold lines RAxML only optimizes the three local branches adjacent to the insertion point either analytically or by the Newton-Raphson method before computing its likelihood value. Since the likelihood of the tree strongly depends on the topology per se this fast pre-scoring can be used to establish a small list of potential alternative trees which are very likely to improve the score of `t_best`. RAxML uses a list of size 20 to store the best 20 trees obtained during one rearrangement step. This list size proves to be a practical value in terms of speed and thoroughness of the search. After completion of one rearrangement step the algorithm performs global branch length optimizations on those 20 best topologies only. Due to the capability to analyze significantly more alternative topologies in less time a higher upper rearrangements setting can be used e.g. 5 or 10 which results in significantly improved final trees. Another important change especially for the

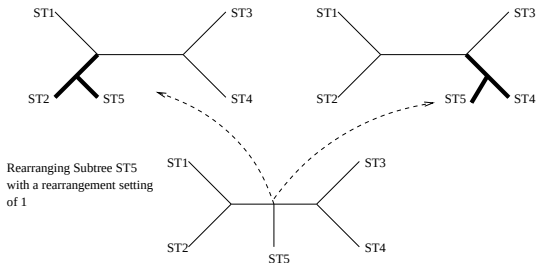

Fig. 1. Rearrangements traversing one node for subtree ST5, branches which are optimized by RAxML are indicated by bold lines

initial optimization phase, i.e. the first 3-4 rearrangement steps, consists in the subsequent application of topological improvements during one rearrangement step. If during the insertion of one specific subtree into an alternative branch a

topology with a better likelihood is encountered this tree is kept immediately and all subsequent subtree rearrangements of the current step are performed on the improved topology. This enables rapid initial optimization of random starting trees [11]. The exact implementation of the RAxML algorithm is indicated in the C-like pseudocode below. The algorithm is passed the user/parsimony starting tree `t`, the initial rearrangement setting `rStart` (default: 5) and the maximum rearrangement setting `rMax` (default: 21). Initially the rearrangement stepwidth ranges from `rL = 1` to `rU = rStart`. Fast analytical local branch length optimization `a` is turned off when functions `rearr()`, which actually performs the rearrangements, and `optimizeList20()` fail to yield an improved tree for the first time. As long as the tree does not improve the lower and upper rearrangement parameters `rL, rU` are incremented by `rStart`. The program terminates when the upper rearrangement setting is greater or equal to the maximum rearrangement setting, i.e. `rU >= rMax`.

```
optimize(tree t, int rStart, int rMax)
{
 int rL, rU;
 boolean a = TRUE, impr = TRUE, stop = FALSE;
 while(!stop){
   if(impr){
     rL = 1;
     rU = rStart;
     rearr(t, rL, rU, a);
    }
   else{
     if(!a){
       a = FALSE;
       rL = 1;
       rU = rStart;
      }
     else{
       rL += rStart;
       rU += rStart;
      }
     if(rU < rMax) rearr(t, rL, rU, a);
     else stop = TRUE;
    }
   impr = optimizeList20();
  }
}
```

Parallel Algorithm: The parallel implementation is based on a simple master-worker architecture and consists of two phases. In **phase I** the master distributes the alignment file to all worker processes if no common file system is available, otherwise it is read directly from the file. Thereafter, each worker independently computes a randomized parsimony starting tree and sends it to the master process. Alternatively, it is possible to start the program directly in **phase II** by specifying a tree file name in the command line. In **phase II** the master initiates the optimization process for the best parsimony or specified starting tree. Due to the high speed of a single topology evaluation, the requirement for atomicity

of a specific subtree rearrangement by function `rearrangeSubtree()` and the high communication cost, it is not feasible to distribute work by single topologies as e.g. in parallel fastDNAml. Therefore, we distribute work by sending the subtree ID along with the currently best topology `t_best`, to each worker. The `sequential` and `parallel` implementation of RAxML on the master-side is outlined in the pseudocode of function `rearr()` which actually executes subtree rearrangements. The worker simply executes function `rearrangeSubtree()`.

```
void rearr(tree t_best, int rL, int rU, boolean a)
{
 boolean impr;
 worker w;
 for(i = 2; i < #species * 2 - 1; i++){
   if(sequential){
     impr = rearrangeSubtree(t_best, i, rL, rU, a);
     if(impr) applySubsequent(t_best, i);
   }
   if(parallel){
     if(w = workerAvailable) sendJob(w, t_best, i);
     else putInWorkQueue(i);
   }
 }
 if(parallel){
   while(notAllTreesReceived){
     w = receiveTree(w_tree);
     if(likelihood(w_tree) > likelihood(t_best)) t_best = w_tree;
     if(notAllTreesSent) sendJob(w, t_best, nextInWorkQueue());
  }
 }
}
```

In the sequential case rearrangements are applied to each individual subtree `i`. If the tree improves through this subtree rearrangement `t_best` is updated accordingly, i.e. subsequent topological improvements are applied. In the parallel case subtree IDs are stored in a work queue. Obviously, the subsequent application of topological improvements during 1 rearrangement step (1 invocation of `rearr()` is closely coupled. Therefore, we slightly modify the algorithm to break up this dependency according to the following observation: Subsequent improved topologies occur only during the first 3–4 rearrangement steps (initial optimization phase). Thereafter, the likelihood is improved only by function `optimizeList20()`. This phase requires the largest amount of computation time, especially with big alignments ($\approx 80\%$ of execution time). Thus, during the initial optimization phase we send only one single subtree ID `i=2,...,#species * 2 - 1` along with the currently best tree `t_best` to each worker for rearrangements. Each worker returns the best tree `w_tree` obtained by rearranging subtree `i` within`t_best` to the master. If `w_tree` has a better likelihood than `t_best` at the master, we set `t_best = w_tree` and distribute the updated best tree to each worker along with the following work request. The program assumes that the initial optimization **phase IIa** is terminated if no subsequent improved topology

has been detected during the last three rearrangement steps. In the final optimization **phase IIb**, we reduce communication costs and increase granularity by generating only $5 * \#workers$ jobs (subtree ID spans). Finally, irrespective of the current optimization phase the best 20 topologies (or $\#workers$ topologies if $\#workers > 20$) computed by each worker during one rearrangement step are stored in a local worker tree list. When all `#species * 2 - 3` subtree rearrangements of `rearr()` have been completed, each worker sends its tree list to the master. The master process merges the lists and redistributes the 20 ($\#workers$) best tree topologies to the workers for branch length optimization, like in parallel fastDNAml. When all topologies have been globally optimized the master starts the next iteration of function `optimize()`. Due to the required changes to the algorithm the parallel program is non-deterministic, since final output depends on the number of workers and on the arrival sequence of results for runs with equal numbers of workers, during the initial optimization **phase IIa**. This is due to the altered implementation of the subsequent application of topological improvements during the initial rearrangement steps which leads to a traversal of search space on different paths.

3 Results

For our experiments we extracted alignments of 1.000 and 10.000 taxa (1000_ARB, 10000_ARB) from the ARB database containing organisms from the domains Eukarya, Bacteria and Archaea. We used the HKY85 model of sequence evolution and a transition/transversion ratio of 2.0. Furthermore, we generated 8 synthetic 4.000 taxon alignments (SIM_1,...,SIM_8) with a length of 2.000 base pairs and distinct parameter settings for comparison of PHYML and RAxML.

Synthetic Data Tests: In Table 1 we list the topological distance to the simulated "true" tree (normalized Robinson-Foulds rate) and execution time in seconds of PHYML and RAxML for the 8 synthetic 4.000 taxon alignments. Details on the generation of the simulated data sets, a discussion of results, and supplementary experiments with real-data are provided in [12].

Table 1. Topological accuracy and execution times for PHYML & RAxML on simulated data

data	PHYML	secs	RAxML	secs	data	PHYML	secs	RAxML	secs
SIM_1	0.065	18944	0.065	9152	SIM_5	0.028	24182	0.035	91178
SIM_2	0.039	22273	0.037	50609	SIM_6	0.027	32614	0.031	176686
SIM_3	0.033	24907	0.027	97962	SIM_7	0.027	34750	0.032	185454
SIM_4	0.030	30870	0.031	85080	SIM_8	0.026	18828	0.036	78061

Scalability Tests: We conducted parallel tests with a fixed starting tree for 1000_ARB. The program was executed on the 2.66GHz Xeon cluster on the RRZE [10] on 1, 4, 8, 16, and 32 processors with an initial rearrangement setting

`rStart` of 5. To calculate the speedup values we only take into account the number of workers, since the master process hardly produces any load. In Figure 2 we plot "fair" and "normal" speedup values obtained for the experiments with the 1000_ARB data set at the RRZE PC-cluster. "Fair" speedup values measure the first point of time at which the parallel code encounters a tree with a better likelihood than the final tree of the sequential run or vice versa. These "fair" values better correspond to real program performance. Furthermore, we also indicate "normal" speedup values which are based on the entire execution time of the parallel program, irrespective of final likelihood values. Since we intend to explore the effect of non-determinism on program performance we executed the parallel code 4 times for each job-size and calculated average "normal"/"fair" execution times and likelihood values.

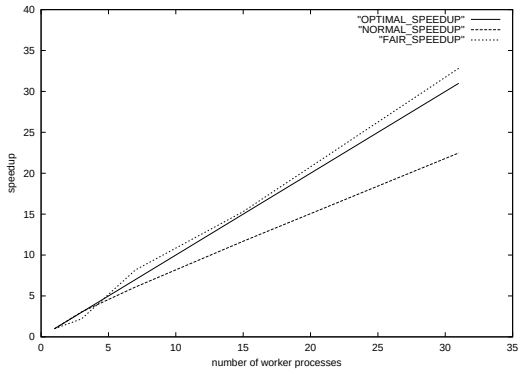

Fig. 2. Normal, fair, and optimal speedup values for 1000_ARB with 3,7,15, and 31 worker processes on the RRZE PC Cluster

Inference of a 10.000-taxon tree: The computation of the 10.000-taxon tree was conducted using the sequential, as well as the parallel version of RAxML. One of the advantages of RAxML consists in the randomized generation of starting trees. Thus, we computed 5 distinct randomized parsimony starting trees sequentially along with the first 3–4 rearrangement steps on a small cluster of Intel Xeon 2.4GHz processors at our institute. This phase required an average of 112.31 CPU hours per tree. Thereafter, we executed several subsequent parallel runs (due to job run-time limitations of 24 hrs) with the respective starting trees on either 32 or 64 processors at the RRZE 2.66GHz Xeon-cluster. The parallel computation required an average of 1689.6 accumulated CPU hours per tree. The best likelihood for 10000_ARB was -949570.16 the worst -950047.78 and the average -949867.27. PHYML reached a likelihood value of -959514.50 after 117.25 hrs on the Itanium2. Note, that the parsimony starting trees computed with RAxML had likelihood values ranging between -954579.75 and -955308.00. The average time required for computing those starting trees was 10.99 hrs. Since

bootstrapping is not feasible for this large data size and in order to gain some basic information about similarities among the 5 final trees we built a consensus tree using the extended majority rule with consense from PHYLIP (consense constantly exited with a memory error message when given more than 5 trees). The consensus tree has 4777 inner nodes which appear in all 5 trees, 1046 in 4, 1394 in 3, 1323 in 2, and 1153 in only 1 tree (average: 3.72).

4 Conclusion

We presented an efficient parallel implementation of recently introduced heuristics for phylogenetic inference under simple models of site substitution which achieves optimal speedup values. Thus, RAxML provides a fast and practicable approach for sequential and parallel inference of large phylogenetic trees containing up to 10.000 organisms. We were able to compute the -to the best of our knowledge- first 10.000-taxon tree with maximum likelihood using RAxML on a medium-size commodity PC cluster. However, at this tree size, there arise new, yet unresolved problems such as, assessment of quality and visualization which require further investigation.

References

1. Feng, X. et al.: Parallel algorithms for Bayesian phylogenetic inference. J. Par. Dist. Comp. (2003) 63:707–718
2. Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. J. Mol. Evol., (1981) 17:368–376,
3. Guindon, S. et al.: A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood. Syst. Biol., (2003) 52(5):696–704
4. Hasegawa,M. et al.: Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. J. Mol. Evol., (1985) 22:160–174
5. Huelsenbeck, J.P. et al.: MRBAYES: Bayesian inference of phylogenetic trees. Bioinf., (2001) 17(8):754–755
6. Ludwig, W. et al.: ARB: A Software Environment for Sequence Data. Nucl. Acids Res., (2004) 32(4):1363–1371
7. Olsen, G. et al.: fastdnaml: A Tool for Construction of Phylogenetic Trees of DNA Sequences using Maximum Likelihood. Comput. Appl. Biosci., (1994) 10:41–48
8. PAUP: PAUP.CSIT.FSU.EDU, visited May 2003
9. PHYLIP: EVOLUTION.GENETICS.WASHINGTON.EDU, visited Nov 2003
10. RRZE: WWW.RRZE.UNI-ERLANGEN.DE, visited Oct 2003
11. Stamatakis, A. et al.: New Fast and Accurate Heuristics for Inference of Large Phylogenetic Trees. Proc. of IPDPS2004 (2004)
12. Stamatakis, A. et al.: RAxML-III: A Fast Program for Maximum Likelihood-based Inference of Large Phylogenetic Trees. Bioinf. to be published
13. Stewart, C. et al.: Parallel Implementation and Performance of fastdnaml - a Program for Maximum Likelihood Phylogenetic Inference. Proc. of SC2001 (2001)
14. Strimmer, K. et al.: Quartet Puzzling: A Maximum-Likelihood Method for Reconstructing Tree Toologies. Mol. Biol. Evol., (1996) 13:964-969
15. Williams, T.L. et al.: An Investigation of Phylogenetic Likelihood Methods. Proc. of BIBE'03 (2003)