



Large Maximum Likelihood Trees

Bui Quang Minh, Le Sy Vinh, Heiko A. Schmidt,
and Arndt von Haeseler

published in

NIC Symposium 2006,
G. Münster, D. Wolf, M. Kremer (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. 32, ISBN 3-00-017351-X, pp. 357-366, 2006.

© 2006 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for
personal or classroom use is granted provided that the copies are not
made or distributed for profit or commercial advantage and that copies
bear this notice and the full citation on the first page. To copy otherwise
requires prior specific permission by the publisher mentioned above.

<http://www.fz-juelich.de/nic-series/volume32>

Large Maximum Likelihood Trees

Bui Quang Minh¹, Le Sy Vinh²,
Heiko A. Schmidt¹, and Arndt von Haeseler¹

¹ Center for Integrative Bioinformatics Vienna
Max F. Perutz Laboratories, Austria
E-mail: {arndt.von.haeseler, heiko.schmidt}@univie.ac.at

² Central Institute for Applied Mathematics
Research Center Jülich, 52425 Jülich, Germany
E-mail: vinh@cs.uni-duesseldorf.de

We introduce the underlying principles of phylogenetic reconstruction, avoiding all technicalities. Since phylogenetic reconstruction based on DNA-sequence data is a computational expensive undertaking, efficient algorithms are required to suggest reasonable solutions with respect to an objective function. However, even efficient programs like IQPNNI cannot cope in reasonable time with extremely large data sets. Thus, we summarize our results on implementing a hybrid parallelization scheme for IQPNNI.

1 Introduction

1.1 Motivation

Charles Darwin (1859) said in his famous *evolutionary theory*⁴ that all species have evolved from a common ancestor under the pressure of *natural selection*. The phylogenetic relationship of contemporary organisms is therefore best represented in a *phylogenetic tree*. It is one of the main objectives in biology to reconstruct this tree.

With the advent of molecular biology and the incredible pace at which new sequences from all sorts of life forms are generated, phylogenetic trees are nowadays inferred from DNA-sequence data. DNA-sequences are easy to read. A DNA is simply a long word over a finite alphabet of four letters: *A*, *C*, *G*, and *T*. This word is subject to subtle changes (mutations) in the course of time. Among mutations, the simple replacement of a letter by another letter is called substitution. These substitution accumulate during time. Thus, a DNA sequence transmitted from a grand-grand-(grand)^k-mother to its contemporary will accumulate mutations. Thus, when comparing the two sequences they will be different. These differences reflect the amount of time (in an appropriate scaling) that went by. As DNA is ubiquitously occurring in all organisms and because DNA cannot be generated de-novo, the history of species can be inferred by simply comparing their DNA sequence. As simple as this sounds, as difficult is the actual implementation of such approaches. We cannot possibly spell out all the details here, and thus refer to the pertinent literature⁶.

One of the reasons, why phylogenetic reconstruction is difficult, is the sheer amount of trees as the number of species increases. If we consider only trivalent, unrooted trees, i.e. those trees -from a graph theoretic point of view - with node degree 3 (interior nodes) and 1 (exterior nodes), where the exterior nodes are labelled with a species name, then the number $t(n)$ of trees with $n \geq 3$ exterior nodes is given by

$$t(n) = 1 \cdot 3 \cdot \dots \cdot (2n - 5). \quad (1)$$

Unfortunately, most currently used tree reconstruction algorithms that aim to optimize an objective function belong to the class of NP-hard problems^{7,5,2}, therefore it seems hopeless to actually find the truly optimal tree if the data at hand comprise more than 10 species or so. To overcome the problem of finding the best tree(s), several heuristics have been suggested such as Neighbor Joining^{12,8}, Quartet Puzzling^{18,20}, Nearest Neighbor Interchange (NNI)⁹.

In a series of studies based on computer generated data, it has been shown that these heuristics perform reasonably in terms of speed (computational efficiency) and in terms of accuracy, i.e. the potential to rediscover the tree from the data at the exterior nodes (the sequences). However, for data sets of thousands of species the heuristics turn out to be too slow, to really evaluate the data based on different models of evolution and on different tree reconstruction methods.

Thus, in recent years, people have invoked parallel computing¹⁹ to reduce the computational burden such as fastDNAm1¹¹, TREE-PUZZLE¹³, RAxML¹⁶, MrBayes¹, pIQPNNI¹⁰. These programs make use of several parallel architectures.

The most popular parallel architecture nowadays is a cluster of Symmetric Multi Processors (SMPs). An SMP is a *shared memory* computer in which several processors have access to the same physical memory space. Such SMPs are clustered by a high bandwidth network to create a hybrid system. Processes on different SMP nodes can communicate with each other using the *Message Passing Interface* (MPI)¹⁵, an industry standard for programming on *distributed memory* systems. Inside an SMP, a process can be furthermore divided into several concurrent threads applying OpenMP, a standard for shared memory programming³. The SMP cluster motivates the application of hybrid programming models with both MPI and OpenMP in order to take full advantage of the hybrid parallel architecture. Care should be taken as such an approach does not always guarantee an improvement over the pure MPI parallelization¹⁴. In the following we will summarize our experience to parallelize the IQPNNI program²⁰.

1.2 The Data

To understand the rest of the paper it is necessary to have a closer look at the data. Consider a dataset of molecular sequences from n species. To account for the different mutation processes acting on the sequences in the course of time, one has to compute first a multiple sequence alignment²¹. We will not explain how a multiple sequence alignment (MSA) is computed. It suffices to say, that a MSA is a two-dimensional table, where each row represents the sequence from an organism and each column represent a position in the sequence

	1	2	3	4	5	6	7	8	9	10	...
Human	A	T	G	C	G	C	A	T	C	A	...
Chimpanzee	A	T	G	C	G	G	G	T	G	T	...
Gorilla	G	C	G	A	G	A	C	T	T	A	...
Rhesus	T	C	C	A	A	G	G	T	C	T	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 1. An example multiple sequence alignment.

that traces back to a common ancestral position (see Fig. 1). In other words sequences are aligned in a matrix of n rows (i.e. presenting species) and m columns, where m denotes the alignment length. In MSA, columns are also called *sites*. All phylogenetic reconstruction approaches start with this type of data and try to reconstruct a tree that explains the variability observed in the MSA. We are interested in a probabilistic framework, that models the evolutionary process. To this end, we introduce a model of sequence evolution that acts on the tree.

Based on such a model it is straightforward to compute the likelihood of a tree relating these sequences⁶. We obtain the likelihood of the tree as the conditional probability of the data given the tree:

$$L(\text{tree}) = P(\text{data}|\text{tree}). \quad (2)$$

Thus, the likelihood acts as an objective function and we want to find the tree(s) that maximize(s) L given the data, i.e. the MSA. The function $P(\cdot)$ represents the evolutionary model, that is the way we think how sequences change.

The computation of the likelihood can be very expensive. Hence, to keep computation tractable, several assumptions are made. One of them is to assume that every site evolves independently of each other. We also assume that the model P is the same for all parts of the tree and for each position in the sequence. According to this, the likelihood can now be rewritten as the product of the likelihood at each site:

$$P(\text{data}|\text{tree}) = \prod_{i=1}^m P(\text{site}_i|\text{tree}). \quad (3)$$

Normally, $P(\text{site}_i|\text{tree})$ is very close to zero, and to eliminate numerical inaccuracies, one typically takes the logarithm of the likelihood function:

$$\log L(\text{tree}) = \sum_{i=1}^m \log P(\text{site}_i|\text{tree}). \quad (4)$$

This so-called likelihood function needs to be maximized by finding the best tree. This is a combinatorial optimization problem.

1.3 IQPNNI Algorithm

The IQPNNI algorithm²⁰ was recently proposed to reconstruct phylogenetic trees. Compared to other approaches IQPNNI performs well with respect to accuracy. Unfortunately, the extra accuracy is paid for by an increased computing time.

The IQPNNI algorithm comprises two major steps (Fig. 2a). In the *initial step*, an initial tree is obtained based on BIONJ tree⁸ combined with fast NNI⁹.

In the subsequent *optimization step*, the tree topology is reorganized by Important Quartet Puzzling (IQP)²⁰ and NNI to improve its likelihood. If the likelihood of the resulting tree exceeds that of the current best tree, then the current best tree is replaced by the new tree. The optimization step is repeated many times to thoroughly search the tree space. Typically the iteration stops after a user-defined number of repetitions.

Because we early noticed that the sequential IQPNNI implementation (sIQPNNI) runs relatively slowly, a pure MPI parallelization (pIQPNNI) was developed¹⁰. pIQPNNI substantially reduced the running time. We could show that its speedup is nearly optimal for up to 30 processors.

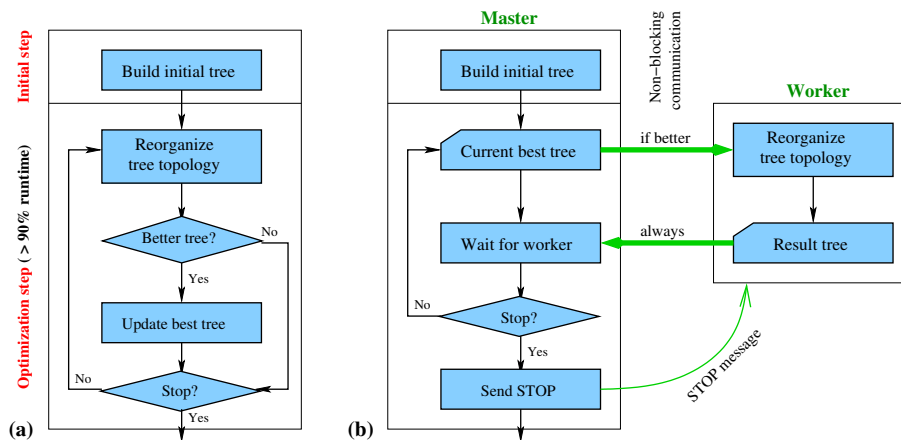


Figure 2. (a) Sequential and (b) MPI parallel scheme of IQPNNI algorithm.

2 Parallelization

2.1 MPI Parallelization

pIQPNNI was described in details in Minh et al.¹⁰ and here we outline the main idea. A preliminary analysis revealed that the optimization step consumes 90% to 99% of the total running time. Hence, the initial step is mainly carried out sequentially and the parallelization of the optimization step was done using a master/worker scheme (Fig. 2b). Starting from the current best tree, every worker runs its own optimization step as explained before. After finishing one iteration, the worker always sends back its resulting tree to the master. The master receives and updates the current best tree if the received tree shows a higher likelihood. In such case, the master will broadcast the better tree to all other workers by non-blocking communication.

The worker now synchronizes with the master and starts the next iteration with its current best tree. In addition, the master checks whether the stop condition applies and, if so, sends a stop message to all workers.

2.2 Hybrid MPI/OpenMP Parallelization

In a hybrid scheme, the program runs with p MPI processes, each process contains t OpenMP threads. That means, a total number of $p \cdot t$ processors are consumed. For p processes, we preserve the master/worker scheme as described in the previous section. For each process, the OpenMP parallelization is done in the following way.

A flow-chart analysis shows, that IQPNNI spends most of its running time (at least 90%) to calculate the likelihood of specific trees. As can be seen from the Equation 4, this involves for-loops, whose iterations are independent of each other. Therefore, we parallelized these loops. Loop-level OpenMP parallelization can be easily employed by adding a pragma directive immediately before any “for” loop involving the computation of the likelihood. A similar approach was also discussed in Stamatakis¹⁷.

By this way, the MPI and the OpenMP codes are independent of each other and one can have a pure MPI or a pure OpenMP version simply by setting $t = 1$ or $p = 1$, respectively. In the subsequent analysis, we tested the program on the JUMP (Juelich Multi Processor) system, a cluster of 41 IBM Regatta p690+ SMP nodes. Each node has 32 Power4+ processors of 1.7 GHz. We used up to 128 CPUs on this supercomputer for the experiments.

3 Performance Analysis

3.1 Datasets

The experiments were conducted on two biological datasets (Table 1). The first data, abbreviated 218dna, comprise a selection of 218 small subunit ribosomal RNA sequences (*ssu rRNA*) from the Ribosomal Database Project II <http://rdp.cme.msu.edu/>. The second dataset, 74aa, was kindly compiled by colleagues H.A. Schmidt and D. Liebers. This data set consists of amino acid sequences.

Type	Name	#Seqs	#Sites	#Iterations	Initial step	Opt. step	Total
DNA	218dna ^a	218	4182	150	70s	47m:55s	49m:05s
AA	74aa ^b	74	4013	50	77s	32m:41s	33m:58s

^a prokaryotic sequences from the small ribosomal subunit.

^b vertebrate amino acid sequences.

Table 1. The datasets used for analysis and sequential runtime of sIQPNNI.

Table 1 also displays the sequential running time of sIQPNNI. The initial step took only about 3% of the whole time on both datasets (about 70 seconds out of 50 and 34 minutes, respectively). Compared to sIQPNNI, the parallel version pIQPNNI needed a batch of about 3 minutes for both datasets using 30 CPUs.

3.2 OpenMP-IQPNNI

Firstly, we measured the performance of the pure OpenMP parallelization on a JUMP node using up to 32 processors as depicted in Fig. 3 using the 218dna and 7aa data. Interestingly, the speedup on both datasets is nearly optimal up to $t = 8$ threads and suddenly drops sharply with more than 8 threads. The runtimes with 16 threads on dataset 218dna and with 32 threads on both datasets are even greater than the sequential time, and thus not shown.

This break-down in overall performance seems to be connected to compiler issues on the JUMP system. Due to the limited time and an expected imbalance with too many threads in the hybrid version, this phenomenon was not further investigated. For that reason, we restricted further analysis to a maximum of 8 threads per process.

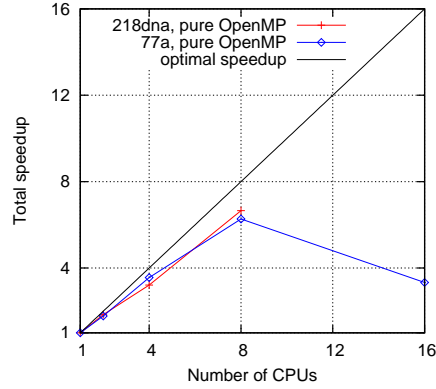


Figure 3. Pure OpenMP speedup for the whole program.

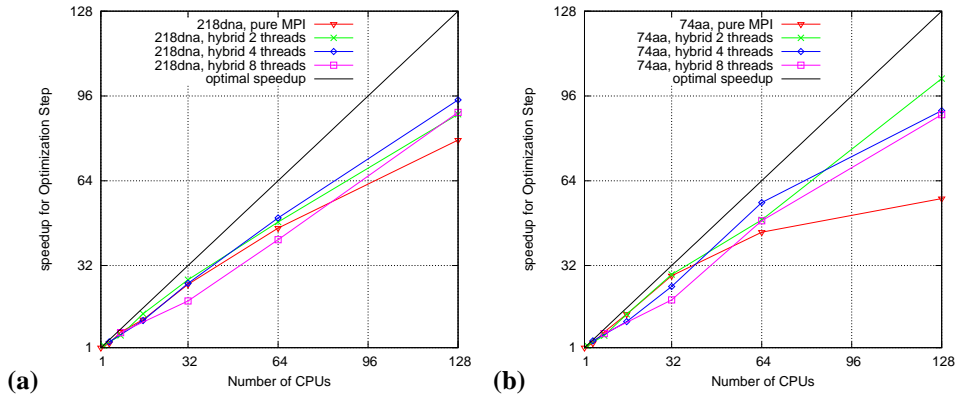


Figure 4. Speedup of optimization step for dataset (a) 218dna and (b) 74aa.

3.3 pIQPNNI vs. Hybrid IQPNNI

We tested the performance of the pIQPNNI against our newly implemented hybrid version with 2, 4 and 8 OpenMP threads per process.

Fig. 4 shows the speedup for the optimization step for 218dna (Fig. 4a) and 74aa (Fig. 4b). For both datasets, the scaling of the pIQPNNI (red line) and the hybrid versions are comparable with a near linear speedup, except that the 8-threads parallelization (pink line) shows a slightly poorer speedup for less than 4 processes, i.e. 32 CPUs. However, this effect disappears with 8 processes or more. This can be explained by the fact that the master process consumes actually only one processor and the other $t - 1$ processors are unused. So a fraction of $\frac{t-1}{pt}$ CPUs are idle during the optimization step. The effect will be large if p is small and otherwise becomes more apparent if p increases.

In addition, we observe that the speedup of the pIQPNNI version on the dataset 74aa is quite poor when running on 128 CPUs. This is, however, due to the fact that we only

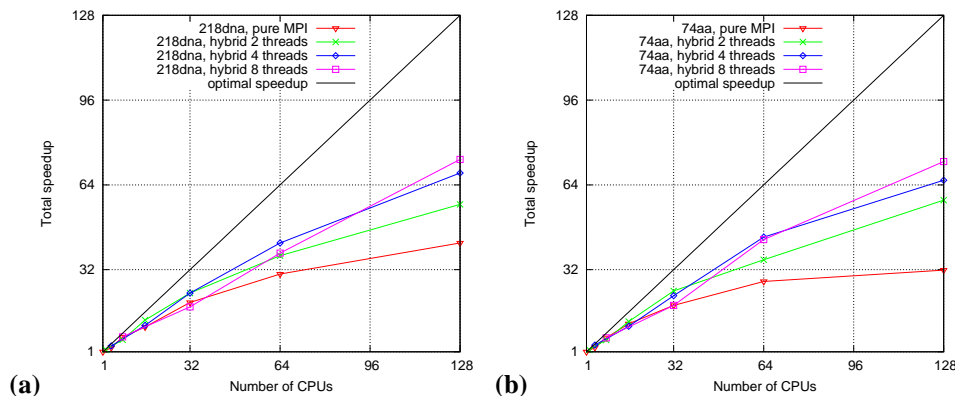


Figure 5. Total speedup for dataset (a) 218dna and (b) 74aa.

#Processes	#Threads	Initial step	Opt. step	Total	Speedup
128	pure MPI	34s	36s	70s	42.1
64	2	20s	32s	52s	56.6
32	4	13s	30s	43s	68.5
16	8	08s	32s	40s	73.6

Table 2. Runtime on dataset 218dna with 128 CPUs.

set the number of iterations to 50 on this dataset. Since each iteration on each worker process consumed roughly the same amount of time, then we would need only 50 workers to finish the whole optimization step. This effect will not occur if we increase the number of iterations to at least the number of MPI processes. Unfortunately, this could not be experimentally tested, due to limitations in computing time.

The total speedup of the pIQPNNI and hybrid-IQPNNI is depicted in Fig. 5. The hybrids with 4- and 8-threads (blue and pink line) scaled best. Whereas the 2-threads hybrid performed a bit worse. The pIQPNNI shows a saturation effect. This shows a tendency that raising the number of threads per process will improve the performance with the growing number of CPUs.

Moreover, the speedup curves are not similar to those for the optimization step since the main part of the initial step is carried out sequentially in the MPI parallelism. As a result, its runtime proportion will be more significant with the increasing number of CPUs. Table 2 displays the running time with 128 CPUs on the dataset 218dna. The initial step of the pIQPNNI consumed 50% of the total time and the time reduction for the hybrid parallelization is mainly due to the shorter time used in the initial step.

4 Conclusions

In this study we gave an overview of different methods to improve the performance of phylogenetic applications. The first is to incorporate efficient heuristics and the second is

to parallelize the current algorithms. To this end, we illustrate an efficient way to parallelize the IQPNNI algorithm.

Furthermore, we studied how to port a pure MPI parallelization of the IQPNNI algorithm into a hybrid MPI/OpenMP parallelism. The loop-level parallelism is applied to the time-consuming for-loops calculating the likelihood of the phylogenies. These loops appeared at a low level and all MPI functions are called outside the parallel OpenMP regions. Hence, this ensures portability even when the MPI libraries are not thread safe.

Analyses on two real datasets showed improved performance of the hybrid parallelization over the pure MPI on a cluster of SMPs. We tested up to 128 CPUs and 8 threads per process. With large numbers of processors, the pure MPI implementation indicated a saturation effect. In contrast, the hybrid version scaled better, especially when increasing the number of threads. This is due to the fact that we make full use of the capabilities of the hybrid architecture.

Acknowledgements

Support from the guest student programme 2005 at FZJ for B.Q.M is greatly appreciated. A.v.H and H.A.S were formerly members of NIC. The use of computing facilities at ZAM/NIC is gratefully acknowledged.

References

1. G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist. Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinformatics*, 20:407–415, 2004.
2. B. Chor and T. Tuller. Maximum likelihood of evolutionary trees is hard. In *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2005)*, volume 3500 of *Lecture Notes in Computer Science*, pages 296–310, New York, USA, May 2005. ACM Press.
3. L. Dagum and R. Menon. OpenMP: An industry-standard API for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5:46–55, 1998.
4. C. Darwin. *On the Origin of Species by Means of Natural Selection*. John Murray, London, 1859.
5. W. H. E. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Syst. Zool.*, 35:224–229, 1986.
6. J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, Massachusetts, 2004.
7. L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Adv. Appl. Math.*, 3:43–49, 1982.
8. O. Gascuel. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Mol. Biol. Evol.*, 14:685–695, 1997.
9. S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, 52:696–704, 2003.
10. B. Q. Minh, L. S. Vinh, A. von Haeseler, and H. A. Schmidt. piQPNNI-parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21(19):3794–3796, 2005.

11. G. J. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek. fastDNAm1: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Comput. Appl. Biosci.*, 10:41–48, 1994.
12. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4:406–425, 1987.
13. H. A. Schmidt, K. Strimmer, M. Vingron, and A. von Haeseler. TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18:502–504, 2002.
14. L. Smith and M. Bull. Development of mixed mode MPI/OpenMP applications. *Scientific Programming*, 9:83–98, 2001.
15. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The Complete Reference - The MPI Core*, volume 1. The MIT Press, Cambridge, Massachusetts, 2 edition, 1998.
16. A. P. Stamatakis, T. Ludwig, and H. Meier. RAXML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21:456–463, 2005.
17. A. P. Stamatakis. An efficient program for phylogenetic inference using simulated annealing. In *Online Proceedings of the 4th IEEE International Workshop on High Performance Computational Biology (HICOMB 2005)*, page 8, Denver, April 2005.
18. K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13:964–969, 1996.
19. O. Trelles. On the parallelisation of bioinformatics applications. *Brief. Bioinform.*, 2:181–194, 2001.
20. L. S. Vinh and A. von Haeseler. IQPNNI: Moving fast through tree space and stopping in time. *Mol. Biol. Evol.*, 21:1565–1571, 2004.
21. M. S. Waterman. *Introduction to Computational Biology*. Chapman and Hall, London, 1995.

