PUZZLE/PPUZZLE: Using Parallel Computing for Maximum Likelihood Phylogenetic Analysis

Heiko A. Schmidt[1], Korbinian Strimmer[2], Arndt von Haeseler[3], and Martin Vingron[1]

[1] Deutsches Krebsforschungszentrum, Theoretische Bioinformatik, Im Neuenheimer Feld 280, D-69120 Heidelberg, Germany
{H.Schmidt|M.Vingron}@DKFZ-Heidelberg.de

[2] GSF - Forschungszentrum MIPS, Max-Planck-Institut für Biochemie, Am Klopferspitz 18a, D-82152 Martinsried/München, Germany
strimmer@biochem.mpg.de

[3] Max-Planck-Institut für Evolutionäre Anthropologie, Evolutionäre Genetik, Inselstr. 22, D-04103 Leipzig, Germany
arndt@eva.mpg.de

Introduction

Phylogenetic reconstruction is used in several fields of genetic sequence analysis. It is applied not only for inferring relationships between species or within gene or protein families. It is also used for classifying organisms, e.g., viruses, and for reconstructing duplication events and their chronological order in genomes. Hence phylogenetic reconstruction plays an important role in understanding development of sequences and genomes. There are several methods for inferring phylogenetic trees from genetic sequence data. Approaches using maximum likelihood (ML) based techniques belong to the most respected. Since Felsenstein (1981) popularized this approach several further methods have been published (e.g., Strimmer & von Haeseler 1996; Olsen *et al.* 1994).

1 The Algorithm

1.1 Computing the ML Value of a Tree

The algorithm introduced by Felsenstein (1981) for computing the maximum likelihood (ML) value for a 'fixed' tree topology models evolution as a reversible Markov process described by a transformation matrix. The ML value describes the probability, that this tree has given rise to the data (i.e. the multiple sequence alignment).
To compute the ML value a root is placed at an arbitrary location in the unrooted tree topology. From this root likelihood values are computed recursively for all subtrees, one for each alignment column. These likelihood values are summed up for the subtrees. In the end all columns are multiplied to a global likelihood value.
All edges, one after another, are now adjusted to maximize this value getting a maximum likelihood value.
For details see Felsenstein (1981) and Felsenstein & Churchill (1996).

1.2 Finding Optimal Trees: Brute Force

Given this algorithm, the best way would be to apply it to the whole set of possible binary tree topologies. After having all ML values one chooses the topology with the best ML value (Fig. 1). Due to the fact that the number of possible binary trees for n sequences is

$$\prod_{k=3}^{n}(2k-5) = \frac{(2n-5)!}{2^{n-3}(n-3)!}$$

this is for most alignments not feasible in practice. For multifurcating trees this number is even worse. Hence heuristic methods are applied.

1.3 The Quartet Puzzling Algorithm

The quartet puzzling algorithm (Strimmer & von Haeseler, 1996) is a quartet based three-step procedure.
In the first step, the maximum likelihood step, all $\binom{N}{4}$ quartet maximum likelihood trees are reconstructed in order to find the most likely relationship of every set of four sequences. Then, in a so-called puzzling step these quartet trees are composed into an overall tree. Since this step highly depends on the order of sequences added to the puzzle tree, many trees with different permutations of the input order are constructed.
In the final step a strict majority rule consensus tree is built from these puzzle trees.

2 Parallelization of the Algorithm

2.1 Approach

Since the construction of the quartet trees as well as the construction of the puzzling trees are independent of each other and the amount of single tasks in each step is sufficiently large, we approached these steps in our parallelization.

2.2 Concepts

In our parallelization we used master/worker concepts. This means one master process coordinates the tasks and sends them to the worker processes, which then do the computation work (Fig. 1).
Besides that we used some kind of *guided self-scheduling* algorithm (GSS, Hagerup 1997).

We adapted the GSS algorithm for the puzzle algorithm, to reduce communication by grouping together tasks, but on the other hand trying to keep all the workers equally busy. For this purpose the scheduling algorithm starts with larger blocks of tasks and ends with a minimum amount per block.

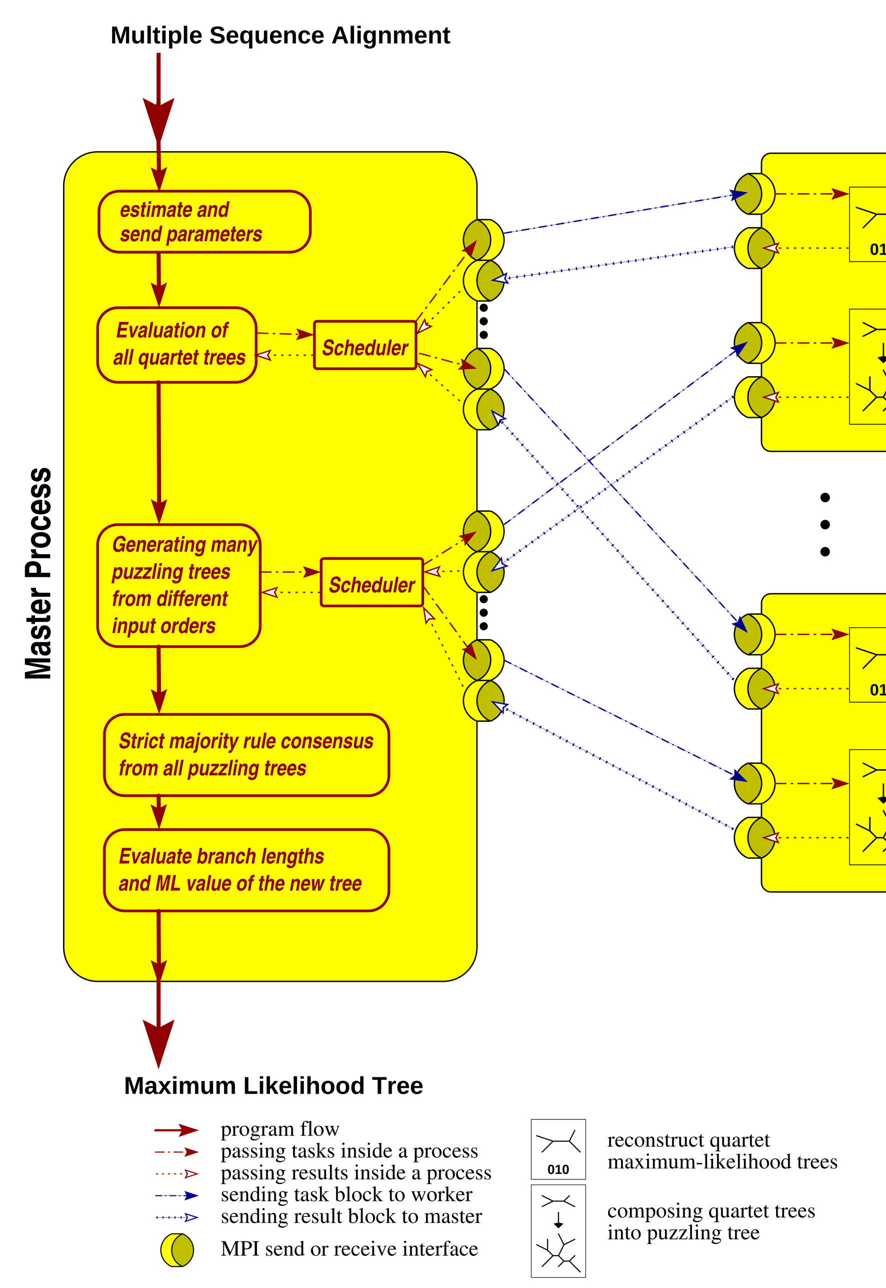

Figure 1: Structure of the parallel implementation `puzzle`. The yellow boxes are separate processes in the parallel run.

2.3 Implementation

For the parallelization we used the Message Passing Interface (MPI, Snir *et al.* 1998; Gropp *et al.* 1998). Since MPI has become the *de facto* standard in parallel computing using message passing, implementations of the MPI libraries are available on almost all parallel platforms from massively parallel computers to clusters of workstations (COW).

3 Results

3.1 On Parallel Computers

In tests on parallel computers like HP S Class and HP V Class our implementation scaled very well with increasing numbers of processors (Fig. 2).

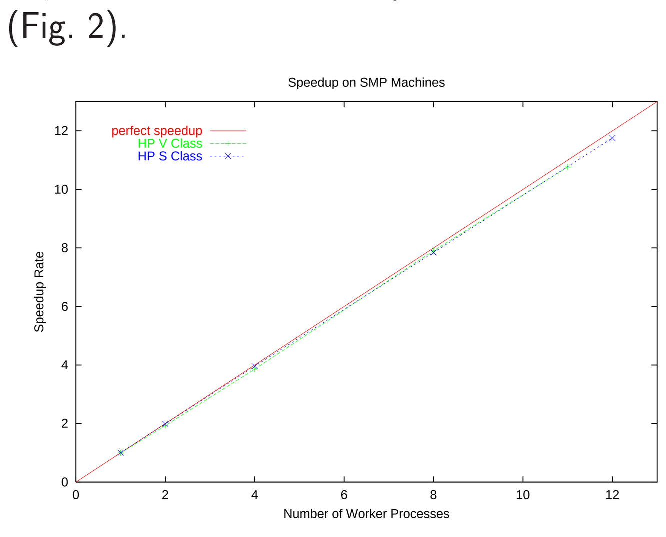

Figure 2: Speedup of `ppuzzle` on SMP parallel computers. The trees were reconstructed from 60 gag sequences from HIV and SIV. The analyses were done with the Dayhoff model and an amount of 10000 puzzling trees.

3.2 On Distributed Systems

Analyses of the Gantt charts and profiles, like Fig. 3, of parallel runs on a COW (cluster of workstations) of 20 SUN workstations show that computational resources remain unused, mainly during the initializing phase and in the consensus step. This is due to the fact that these parts of the program are sequential and the worker processes are not part of the computation.
The COW used here included a wide variety of different SUN computers (Sparc 20, Sparc 4/5, Ultra 1, Ultra 5). Although the machines in the COW vary in their speed a lot (from 65MHz to 300MHz), the Gantt chart shows quite a good performance on this kind of parallel system.

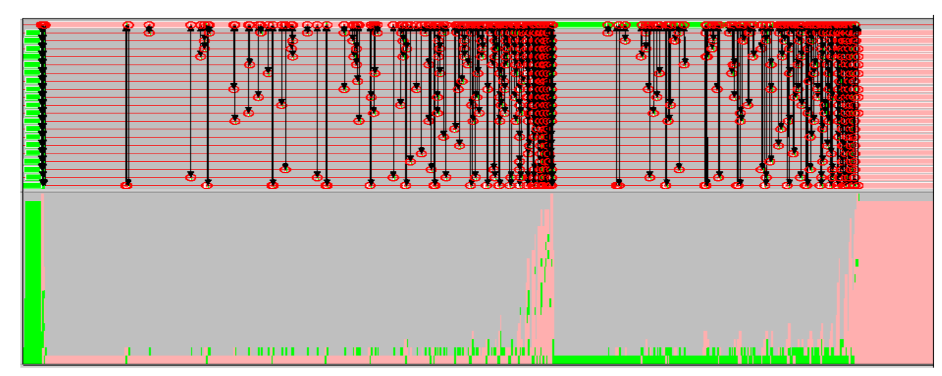

Figure 3: Gantt chart and profile of a parallel run of the `puzzle` program for 32 gag sequences on a COW of 20 SUNs. (a) The Gantt chart shows the communication between the processes in each line. (b) Profile: Grey means computation, the other colors show communication overhead and waiting for communication.

3.3 Conclusion

Our tests have shown that our implementation is useful for parallel as well as for distributed platforms like COWs. We could reach tree reconstruction times of about 5.5 hours for a dataset of 105 nucleosome sequences with 5000 puzzling trees during daytime load of the COW. The same reconstruction with the sequential `puzzle` program took on an SUN Enterprise with 336 MHz processors (about twice as fast as the average machine in the COWs) about 56 hours.
Due to the reduction of the wall clock time needed for computation, this implementation gives us the possibility to reconstruct trees from larger sets of sequences.

4 Future Work

We intend to change the scheduling algorithm to cope better with problems occurring on COWs, like changing load on single Workstations. For this we will apply a dynamically weighted scheduling algorithm to reduce the task block size for machines, if their load increases.

5 The Program

The requirements for running `puzzle/ppuzzle` are:

- an ANSI C compiler (e.g. `gcc`, freely available)
- a Message Passing Interface (MPI) library 1.1 (e.g. MPICH or LAM, freely available)
- the `puzzle/ppuzzle` source code, will be part of the next PUZZLE release (`http://members.tripod.de/korbi/puzzle/`)
- ...and you should have a parallel computing platform available: e.g. a Cluster of Workstations or a parallel machine (SMP or MPP).

Because of using only standard software and libraries, which are also available for free the code should be portable to all kinds of platforms without changes.

References

Felsenstein, J. (1981). Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.* **17**, 368–376.

Felsenstein, J. & Churchill, G. A. (1996). A hidden markov model approach to variation among sites in rate of evolution. *Mol. Biol. Evol.* **13**, 93–104.

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., & Marc Snir, M. (1998). *MPI: The Complete Reference - The MPI Extensions*, volume 2. Cambridge, Massachusetts: The MIT Press, 2 edition.

Hagerup, T. (1997). Allocating independent tasks to parallel processors: An experimental study. *J. Parallel Distrib. Comp.* **47**, 185–197.

Olsen, G. J., Matsuda, H., Hagstrom, R., & Overbeek, R. (1994). fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *CABIOS*, **10**, 41–48.

Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., & Dongarra, J. (1998). *MPI: The Complete Reference - The MPI Core*, volume 1. Cambridge, Massachusetts: The MIT Press, 2 edition.

Strimmer, K. & von Haeseler, A. (1996). Quartet puzzling: A quartet maximum–likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.* **13**, 964–969.

Acknowledgments

This work is supported by *Deutsche Forschungsgemeinschaft* under grants no. VI 160/3-1 and Ha 1628/4-1.