RAxML-OMP: An Efficient Program for Phylogenetic Inference on SMPs*

Alexandros Stamatakis[1], Michael Ott[2], and Thomas Ludwig[3]

[1] Institute of Computer Science, Foundation for Research and Technology-Hellas, P.O. Box 1385, GR-71110 Heraklion, Crete, Greece
[2] Technical University of Munich, Department of Computer Science, Boltzmannstr. 3, D-85748 Garching b. München, Germany
[3] Ruprecht-Karls University, Department of Computer Science, Im Neuenheimer Feld 348, D-69120 Heidelberg, Germany

Abstract. Inference of phylogenetic trees comprising hundreds or even thousands of organisms based on the Maximum Likelihood (ML) method is computationally extremely intensive. In order to accelerate computations we implemented RAxML-OMP, an efficient OpenMP-parallelization for Symmetric Multi-Processing machines (SMPs) based on the sequential program RAxML-V (Randomized Axelerated Maximum Likelihood). RAxML-V is a program for inference of evolutionary trees based upon the ML method and incorporates several advanced search algorithms like fast hill-climbing and simulated annealing. We assess performance of RAxML-OMP on the widely used Intel Xeon, Intel Itanium, and AMD Opteron architectures. RAxML-OMP scales particularly well on the AMD Opteron architecture and achieves even super-linear speedups for large datasets (with a length $\geq$ 5.000 base pairs) due to improved cache-efficiency and data locality. RAxML-OMP is freely available as open source code.

1 Introduction

Phylogenetic (evolutionary) trees are used to represent the evolutionary history of a set of n organisms which are often also called taxa within this context. A multiple alignment of a—in a biological context—suitable small region of their DNA or protein sequences can be used as input for the computation of phylogenetic trees. Note, that a high-quality multiple alignment of the organisms is a necessary prerequisite to conduct a phylogenetic analysis: *The quality of the evolutionary tree can only be as good as the quality of the multiple alignment!* Other computational approaches to phylogenetics also use gene order data [24].

In a computational context phylogenetic trees are usually strictly bifurcating (binary) unrooted trees. The organisms of the alignment are located at the tips (leaves) of such a tree whereas the inner nodes represent extinct common

* This work is funded by a Postdoc-fellowship granted by the German Academic Exchange Service (DAAD) and by the "Competence Network for Technical, Scientific High Performance Computing in Bavaria (KONWIHR)".

V. Malyshkin (Ed.): PaCT 2005, LNCS 3606, pp. 288–302, 2005.

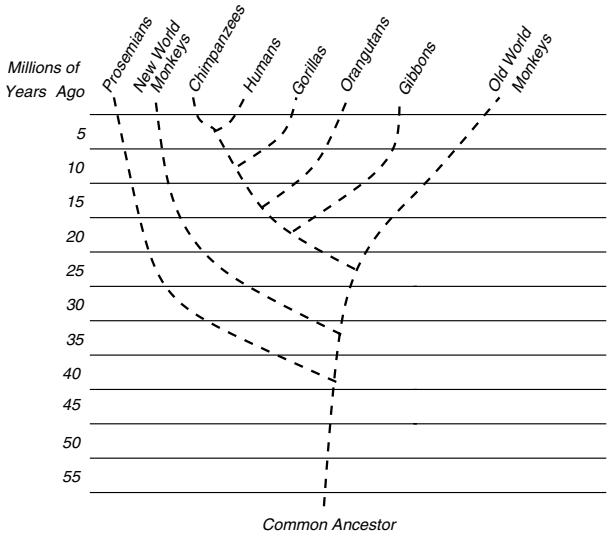

Fig. 1. Phylogenetic tree representing the evolutionary relationship between monkeys and the homo sapiens

ancestors. The branches of the tree represent the time which was required for the mutation of one species into another—new—one. An example for the evolutionary tree of the monkeys and the homo sapiens is provided in Figure 1. Note, that the tree need not be *the* model of evolution. Therefore, approaches using phylogenetic networks are becoming more popular recently [8].

The inference of phylogenies with computational methods has many important applications in medical and biological research, such as e.g. drug discovery and conservation biology. A paper by D. Bader *et al* [1] addresses potential industrial applications of evolutionary tree inference and contains numerous useful references to important biological results obtained via phylogenetic analysis.

Due to the rapid growth of available sequence data over the last years and the constant improvement of multiple alignment methods it has now become feasible to compute very large trees which comprise more than 1.000 organisms. The computation of the tree-of-life containing representatives of all living beings on earth is considered to be one of the *grand challenges* in Bioinformatics.

The most fundamental algorithmic problem computational phylogeny faces consists in the immense amount of potential alternative tree topologies. This number grows exponentially with the number of sequences n, e.g. for $n = 50$ organisms there already exist $2.84 * 10^{76}$ alternative topologies; a number almost as large as the number of atoms in the universe ($\approx 10^{80}$). Thus, given some—biologically meaningful—optimality criterion for evaluating all alternative configurations (topologies) in order to search for the best tree, one can quickly assume that the problem might be NP-hard. In fact, this has already been demonstrated for the general version of the *perfect phylogeny* problem [3] and *maximum parsimony* (MP) [4]. The *maximum likelihood* (ML) criterion [5]

is also believed to be NP-hard, though this could not be demonstrated so far because of the significantly superior mathematical complexity of the model. Due to the large amount of alternative trees, intelligent search space heuristics have to be deployed for ML-based phylogenetic inference. Another important aspect for the design of such heuristics consists in the very high degree of accuracy (difference to the score of the optimal or best-known solution) which is required to obtain reasonable biological as well as topologically closely related results. While an accuracy of 90% is considered to be a "good" value for heuristics designed to solve other NP-hard optimization problems, e.g. the traveling salesman problem, recent results [29] suggest that phylogenetic analyses require an accuracy $\geq 99.99\%$, in particular for large trees. This observation yields the whole field more difficult and challenging.

When comparing the various optimality criteria which have been devised for phylogenetic trees one can observe a *trade-off* between speed and quality. This means that a phylogenetic analysis conducted with an elaborate model such as maximum likelihood requires significantly more computation time but yields trees with superior accuracy than e.g. *neighbor joining* [6] (NJ) or MP [7] [28]. However, due to the higher accuracy it is desirable to infer large and complex trees with maximum likelihood or closely related Bayesian methods.

Within this context it is important to note that the design of maximum likelihood programs is primarily an *algorithmic discipline*, due to the gigantesque number of alternative tree topologies and the high computational cost of the likelihood function. Thus, progress in the field has mainly been attained via algorithmic improvements rather than by brute force allocation of all available computational resources. As an example consider the performance of parallel fastDNAml [21] (state-of-the-art parallel ML program in 2001) and RAxML-V [19] (Randomized Axelerated Maximum Likelihood, one of the fastest sequential ML programs in 2004) on a 1.000-organism alignment: For this large alignment parallel fastDNAml consumed approximately 9.000 accumulated CPU hours on a Linux PC cluster in contrast to less than 20 hours required by RAxML-V on a single Intel Xeon processor. In addition, the likelihood of the tree computed by RAxML-V was *significantly better* than the likelihood score obtained by parallel fastDNAml.

However, as algorithmic research in phylogenetics comes of age and novel powerful algorithms allow for computation of trees which comprise more than 500 sequences, a new category of problems arises. Those problems mainly concern memory shortage, cache efficiency, and a still very large demand for computation time. Thus, the main focus of this paper is on the deployment of the shared memory programming paradigm for the computation of large trees (containing ≥ 500 sequences) based on statistic models of sequence evolution.

The remainder of this paper is organized as follows: Section 2 describes related work in the area of ML phylogeny programs. The following Section 3 briefly describes the main components of the sequential version of RAxML-V. In Section 4 the computation of the likelihood score for a tree is explained and the OpenMP [14] parallelization of RAxML-V is outlined. In Section 5 we report

RAxML-OMP speedups on Xeon, Itanium, and Opteron SMPs. Finally, Section 6 provides a conclusion and briefly addresses current and future issues of work.

2 Related Work

The survey of related work is restrained to statistical phylogeny methods since they have shown to be the most accurate methods currently available. On the one hand there exist "traditional" maximum likelihood methods and a large variety of programs implementing maximum likelihood searches. The recently updated site maintained by J. Felsenstein [17] lists most available programs. On the other hand there exist Bayesian methods which are relatively new compared to maximum likelihood and have experienced great impact, especially through the release of a program called MrBayes [9].

A thorough comparison of popular phylogeny programs using statistical approaches such as fastDNAml [13], MrBayes, PAUP [15], and TREE-PUZZLE [22] on *small* simulated datasets (up to 60 sequences) has been conducted by T.L. Williams *et al* [28]. The most important result of this paper is that MrBayes outperforms all other phylogeny programs in terms of speed and tree quality. However, the results of this survey do not necessarily apply to large real data sets since simulated alignment data has different properties and a significantly stronger phylogenetic signal than real world data (see [20] for a discussion), i.e. typically much more computational effort is required to find a "good" phylogenetic tree for real-world data. Due to these significant differences between real and simulated datasets comparative surveys should include collections of simulated *and* real datasets in order to yield a more complete image of program performance. In fact, there exist some real datasets for which MrBayes fails to converge to acceptable likelihood values within reasonable time [19]. Huelsenbeck *et al* [10] provide an in-depth discussion of potential pitfalls of Bayesian inference.

More recently, Guidon and Gascuel published an interesting paper about their new program PHYML [7], which is very fast and seems to be able to compete with MrBayes. PHYML is a "traditional" maximum likelihood hill-climbing program which seeks to find the optimal tree in respect to the likelihood value. Moreover, the respective performance analysis includes larger simulated datasets of 100 sequences and two well-studied real data sets containing 218 and 500 sequences. Their experiments show that PHYML is extremely fast on real and simulated data. However, the accuracy on real data needs improvement [19]. Moreover, the results show that well-established sequential programs like PAUP* [15], TREE-PUZZLE [22], and fastDNAml [13] are prohibitively slow on datasets containing more than 200 sequences, at least in sequential execution mode.

Vinh *et al* [27] recently published a program called IQPNNI which yields better trees than PHYML on real world data but is significantly slower.

Finally, the current hill-climbing and simulated annealing algorithms implemented in RAxML-V clearly outperform PHYML and IQPNNI on real world data, both in terms of execution time and final tree quality [20].

The main problem which parallel implementations of ML analyses face is that technical development drags behind algorithmic development. This means that programs are parallelized that do not represent the state-of-the-art algorithms any more. Thus, it can be observed that parallel or distributed codes like parallel fastDNAml [21], DPRml [12] (both based on a search algorithm from 1994) or parallel TREE-PUZZLE [18] are just as good as the currently best sequential codes in terms of tree quality. However, they require *significantly* more CPU hours to attain the same results. The above programs have all been parallelized with MPI.

To the best of our knowledge, apart from RAxML-OMP, there exists only one distributed shared-memory implementation of an ML program for NUMA architectures: veryfastDNAml [26] which is based on the TreadMarks library [25]. The veryfastDNAml implementation is also based on the old and slow fastDNAml algorithm from 1994. The technical details of the veryfastDNAml implementation have not been published anywhere such that it is not known if the parallelization is based on loop-level parallelism or a coarse-grained master-worker scheme.

3 RAxML-V

In this Section we provide a brief outline of the basic components and algorithms of RAxML-V, which are required to understand the structure of the parallelization. The program initially computes a starting tree which contains all sequences of the alignment using a fast greedy MP search. The MP search is performed by an appropriately modified version of Joe Felsenstein's dnapars program [17]. One important property of dnapars is that it yields distinct starting trees depending on the input order permutation of the sequences. By randomizing the sequence input order, the program can start the optimization from different points of search space each time it is executed. Therefore, by executing several RAxML-V runs it is more likely to find good trees and avoid local maxima since each run will yield a distinct final tree. Thus, the confidence into the final results obtained by RAxML-V is higher than for strictly *deterministic* programs.

The procedure by which the parsimony score is computed in dnapars is very similar to ML. Thus, the loop-level parallelization of the parsimony component is analogous to that for ML which we describe in more detail in the following Section 4.

After the computation of the parsimony starting tree, the likelihood of the candidate topology is improved by subsequent application of topological alterations. To evaluate and select candidate alternative topologies RAxML-V uses a mechanism called *lazy subtree rearrangements* [19]. This mechanism initially performs a rapid pre-scoring of a comparatively large number of alternative topologies. After the pre-scoring step a few of the best pre-scored topologies are analyzed more thoroughly. The fact, that RAxML-V is currently the fastest *and*

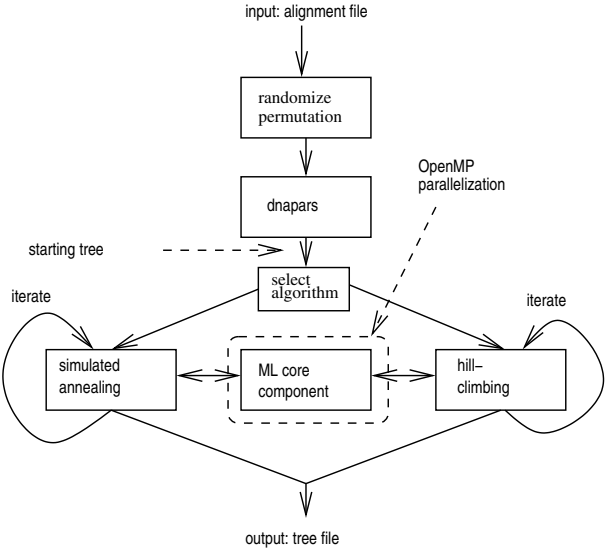

Fig. 2. Basic components of RAxML-V

most accurate program on real alignment data is due to this ability to quickly evaluate (pre-score) a large number of alternative tree topologies. Furthermore, RAxML-V currently implements two basic search procedures which exploit the lazy subtree rearrangement mechanism:

1. A strict hill-climbing procedure which applies lazy subtree rearrangements until the candidate tree can not be improved upon any more [19].
2. A simulated annealing algorithm which is slightly slower than hill-climbing on the one hand but able to escape local maxima on the other hand [20].

Finally, it is important to know that both search algorithms use the same core component to calculate maximum likelihood values, such that the parallelization applies to both search strategies. Figure 2 provides an overview of RAxML-V as described in this Section.

4 Parallelization

The current Section does not intend to provide a detailed introduction to ML for phylogenetic trees. The goal is to give a notion of the complexity and amount of arithmetic operations required to compute the maximum likelihood score for one *single* tree topology. Furthermore, it aims to explain where the intrinsic loop-level parallelism occurs and how it can be exploited.

The seminal paper by Felsenstein [5] which actually introduces the application of ML to phylogenetic trees and the comprehensive and readable chapter by Swofford et al. [23] provide detailed descriptions of the mathematical background and models of nucleotide substitution (see below).

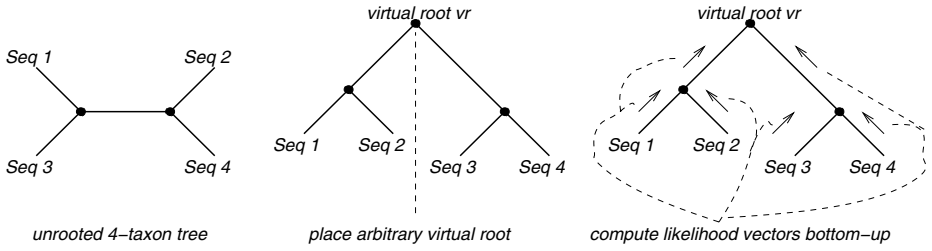

Fig. 3. Computation of the likelihood vectors of a 4-taxon tree

To calculate the likelihood of a tree topology with given branch lengths one requires a probabilistic model of nucleotide substitution $P_{ij}(t)$ which allows for computing the probability P that a nucleotide i (e.g. A) mutates to another nucleotide j (e.g. G) within time t (branch length).

Given the model of nucleotide substitution and an *unrooted* tree topology with fixed branch lengths where the data (the individual sequences of the multiple alignment) is located at the tips, one can proceed with the computation of the likelihood score for that tree. In order to compute the likelihood a *virtual root* (*vr*) has to be placed into an *arbitrary* branch of the unrooted tree in order to calculate/update the individual entries of each *likelihood vector* with length n (alignment length) in the tree bottom-up, i.e. starting at the tips and moving towards *vr*. It is important to note, that the likelihood of the tree is identic irrespectively of where *vr* is placed. After having updated all likelihood vectors the vectors to the right and left of *vr* can be used to compute the overall likelihood value of the tree. The process of rooting and updating the likelihood vectors for a 4-taxon tree is outlined in Figure 3.

To understand how the individual likelihood vectors are updated consider a subtree rooted at node `p` with immediate descendants `r` and `q` and likelihood vectors `l_p`, `l_q`, and `l_r` respectively. When the likelihood vectors `l_q` and `l_r` have been computed the entries of `l_p` can be calculated—in an extremely simplified manner—as outlined by the pseudo-code below and in Figure 4:

```
for(i = 0; i < n; i++)
   l_p[i] = f(g(l_q[i], b_pq), g(l_r[i], b_pr));
```

where `f()` is a simple function, i.e. requires just a few FLOPs, to combine the values of `g(l_q[i], b_pq)` and `g(l_r[i], b_pr)`. The `g()` function however is more complex and computationally intensive since it contains the evaluation of $P_{ij}(t)$. The parameter t corresponds to the branch lengths `b_pq` and `b_pr` respectively. Since entries `l_p[i]` and `l_p[i + 1]` can be computed *independently* this `for`-loop can be parallelized by insertion of an appropriate OpenMP directive to exploit the inherent loop-level parallelism:

```
#pragma omp parallel for private(...)
for(i = 0; i < n; i++)
   l_p[i] = f(g(l_q[i], b_pq), g(l_r[i], b_pr));
```

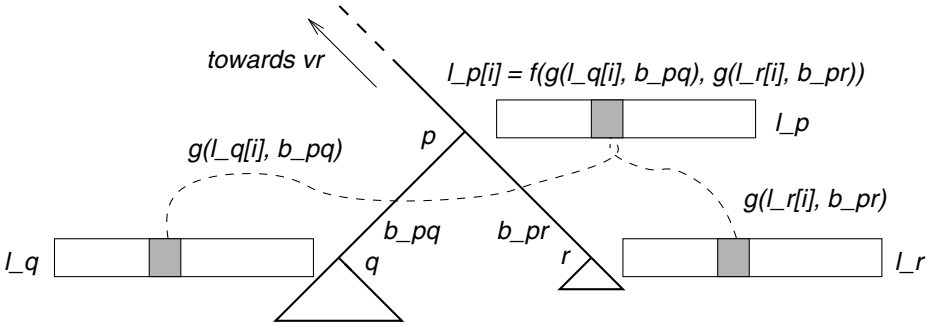

Fig. 4. Updating the likelihood vector of node p at position i

Up to this point it has been described how to compute the likelihood of a tree given some arbitrary branch lengths. However, in order to obtain the *maximum* likelihood value for a given tree topology the length of *all* branches in the tree has to be optimized. Since the likelihood of the tree is not altered by distinct rootings of the tree the virtual root can be subsequently placed into all branches of the tree. Each branch can then be optimized individually to improve the likelihood value of the entire tree. In general—depending on the implementation—this process is continued until no further branch length alteration yields an improved likelihood score. Branch length optimization can be regarded as maximization of a one-parameter function $lh(t)$ where lh is the phylogenetic likelihood function and t the current branch length at vr.

Typically, the three basic operations: computation of the likelihood vectors, optimization of the branch lengths, and computation of the overall likelihood value require $\approx 90\%$ of the complete execution time of every ML implementation. For example 92.72% of total execution time for a typical dataset with 150 sequences in PHYML and 92.89% for the same dataset in RAxML-V. Thus, an acceleration of these functions on a technical level by optimization of the C code, the memory access behavior and consumption, as well as the exploitation of loop-level parallelism can lead to substantial performance improvements. The structure of the loops in the three basic functions is very similar to the abstract pseudocode representation provided above. The main `for`-loops of RAxML have been parallelized in an analogous way.

Memory consumption is becoming a problem for inference of large phylogenetic trees containing more than 1.000 sequences. Table 1 provides some figures for memory requirements of RAxML, PHYML, and MrBayes for large datasets. Note that MrBayes could not handle the 10.000-taxon dataset, even when compiled on a 64-bit architecture. In fact only the sequential RAxML-version could still be executed on a 32-bit processor with this large dataset. The memory requirements of RAxML-V are directly proportional to the alignment size, i.e. $\Theta(n * m)$ where n is the number of sequences and m the number of base pairs (length of the alignment). Figure 5 depicts how the memory allocated by RAxML-OMP is accessed by 2 individual threads, each running on a separate CPU. The situation is particularly favorable because memory accesses are inde-

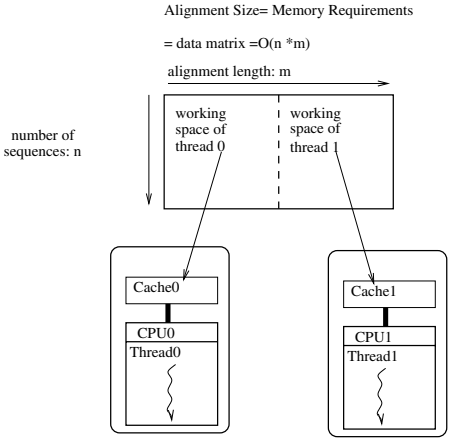

Fig. 5. Memory access scheme of RAxML-OMP

Table 1. Memory consumption of RAxML-III, MrBayes, and PHYML for large data sets

Program	1.000 taxa	10.000 taxa
RAxML-III	200 MB	750 MB
PHYML	900 MB	8.8 GB
MrBayes	1.2 GB	not available

pendent and equally distributed among threads, i.e. *thread 0* in the figure works exclusively on the left half of the data matrix and *thread 1* on the right half.

Thus, we believe that RAxML-OMP provides a viable approach to resolve both memory shortage problems and allow for higher cache efficiency at the same time. It is important to emphasize that memory efficiency is becoming an important issue because evolutionary biology has already entered the whole-genome era. This means that alignments used for phylogenetic analyses will particularly grow in length m which will have typical values of around 10.000 or 20.000 base pairs and do not fit into cache any more. Moreover, inferences of large trees containing more than 1.000 sequences which are now becoming algorithmically feasible also require *long* alignments in terms of m to produce a reliable phylogenetic signal [2]. Finally, RAxML-OMP can serve as a basis for hybrid MPI/OpenMP implementations on constellations of PC-clusters which are widely available nowadays.

5 Results

Initially, we provide a brief description of the test platforms and datasets used in this study. Thereafter, we provide measured speedup values for various plat-

form/dataset combinations and compare the performance on the different SMP architectures.

Test data, platforms and experimental setup: For measuring the efficiency of RAxML-OMP we executed the program on three common SMP architectures: a dual-processor Intel Xeon 2.4GHz with 4 Gbyte of main memory, a quad-processor Intel Itanium2 1.3GHz with 8 Gbyte of main memory, and a quad-processor AMD Opteron 850 2.4 GHz with 8 Gbyte of memory. We used several real world alignment data sets containing 150, 218, 500, and 1.000 taxa (150_SC, 218_RDPII, 500_ARB, 1000_ARB). In addition we generated 3 simulated alignment data sets with 300 sequences (sim300_1000, sim300_5000, sim300_10000) to evaluate the effect of increasing alignment length on program performance. For the sake of completeness we indicate the alignment lengths (# of base pairs) of all datasets we used in Table 2.

Table 2. Alignment lengths

Dataset	# bp	Dataset	# bp
150_SC	1.130	sim300_1000	1.000
218_RDPII	1.847	sim300_5000	5.000
500_ARB	2.751	sim300_10000	10.000
1000_ARB	3.364		

We compiled RAxML-OMP with the native Intel compiler `icc -O3` and the respective OpenMP flags for the Itanium and Xeon architectures. For the Opteron we used the PGI [16] compiler `pgcc -O3`. In order to measure execution times and calculate speedup values we executed RAxML-OMP with 1 and 2 threads on the Xeon, and 1,2, and 4 threads on the Itanium and Opteron processors respectively. We executed 3 runs for each dataset/architecture/number-of-threads combination and report average values. To be able to reproduce comparable results we used a fixed parsimony starting tree. This was achieved by using a standard input sequence permutation order instead of a randomized one. Moreover, we also measured the execution times of the parsimony and maximum likelihood components separately to analyze the efficiency for each part. A separate analysis is of particular interest since the parsimony component exclusively performs integer operations while maximum likelihood performs mainly a large number of floating point operations. Moreover, ML requires approximately 5 times higher per-loop execution times than MP, e.g. for the 150_SC dataset 18.3 μs for one complete iteration of a parsimony `for`-loop and 97.2 μs for an ML `for`-loop[1]

Experimental results: A complete analytical table containing the execution times of *all* experiments conducted within the framework of this study is available

[1] These times have been measured on an Intel Centrino.

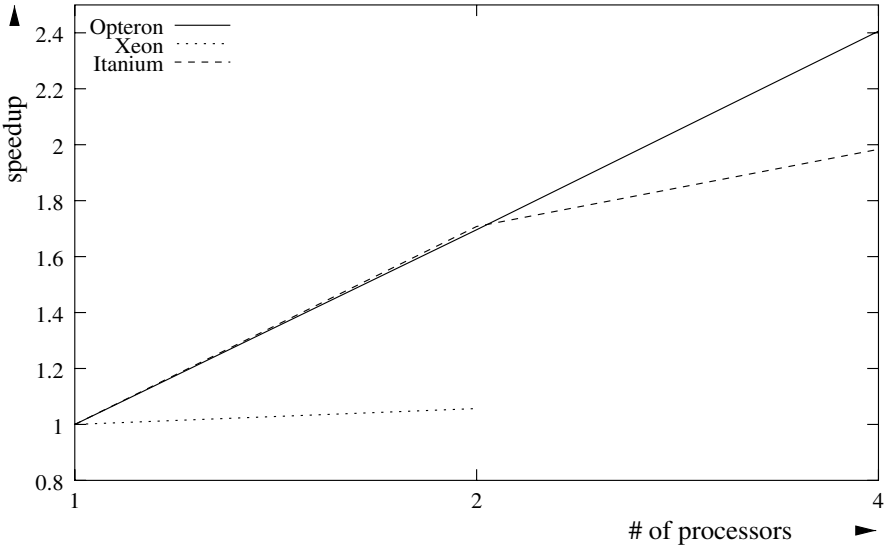

Fig. 6. Speedup on Xeon, Opteron, and Itanium for 218_RDPII

at: wwwbode.in.tum.de/~ottmi/results_jan_05.html. Therefore, we present some representative examples of RAxML-OMP performance.

Figure 6 indicates the speedup values for the relatively small—in terms of alignment length m (see Table 2 and Figure 5)—dataset 218_RDPII on Xeon,

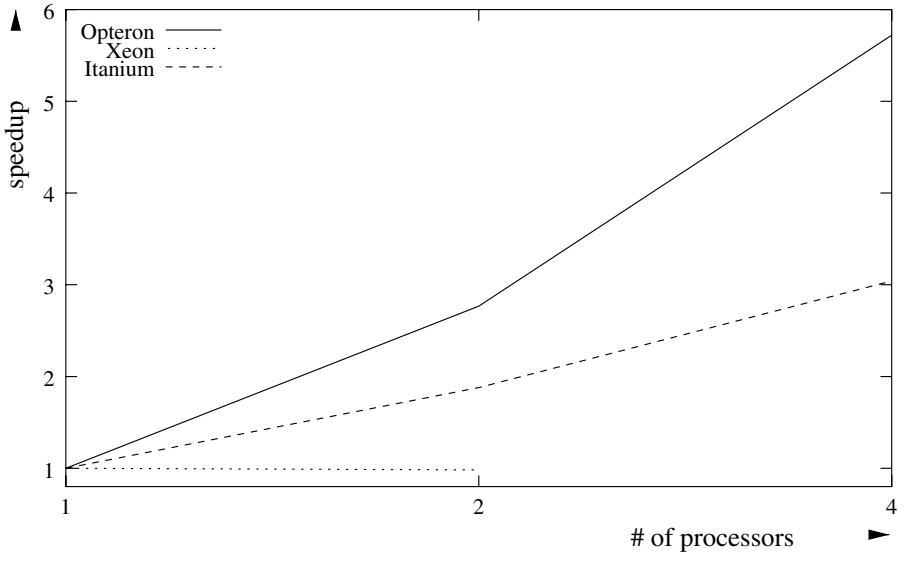

Fig. 7. Speedup on Xeon, Opteron, and Itanium for sim300_10000

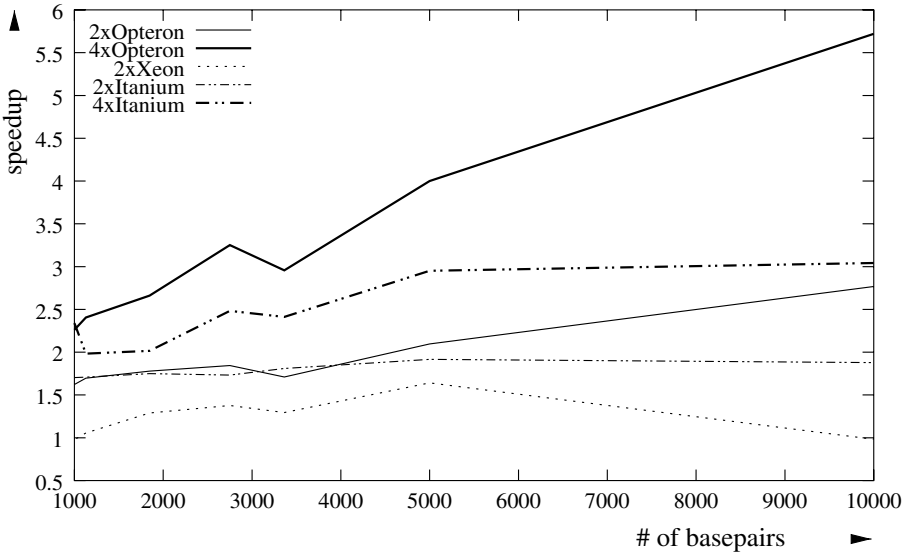

Fig. 8. Speedup over alignment length (number of base pairs) per processor type and number of CPUs

Opteron, and Itanium architectures. The generally better scalability of the Opteron processor is most probably due to the HTT (Hyper Transport Technology [11]) memory access architecture which suits the program structure of RAxML-OMP. However, this issue requires further investigation. On the other hand, due to an unfavorable memory access design the Xeon processor yields only marginal speedups.

Figure 7 provides the speedup values for the sim300_10000 dataset. Though, comparable in size in respect to the number of taxa with 218_RDPII, the length of this alignment and consequently the length of the parallelized `for`-loops is significantly longer: 10.000 nucleotides = 10.000 iterations (also called base pairs). Note, that the speedup on the AMD Opteron on 2 *and* 4 CPUs is clearly *super-linear* (≈ 2.8 and ≈ 5.6 respectively). This is due to the improved cache efficiency and data locality inherent to RAxML-OMP in conjunction with AMD's HTT and a "long" alignment. In order to demonstrate the impact of alignment length on speedup values in Figure 8 we plot the speedup over the number of base pairs—for all datasets used in this study—per processor type and number of CPUs. The general tendency is that the parallel efficiency increases with alignment length due to the aforementioned reasons on the Opteron. Note, that for an AMD Opteron equipped with significantly less main memory (512MB) and a smaller cache the speedups became already super-linear at significantly lower alignment lengths ($\geq$ 2.000 base pairs). Another point worth mentioning is that a "large" number of taxa n (see Figure 5) in the alignment has a negative effect on speedup-values since the amount of allocated memory increases significantly. This explains the buckling which can be observed at $\approx$ 3.500 base pairs. This value corresponds to the large—in terms of taxa—1000_ARB dataset. In Fig-

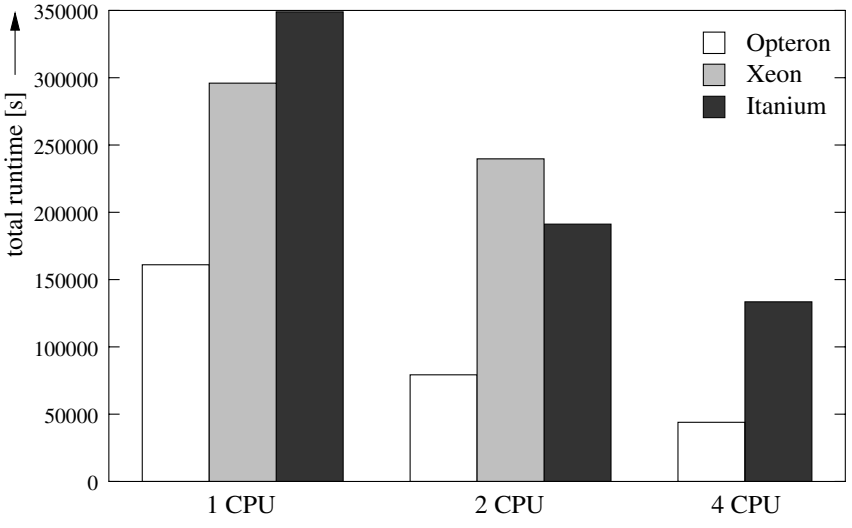

Fig. 9. Accumulated average execution times over all datasets per processor type and number of CPUs

ure 9 we present the accumulated average runtime over all datasets per number of CPUs for the Itanium, Opteron, and Xeon architectures. In all cases RAxML-OMP is at least $\approx 50\%$ faster on Opteron than on the Xeon and Itanium.

Finally, as expected the parallel efficiency of the ML component was significantly better than for MP due to the aforementioned reasons (please refer to the results web-site for exact figures).

6 Conclusion, Availability and Future Work

We have presented an efficient OpenMP parallelization of RAxML-V which scales particularly well on the AMD Opteron SMP architecture. Due to improved cache efficiency and data locality RAxML-OMP yields clearly superlinear speedups for long (in terms of base pairs) datasets on 2-way and 4-way Opteron nodes. Moreover, the current implementation allows for inference of large 1.000-taxon trees on a single Opteron node in less than 6 hours. The program is freely available for download as open source code at www.ics.forth.gr/~stamatak. Currently, we are working on an OpenMP-version of PHYML which faces more serious memory problems than RAxML.

Since scalability of parallel programs which exploit fine-grained loop-level parallelism is limited, future work will mainly cover the implementation of a mixed MPI/OpenMP parallelization of RAxML for hybrid supercomputer architectures. Moreover, the architectural causes for the relatively bad performance of RAxML-OMP on both Intel architectures in comparison to the efficiency on the Opteron need to be further investigated.

References

1. Bader, D.A., Moret, B.M.E., Vawter, L.: Industrial Applications of High-Performance Computing for Phylogeny Reconstruction. Proceedings of SPIE IT-Com: Commercial Applications for High-Performance Computing **4528** (2001) 159–168
2. Bininda-Emonds, O.R.P., Brady, S.G., Sanderson, M.J., Kim, J.: Scaling of accuracy in extremely large phylogenetic trees. Proceedings of Pacific Symposium on Biocomputing (2000) 547–558
3. Bodlaender, H.L., Fellows, M.R., Hallett, M.T., Wareham, T., Warnow, T.: The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs. Theor. Comp. Sci. **244** (2000) 167–188
4. Day, W.E., Johnson, D.S., Sankoff, D.: The computational Complexity of inferring rooted phylogenies by parsimony. Math. Bios. **81** (1986) 33–42
5. Felsenstein, J.: Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach. J. Mol. Evol. **17** (1981) 368–376
6. Gascuel, O.: BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. Mol. Biol. Evol. **14** (1997) 685–695
7. Guindon, S., Gascuel, O.: A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood. Syst. Biol. **52(5)** (2003) 696–704
8. Gusfield, D., Eddhu, S., Langley, C.: Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination. Proceedings of 2nd IEEE Computer Society Bioinformatics Conference (2003) 363–371
9. Huelsenbeck, J.P., Ronquist, F., Nielsen, R., Bollback, J.P.: Bayesian Inference and its Impact on Evolutionary Biology. Science **294** (2001) 2310–2314
10. Huelsenbeck, J.P., Larget, B., Miller, R.E., Ronquist, F.: Potential Applications and Pitfalls of Bayesian Inference of Phylogeny. Syst. Biol. **51(5)** (2002) 673–688
11. Hyper Transport Technology: WWW.HYPERTRANSPORT.ORG.
12. Keane, T.M., Naughton, T.J., Travers, S.A.A., McInerney, J.O., McCormack, G.P.: DPRml: Distributed Phylogeny Reconstruction by Maximum Likelihood. Bioinformatics **21(7)** (2005) 969–974
13. Olsen, G., Matsuda, H., Hagstrom, R., Overbeek, R.: fastdnaml: A Tool for Construction of Phylogenetic Trees of DNA Sequences using Maximum Likelihood. Comput. Appl. Biosci.**10** (1994) 41–48
14. OpenMP: WWW.OPENMP.ORG/DRUPAL.
15. PAUP project site: PAUP.CSIT.FSU.EDU.
16. Portland Group High-Performance Compilers and Tools: WWW.PGROUP.COM.
17. PHYLIP downlaod site and list of phylogeny software: EVOLUTION.GENETICS.WASHINGTON.EDU.
18. Schmidt, H.A., Strimmer, K., Vingron, M., Haeseler, A.v.: TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. Bioinformatics **18** (2002) 502–504
19. Stamatakis, A., Ludwig, T., Meier, H.: RAxML-III: A Fast Program for Maximum Likelihood-based Inference of Large Phylogenetic Trees. Bioinformatics **21(4)** (2005) 456–463
20. Stamatakis, A.: An Efficient Program for phylogenetic Inference Using Simulated Annealing. Proceedings of 19th International Parallel and Distributed Processing Symposium (2005) to be published
21. Stewart, C., Hart, D., Berry, D., Olsen, G., Wernert, E., Fischer, W.: Parallel Implementation and Performance of fastdnaml - a Program for Maximum Likelihood Phylogenetic Inference. Proceedings of SC2001 (2001)

22. Strimmer, K., Haeseler, A.v.: Quartet Puzzling: A Maximum-Likelihood Method for Reconstructing Tree Topologies. Mol. Biol. Evol. **13** (1996) 964–969
23. Swofford, D.L., Olsen, G.J., Wadell, P.J., Hillis, D.M.: Phylogenetic Inference. Hillis, D.M., Moritz, C., Mabel, B.K., (editors) Molecular Systematics, **Chapter 11** (1996) Sinauer Associates, Sunderland, MA
24. Tang, J., Moret, B.M.E., Cui, L., dePamphilis, C.W.: Phylogenetic reconstruction from arbitrary gene-order data. Proc. 4th IEEE Conf. on Bioinformatics and Bioengineering BIBE'04 (2004) 592–599
25. The TreadMarks Distributed Shared Memory (DSM) System: www.cs.rice.edu/~willy/TreadMarks/overview.html
26. VeryFastDNAml: www-bioweb.pasteur.fr/seqanal/soft-pasteur.html#veryfastdnaml
27. Vinh L.S., Haeseler, A.v.: IQPNNI: Moving fast through tree space and stopping in time. Mol. Biol. Evol. **21(8)** (2004) 1565–1571
28. Williams, T.L., Moret, B.M.E.: An Investigation of Phylogenetic Likelihood Methods. Proceedings of 3rd IEEE Symposium on Bioinformatics and Bioengineering (2003)
29. Williams, T.L., Berger-Wolf, B.M., Roshan, U., Warnow, T.: The relationship between maximum parsimony scores and phylogenetic tree topologies. Tech. Report, **TR-CS-2004-04** (2004) Department of Computer Science, The University of New Mexico