Evolutionary Placement of Short Sequence Reads on Multi-Core Architectures

Alexandros Stamatakis, Zsolt Komornik, Simon A. Berger
Dept. of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching b. München, Germany
Email: stamatak@in.tum.de, z.komornik@gmail.com, simon.berger@in.tum.de

***Abstract*—The application of high performance computing methods in bioinformatics becomes increasingly important because of the masses of data generated by novel short-read DNA sequencers. One important application of such short reads, is the analysis of microbial communities where the anonymous short reads need to be identified by sequence comparison to a set of reference sequences. This identification is required to analyze the microbial composition and biological diversity of the sample. We briefly introduce a new algorithm for evolutionary (phylogenetic) placement of short reads under the Maximum Likelihood criterion and implement it in RAxML. While this algorithm is significantly more accurate than plain pair-wise sequence comparison it can become highly compute-intensive when a typical number of 100,000 reads and more need to be placed into an existing phylogenetic tree. Therefore, we deploy multi-grain parallelism to improve parallel efficiency of this algorithm on 16-core and 32-core architectures. Via this multi-grain approach, we achieve parallel execution time improvements of 25% and super-linear speedups on 16 cores, as well as near-linear speedups and improvements exceeding 50% on 32-cores on two large real-world microbial datasets. Evolutionary placement of 100,000 reads into a tree with more than 4,000 taxa now only requires less than 2 hours of execution time on 32 cores.**

***Index Terms*—Evolutionary Placements; Short Reads; Maximum Likelihood; Pthreads; Multi-Grain Parallelism; RAxML;**

I. Introduction

The application of high performance computing methods to bioinformatics applications faces two challenges: the *many-core revolution* and the *biological data flood* that is driven by novel wet-lab sequencing techniques. These new sequencing techniques can generate more than 100,000 short read DNA sequences with a length ranging between 30 to 450 nucleotides in a single run. Besides full-genome assembly (see, e.g., [1]), another important application is the *in-vivo* sampling of microbial communities, e.g., in the human gut [2] or on human hands [3]. Given the short reads, the first step in the analysis process of microbial communities consists of identifying the anonymous reads by determining to which of the known (identified) organism sequences they are most closely related to. This is usually done by pair-wise sequence comparisons, i.e., each of the short reads is compared against all reference sequences via a respective BLAST [4] search or, e.g., pairwise sequence alignment [5]. An assignment of anonymous reads to known organisms allows for comparison of microbial communities from different samples [2], for instance, samples from different habitats, and allows to determine their respective phylogenetic diversity [6].

Here we present the parallelization of a novel phylogenetic identification (placement) algorithm for anonymous reads (henceforth denoted as Query Sequences (QS)), using a set of full length Reference Sequences (RS). While, as already mentioned, the most straight-forward approach for identifying the QS consists of using similarity methods such as BLAST and pair-wise alignment, this standard approach faces some limitations: *Firstly,* the QS might be too short to yield sufficient signal for a correct placement. *Secondly,* this approach can yield misleading assignments of QS to RS if the RS sample does not contain sequences that are sufficiently closely related to the QS. Thus, the most important argument against a BLAST-based approach is, that it will not unravel potential problems in the sampling of the RS (see [7]). For instance, given two RS A and B, a QS Q may be assigned to A by BLAST, while it is in reality most closely related to a RS C that was not included in the reference sequence set. Thus, many studies of microbial communities deploy phylogenetic (evolutionary) trees for QS placement, despite the significantly higher computational cost. If a QS falls into an inner branch of the phylogenetic reference tree comprising the RS, i.e., it is not located near a leave of the tree that represents a currently living species, this indicates that the sampling of the RS is insufficient to correctly place and capture the diversity of the QS. This also provides vital information about the parts of the reference tree whose sampling needs to be improved. In fact, this information can be used to guide efforts for full-length DNA sequencing in order to improve the taxon sampling of the organisms under study and fill in the missing parts in our view of evolutionary history. Note that, the accuracy of phylogeny reconstruction increases with taxon sampling [8].

To date, phylogeny-based placement of reads (environmental samples) is conducted as follows: the QS are aligned with respect to a Reference Alignment (RA) for the Reference Sequences (RS), and then inserted into the Reference Tree (RT), either via a complete *de novo* tree reconstruction, a constrained tree search (using the RT as a constraint topology), or some fast and approximate QS addition algorithm as implemented for instance in ARB [9]. This approach yields a fully resolved bifurcating tree that can comprise more than 10,000 sequences from a single gene, typically 16S or 18S rRNA [2], [3]. The reconstruction of large single-gene trees is hard because of a weak signal, i.e., reconstruction accuracy decreases for trees with many short sequences [10], [11], and even more so for short-read QS. Hence, we advocate a different approach that only computes the optimal insertion branch, i.e., the optimal placement, for every QS in the RT with respect to its Maximum Likelihood (ML [12]) score. This approach is faster than a de novo tree reconstruction and at the same time significantly more accurate than BLAST, in particular when the taxon sampling of the reference tree is sparse or inadequate with respect to the sample. Moreover, using appropriate additional heuristics, the placement algorithm is as fast as BLAST. The complexity of both alternative approaches is $O(qr)$, where q is the number of QS and r the number of RS. Results regarding the accuracy assessment of our novel algorithm including a detailed comparison to BLAST, impact of QS length on accuracy etc. are outside the scope of this paper. While we briefly sketch the results here, a draft manuscript (currently under review) providing a detailed experimental setup, accuracy assessment, and run-time comparison is available also available [13].

The evolutionary placement algorithm, including the parallelization described here, is already available in the latest open source code release of RAxML (Randomized Axelerated Maximum Likelihood [14], version 7.2.6, February 2010, http://wwwkramer.in.tum.de/exelixis/software.html) which is a widely used program for phylogenetic inference (over 6,700 downloads from distinct IPs to date, approximately 700 citations of the three main papers according to Google Scholar).

In this paper we focus on deploying multi-grain parallelism on multi-core architectures to accommodate the computational requirements of the placement algorithm by using three real-world 16S rRNA datasets with 4,412 [3], 16,307 [2], and 100,627 QS (unpublished) that are classified into a single-gene bacterial reference tree with 4,874 species and a length of 1,287 base pairs. We show that fine-grain parallelism in the phylogenetic likelihood function [15], [16] does not scale well for the typical input datasets (many taxa, few sites) of our algorithm and propose a multi-grain approach that dynamically switches from a fine-grain parallelization scheme to a more coarse-grain scheme during program execution. This multi-grain approach yields near-linear and even super-linear speedups as well as parallel run time improvements of more than 50% on two general purpose multi-core systems. Using our evolutionary placement method in combination with multi-grain parallelism, it is possible to conduct phylogenetic classification runs on large numbers of QS (100,627) in under 2 hours on a 32 core system.

The remainder of this paper is organized as follows: In Section II we cover related work on parallel phylogenetic inference. In Sections III and IV we describe the ML method and provide an abstract description of the novel evolutionary placement algorithm. In Section V we summarize the results of the accuracy assessment for the sequential algorithm and the compare it to plain sequence similarity-based approaches. Thereafter, we outline the multi-grain parallelization strategy (Section VI). The experimental setup and results are covered in Section VII and we conclude in Section VIII.

II. Related Work

This paper introduces a novel algorithm for phylogenetic placement. Thus, we are not aware of any directly related work that covers the problems associated with the parallelization of such an algorithm.

Own previous work mainly focused on exploiting loop-level parallelism for the ML function (see next Section) in RAxML on shared memory and multi-core systems [16], the IBM Cell [17], and the IBM BlueGene/L [18]. While the fine-grain loop-level parallelism of the ML function scales well on very large (in terms of number of base pairs) phylogenomic alignments up to thousands [18] of processors, scalability is limited for large single-gene alignments. Minh *et al* [19] implemented a hybrid OpenMP/MPI version of IQPNNI that exploits loop-level and coarse-grain parallelism. The ML function in GARLI [20] has also been parallelized with OpenMP.

III. The Maximum Likelihood Model

The input of a phylogenetic analysis consists of a multiple sequence alignment with n sequences (taxa/tips) and m alignment columns (also called alignment sites). The output is an unrooted binary tree; the n taxa are located at the leaves of the tree and the inner nodes represent common (extinct) ancestors. The branch lengths represent the relative time of evolution between nodes in the tree. In order to compute the likelihood on a fixed tree topology, one requires several Maximum Likelihood (ML) model parameters: the instantaneous nucleotide substitution matrix Q which contains the transition probabilities for time dt between nucleotide (4x4 matrix) characters and the prior probabilities of observing the nucleotides: $\pi_A, \pi_C, \pi_G, \pi_T$. Finally, one also requires the $2n-3$ branch lengths in the unrooted tree topology.

Given these parameters, to calculate the likelihood on a fixed tree, one needs to compute the entries for all ancestral probability vectors at inner nodes, that contain the probabilities $P(A), P(C), P(G), P(T)$, of observing an `A, C, G` or `T` at a site c, where $c = 1...m$, of the ancestral probability profile of this inner node. The probability vectors are computed bottom-up from the tips towards a virtual root that can be placed into any branch of the tree via the Felsenstein pruning algorithm [12]. If the substitution model is time-reversible, the likelihood score will be the same for any placement of the virtual root and thereby greatly simplifies the computation of the likelihood score. However, there have been proposals, for relatively easy to compute non-reversible substitution models [21].

Given a parent (ancestral) node k, and two child nodes i and j (with respect to the virtual root), their probability vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$, the respective branch lengths leading to the children b_i and b_j, and the transition probability matrices $P(b_i), P(b_j)$, the probability of observing an `A` at position c of the ancestral (parent) vector $\vec{L}_A^{(k)}(c)$ is computed as follows:

$$\vec{L}_A^{(k)}(c) = \Big(\sum_{S=A}^{T} P_{AS}(b_i)\vec{L}_S^{(i)}(c)\Big)\Big(\sum_{S=A}^{T} P_{AS}(b_j)\vec{L}_S^{(j)}(c)\Big) \quad (1)$$

The transition probability matrix $P(b)$ for a given branch length is obtained from Q via $P(b) = e^{Qb}$, i.e., via an Eigenvalue/Eigenvector decomposition. Once the two probability vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$ to the left and right of the virtual root (vr) have been computed, the likelihood score $l(c)$ for an alignment column c can be calculated as follows, given the branch length b_{vr} between nodes i and j:

$$l(c) = \sum_{R=A}^{T} \left(\pi_R \vec{L}_R^{(i)}(c) \sum_{S=A}^{T} P_{RS}(b_{vr}) \vec{L}_S^{(j)}(c)\right) \qquad (2)$$

The overall score is then computed by summing over the per-column log likelihood scores:

$$LnL = \sum_{c=1}^{m} log(l(c)) \qquad (3)$$

An important property of the likelihood function is, that sites evolve independently, i.e., all entries c of a vector $\vec{L}$ can be computed simultaneously, which is the main source of fine-grain parallelism [22], [23].

In order to compute the *Maximum* Likelihood value on a fixed tree all individual branch lengths and the Q matrix must also be optimized via ML. For Q the most common approach in state-of-the-art ML implementations consists of using Brent's algorithm [24]. To evaluate changes in Q the entire tree needs to be re-traversed, i.e., a *full* tree traversal needs to be conduct to re-compute the likelihood. For the optimization of branch lengths the Newton-Raphson method is commonly used (see, e.g., [25]). The branches of the tree are repeatedly visited and optimized one by one until the induced branch length change is smaller than a pre-defined ϵ. The Newton-Raphson method only operates on a single pair of likelihood vectors $\vec{L}^{(i)}, \vec{L}^{(j)}$ that are located at either end of the branch to be optimized. Note that in the parallel version of RAxML, a reduction operation to compute the overall score (over all columns) for the first and second derivative is required at *every* iteration of the Newton-Raphson procedure. Hence, we need to synchronize threads between all iterations of the Newton-Raphson method for optimizing a single branch in a fine-grain parallelization of the likelihood function [26].

An important issue is the assignment of memory space for ancestral probability vectors. There exist two alternative approaches: a separate vector can be assigned to each of the three outgoing branches of an inner node, or only one vector can be assigned to each inner node. In the latter case, the probability vectors always maintain a rooted view of the tree, i.e., they are oriented towards the current virtual root of the tree. In the case that the virtual root is then relocated to a different branch (for instance to optimize the respective branch length), a certain number of vectors, for which the orientation to the virtual root has changed need to be re-computed. If the tree is traversed in an intelligent way for branch length optimization, the number of probability vectors that will need to be re-computed can be kept to a minimum. An example for this data organization as used in RAxML is provided in Figure 1.

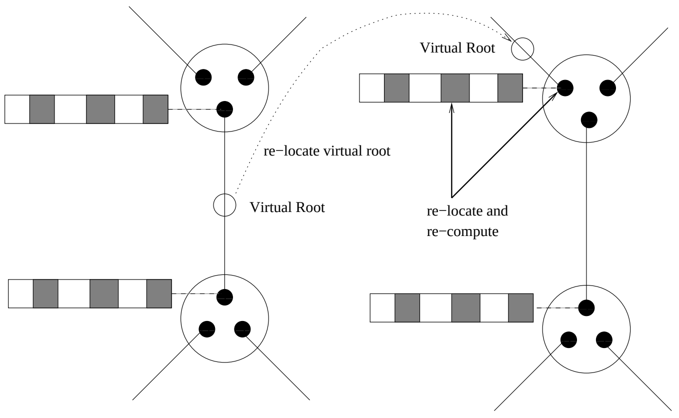

Fig. 1. Rooted organization of the probability vectors at inner nodes.

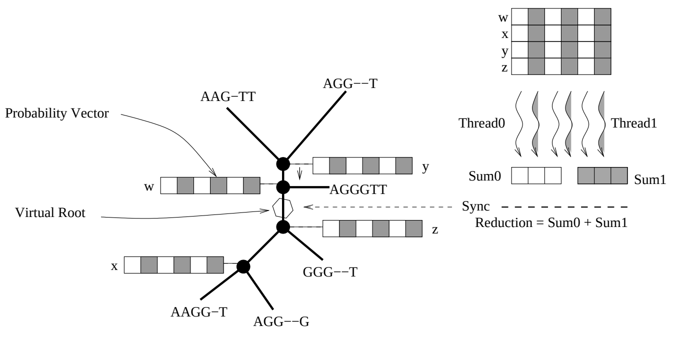

Fig. 2. Parallel fine-grain computation of the likelihood score on a given tree with given branch lengths.

The main bulk of all of the above likelihood computations consists of `for`-loops over the length m of the multiple sequence input alignment which typically require 95% of total execution time in all state-of-the-art likelihood-based software packages. The general parallelization scheme of the Pthreads-based version of RAxML is outlined in Figure 2 for a *full* tree traversal. We use a cyclic distribution of the m alignment columns to allow for better load-balance in phylogenomic datasets that can contain DNA as well as AA (protein); protein data requires more FLOPS per alignment site.

The master thread steers the tree search and orchestrates the optimization of the branch lengths and model parameters. For the model parameter optimization phase, which is relevant to the algorithm presented here, the tree needs to be fully traversed to optimize Q. Hence, the master thread generates a full tree traversal list, that remains fixed during the model parameter optimization process because the tree topology is not being changed. When a model parameter has been changed, every worker thread can independently update its fraction of the vector entries for the full tree traversal and the threads only need to be synchronized when the virtual root is reached (see Figure 2). Therefore, every thread can conduct a relatively large fraction of independent work per alignment column during the model parameter optimization phase. Since the branch length optimization process requires several iterations at every individual branch, the synchronization to computation ratio for branch length optimization is less favorable (see [26]). While this fine-grain parallelization approach is highly efficient and scales up to 1,024 CPUs for large phylogenomic analyses [18], [27] with $m \gg 1,000$, scalability for single-gene analyses with $m \approx 1,000$ is limited. In Section VI we provide solutions to this single-gene scalability problem for the evolutionary placement algorithm. Note however, that similar solutions using a scatter-gather operation on probability vectors can potentially also be applied to improve parallel efficiency and scalability for standard single-gene ML searches.

IV. Evolutionary Placement Algorithm

As already mentioned the input for our algorithm consists of a reference tree comprising r full-length RS (Reference Sequences), and a comprehensive alignment that contains the r RS as well as the q QS (Query Sequences). The QS can, e.g., be aligned to the reference alignment with ARB [9] or NAST [28]. The sequence alignment programs MAFFT [29] and MUSCLE [30] can also be deployed to extend and merge alignments. One key assumption is that the reference tree is biologically well-established or has been obtained via a preceding thorough ML analysis of the RS. Initially, the algorithm will read in the reference tree and reference alignment and mark all sequences of the alignment that are *not* contained in the reference tree as QS. Thereafter, the ML model parameters and branch lengths on the reference tree will be optimized.

Once the model parameters and branch lengths have been optimized, the placement algorithm is invoked. It will visit the $2r - 3$ branches of the reference tree in depth first-order, starting at an arbitrary branch of the tree. At each branch, initially the probability vectors of the reference tree to the left and the right will be computed (if they are not already oriented towards the current branch). Thereafter, the program will successively insert (and remove again) one QS at a time into the current branch and compute the likelihood (we henceforth denote this as insertion score) of the respective tree containing $r + 1$ taxa. The insertion score is stored in a $q \times (2r - 3)$ table that keeps track of the insertion scores for all q QS into all $2r - 3$ branches of the reference tree. Hence, the complexity of our placement algorithm is $O(rqm)$ since we need to compute $q \times (2r - 3)$ insertion scores. Note that, the time for the computation of an insertion score is a linear function of the alignment length m (see Equations 2 and 3).

In order to more rapidly compute the per-branch insertions of the QS we use an approximation that is comparable to the Lazy Subtree Rearrangement (LSR) moves that are deployed in the standard RAxML search algorithm [14]. After inserting a QS into a branch of the reference tree we would normally need to re-optimize all branch lengths of the thereby obtained new—extended by one QS—tree topology to obtain the *Maximum* Likelihood insertion score. Instead, we only optimize the three branches adjacent to the insertion node of the QS (see Figure 3) before computing the insertion score. In analogy to the LSR moves, we also use two methods to re-estimate the three branches adjacent to the insertion branch, a *fast* method that does not make use of the Newton-Raphson method

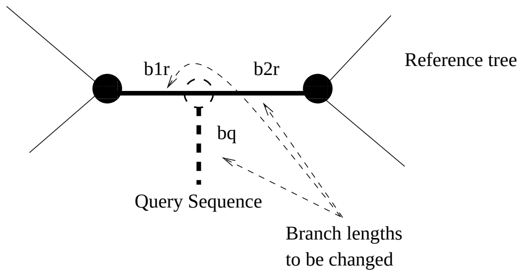

Fig. 3. Local Optimization of branch lengths for the insertion of a Query Sequences (QS) into the reference tree.

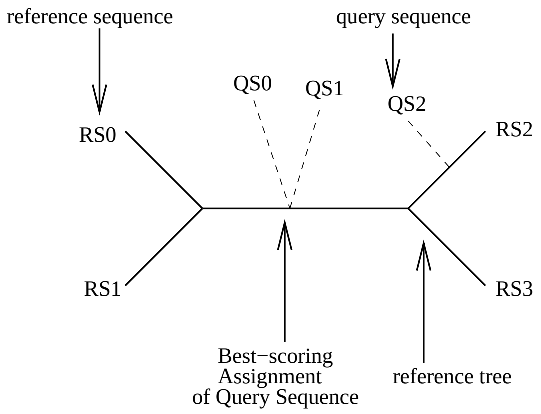

Fig. 4. Evolutionary placement of 3 Query Sequences (QS0, QS1, QS2) using a 4-taxon reference tree.

and a *slow* method. The *fast* insertion method just splits the branch of the reference tree b_r into two parts b_{r1} and b_{r2} by setting $b_{r1} := \sqrt{b_r}$, $b_{r2} := \sqrt{b_r}$, and the branch leading to the QS is set to $b_q := 0.9$, where 0.9 is the default RAxML value to initialize branch lengths. The *slow* method repeatedly applies the Newton-Raphson procedure to all three branches b_{r1}, b_{r2}, b_q until the branch length changes are $\leq \epsilon$, where $\epsilon = 0.00001$.

The output of this procedure is the input reference tree, enhanced by assignments of the QS to the respective best-scoring insertion branch. This means that QS are attached to branches that yield the best insertion score for the respective QS. Hence, the algorithm will return a potentially multi-furcating tree, if more than one QS is assigned to the same branch. An example output tree for 4 RS and 3 QS is provided in Figure 4.

Moreover, RAxML can also conduct a standard phylogenetic bootstrap [31], i.e., repeat the evolutionary placement procedure several times under slight perturbations of the input alignment. This allows to account for uncertainty in the placement of the QS as shown in Figure 5. Thus, a QS might be placed into different branches of the reference tree with various levels of support. For the Bootstrap replicates we also use additional heuristics to accelerate the insertion process. During the insertions on the original input alignment we keep track of the insertion scores for *all* QS into *all* branches of the reference tree. For every QS we can then sort the insertion

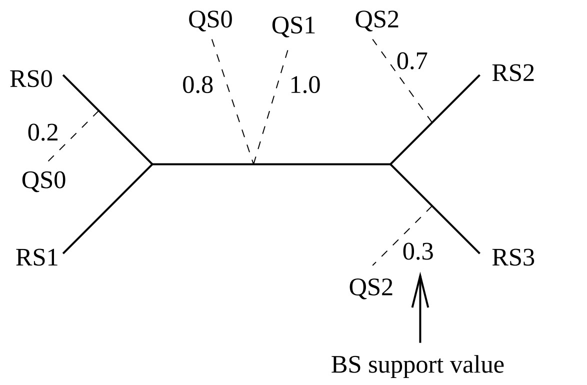

Fig. 5. Evolutionary placement of 3 query sequences (QS0, QS1, QS2) into a 4-taxon reference tree with Bootstrap support.

branches by their scores and only conduct insertions for a specific QS into the 10% best-scoring branches. Evidently, this reduces the number of insertion scores to be computed per QS by 90% and yields an approximately ten-fold speedup. The bootstrap procedure can for instance be used to compute the distribution of the phylogenetic diversity of the QS in the environmental sample.

Finally, we are currently also experimenting with methods (that are already implemented) to simultaneously align and insert QS to a specific branch of the reference tree using a Maximum Likelihood and Maximum Parsimony-based [32] dynamic programming algorithms. This will allow for direct and more convenient analyses of environmental reads, because users will be able to conduct the alignment and placement step in one single run. Evidently, the complexity will increase to $O(rqm^2)$ because of the dynamic programming approach. Therefore, the framework we develop here can be used to integrate and accelerate the future release of the simultaneous alignment and placement procedures.

V. Performance of Sequential Algorithm

As already mentioned, a detailed performance analysis as well as a description of the experimental setup to assess the accuracy of our algorithm is available [13]. We assessed the accuracy of our evolutionary placement algorithm on 8 real-world (mostly single-gene) datasets that were hand-aligned by biologists. Initially, we computed best-known ML trees and bootstrap support values on these alignments using RAxML. We then pruned a single taxon (a virtual Query Sequence) at a time from these trees. Thereafter, we tried to re-insert the virtual Query Sequence into the tree using our evolutionary placement algorithm, BLAST, and a simple sequence comparison algorithm. This sequence-based comparison algorithm makes use of the aligned sequences to compute all pair-wise edit distances to the taxa of the Reference Alignment that are located at the leaves of the tree. We only selected taxa (virtual QS) for pruning that had bootstrap support $\geq 75\%$ ([13] for selection details and rationale), i.e., taxa with a sufficiently well-supported position in the tree. In addition, we artificially shortened the virtual QS in order to generate sequences that

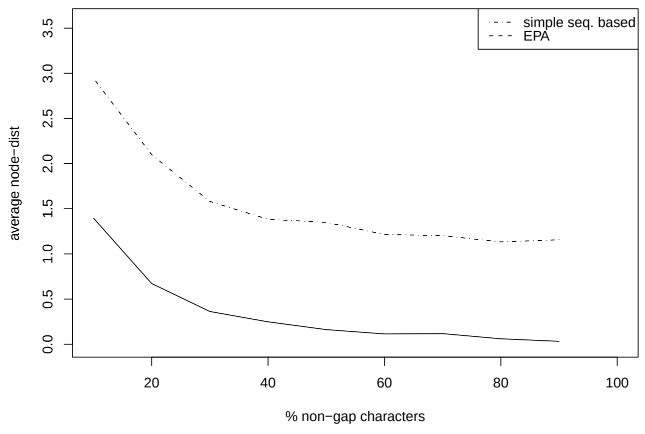

Fig. 6. Accuracy of sequence similarity-based and ML-based placement of short-read sequences as a function of the percentage of non-gap characters, averaged over all test datasets.

have the same length as 454 reads, i.e., approximately 200-400 base pairs.

The accuracy was then assessed by determining the number of nodes that lie between the original "true" pruning position of the virtual QS and the respective placement branch as computed by our algorithm, BLAST and the sequence comparison algorithm. In Figure 6, we depict the average node distance from the correct insertion position, averaged over all datasets. The x-axis indicates the percentage of non gap-characters in the virtual—artificially shortened—Query Sequences with respect to the full alignment length. Figure 6, clearly shows that our algorithm is twice as accurate as the sequence-based placement approach.

In a second set of experiments we shortened the full-length virtual query sequence by generating artificial paired-end reads, i.e., just kept 100 base-pairs of nucleotide data at the left and right end of the full length sequence and inserted gaps in the middle. We then used our placement algorithm and BLAST to compute insertion positions for those query sequences. In Figure 7, we provide a histogram for a DNA dataset of 855 taxa that depicts the number of inserted query sequences at distances of 0, 1, 2, etc. nodes away from the original "correct" position. This histogram shows that ML-based placement is also significantly more accurate than a BLAST-based approach.

Finally, we have developed several additional heuristics to accelerate the evolutionary placement algorithm (similar to the procedure for the acceleration of bootstrapping outlined in Section IV), that are only 1.5-2 times slower than BLAST (see Figures 10.a and 10.b in [13]), but are almost as accurate as the slow standard evolutionary placement algorithm. The parallelization of all these additional heuristics and the bootstrapping procedure is analogous to the general parallelization scheme which we present in the following Section.

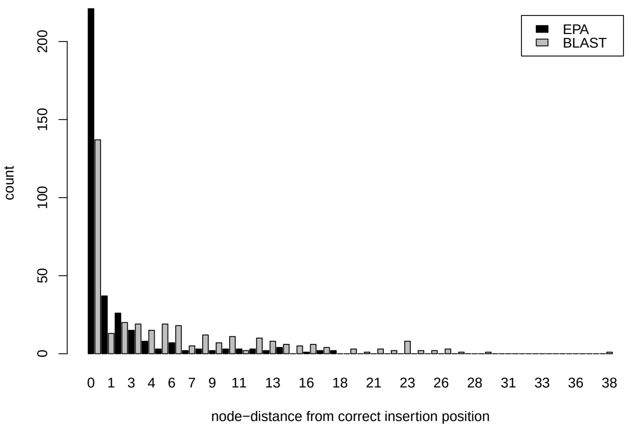

Fig. 7. Accuracy of BLAST- and ML-based placement of short-read sequences as a function of the percentage of non-gap characters, averaged over all test datasets.

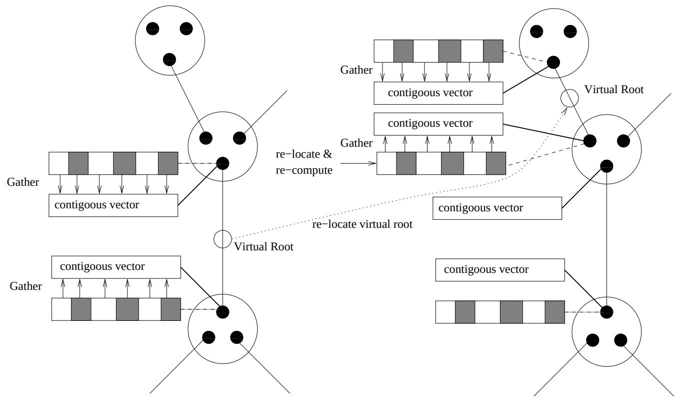

Fig. 8. Multi-grain gather operation of the per-branch probability vectors on the reference tree.

VI. Multi-Grain Parallelization

The current release of RAxML (v7.2.6) contains a full fine-grain Pthreads-based parallelization of the likelihood function (see Section III and [16]). The initial parallelization of the evolutionary placement algorithm was hence straight-forward because we used the existing parallel framework. However, the scalability of the fine-grain parallelization is limited on datasets with few sites such as the real-world single gene bacterial datasets we use in our experiments.

The main parallelization challenge lies in the two different phases of the program that exhibit varying degrees of parallelism: In the *initial phase* during which ML model parameters are optimized, the branch length optimization procedure is hard to be further parallelized because of intrinsic dependencies between individual branch length optimization steps. The *insertion phase* consists of computing the $q \times (2r-3)$ insertion scores of the QS on the reference tree and is easier to parallelize. Hence we need to deploy multi-grain parallelism using the fine-grain parallelization scheme for the *initial phase* and a more coarse-grain parallelization strategy for the *insertion phase*. One can parallelize the insertion phase by essentially conducting all $q \times (2r - 3)$ insertion score computations simultaneously. Here, we parallelize by branches, i.e., we use a cyclic distribution of reference tree branches to threads where every thread computes insertion scores for insertions of *all* q QS into one specific branch. Given the number of branches in real-world reference trees (9,745 insertion branches, see Section VII) which is expected to further increase in the future, this represents a sufficiently fine-grain source of parallelism for this algorithm. In order to be able to compute QS insertions simultaneously we require to compute and store all pairs of probability vectors for all branches in the tree. Hence, for every branch we need a contiguous (remember that for the fine-grain parallelization we use a cyclic distribution of the probability vector columns) probability vectors attached to the left and right end of each branch. This allows a single thread to independently compute all insertions of the QS into one branch, since only those two vectors are required to compute the insertion score.

In order to store and compute contiguous probability vectors we initially allocate a data-structure of length $2r - 3$ that corresponds to the total number of branches in the reference tree. For every entry of this branch data structure we allocate two contiguous (full-length) probability vectors that are used to store the probability vectors to the left and the right of that branch in the reference tree. In order to fill the per-branch probability vectors we proceed as follows: We traverse the entire reference tree in depth-first order and place the virtual root into the branch that is currently being visited and re-compute the probability vectors (if necessary) to the left and right of the current branch. Once the strided vectors using the cyclic probability vector column distribution have been computed, we conduct a gather operation to store the per-thread columns contiguously in the vectors attached to the branch data structure. This traverse and gather procedure is outlined in Figure 8. The overhead of this full tree traversal and the gather operation is negligible (less than 0.5% of overall execution time) compared to the overall execution time as well as the time required for initial model parameter optimization in fine-grain mode. Once the probability vector pairs for all branches have been stored, we can simply distribute work to threads in a cyclic way, i.e., thread 0 will insert all QS into branch 0, thread 1 will insert all QS into branch 1, etc.

An additional rationale for deploying a per-branch parallelization strategy is a planned distributed memory parallelization of the evolutionary placement algorithm with MPI (Message Passing Interface). If we parallelized by insertion sequences we would need to hold all insertion branch vectors in the memory of every process which might lead to memory shortage on systems such as the IBM BlueGene/L or BlueGene/P.

VII. Experimental Setup & Results

We used three real-world 16S rRNA datasets with 4,412 [3] (HAND dataset), 16,307 [2] (GUT dataset), and 100,627 (unpublished) QS that are classified into a single-gene bacterial reference tree with 4,874 species and a reference alignment length of 1,287 base pairs. Those datasets represent typical current use cases for our algorithm and the two smaller ones are freely available for download together with the multi-grain parallelization of the source-code at http://wwwkramer.in.tum.de/exelixis/software.html. As test systems for assessing scalability we used a 4-way quad-core AMD Barcelona system with a total of 16 cores running at 2.2 GHz and 128 GB of main memory, and a Sun x4600 8-way quad-core AMD Shanghai system with a total of 32 cores running at 2.7 GHz and 64 GB of main memory.

Computational experiments were conducted as follows: On the AMD Barcelona system we executed one sequential run and Parallel Fine-Grain (PFG) as well as Parallel Multi-Grain (PMG) runs on 2, 4, 8, and 16 cores for the *fast* insertion method. On the x4600 we executed the same runs, but up to 32 cores. We also executed two runs using the *slow* insertion method with 16 threads for the PFG and PMG parallelization on the 16-core system. The execution time for the slow method using the PFG approach on the HAND dataset was 35 hours compared to *only* 22 hours using the PMG implementation. Thus, for the slower (and more accurate) insertion algorithm on the HAND dataset we achieve a parallel efficiency improvement exceeding 35%. The execution times for the slow method on the larger GUT dataset where 139 hours (PFG) and 76 hours (PMG) respectively, which corresponds to a parallel run time improvement of 45%.

In Figure 9 we depict the speedups for the fast insertion algorithm using the PFG and PMG parallelizations on the Barcelona system for datasets HAND and GUT. Figure 9 shows that the PFG version scales reasonably well on the 16-core system and even achieves super-linear speedups due to improved cache efficiency up to 8 threads. However, as outlined in Sections III and VI the scalability of the fine-grain approach is limited for single-gene alignments. The impact of an increasing amount of synchronization events per computation in the PFG parallelization is demonstrated by the sub-linear speedups on 16 cores. The increase in parallel efficiency of the PMG over the PFG approach on 16 cores for the fast insertion method is slightly lower than for the slow insertion method, but we still achieve a 25% improvement on 16 cores. This is because the contribution of the less scalable fine-grain initial model optimization phase to overall execution time is larger for the fast than for the slow insertion method. Nonetheless, speedups are significantly super-linear up to 16 cores for the PMG version. The overall speedups are slightly lower on the 32-core system (Figure 10), because of the less scalable fine-grain initial model optimization phase; stand-alone placement without model optimization scales linearly. However, the increase in parallel efficiency of the PMG over the PFG approach with 32 cores exceeds 50%.

Figures 9 and 10 show that the parallel efficiency of the evolutionary placement algorithm is generally higher on the AMD Barcelona system. The reason for this is that the improved cache efficiency of the parallel algorithm has a larger impact

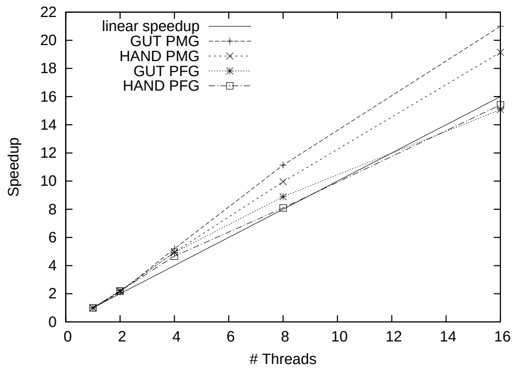

Fig. 9. Speedups for overall execution times on a 16-core AMD Barcelona system using the fast insertion algorithm.

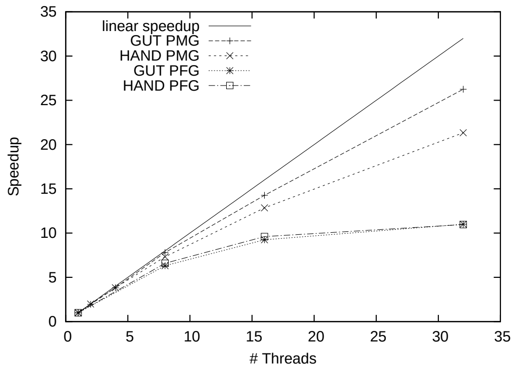

Fig. 10. Speedups for overall execution times on a 32-core x4600 system using the fast insertion algorithm.

on the Barcelona cores that have 2MB L3 caches compared to 6MB L3 caches on the Shanghai cores. In Figure 11 we plot the speedup of the Shanghai over the Barcelona CPUs as a function of the number of threads. When only one Shanghai core is used, the program is almost twice as fast as on a Barcelona core which can not be explained by the higher clock rate and improved micro-architecture alone. The main cause is the three times larger L3 cache size. If the number of threads is increased and cache misses are thereby decreased, the speedup converges to 1.23 which corresponds to the clock rate ratio $(2.7GHz/2.2GHz)$.

Finally, we also conducted experiments on a challenging dataset with 100,627 QS to explore the limits of our approach on the x4600 system. We obtained overall speedups of 14.5 and 27.3 for fast insertion runs with 16 and 32 cores respectively. On 32 cores the overall runtime for the placement of 100,627 QS is less than 1.5 hours which makes our approach a viable and more accurate alternative to BLAST and other sequence comparison-based approaches that do not take into account the evolutionary history of the sequences under study.

VIII. Conclusion & Future Work

We have introduced a new, accurate, algorithm for ML-based evolutionary placement of short reads which is already being used by several research groups (personal communi-

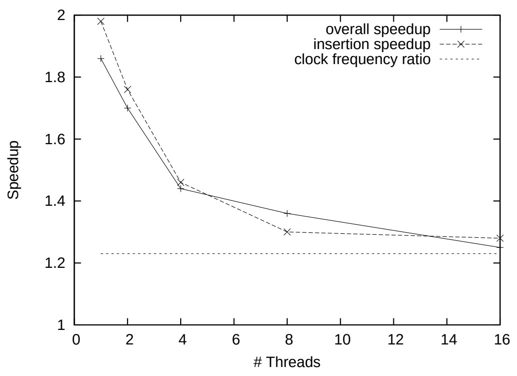

Fig. 11. Speedup of the 32-core (Shanghai) over the 16-core (Barcelona) architecture.

cation) for real-world studies of microbial communities. We show that fine-grain parallelism does not scale well for the analysis of real-world microbial datasets and propose a more appropriate multi-grain parallelization approach. Our parallelization yields super-linear speedups on multi-core systems with small L3 caches and parallel run time improvements of 25% up to 50% compared to the fine-grain approach.

Current work covers the full integration of the multi-grain parallelization with RAxML, i.e., the support of all data types (morphological, DNA, secondary structure, protein data) and models of rate heterogeneity. We also plan to devise an MPI-based parallelization. Future work, will focus on using multi-grain parallelism for an insertion algorithm that can simultaneously place *and* align the QS.

Acknowledgment

We wish to thank R. Knight and M. Hamady for providing test datasets, M. Stark and C.v. Mehring for productive discussions on algorithm design, and B. Moret for access to the Barcelona system.

References

[1] T. Wicker, E. Schlagenhauf, A. Graner, T. Close, B. Keller, and N. Stein, “454 sequencing put to the test using the complex genome of barley,” *BMC genomics*, vol. 7, no. 1, p. 275, 2006.

[2] P. Turnbaugh, M. Hamady, T. Yatsunenko, B. Cantarel, A. Duncan, R. Ley, M. Sogin, W. Jones, B. Roe, J. Affourtit *et al.*, “A core gut microbiome in obese and lean twins,” *Nature*, vol. 457, no. 7228, pp. 480–484, 2008.

[3] N. Fierer, M. Hamady, C. Lauber, and R. Knight, “The influence of sex, handedness, and washing on the diversity of hand surface bacteria,” *Proc. Nat. Acad. Sci.*, vol. 105, no. 46, p. 17994, 2008.

[4] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” *Nucleic acids research*, vol. 25, no. 17, p. 3389, 1997.

[5] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *J. Mol. Bwl*, vol. 147, pp. 195–197, 1981.

[6] D. Faith, “Conservation evaluation and phylogenetic diversity.” *Biological Conservation*, vol. 61, no. 1, pp. 1–10, 1992.

[7] L. Koski and G. Golding, “The closest BLAST hit is often not the nearest neighbor,” *J. Mol. Evol.*, vol. 52, no. 6, pp. 540–542, 2001.

[8] D. Zwickl and D. Hillis, “Increased taxon sampling greatly reduces phylogenetic error,” *Systematic Biology*, vol. 51, no. 4, pp. 588–598, 2002.

[9] W. Ludwig, O. Strunk, R. Westram, L. Richter, H. Meier, A. Buchner, T. Lai, S. Steppi, G. Jobb, W. Forster *et al.*, “ARB: a software environment for sequence data.” *Nucleic Acids Research*, vol. 32, no. 4, p. 1363, 2004.

[10] B. M. E. Moret, U. Roshan, and T. Warnow, “Sequence-length requirements for phylogenetic methods,” in *WABI 2002*, 2002, pp. 343–356.

[11] O. R. P. Bininda-Emonds, S. G. Brady, M. J. Sanderson, and J. Kim, “Scaling of accuracy in extremely large phylogenetic trees.” in *Pacific Symposium on Biocomputing*, 2000, pp. 547–558.

[12] J. Felsenstein, “Evolutionary trees from DNA sequences: a maximum likelihood approach,” *J. Mol. Evol.*, vol. 17, pp. 368–376, 1981.

[13] S. A. Berger and A. Stamatakis, “Evolutionary placement of short sequence reads,” TU Munich, Tech. Rep., November 2009. [Online]. Available: http://arxiv.org/abs/0911.2852v1

[14] A. Stamatakis, “RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models,” *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[15] A. Stamatakis and M. Ott, “Exploiting Fine-Grained Parallelism in the Phylogenetic Likelihood Function with MPI, Pthreads, and OpenMP: A Performance Study.” in *PRIB*, ser. Lecture Notes in Computer Science, vol. 5265. Springer, 2008, pp. 424–435.

[16] ——, “Efficient computation of the phylogenetic likelihood function on multi-gene alignments and multi-core architectures.” *Phil. Trans. R. Soc. series B, Biol. Sci.*, vol. 363, pp. 3977–3984, 2008.

[17] F. Blagojevic, D. Nikolopoulos, A. Stamatakis, and C. Antonopoulos, “Dynamic Multigrain Parallelization on the Cell Broadband Engine,” in *Proc. of PPoPP 2007*, San Jose, CA, March 2007.

[18] M. Ott, J. Zola, S. Aluru, and A. Stamatakis, “Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L,” in *Proc. of IEEE/ACM Supercomputing Conference 2007 (SC2007)*, 2007.

[19] B. Minh, L. Vinh, H. Schmidt, and A. Haeseler, “Large maximum likelihood trees,” in *Proc. of the NIC Symposium 2006*, 2006, pp. 357–365.

[20] D. Zwickl, “Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion,” Ph.D. dissertation, University of Texas at Austin, April 2006.

[21] B. Boussau and M. Gouy, “Efficient likelihood computations with nonreversible models of evolution,” *Systematic biology*, vol. 55, no. 5, pp. 756–768, 2006.

[22] N. Alachiotis, A. Stamatakis, E. Sotiriades, and A. Dollas, “An Architecture for the Phylogenetic Likelihood Function,” 2009, accepted for publication.

[23] A. Stamatakis, M. Ott, and T. Ludwig, “RAxML-OMP: An Efficient Program for Phylogenetic Inference on SMPs.” *PaCT*, pp. 288–302, 2005.

[24] R. Brent, *Algorithms for Minimization without Derivatives*. Prentice Hall, 1973.

[25] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, “Numerical recipes in C,” *The art of scientific computing (Cambridge: University Press*, vol. 3, no. 2, 1992.

[26] A. Stamatakis and M. Ott, “Load Balance in the Phylogenetic Likelihood Kernel,” in *Proceedings of ICPP 2009*, 2009, accepted for publication.

[27] M. Ott, J. Zola, S. Aluru, A. Johnson, D. Janies, and A. Stamatakis, “Large-scale Phylogenetic Analysis on Current HPC Architectures,” *Scientific Programming*, vol. 16, no. 2-3, pp. 255–270, 2008.

[28] T. DeSantis Jr, P. Hugenholtz, K. Keller, E. Brodie, N. Larsen, Y. Piceno, R. Phan, and G. Andersen, “NAST: a multiple sequence alignment server for comparative analysis of 16S rRNA genes,” *Nucleic Acids Research*, vol. 34, no. Web Server issue, p. W394, 2006.

[29] K. Katoh and H. Toh, “Recent developments in the MAFFT multiple sequence alignment program,” *Briefings in Bioinformatics*, vol. 9, no. 4, p. 286, 2008.

[30] R. Edgar, “MUSCLE: multiple sequence alignment with high accuracy and high throughput,” *Nucleic acids research*, vol. 32, no. 5, p. 1792, 2004.

[31] J. Felsenstein, “Confidence Limits on Phylogenies: An Approach Using the Bootstrap,” *Evolution*, vol. 39, no. 4, pp. 783–791, 1985.

[32] W. Fitch and E. Margoliash, “Construction of phylogenetic trees,” *Science*, vol. 155, no. 3760, pp. 279–284, 1967.