Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L

Michael Ott
Technical University of Munich
Department of Computer Science
ottmi@in.tum.de

Jaroslaw Zola
Iowa State University
Department of Electrical and Computer Engineering
zola@iastate.edu

Srinivas Aluru
Iowa State University
Department of Electrical and Computer Engineering
aluru@iastate.edu

Alexandros Stamatakis
École Polytechnique Fédérale de Lausanne
School of Computer and Communication Sciences
Alexandros.Stamatakis@epfl.ch

ABSTRACT

Phylogenetic inference is a *grand challenge* in Bioinformatics due to immense computational requirements. The increasing popularity of multi-gene alignments in biological studies, which typically provide a stable topological signal due to a more favorable ratio of the number of base pairs to the number of sequences, coupled with rapid accumulation of sequence data in general, poses new challenges for high performance computing. In this paper, we demonstrate how state-of-the-art Maximum Likelihood (ML) programs can be efficiently scaled to the IBM BlueGene/L (BG/L) architecture, by porting RAxML, which is currently among the fastest and most accurate programs for phylogenetic inference under the ML criterion. We simultaneously exploit coarse-grained and fine-grained parallelism that is inherent in every ML-based biological analysis. Performance is assessed using datasets consisting of 212 sequences and 566,470 base pairs, and 2,182 sequences and 51,089 base pairs, respectively. To the best of our knowledge, these are the largest datasets analyzed under ML to date. The capability to analyze such datasets will help to address novel biological questions via phylogenetic analyses. Our experimental results indicate that the fine-grained parallelization scales well up to 1,024 processors. Moreover, a larger number of processors can be efficiently exploited by a combination of coarse-grained and fine-grained parallelism. Finally, we demonstrate that our parallelization scales equally well on an AMD Opteron cluster with a less favorable network latency to processor speed ratio. We recorded super-linear speedups in several cases due to increased cache efficiency.

Keywords

Phylogenetic Inference, Maximum Likelihood, RAxML, IBM BlueGene/L

1. INTRODUCTION

Phylogenetic trees are used to represent the evolutionary history of a set of n organisms. An alignment of DNA or protein sequences that represent these n organisms can be used as input for phylogenetic inference. In a phylogeny the organisms of the input dataset are located at the tips (leaves) of the tree and the inner nodes represent extinct common ancestors. The branches of the tree represent the time which was required for the mutation of one species into another, new one. Phylogenetic trees have many important applications in medical and biological research (see [3] for a summary).

Due to the rapid growth of sequence data over the last years there is an increasing demand to compute large trees which often comprise more than 1,000 organisms and sequence data from several genes (so-called multi-gene alignments). Since alignments continuously grow in the number of organisms *and* in sequence length, there exists an increasing need for efficient parallel phylogeny programs.

It has recently been shown that the Maximum Likelihood (ML) phylogeny problem is NP-hard [8]. The inherent algorithmic complexity of this problem is a result of the vast number of alternative tree topologies which grows exponentially with the number of organisms n, e.g. for $n = 50$ there exist $2.84 * 10^{76}$ alternative trees. In order to find *the* Maximum Likelihood tree, all potential alternative trees would have to be enumerated and evaluated under ML. Thus, efficient heuristic tree search algorithms are required to reduce the search space. Significant progress in the field of heuristic ML search algorithms has been made over the last years with the release of programs such as IQPNNI [23], PHYML [18], GARLI [38] and RAxML [29, 32], to name only a few. Note that none of these heuristics is guaranteed to find *the* Maximum Likelihood tree topology, but will only yield a best-known ML-based tree. In order to explore the search space more thoroughly tree searches from different starting points (starting trees) are performed, so-called multiple ML tree searches.

In addition to the algorithmic difficulty, ML-based inference of phylogenies is very memory- and floating point-intensive. In fact, both memory consumption as well as inference times grow linearly with the number of distinct alignment columns (see Section 3.1). Due to the continuous accumulation of sequence data, the application of high performance computing techniques can significantly contribute to the reconstruction of larger and more accurate trees.

RAxML-VI-HPC [29] (Randomized Axelerated Maximum Likelihood version VI for High Performance Computing) is a program for large-scale ML-based [12] inference of evolutionary trees using multiple alignments of DNA or AA (Amino Acid) sequences. Since August 2006, the program has been downloaded over 660 times from distinct IP addresses and is an integral component of some of the prominent web resources including CIPRES (CyberInfrastructure for Phylogenetic RESearch, www.phylo.org) project and the greengenes workbench [10] (greengenes.lbl.gov). Moreover, some of the largest published ML-based phylogenetic analyses to date have been conducted with RAxML [17, 26, 22]. Typical applications in current medical and biological research include the evolution of Papilloma-viruses [15, 16] which are associated with cervical cancer or the analysis of microorganisms living in permafrost soils [14]. Finally, the sequential version of the program has been used to compute trees on the two largest data matrices analyzed under ML to date: a 25,057-taxon alignment of protobacteria (length: 1,463 nucleotides, [9]) and a 2,182-taxon alignment of mammals (length: 51,089 nucleotides, [5]). A recent performance study [29] on real world datasets with more than 1,000 sequences reveals that RAxML is able to find better trees in less time and with lower memory consumption than other current ML programs (IQPNNI, PHYML, GARLI, MrBayes). Recently, novel heuristics have been introduced in RAxML that further accelerate the program by a factor of approximately 2.5 while yielding equally good trees [30].

Porting RAxML and any other program to the IBM BlueGene/L (BG/L) represents a particular scaling challenge due to the large number of relatively low performance processors, limited amount of RAM memory (512 MB per node on our system, which reduces to half that if both processors of a node are used for computation; memory shortage is a prevalent problem for large-scale ML analyses) and limited functionality of the operating system as well as of the message passing layer. Despite the fact that RAxML is currently the most memory-efficient ML implementation [29], the total memory footprint of typical large-scale phylogenetic analyses can easily exceed 1GB. Thus, we devise an efficient mechanism to distribute data structures across nodes and orchestrate computations accordingly.

We assess performance of our parallelization under the GTR+Γ model of evolution [37] using the two largest datasets that have been analyzed under ML to date, in terms of input matrix dimensions and memory footprint:

- A multi-gene alignment of 2,182 mammalian sequences with 51,089 nucleotide positions.
- An alignment of non-redundant SNPs (Single Nucleotide Polymorphisms) on the human chromosome 1 that consist of 212 sequences and 566,470 base pairs.

A full analysis on these datasets is currently not feasible on conventional machines. In general, adapting applications to the BG/L architecture is important since 12 out of the 30 top-ranked supercomputers in the current top 500 list, including the number one system at Lawrence Livermore National Laboratories, are based on the BG/L architecture. Moreover, software which scales well on BG/L, will probably scale equally well on forthcoming petascale systems such as the BlueGene/P and BlueGene/Q.

The BG/L supercomputer is a massively parallel architecture with 360 Teraflops of peak performance in the full 64-rack configuration. Because it has been extensively described in other papers (see e.g. [21]), we only highlight the issues which are crucial for the effective porting of ML code. A single node of BG/L consists of two PowerPC 700 MHz processors, each enhanced with two 64-bit floating point units (so called double-hummer). Note that these FPUs cannot be addressed separately. Each of the two CPUs on a node has a non-coherent L1 data cache with 32 KB and both CPUs share a small L2 cache which serves as a prefetch buffer, as well as a common 4 MB L3 cache. There are two working modes: co-processor, where one of the two processors is used to perform computations while the other one is responsible for handling communication, e.g. reduction operations. The other mode is virtual node, where both CPUs are running computations and share cache, memory, and the network. The BlueGene/L system has a high processors to memory ratio, with each node on our system having only 512 MB of RAM memory. Nodes are connected using five different networks. Three of those are of interest to the application programmer: a point-to-point network organized into 3D mesh or torus (bandwidth: 154 MB/s/link, latency $\approx$ 3.35 μs, plus 90 ns per hop); a fast collective network for reduction and broadcast operations (bandwidth 337 MB/s, latency 2.5 μs); a global interrupt network which allows to perform full-system synchronization within 1.5 μs. To program this system one can use the IBM MPI implementation that provides customized routines for most MPI functions [1, 2].

The remainder of this paper is organized as follows: First, we review related work on parallelization of ML programs and adaptations to the BG/L (Section 2). In Section 3 we describe the BlueGene-specific parallelization of RAxML at fine-grained (Section 3.1) and coarse grained (Section 3.2) levels. Our experimental setup and the respective results are provided in Section 4. We conclude the paper with Section 5.

2. RELATED WORK AND PREVIOUS PARALLELIZATIONS OF RAXML

RAxML exploits two levels of parallelism: fine-grained loop-level parallelism and coarse-grained embarrassing parallelism. The program has been parallelized with OpenMP to exploit loop-level parallelism. Like every ML-based program, RAxML exhibits a source of loop-level parallelism in the likelihood functions which typically consume over 95% of the overall computation time. The OpenMP implementation scales particularly well on large multi-gene alignments due to increased cache efficiency [34]. Note that loop-level parallelism can be exploited at two levels of granularity: at a relatively coarse-grained OpenMP level and at a fine-grained CPU level via SIMD instructions. These two layers of loop-level parallelism have been exploited in a recent RAxML porting to the IBM CELL processor [6, 7]. However, the main focus of porting RAxML on Cell was on exploring programming and scheduling techniques for this architecture, using a complex bioinformatics application. In contrast to the current paper, the work on Cell represents a proof-of-concept implementation, rather than a parallelization for large-scale production runs.

The MPI version of RAxML exploits the embarrassing parallelism that is inherent to every real-world phylogenetic analysis. In order to conduct such an analysis (see [17] for an example), about 20–200 distinct tree searches (multiple inferences) to find a best-scoring tree on the original alignment as well as a large number of (100–1,000) bootstrap analyses have to be conducted. *Bootstrap Analyses* are required to assign confidence values ranging between 0.0 and 1.0 to the inner nodes of the best-known/best-found ML tree. This allows to determine how well-supported certain parts of the tree are and is important for drawing biological conclusions. Bootstrapping is essentially very similar to multiple inferences. The only difference is that inferences are conducted on a randomly re-sampled alignment (a certain number of alignment columns are re-weighted) for every bootstrap run. This is performed in order to assess the topological stability of the tree under slight alterations of the input data.

All those individual tree searches, be it bootstrap or multiple inferences, are completely independent from each other and can thus be exploited by a simple master-worker scheme. If the dataset is not extremely large or the available memory per CPU is not too small, this represents the most efficient approach to exploit HPC platforms for production runs.

Most other parallel implementations of ML programs [11, 23, 31, 35, 38] have mainly focused on the intermediate level of parallelism (inference/search algorithm parallelism) which is situated between the loop-level parallelism and coarse-grained parallelism currently exploited in RAxML. The work on the exploitation of inference parallelism mentioned above mainly deals with highly algorithm-specific and mostly MPI-based parallelization of various hill-climbing, genetic, as well as divide-and-conquer search algorithms. Typically, such parallelizations yield a lower parallel efficiency compared to the embarrassing and loop-level types due to hard-to-resolve dependencies in the respective search algorithms. Moreover, these parallelizations are much more program-specific and thus not generally applicable. Minh *et al.* [24] recently implemented a hybrid OpenMP/MPI version of IQPNNI which exploits loop-level and inference parallelism.

Finally, Feng *et al.* [13] describe PBPI (Parallel Bayesian Phylogenetic Inference), a fast parallel implementation of a Bayesian phylogenetic algorithm on BlueGene/L. However, we are not aware of any published real biological study based on PBPI. This is probably due to the fact that PBPI currently only implements the simple Jukes-Cantor model of sequences evolution [20] and does not offer protein substitution models nor account for rate heterogeneity among sites. According to the PBPI web-page [25] a version with a larger variety of models is in preparation. Nonetheless, the datasets for which trees have been inferred with PBPI on BG/L can easily be handled by RAxML or GARLI under a more realistic model of nucleotide substitution on a single CPU. For example, on a single 2.4 GHz AMD Opteron processor the sequential version of RAxML requires an average of 3,674 seconds for one full ML tree search on the largest dataset (218 sequences, 10,000 base-pairs) that has currently been analyzed with PBPI. This means that, datasets of this size do not represent a computational challenge and do not require expensive supercomputers. Finally, we are concerned about the exclusive usage of "perfect" simulated data, that does not contain gaps and typically requires less powerful search mechanisms (see [32] for a discussion and related experiments) to evaluate the performance of the topology proposal mechanism in PBPI.

With respect to the application of Bayesian searches to large — in terms of number of sequences — datasets, a recent study by D.E. Soltis *et al.* [27] describes potential pitfalls concerning apparent stationarity of MC^3 chains (see also [33]), by example of a 567-taxon dataset of Angiosperms.

3. PARALLELIZATION OF RAXML ON THE BLUEGENE/L

The current section covers the adaptation of the two levels of parallelism in RAxML to the BG/L using an MPI/MPI-based approach. Depending on the alignment dimensions it provides sufficient flexibility to simultaneously compute many jobs on a relatively short alignment or to use a large number of processors for jointly computing the likelihood function on very long and memory-intensive alignments.

3.1 Fine-Grained Parallelism

As already mentioned the computation of the likelihood function consumes over 90-95% of total execution time in all current ML implementations. Due to its intrinsic fine-grained parallelism coupled with a low number of dependencies, the ML functions represent ideal candidates for parallelization at a low level of granularity.

To compute the likelihood of a fixed *unrooted* tree topology with given branch lengths one needs to compute the entries for all likelihood vectors, which are located at the inner nodes of the tree, bottom-up towards a virtual root that can be located at any branch of the tree. The sequences of the alignment are located at the tips of the tree topology and are represented by tip vectors. Once this is done, the log likelihood value can then be computed by summing up over the likelihood vector values to the left and right of the virtual root. In order to obtain the *Maximum* Likelihood value, all individual branch lengths must be optimized with respect to the overall likelihood score. For a more detailed description please refer to [12] or [28]. Note that, most current search algorithms such as GARLI, RAxML or PHYML, do not re-optimize *all* branch lengths after a change in tree topology but rather carry out local optimizations in the neighborhood of the tree that is most affected by the change. The main bulk of these computations consist of `for`-loops over the length m of the alignment, or more precisely over the number m' of distinct patterns in the alignment.

The data-structures required to store the n sequences of the alignment at the tips (tip vectors) of the tree and the $n-2$ likelihood vectors at the inner nodes account for more than 90% of the total memory footprint of RAxML. In fact, the memory consumption of all ML and Bayesian implementations is largely dominated by these data structures. The n tip vectors consist of simple `char*` arrays of length m' to which the AA and DNA alphabet are mapped (see [4] for more implementation details). For DNA data each of the m' entries at inner likelihood vectors consists of 4 double values (20 double values for amino acids) that contain the probabilities of observing an `A, C, G` or `T` (the 4 DNA-bases adenine,

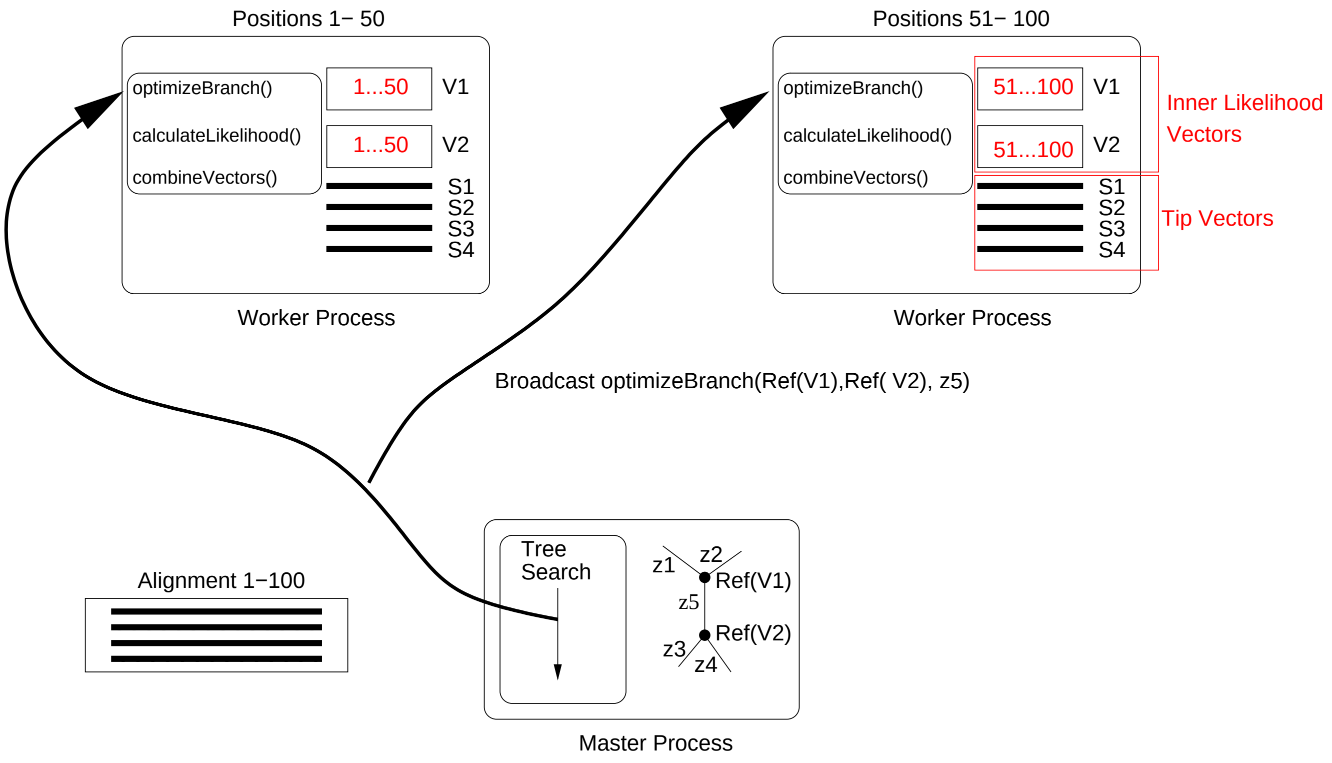

Figure 1: Simplified representation of the fine-grained parallelization strategy

guanin, cytosine, and thymine) at the specific internal node of the tree. If, as in the present case, the discrete Γ model of rate heterogeneity [37] with 4 discrete rates is used, each entry of the likelihood vector consists of 16 (80 for AA) double values, 4 (20) for each discrete rate. Furthermore, each likelihood vector entry contains an additional single integer value that is used to record the number of times the specific position has been scaled to avoid numerical underflow. Note that, the individual iterations of the `for`-loops over tip and likelihood vectors are independent from each other. This property is due to one of the fundamental assumptions of the ML model which states that individual columns evolve independently from each other [12].

We summarize the three basic operations at an abstract level and provide their approximate contributions to overall run-time. The computations consist in combining the values of two or three likelihood and/or tip vectors via a relatively large number of floating point operations:

1. **Computation of Partial Likelihood Vectors (approximately 55-60% of run-time):** This operation computes the entries of a likelihood vector located at an inner node p by combining the values of the likelihood or tip vectors and branch lengths of its two descendants. Thus, this function operates on 3 likelihood/tip vectors but does not require any reduction operations.

2. **Log Likelihood Value Computation (approximately 5% of run-time):** this function just combines the values of two likelihood/tip vectors at the nodes located at either end of the branch where the virtual root has been placed into the log likelihood value for the tree. It requires a global reduction operation.

3. **Branch Length Optimization (approximately 30-35% of run-time):** this operation optimizes a specific branch between two nodes of the tree (two likelihood/tip vectors) by applying e.g. a Newton-Raphson procedure. In order to perform this operation synchronization between the individual iterations of the Newton-Raphson method is required in addition to reduction operations to compute the derivatives of the likelihood function.

In the following we describe how this type of parallelism which is typically exploited with OpenMP on SMP systems [24, 34] can be mapped to the BG/L using appropriate MPI collective communication operations. In order to achieve this we have implemented a master-worker approach to exploit the fine-grained parallelism of the ML function. The master process maintains the only copy of the tree topology and orchestrates the actual tree search as outlined in [32] by issuing the three distinct types of likelihood vector combination instructions to the worker processes.

At initialization each of the p worker processes allocates a fraction m'/p space for the n tip and $n-2$ inner likelihood vectors, i.e. the memory space for tip/likelihood vectors is equally distributed among the processes. These vectors are consistently enumerated in all workers and the master, despite the fact that no memory is actually allocated in the master. The worker processes are relatively lightweight because they only implement the actual mathematical operations on the tip and likelihood vectors. Thus, the master process simply has to broadcast commands such as optimize the branch length between vectors number x and y given the current branch length z. Global reduction operations, which in both cases (log likelihood computation & branch length

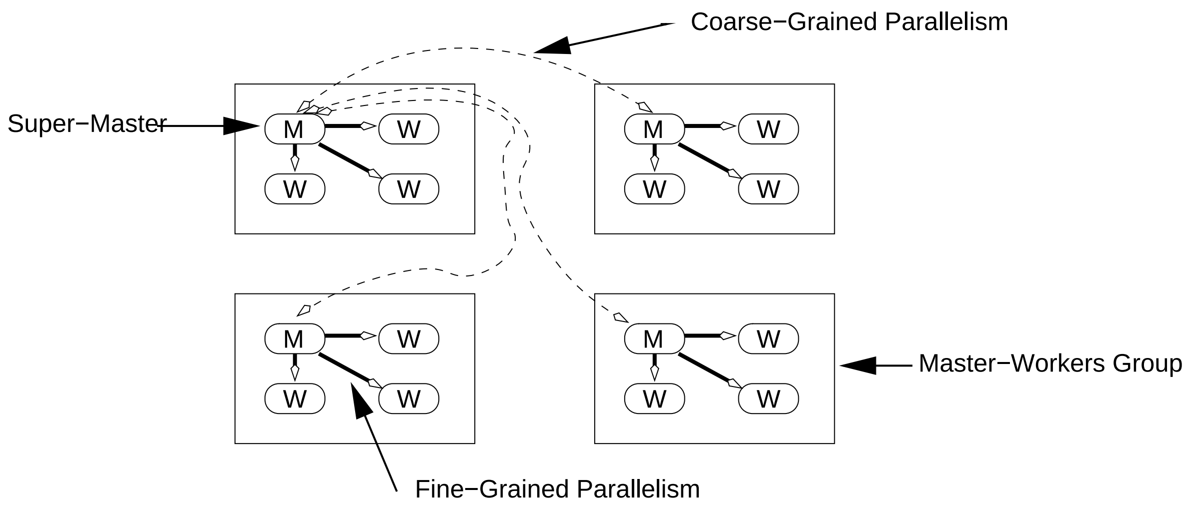

Figure 2: Hybrid MPI/MPI parallelization of RAxML on BlueGene/L

optimization) are simply an addition over m' double values, are performed via the respective MPI collective reduction operation. It is important to note that this represents the most reasonable approach to exploit the BG/L architecture. The IBM MPI implementation uses the specialized very low latency network to implement both collective operations (broadcast and reduction). In contrast to the other two operations (branch length optimization and likelihood computation), the computation of inner likelihood vectors frequently consists of a series of recursive calls, depending on how many vectors must be updated due to changes in the tree topology or model parameters. In order to reduce the communication frequency such series of recursive calls are transformed into an iterative sequence of operations by the master. The master then sends the whole iterative sequence of inner likelihood vector updates via a single broadcast to each worker. The above mechanism is particularly efficient during the ML model parameter optimization phase: after the modification of a model parameter the $n-2$ likelihood vectors of the entire tree need to be updated to compute the log likelihood under the changed model. Thus, we can reduce the number of messages required from $n-2$ to 1. Note that the model optimization phase typically accounts for 10-20% of overall inference time and that n can easily become larger than 1,000.

In order to make efficient use of the 64-bit floating point unit we re–organized the `for`-loops in the ML functions to facilitate compiler-based (IBM XL C) loop unrolling and exploitation of memory access patterns. Moreover, we used the highly-tuned implementations of the compute-intensive mathematical functions `log` and `exp` that are provided by the IBM MASS library (Mathematical Acceleration SubSystem, www.ibm.com/software/awdtools/mass). We were not able to use SIMD vector instructions on the double-hummer due to highly specific requirements for relative data alignment which would induce significant changes to the RAxML source code. Nonetheless, we have identified the problem and are confident that we can overcome this limitation in the near future. At this point we should mention that because we are not using double-hummer and the network communication is relatively infrequent we are able to exploit virtual node mode very efficiently (see Section 4.4). A similar behavior has been observed in several other applications [21], however it is not common.

Note that, we have only ported the likelihood functions for the commonly used GTR+Γ model to demonstrate the general applicability of our parallelization strategy since the CAT approximation of rate heterogeneity is currently a specialty of RAxML.

In Figure 1 we provide a simplified view of the parallel implementation for an alignment with 4 sequences and 100 distinct patterns ($m' = 100$). The two likelihood ($V1$, $V2$, large rectangles) and four tip vectors ($S1 - S4$, thick black lines) are split up equally among both worker processes. The master only maintains the tree data structure and executes the RAxML search algorithm. In this example the master broadcasts a request for branch length optimization of branch $z5$ which is performed by executing computations on the likelihood vectors $V1$ and $V2$ in the workers. Note that, the master only needs to send the vector reference numbers $Ref(V1)$, $Ref(V2)$ to the workers.

3.2 Coarse-Grained Parallelism

As outlined in Section 2, RAxML also exploits the embarrassing parallelism inherent to every ML-based production run on real biological data via a simple master-worker scheme. A centralized master distributes tree inference jobs on distinct starting trees or distinct bootstrap replicates to the worker processes.

For the BG/L porting we modified the above scheme to exploit the hybrid parallelism of RAxML using MPI for both layers: coarse-grained work distribution and fine-grained parallelism as described in the previous Section. In a small example scenario a set of 4 individual master processes might be working on individual ML searches on 20 distinct starting trees or bootstrap replicates. Those 4 masters can e.g. use 3,7, or 15 individual worker processes — depending on the dataset size — to perform likelihood computations. In order to distribute coarse-grained work at tree search level one of those 4 masters has to assume the role of a super-master. Apart from scheduling fine-grained work to its private set of workers the super-master also needs to distribute coarse-grained work to the remaining masters. For this purpose we slightly modified the straight-forward master-worker scheme of the standard RAxML distribution which uses a work queue. The rationale for these modifications is to avoid frequent perturbations of fine-grained work scheduling at the super-master by coarse-grained work distribution to other master processes (the execution time for a tree or bootstrap search typically takes at least several minutes — if not hours — i.e. master/super-master communication is relatively scarce).

Initially, we divide the `MPI_COMM_WORLD` communicator into the respective subgroups (4 subgroups in our example) by using the `MPI_Comm_split` command. On BG/L we apply the following scheme: Each resulting sub-communicator is built as 3D mesh with dimensions x, y, z, such that $x \approx y \approx z$. The master node has coordinates (0,0,0). This mechanism to create partitions and place the master node is well–adapted to the collective operations implementation in IBM MPI. When `MPI_COMM_WORLD` is used `MPI_Bcast` and `MPI_Reduce` utilize the specialized low-latency network for collective communication. However, when custom communicators are used collective communication is handled over the point-to-point network which has a higher latency. The exact algorithm deployed in this case depends on the message size and the shape of the communicator, e.g. if it is rectangular. In our case the shape of the communicator guarantees that optimal algorithms, which e.g. benefit from deposit bits, are employed on the point-to-point network. Consequently, the latency is only slightly higher compared to the latency of the `MPI_COMM_WORLD` communicator (3.35 μs, plus 90 ns per hop versus 2.5 μs).

Once the communicators have been set up the master of subgroup 0 becomes the super-master. At program initialization, each master process immediately starts computations on bootstrap replicates or ML searches without communicating with the super-master. Every time a master has completed the computations on a tree it sends a message to the super-master and locally stores the tree in a list. This message contains the number of trees that have been computed so far by this specific master. Every time the super-master receives such a message it checks if the total number of trees specified by the user (20 in our example) has already been computed. If that is the case, the super-master sends a termination message to all other master processes. When a master receives the termination message it sends all locally stored trees to the super-master, which prints them to file. Thereafter, each worker terminates along with the respective worker processes. When all tree topologies have been written to file, the super-master exits as well. The above modification avoids the perturbation of fine-grained work scheduling at the super-master, since the actual tree topologies are only sent at the end of the computation. In theory it could occur that more trees than necessary are computed if two masters finish tree computations at the same time. However, we have not observed such a case in our experiments because the probability of such an event is low. In addition, the computation of potentially more trees than requested by the user within the same amount of time does not represent a disadvantage. The check for pending master/super-master communication messages has been integrated into the master function that schedules computations of partial likelihood vectors to workers. We chose to integrate the check into this method because it is by far the most frequently invoked ML function and thereby provides a fine enough granularity to rapidly detect master/super-master communication requests.

# SEQS	# BP	# PATT
50	5,000	3,066
50	50,000	23,385
50	500,000	216,025
250	250,000	202,482
250	500,000	403,581

Table 1: Number of distinct alignment patterns in the dataset subsamples

Figure 2 outlines a typical setup with 4 masters that use 3 worker processes each for ML computations. Thick black arrows indicate frequent fine-grained ML communications within each master-worker group. Dotted thin lines show the infrequent and less time-critical communications between the super-master and the remaining masters.

4. EXPERIMENTAL SETUP AND RESULTS

In this section, we describe the experimental setup and platforms used (Section 4.1). We also provide performance data for the fine-grained parallelization in Section 4.2 as well as for the hybrid parallelization (Section 4.3). Finally, in Section 4.4 we give execution times for analyses on the complete biological datasets.

4.1 Experimental Setup

To test the scalability of our parallelization we used two large and challenging real-world datasets:

1. A multi-gene alignment of 2,182 mammalian sequences with 51,089 base-pairs that comprises data from 67 different genes. Despite the fact that a few ML trees could already be computed with RAxML on a 4-way AMD Opteron, the execution times do not allow for a full bootstrap analysis. Large-scale analyses of mammalian phylogenies have recently received considerable attention (including the popular press), since they can be used for applications such as dating the rise of present-day mammals [5].

2. An alignment of 212 sequences and 566,470 base-pairs. This dataset contains genotype data for non-redundant SNPs on the human chromosome 1 (a so called haplotype map) in sorted order for 210 unrelated individuals in the HapMap project [9]. A chimpanzee as well as a gorilla sequence have been added to this alignment as outgroups. Each SNP corresponds to two positions within the alignment representing the two alleles and allowing for heterozygosity in the human population. A phylogenetic analysis of this dataset represents a novel approach where individual humans are represented as leaves on a phylogenetic tree that is reflective of their ancestral history. The combination of genotypic and phenotypic data on phylogenetic trees makes new types of correlative inference possible that differ from standard linear approaches [19]. Phylogenetic trees inferred on BG/L could be combined and

# SEQS	# BP	# Workers									
		1	3	7	15	31	63	127	255	511	1,023
50	5,000	1,400s	498s	226s	120s	67s	44s	33s	31s	29s	29s
50	50,000	14,326s	4,653s	2,008s	948s	472s	252s	142s	87s	60s	48s
50	500,000		32,659s	14,187s	6,531s	3,055s	1,533s	798s	436s	256s	169s
250	250,000					22,981s	11,495s	6,289s	3,762s	2,381s	1,689s
250	500,000					145,739s	70,617s	35,056s	18,025s	9,375s	5,105s

Table 2: Absolute run-times on BlueGene/L

# SEQS	# BP	# Workers						
		1	3	7	15	31	63	127
50	5,000	664s	182s	92s	62s	59s	72s	120s
50	50,000	7,706s	2,332s	937s	390s	218s	157s	181s
50	500,000	51,569s	17,084s	7,766s	3,462s	1,717s	884s	440s
250	500,000				192,591s	90,901s	45,306s	23,111s

Table 3: Absolute run-times on AMD Opteron

correlated with publicly available biological phenotype data from the same human samples [36].

In order to test scalability on various dataset-sizes we extracted appropriate sub-alignments from the above datasets. From the mammalian alignment we extracted sub-alignments containing 50 sequences with 5,000 and 50,000 base-pairs each. The human chromosome alignment was sub-sampled to 50 sequences with 500,000 base pairs. Additionally we extracted alignments with 250 sequences and lengths of 250,000 and 500,000 base-pairs from the same HapMap database we derived the 210 sequences dataset from. Table 1 provides the number m' of distinct patterns (# PATT) for each subsample of the mammalian and haplotype alignments. Note that m' reflects the length of the respective `for`-loops in each test dataset.

For the assessment of scalability on a more common cluster architecture we used a system of 32 4-way AMD 2.4 GHz Opteron 850 processors with 8GB of main memory per node which are interconnected by Mellanox Technologies MT23108 Infiniband host channel adapters and an MTEK43132 Infiniband switch (latency for small messages $\leq 5\mu s$). The BG/L system we used is a one–rack machine with 1,024 nodes (2,048 CPUs) and a peak performance of 5.734 Teraflops.

As RAxML uses randomized algorithms for the creation of starting trees and for bootstrapping, the run-times as well as the results of the tree search differ for every individual program run. In order to obtain reproducible results and run-times, we used a fixed seed for the random number generator.

4.2 Scalability of Fine-grained Parallelism

We provide speedup values for the fine-grained parallelization based on the number of workers for various dataset sizes in Figures 3 through 8. Absolute run-times for BlueGene/L and AMD Opteron are given Tables 2 and 3, respectively. Note that the total run-time for different datasets does not scale linearly with the number of sequences in the dataset or the sequences' length: the search algorithm will take different paths through the search space for different input datasets. Experimental results regarding the scalability of RAxML with respect to the number of taxa are provided in [29].

Plots 3 and 4 depict speedups for mammalian subsets with 50 sequences consisting of 3,066 and 23,385 alignment patterns, respectively. The poor performance for more than 15 workers shown in Figure 3 can be explained by the rather small problem size. Note that performance on the Opteron cluster is slightly super-linear up to 31 worker processes in Figure 4 because of an increased cache efficiency. Another interesting observation is that BG/L scales significantly better for this setting with more than 63 workers. However, this was expected as the BlueGene system provides a better communication to computation ratio due to it's very low latency network and only moderate computing power per CPU. Thus it scales better even for small problem sizes.

In Diagram 5 we depict speedup values for a 50-taxa haplotype subset with 216,025 distinct patterns. Since the `for`-loops for this alignment are longer by one order of magnitude, the communication to computation ratio improves roughly by a factor of 10. As a result, the scalability both on the Opteron cluster as well as on BG/L is nearly linear.

Finally, in Figures 6, 7 and 8 we show how the program scales on 250 haplotype sequences with 202,482 and 403,581 distinct patterns, respectively, up to 1,024 CPUs. Note that, plots 7 and 8 provide relative scalability compared to a run with 15 workers. This is due to the fact that we were not able to execute the program with a smaller number of workers because of memory shortage. However, as shown in Figures 4 and 5, the program scales linearly for up to 15 workers for these numbers of patterns.

In general, one can conclude that the scalability of the fine-grained parallelism directly depends on the length of the alignment or rather the number of distinct patterns, as the computation/communication ratio increases with the length of the likelihood vectors. The number of sequences and thus the number of nodes in the tree influence performance only slightly. On the one hand, the computation of partial likelihood vectors benefits from the increased number of internal nodes, as more vector updates can be aggregated into one single call and thus again improve computation/communication ratio. On the other hand, the total number of branches whose lengths need to be optimized also increases with the number of sequences. However, this function is very costly with respect to communication because of the frequent reduce operations which need to be performed on every iteration of the Newthon-Raphson procedure. Furthermore, the cost of these reduce operations also increases with the number of processes involved. In total the positive and negative effects of increasing the number of sequences

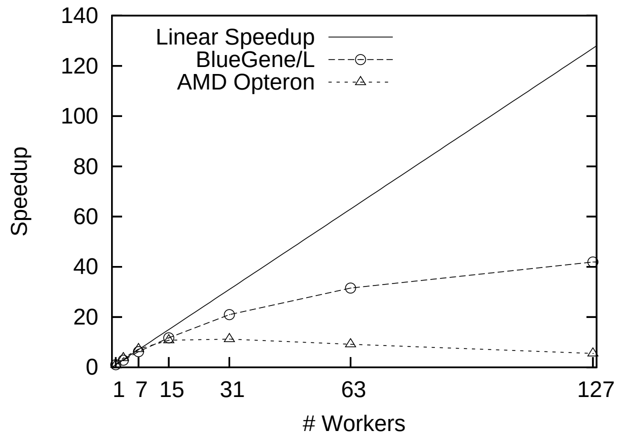

Figure 3: Speedup on 50 mammalian sequences with 3,066 distinct patterns

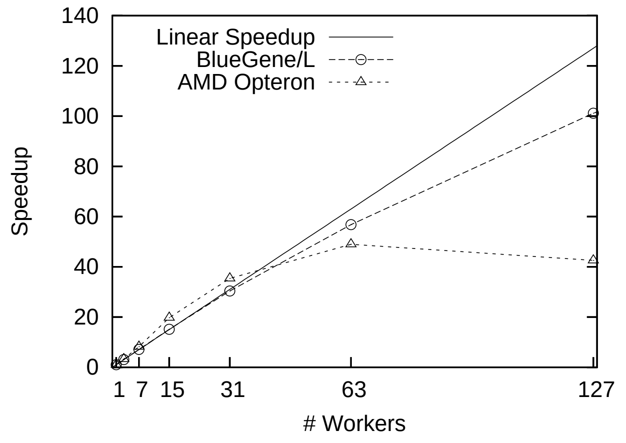

Figure 4: Speedup on 50 mammalian sequences with 23,385 distinct patterns

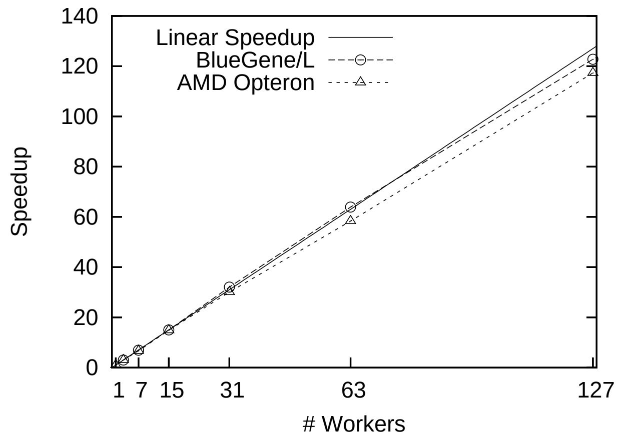

Figure 5: Speedup on 50 haplotype sequences with 216,025 distinct patterns

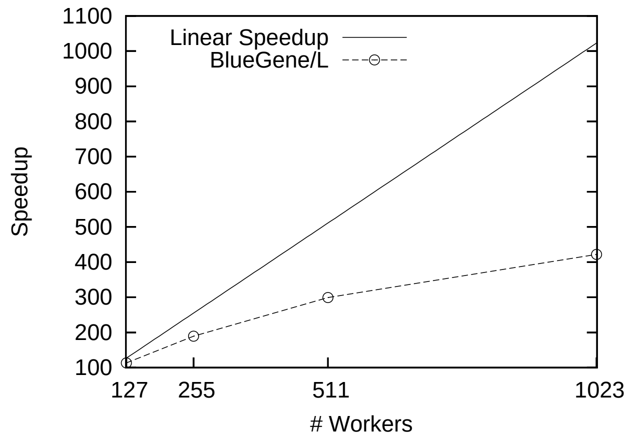

Figure 6: Speedup on 250 haplotype sequences with 202,482 distinct patterns

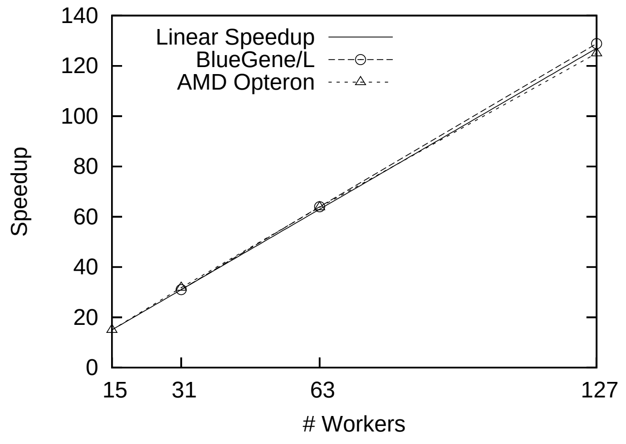

Figure 7: Speedup on 250 haplotype sequences with 403,581 distinct patterns on 15–127 workers

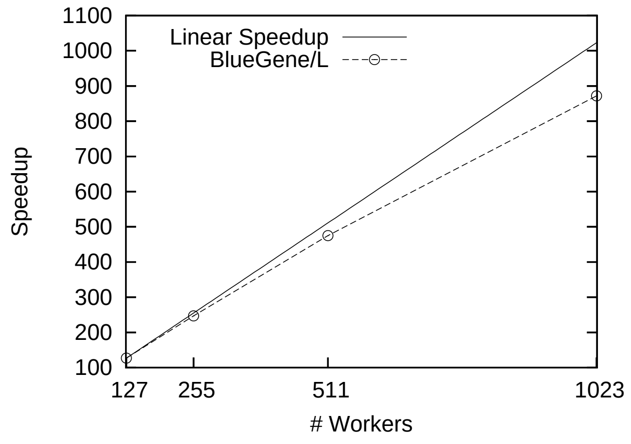

Figure 8: Speedup on 250 haplotype sequences with 403,581 distinct patterns on 127–1,023 workers

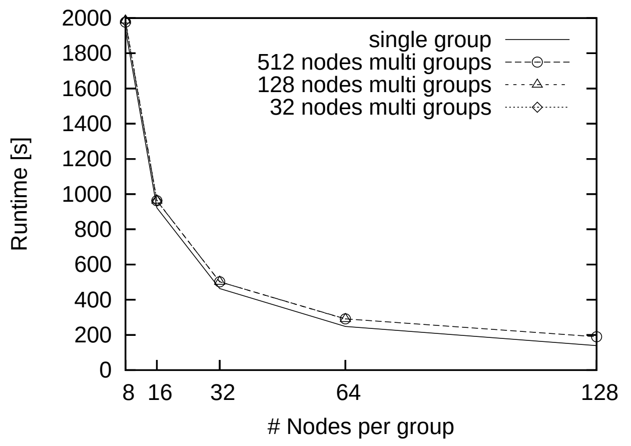

Figure 9: Execution times of multiple groups setup on mammalian subsample with 50 sequences and 23,385 distinct patterns

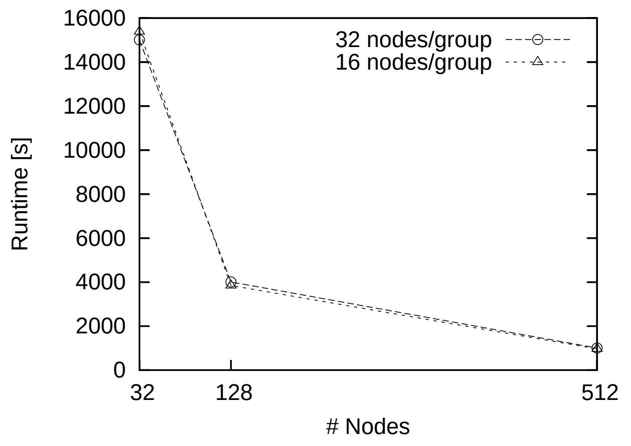

Figure 10: Scalability of coarse-grained parallelization: Execution times for 32 distinct tree searches on mammalian subsample with 50 sequences and 23,385 distinct patterns

# total nodes	# nodes/group				
	8	16	32	64	128
32	1,984s	963s			
128	1,986s	964s	502s	291s	
512	1,977s	963s	502s	291s	190s

Table 4: Absolute execution times for multiple groups on BlueGene/L

# total nodes	# nodes/group	
	16	32
32	15,387s	15,008s
128	3,850s	4,006s
512	963s	1,005s

Table 5: Absolute execution times for 32 distinct tree searches on BlueGene/L

almost keep balance with a slight bias to improve performance.

4.3 Scalability of Hybrid Parallelism

We assessed the performance of the coarse–grained parallelization using the mammalian dataset with 50 sequences and 23,385 distinct patterns. The experiments were conducted on BlueGene/L partitions of 32, 128, and 512 nodes. For the sake of completeness, Tables 4 and 5 provide the absolute run-times of these experiments.

Figure 9 shows execution times for individual tree inferences using groups of 8, 16, 32, 64, and 128 nodes. The straight black line shows the time for a single master–worker group (see Table 2). The remaining three graphs show execution times for multiple master–worker groups on the aforementioned BG/L partitions which have been split into 4, 8, 16, 32, and 64 groups (where applicable) using `MPI_Comm_split` as described in Section 3.2.

As we expected, the run-times observed for multiple groups setup are slightly higher than the corresponding run-times of a single master–worker group. This is due to the fact that on multi–group setups all messages are sent over the higher latency peer-to-peer network while a single group can utilize the faster specialized collective network (see section 3.2).

Figure 9 shows that the total number of distinct groups does not influence the run-time of the individual tree searches. This means that communication between masters and super–master is infrequent enough to not influence the fine–grained parallelism within each group.

Figure 10 provides the total run-times for 32 distinct tree searches on 32, 128, and 512 nodes. The nodes have been split into groups of 16 and 32 nodes. So for example in the case of 512 nodes and 32 nodes per group, 16 masters with their private set of 31 workers perform 32 distinct tree searches in parallel, two for each group. The plot shows that the total execution time decreases linearly with an increasing number of total nodes used for computation. Furthermore, one can see that groups of 16 nodes perform slightly better than groups of 32 nodes — as expected, given the absolute run-times for single groups in Table 2.

4.4 Full Analyses

On the mammalian dataset (2,182 sequences, 51,089 base pairs) we were able to conduct 30 bootstrap analyses on 2,048 CPUs (1,024 nodes in virtual node mode) within 41 hours by using 8 groups with 255 workers per group.

On the haplotype alignment (212 sequences, 566,470 base pairs) we performed a full phylogenetic analysis: Initially we conducted ML-searches for the best-scoring tree on 1,024 CPUs (1,024 nodes in co-processor mode) in single-group configuration. Within 14 hours we were able to complete 7 distinct tree searches. The bootstrap analysis was performed on 2,048 CPUs in virtual node mode which have been divided into 8 groups. It took 26 hours to complete 64 bootstraped tree searches.

We also used the full haplotype alignment to test the scalability of our approach in virtual node mode. The average run-time of the aforementioned 7 tree searches in co-processor mode is 6,838 seconds. The average execution time on 512 nodes in virtual node mode, i.e. 1,024 CPUs, amounts to 7,000 seconds. The efficiency achieved is thus

97.69%.

The above results demonstrate that a full real-world analysis of these challenging datasets is feasible on BlueGene/L. Additionally, as we have shown in the experiments, the fine-grained parallelism is capable to efficiently exploit hundreds of CPUs. Provided that the alignments used for phylogenetic analyses continue to grow and that real-life problems require hundreds of tree searches and bootstrap runs to be conducted, we may expect that our software will be able to efficiently exploit forthcoming petascale systems.

5. CONCLUSION AND FUTURE WORK

We have presented a generally applicable porting strategy for ML-based phylogeny programs to the BG/L. Moreover, we have demonstrated that our approach scales well up to 2,048 processors on the BG/L and up to 128 CPUs on a common cluster architecture. Performance has been assessed by conducting tree searches on the two largest DNA alignments analyzed under ML to date, to the best of our knowledge.

Being able to handle and scale well on such large datasets, the presented version of RAxML may open up new perspectives towards the computation of whole-genome phylogenies. Due to the steadily accelerating accumulation of sequence data because of novel sequencing techniques the proposed parallelization scheme for ML provides a viable solution for future computational needs in phylogenetics.

Future work will cover biological production runs on those two datasets as well as a full porting of all substitution models offered by RAxML to BG/L. We also plan to execute an appropriate fraction of likelihood computations at the master processes in order to further improve parallel efficiency. Moreover, we will investigate how to accelerate the likelihood functions by additional BG/L-specific low-level code optimizations. We intend to assess the performance of our approach compared to the standard OpenMP-version on current multi-core architectures. We also plan to devise a method which, given a dataset, will automatically determine the optimal number of workers for fine-grained parallelism.

Finally, we will integrate an accelerated algorithmic technique for bootstrapping that has the potential to yield accelerations of one order of magnitude while returning support values that are highly correlated (coefficient 0.94-0.96) to the "standard" technique. This will allow for full ML analyses on even larger datasets in the near future.

6. ACKNOWLEDGEMENTS

We would like to thank Olaf Bininda-Emonds at the University of Jena as well as Andrew Johnson and Daniel Janies from Ohio State University for providing us their sequence data to assess RAxML performance on IBM BlueGene.

This research is supported in part by the National Science Foundation under CNS-0521568 and CCF-0431140.

7. REFERENCES

[1] G. Almasi, C. Archer, J. G. Castanos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. D. Steinmacher-Burow, W. Gropp, and B. Toonen. Design and implementation of message-passing services for the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3), 2005.

[2] G. Almasi, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of MPI collective communication on BlueGene/L systems. In *Proc. of SuperComputing 2005*, pages 253–262, 2005.

[3] D. Bader, B. Moret, and L. Vawter. Industrial Applications of High-Performance Computing for Phylogeny Reconstruction. In *Proc. of SPIE ITCom*, volume 4528, pages 159–168, 2001.

[4] D. Bader, U. Roshan, and A. Stamatakis. *Advances in Computers*, chapter Computational Grand Challenges in Assembling the Tree of Life: Problems & Solutions. Elsevier, 2006.

[5] O. R. P. Bininda-Emonds, M. Cardillo, K. E. Jones, R. D. E. MacPhee, R. M. D. Beck, R. Grenyer, S. A. Price, R. A. Vos, J. L. Gittleman, and A. Purvis. The delayed rise of present-day mammals. *Nature*, 446:507–512, 2007.

[6] 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.

[7] F. Blagojevic, D. S. Nikolopoulos, A. Stamatakis, and C. D. Antonopoulos. RAxML-Cell: Parallel Phylogenetic Tree Inference on the Cell Broadband Engine. In *Proc. of IPDPS 2007*, 2007.

[8] B. Chor and T. Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(1):97–106, 2005.

[9] T. I. H. Consortium. The International HapMap Project. *Nature*, 426:789–796, 2003.

[10] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. Greengenes, a Chimera-Checked 16S rRNA Gene Database and Workbench Compatible with ARB. *Appl. Environ. Microbiol.*, 72(7):5069–5072, 2006.

[11] Z. Du, F. Lin, and U. Roshan. Reconstruction of large phylogenetic trees: a parallel approach. *Computational Biology and Chemistry*, 29(4):273–280, 2005.

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

[13] X. Feng, K. W. Cameron, C. P. Sosa, and B. Smith. Building the Tree of Life on Terascale Systems. In *Proc. of IPDPS 2007*, 2007.

[14] L. Ganzert, G. Jurgens, U. Munster, and D. Wagner. Methanogenic communities in permafrost-affected soils of the Laptev Sea coast, Siberian Arctic, characterized by 16S rRNA gene fingerprints. *FEMS Microbiology Ecology*, 59(2):476–488, 2007.

[15] M. Gottschling, A. Köhler, E. Stockfleth, and I. Nindl. Phylogenetic analysis of beta-papillomaviruses as inferred from nucleotide and amino acid sequence data. *Mol Phylogenet Evol.*, 42(1):213–222, 2007.

[16] M. Gottschling, A. Stamatakis, I. Nindl, E. Stockfleth, A. Alonso, L. Gissmann, and I. G. Bravo. Multiple evolutionary mechanisms drive papillomavirus diversification. *Mol. Biol. Evol.*, 2007.

[17] G. W. Grimm, S. S. Renner, A. Stamatakis, and V. Hemleben. A Nuclear Ribosomal DNA Phylogeny of Acer Inferred with Maximum Likelihood, Splits

Graphs, and Motif Analyses of 606 Sequences. *Evolutionary Bioinformatics Online*, 2:279–294, 2006.

[18] S. Guindon and O. Gascuel. A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood. *Syst. Biol.*, 52(5):696–704, 2003.

[19] F. Habib, A. D. Johnson, R. Bundschuh, and D. Janies. Large scale genotype-phenotype correlation analysis based on phylogenetic trees. *Bioinformatics*, 2007.

[20] T. Jukes and C. Cantor. *Evolution of protein molecules*, chapter III, pages 21–132. Academic Press, New York, 1969.

[21] O. Lascu, N. Allsopp, P. Vezolle, J. Follows, M. Hennecke, F. Ishibashi, M. Paolini, S. Prakash, H. Reddy, C. Sosa, A. Tabary, and D. Quintero. *Unfolding the IBM eServer Blue Gene Solution*. IBM, 2006.

[22] R. E. Ley, J. K. Harris, J. Wilcox, J. R. Spear, S. R. Miller, B. M. Bebout, J. A. Maresca, D. A. Bryant, M. L. Sogin, and N. R. Pace. Unexpected Diversity and Complexity of the Guerrero Negro Hypersaline Microbial Mat. *Appl. Envir. Microbiol.*, 72(5):3685 – 3695, May 2006.

[23] B. Minh, L. Vinh, A. Haeseler, and H. Schmidt. pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21(19):3794–3796, 2005.

[24] B. Minh, L. Vinh, H. Schmidt, and A. Haeseler. Large Maximum Likelihood Trees. In *Proc. of the NIC Symposium 2006*, pages 357–365, 2006.

[25] PBPI web-page, http://people.cs.vt.edu/˜fengx/pbpi/overview.htm.

[26] C. Robertson, J. Harris, J.R.Spear, and N. Pace. Phylogenetic diversity and ecology of environmental Archaea. *Current Opinion in Microbiology*, 8:638–642, 2005.

[27] D. E. Soltis, M. A. Gitzendanner, and P. S. S. PS. A 567-taxon data set for angiosperms: The challenges posed by bayesian analyses of large data sets. *INTERNATIONAL JOURNAL OF PLANT SCIENCES*, 168(2):137–157, 2007.

[28] A. Stamatakis. *Distributed and Parallel Algorithms and Systems for Inference of Huge Phylogenetic Trees based on the Maximum Likelihood Method*. PhD thesis, Technische Universität München, Germany, October 2004.

[29] A. Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[30] A. Stamatakis, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos. Exploring new Search Algorithms and Hardware for Phylogenetics: RAxML meets the IBM Cell. *Journal of VLSI Signal Processing Systems,*, 2007. in press.

[31] A. Stamatakis, T. Ludwig, and H. Meier. Parallel Inference of a 10.000-taxon Phylogeny with Maximum Likelihood. In *Proc. of Euro-Par 2004*, pages 997–1004, September 2004.

[32] A. Stamatakis, T. Ludwig, and H. Meier. RAxML-III: A Fast Program for Maximum Likelihood-based Inference of Large Phylogenetic Trees. *Bioinformatics*, 21(4):456–463, 2005.

[33] A. Stamatakis, H. Meier, and T. Ludwig. New Fast and Accurate Heuristics for Inference of Large Phylogenetic Trees. In *Proc. of IPDPS2004*, HICOMB Workshop, Proceedings on CD, Santa Fe, New Mexico, 2004.

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

[35] C. Stewart, D. Hart, D. Berry, G. Olsen, E. Wernert, and W. Fischer. Parallel Implementation and Performance of FastDNAml – A Program for Maximum Likelihood Phylogenetic Inference. In *Proc. of SC2001*, Denver, CO, Nov. 2001.

[36] B. E. Stranger, M. S. Forrest, A. G. Clark, M. J. Minichiello, S. Deutsch, R. Lyle, S. Hunt, B. Kahl, S. E. Antonarakis, S. Tavare, P. D. P., and E. T. Dermitzakis. Genome-wide associations of gene expression variation in humans. *PLoS Gene*, 1(6), 2005.

[37] Z. Yang. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites. *J. Mol. Evol.*, 39:306–314, 1994.

[38] D. Zwickl. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion*. PhD thesis, University of Texas at Austin, April 2006.