Novel Parallelization Schemes for Large-Scale Likelihood-based Phylogenetic Inference

Alexandros Stamatakis, Andre J. Aberer
The Exelixis Lab, Scientific Computing Group
Heidelberg Institute for Theoretical Studies
Heidelberg, Germany
Alexandros.Stamatakis@h-its.org, Andre.Aberer@h-its.org

***Abstract*—The molecular data avalanche generated by novel wet-lab sequencing technologies allows for reconstructing phylogenies (evolutionary trees) using hundreds of complete genomes as input data. Therefore, scalable codes are required to infer trees on these data under likelihood-based models of molecular evolution. We recently introduced a checkpointable and scalable MPI-based code for this purpose called RAxML-Light and are currently using it for several real-world data analysis projects. It turned out that the scalability of RAxML-Light is nonetheless still limited because of the fork-join parallelization approach that is deployed. To this end, we introduce a novel, generally applicable, approach to computing the phylogenetic likelihood in parallel on whole-genome datasets and implement it in ExaML (Exascale Maximum Likelihood). ExaML executes up to 3.2 times faster than RAxML-Light because of the more efficient parallelization and communication scheme, while implementing exactly the same tree search algorithm. Moreover, the new parallelization approach exhibits lower code complexity and a more appropriate structure for implementing fault tolerance with respect to hardware failures.**

***Keywords*-phylogenetics; likelihood; parallelization; MPI;**

I. Introduction

Next generation sequencing technologies allow biologists to generate an unprecedented amount of molecular raw data that needs to be analyzed. Sequencing a genome has become relatively cheap at present, hence, the main challenge consists in analyzing the data.

Our research group is currently involved in the 1KITE (one thousand insect transcriptome) sequencing project www.1kite.org that was initiated in 2011. To date, 100 full insect transcriptomes have already been sequenced and we expect the sequencing of all 1000 transcriptomes to be completed by the end of 2012.

Reconstructing phylogenetic trees on such datasets to disentangle evolutionary relationships under complex statistical models that deploy the likelihood criterion [1], be it in a Maximum Likelihood [1] or Bayesian setting (see, e.g., [2] for an early paper), requires a huge amount of computational resources. This holds true, both for the amount of memory, as well as the CPU time that is required. Just computing the likelihood score on a DNA dataset with 20,000,000 base pairs (also referred to as *sites*) and 1,500 taxa (sequences) under a simple model of DNA substitution that does not even take into account rate heterogeneity among sites [3] already requires 1TB of RAM [4].

To address these challenges, we recently released a tool called RAxML-Light [4] that is checkpointable and scales across several shared-memory nodes using MPI (Message Passing Interface). In a proof-of-concept execution we demonstrated that, RAxML-Light can infer trees on datasets such as the above (1500 taxa, 20,000,000 sites) using more than 600 cores. However, the limitations of the classical fork-join approach for computing the likelihood in parallel soon became apparent, in particular for partitioned datasets.

By partitioned datasets we refer to multi-gene or whole genome datasets that are sub-divided into distinct partitions (sets of alignment sites/columns). When datasets are partitioned, likelihood model parameters such as the α shape parameter of the Γ model that accommodates rate heterogeneity among sites [3], the branch lengths, or the rate parameters in a GTR (General Time Reversible) nucleotide substitution matrix [5] are optimized (Maximum Likelihood) or sampled (Bayesian inference) independently for each data partition. Frequently, such large datasets are partitioned on a per-gene basis, that is, likelihood model parameters are inferred separately for each gene. Often, DNA datasets are also partitioned with respect to the 1st, 2nd, and 3rd codon position. Biologically, partitioning such large datasets 'makes sense', because different genes evolve at different speeds due to heterogeneous evolutionary pressures. Identifying a 'good' or optimal partitioning scheme (i.e., how a dataset is best partitioned) represents a topic of current research [6], [7] and is outside the scope of this paper. Here, we assume that, a fixed partition scheme is given. Based on our interactions with the large RAxML user community, we observed that biologists typically desire to infer trees on ever growing datasets. At the same time, they also wish to increase the number of data partitions such as to model evolutionary processes in a more realistic way.

Given this trend toward partitioned analyses of whole-genome datasets, we have radically re-designed the parallelization scheme that is used in RAxML and developed a code called ExaML (Exascale Maximum Likelihood) to accommodate these new computational requirements. ExaML

is up to three times faster than RAxML-Light because we drastically reduced the number of collective communication operations, and, more importantly, the amount of data that needs to be transferred during each collective communication call that triggers a parallel region. We achieved this by abandoning the classical fork-join approach that used a dedicated master process for orchestrating the tree search and the associated likelihood computations across all partitions and processes in favor of a de-centralized approach. The fundamental idea of the decentralized approach is that each process executes a local, yet consistent, copy of the search algorithm. Processes only need to communicate with each other when the overall likelihood score (or the first and second derivative of the likelihood function) over all partitions needs to be computed.

The code is available as open-source code at https://github.com/stamatak/ExaML and has already been used at production level for the 1KITE project (see above) as well as for a bird phylogenomics project at various supercomputing facilities (San Diego Supercomputer Center, Munich Supercomputing Center).

The remainder of this paper is organized as follows: In Section II we briefly review related work on parallelizing the likelihood function using the fork-join paradigm. Thereafter, we provide details on the new parallelization scheme in Section III. In the subsequent Section IV we describe the experimental setup and provide performance results. We conclude in Section V.

II. Related Work

To the best of our knowledge, the current paper is the first to identify performance problems pertaining to the fork-join approach for parallel likelihood computations and to propose an alternative solution. Hence, our review of related work covers previous approaches to parallelizing likelihood-based codes using the fork-join approach as well as work related to load balance issues that arise with partitioned datasets.

A detailed overview over the fork-join approach to parallelizing likelihood computations on multi-core systems, distributed memory clusters, and accelerator cards (e.g. GPUs) is provided in [8].

The first MPI-based fork-join implementations were introduced with PBPI [9], a simple proof-of-concept program for Bayesian phylogenetic inference, and a predecessor of RAxML-Light that was specifically tuned for the IBM BlueGene/L supercomputer [10], [11]. This RAxML code version also forms part of the SPEC (Standard Performance Evaluation Corporation) MPI benchmark suite [12].

The fork-join paradigm is also used *implicitly* in all parallel implementations of the likelihood function that deploy OpenMP for multi-core systems [13], [14], [15]. It is also used *explicitly* in the PThreads version of RAxML [16]. Moreover, it represents the standard approach for orchestrating/off-loading likelihood computations to GPUs [17], [18], [19]. In this scenario, a CPU will steer the tree search and offload the likelihood computations to the GPU(s). In other words, the GPU does not need to be aware of the fact that, operations are conducted on a tree data structure. Instead, the GPU simply executes the floating point operations that are required for calculating the likelihood. Note that, widely-used codes such as MrBayes [20], PHYML [21], GARLI [14], or RAxML [22] spend more than 90% of total execution time in likelihood calculations. Hence, computationally, all of those standard codes face similar challenges.

An issue that has rarely been addressed is that of load-balance and data distribution for partitioned analyses. In [23] we showed that, when datasets are partitioned, changes to per-partition model parameters need to be proposed and evaluated simultaneously for all partitions to increase parallel efficiency under the fork-join approach. Improved efficiency is obtained by reducing the number of parallel regions and increasing the amount of work that is conducted per parallel region. This also holds true for the novel parallelization scheme we present here. In the Bayesian context, this implies that, MCMC proposals to change, for instance, the α shape parameter of the Γ model, need to be applied to and be evaluated simultaneously for all partitions. This would also require a modification of the Hastings ratio calculations. To date, this type of simultaneous proposals/parameter changes, is only implemented in the parallel versions of RAxML (PThreads) and RAxML-Light (PThreads & MPI).

Another critical issue pertaining to parallel analysis of large, partitioned datasets with 100 or more partitions is data distribution. As outlined in [24] and [4], significant performance improvements (up to one order of magnitude) can be achieved by distributing/assigning partitions monolithically to processors, rather than performing data distribution at a higher granularity in a cyclic, site-by-site fashion. However, achieving an optimally balanced data distribution when partitions are assigned monolithically to processors is NP-hard, because it is equivalent to the multi-processor scheduling problem [24]. The option to distribute data on a per-partition basis is available in RAxML-Light and ExaML (`-Q` option). Again, we are not aware of any other likelihood-based code that also addresses this problem.

Despite our previous efforts to improve load balance and data distribution in RAxML-Light, parallel efficiency on partitioned datasets is still limited by the communication effort that comes with the fork-join approach. We describe and address this problem in the following Section.

III. Parallelization Scheme

Initially we discuss the limitations of the fork-join approach. Thereafter, we present the new, de-centralized approach in Section III-B.

A. Limitations of the Fork-Join Approach

The fork-join approach to parallelizing likelihood calculations was introduced several years ago (see, e.g., [13] for one early paper from 2005), when input alignments still used to be small by current standards. In many cases, input alignments contained at most 10 to 20 genes (partitions) and only a few phylogenetic inference programs offered the possibility to conduct partitioned analyses.

Since execution times of likelihood-based phylogenetic inference programs are largely dominated by likelihood calculations, the likelihood functions represent the natural candidates for parallelization. Parallelization is fairly straight-forward, since the likelihood model assumes that sites evolve independently [1]. This means that, given a tree for an alignment of 100 sites (alignment columns), the corresponding 100 per-site log likelihood scores can be computed independently and simultaneously in parallel. To obtain the overall log likelihood score for the given tree, one only needs to execute a final reduction operation on the 100 per-site log likelihood scores.

Bayesian inference programs typically use two main functions: one for computing conditional likelihood arrays at inner nodes of the tree (also called ancestral probability vectors) according to the Felsenstein pruning algorithm [1] and one for computing the overall log likelihood score at the virtual root of the tree (including the parallel reduction operation mentioned above). Maximum likelihood programs typically use an additional, third function, for computing the first and second derivative of the phylogenetic likelihood which is required for direct numerical optimization of branch lengths to maximizing the likelihood (typically using the Newton-Raphson procedure). Note that, this also requires parallel reduction operations because the overall first and second derivatives of the likelihood score (across all sites) are required. For a more detailed description of likelihood computations please refer to [8].

In the basic fork-join approach, the tree search (or MCMC proposal mechanism) is executed by a dedicated master thread/process, which is the only process that maintains a tree data structure and the current state of the search. The worker processes are agnostic regarding the semantics of the tree search and only execute one of the three likelihood functions (overall likelihood, conditional likelihood vectors, derivatives of the likelihood) on the fraction of the data (proportion of sites) that has been assigned to them. Thus, every time the master process needs to re-calculate the likelihood of the tree (either because the tree topology was changed, or a model parameter such as α has been altered), it triggers a parallel region. These parallel regions are terminated, either by a barrier (conditional likelihood arrays) or by a parallel reduction operation (overall likelihood and derivatives of the likelihood). Note that, these parallel regions are extremely fine-grain and invoked frequently. The PThreads version of

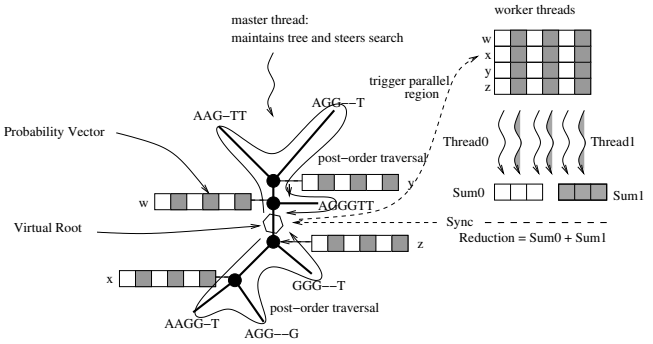


Figure 1: Schematic outline of the fork-join parallelization approach with one master and two worker threads.

RAxML executes between 100,000 - 800,000 barriers within 10 seconds of run time on a 16 core system [25] on medium-sized datasets.

The type and sequence of operations that worker processes need to execute are specified in a so-called traversal descriptor (the Beagle library uses an analogous concept [15]). Assume, for instance, that the value of the α shape parameter has been changed. In this case, the conditional likelihood arrays of the entire tree need to be re-computed via a recursive post-order traversal to correctly calculate the resulting likelihood. This post-order is computed at the master node and stored in the traversal descriptor. In the shared-memory case, the worker threads can simply access this pre-computed traversal descriptor via shared memory. On distributed memory systems, one needs to explicitly broadcast the traversal descriptor (see on-line supplement of [4] for details) for each parallel region. Also note that, when one of the model parameters is changed (e.g., a rate in the GTR matrix or the α shape parameter), they also need to be broadcasted to all worker processes. A schematic representation of the fork-join approach is provided in Figure 1.

The fork-join approach, in particular under the distributed memory setting we consider here, works well as long as entering and exiting parallel regions is latency-bound. However, because of the high number of extremely fine-grain parallel regions, this approach becomes inefficient when entering and exiting parallel regions (where all processes wait for communication without executing any useful likelihood computations) is bandwidth-bound. Unfortunately, this is exactly what happens for partitioned analyses:

Assume that, we are analyzing a large, partitioned dataset with 1000 partitions. When trying to optimize, for instance, the 1000 distinct α shape parameters (one per partition), the process that steers this numerical optimization (in the maximum likelihood case) or proposes new α values for all 1000 partitions (in the Bayesian case) is the master process. Thus, for each set of 1000 new α values that we want to evaluate, we need to broadcast an array of 1000

double values to all worker processes. Thus, as we show in Section IV, triggering parallel regions becomes bandwidth-bound, instead of latency-bound and thereby substantially decreases parallel efficiency. This is a problem which is inherent to the general fork-join based parallelization approach.

Therefore, we require a fundamentally different approach to reduce the amount of data that is being communicated per parallel region. At the same time we need to decrease the overall number of collective communication operations to an absolute minimum.

B. The De-Centralized Approach

The solution to improving parallel performance is fairly straight-forward. Instead of using a fork-join paradigm, we deploy a de-centralized approach where each process executes a local, consistent copy of the tree search algorithm. By consistent we mean that, all processes operate on *exactly* the same tree topology with *exactly* identical model parameter values (GTR rates, branch lengths, α parameters, etc.). In this setting, all processes execute the same search steps on different parts of the data. To make sure that, all processes are always in the same state, they simply need to obtain the same overall values (across the entire alignment that is distributed across processors) for the log likelihood score of the tree and the first and second derivatives of the likelihood function. In other words, processes only need to communicate with each other for exchanging results (computed on the process-specific sub-set of the alignment) that are required to decide which topology to evaluate next in the course of the tree search (in contrast to the fork-join paradigm, where this decision is exclusively taken by the master process). This can easily be achieved by inserting two `MPI_Allreduce()` calls into the code. One call needs to be inserted in the function that computes the overall log likelihood and the other invocation into the routine that computes the derivatives of the likelihood function. In the Bayesian case, only a single `MPI_Allreduce()` needs to be integrated into the code, since computing the derivatives of the likelihood function is not required. In contrast to the fork-join approach we thus completely avoid broadcasting traversal descriptors and arrays with changed model parameters. Note that, in the fork-join approach `MPI_Reduce()` calls are required instead of `MPI_Allreduce()` invocations. Therefore, the parallel performance of our new approach solely depends on the efficiency of the `MPI_Allreduce()` implementation. It is important to note that, `MPI_Allreduce()` needs to yield exactly identical numerical values at all processors because otherwise processes may end up in inconsistent states (chose different trees).

A schematic outline of this de-centralized parallelization approach is provided in Figure 2. The Figure shows that each process has a copy of the tree data structure and computes the traversal order of the tree locally. It also shows how the input data (an alignment of 6 taxa with 6 sites/columns) is distributed among processes. The key differences to Figure 1 are: (i) only a single (instead of two) collective communication operation for computing the likelihood at the virtual root is required, (ii) no dedicated master thread/process is used, (iii) less data is communicated for obtaining the likelihood of the tree, and (iv) Figure 2 is less complex.

In fact, the reduced code complexity of our new approach represents an additional advantage. Except for some additional MPI calls to initially distribute the data, handle the CAT model of rate heterogeneity, and to integrate `MPI_Allreduce()` invocations into the three likelihood functions (evaluation, derivatives, conditional likelihoods), the code is identical to the sequential version of RAxML-Light. The ExaML source code needs less than 50 calls to MPI routines, while RAxML-Light uses more than 100 invocations.

The initial coding of the proof-of-concept ExaML version took A. Stamatakis (who has been developing RAxML for more than 10 years) a single day. Subsequent code extension (e.g., including the CAT model of rate heterogeneity and checkpointing) required another 3-4 days. This is far less coding effort than required for designing RAxML-Light. Moreover, the de-centralized approach was implemented directly in the sequential code and the only more complex modifications were required for initial data distribution using the algorithm developed in [24].

As such, it will be relatively easy to also parallelize other codes, such as, for instance MrBayes using the scheme introduced here. As stated before, the only difficulty regarding Bayesian codes (and other maximum likelihood codes as well) will be that, the proposal mechanism will have to be modified in such a way that it proposes simultaneous parameter changes for all partitions. Another advantage is that, one can completely avoid the concept and broadcasts of traversal descriptors. While traversal descriptors are usually relatively short (i.e., do not represent a full tree traversal containing all nodes, but just a partial tree traversal comprising 4-5 nodes on average; for details see supplement of [4]) broadcasting them for essentially every parallel region still induces a performance penalty.

Thus, ExaML yields better parallel efficiency at substantially lower code complexity.

IV. Experimental Setup and Results

In this Section, we assess the parallel efficiency of ExaML for the typical usage scenarios and compare it to RAxML-Light. We also examine the additional communication cost incurred by broadcasting the traversal descriptor.

A. Cluster System

As test platform we used our institutional cluster at the Heidelberg Institute for Theoretical Studies. The cluster

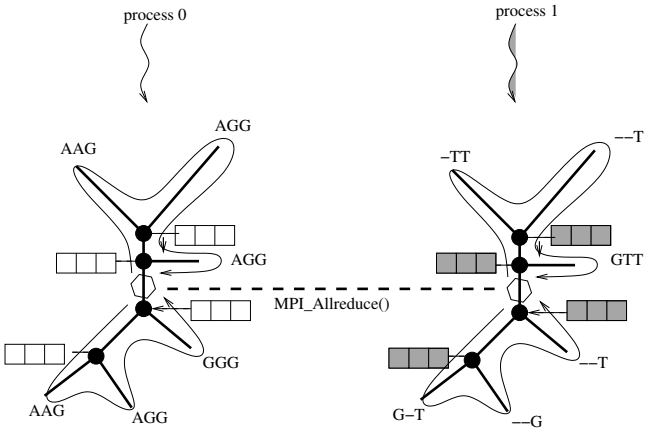


Figure 2: Schematic outline of the de-centralized parallelization approach with two processes.

comprises 50 AMD Magny-Cours nodes. Every node has 6 AMD Opteron 6174 processors with 8 cores each (48 cores per node). The per-core cache size is 512 kB. The nodes are connected via a QLogic Infinband interconnect. Four nodes out of 50 nodes are equipped with 256 GB of RAM, the remaining 46 nodes have 128 GB RAM. We compiled RAxML-Light and ExaML using the GNU gcc compiler (version 4.7.1) and performed runs with the mvapich2 (version 1.5) MPI implementation.

B. Test Datasets

As outlined in Section I, current phylogenetic inference software faces two major scalability challenges: (i) huge datasets in terms of taxa and/or alignment length, and (ii) datasets with an increasing number of data partitions.

For addressing challenge (i), we executed ExaML on a simulated DNA dataset that was used in [4] as an example for a large current dataset. It consists of 150 sequences and has a length of 20,000,000 base pairs. The total number of unique site patterns which is relevant for parallel performance and scalability in this alignment is 12,597,450. Note that, identical sites/columns in an alignment can be compressed into site patterns, hence the number of unique site patterns actually determines the length of the conditional likelihood arrays our algorithms operate on.

For assessing the impact of the number of partitions on ExaML execution times (challenge (ii)), we created a set of alignments with an increasing number of partitions based on a real-world biological dataset with 52 species. The typical average gene length and, hence partition size, amounts to approximately 1000 base pairs. Therefore, we divided the original alignment into partitions of this size. We generated 5 datasets from this real world alignment by extracting the first 10 (10,000 bp), 50 (50,000 bp), 100 (100,000 bp), 500 (500,000 bp), and 1,000 (1,000,000 bp) partitions (sites), respectively.

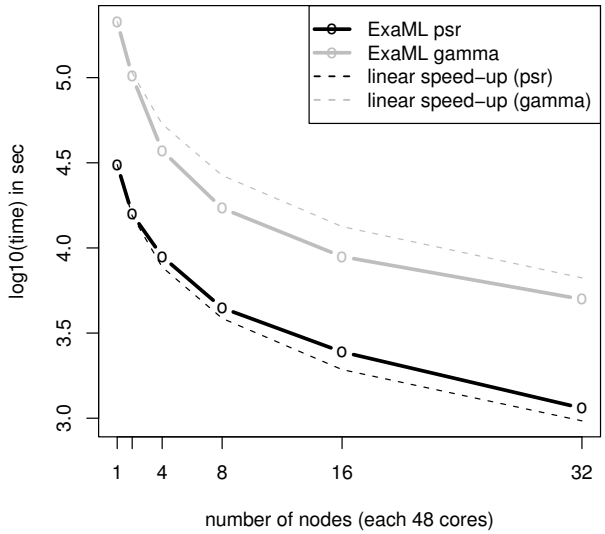


Figure 3: Log-scaled runtimes of ExaML under the PSR (black) and Γ models (gray) of rate heterogeneity on the 150 taxon $\times$ 20,000,000 bp alignment. Dashed lines indicate the linear speedup with respect to single-node runtime.

All programs in the RAxML-family implement two models for accommodating rate heterogeneity among sites: (i) the standard Γ model of rate heterogeneity [3] and (ii) the CAT model of rate heterogeneity [26]. Note that, in ExaML we decided to rename CAT to PSR (Per Site Rate model) to avoid (frequent) confusion with the CAT model as implemented in PhyloBayes [27] which is fundamentally different. Because for the PSR model, we need to optimize per-site evolutionary rates, the MPI communication patterns for PSR and Γ exhibit different patterns. We executed test runs for both models, since both are being used in practice.

C. Runtimes on a large unpartitioned Alignment

Figure 3 depicts execution times for ExaML on the unpartitioned 150 taxon $\times$ 20,000,000 bp alignment. For the runs on 1, 2, and 4 nodes, we executed ExaML on cluster nodes with 256 GB RAM. Nonetheless, for runs under the Γ model of rate heterogeneity on one and two nodes, memory requirements exceeded main memory capacity and led to a performance degradation because of swapping. Note that, the PSR model of rate heterogeneity requires four times less memory than Γ (see [26] for details) which represents *the* main advantage of the PSR model.

In general, we observed that, the alternative communication scheme implemented in ExaML scales well up to 32 nodes. For the PSR model, we obtain a speedup of 6.9 on 8 nodes and a speedup of 26.9 on 32 nodes relative to

the execution time on a single 48-core node. Runs under the Γ model yielded super-linear speedups with respect to single-node execution times, because of the aforementioned memory-shortage that triggered the swapping mechanism. If we use the run on 8 nodes as reference, we obtain speedups of 1.9 and 3.4, for runs on 16 and 32 nodes, respectively.

On 32 nodes, ExaML requires 4990 seconds to complete under Γ, while RAxML-Light needed 6108 seconds. The execution times under PSR are similar for RAxML-Light and ExaML. This amounts to a performance increase under the widely used Γ model between 6.0 and 35.8%. Note that, the PSR model is currently only implemented in RAxML and FastTree 2.0 [28]. All other widely-used programs only deploy the Γ model. Also note that, these runs were *unpartitioned*, hence the performance improvement is exclusively due to the reduction in the number of collective communication operations in ExaML and not because less data is communicated (as in the partitioned case).

D. Runtimes on partitioned Alignments

We executed RAxML-Light and ExaML on the set of alignments with an increasing number of partitions and increasing total alignment length. As described in Section II, we have to assign/distribute entire partitions monolithically to processors [24], if there are substantially more partitions than processors available. This is henceforth denoted as MPS option. MPS data distribution can be activated via the `-Q` command line switch in ExaML and RAxML-Light. We executed both programs on 4 nodes (192 cores in total) for all alignments and enabled the MPS option (`-Q`) for alignments with $\geq$ 500 partitions. In Figure 4, data points for runs with and without the MPS option are not connected to underline this altered parameter setting.

For 10, 50 and 100 partitions under the PSR model of rate heterogeneity, ExaML performs equally well as RAxML-Light (see Figure 4(a)). In contrast, ExaML substantially outperforms RAxML-Light on the three smaller alignments (10, 50, 100 partitions), when the search is conducted under Γ. For all three alignments, the execution time is reduced by approximately 30%. For the two large datasets (500 and 1000 partitions), this performance advantage becomes more pronounced. Under Γ, ExaML executes 3.1 times faster on 500 partitions and 2.6 times faster on 1000 partitions. ExaML also outperforms RAxML-Light on the two large datasets under the PSR model. Here, ExaML is 3.2 times faster on 500 partitions and 2.7 times faster on 1,000 partitions.

We repeated the above experiments for *individual, per-partition, branch length optimization* (`-M` command line option). Under this setting, the branch lengths for each partition are optimized individually and independently rather than being optimized jointly across all partitions. This increases overall inference times because more parameters need to be optimized. A tree with n taxa has $2n-3$ branches. Hence, instead of optimizing $2n-3$ branch lengths, under `-M`, we need to optimize $p(2n-3)$ branch lengths, where p is the number of partitions. This setting is of interest here, because it substantially increases the size of the traversal descriptor (which needs to hold all $p(2n-3)$ branch lengths) as well as message sizes when derivatives of the likelihood function are communicated among processes.

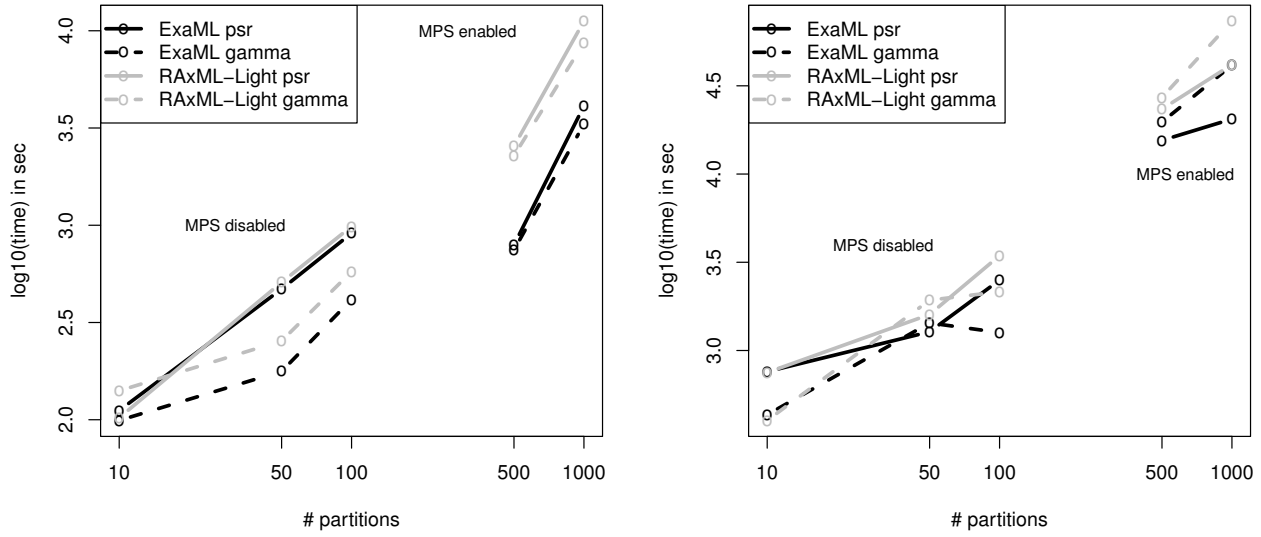
Note that, under this setting, the run time differences between Γ and the PSR model are less pronounced (see Figure 4(b)). In general, ExaML also outperforms RAxML-Light in execution time or performs slightly worse than RAxML-Light. For runs without MPS enabled, ExaML is 1.7 times faster than RAxML-Light in the best case (Γ, 100 partitions). Under the PSR model on 1000 partitions, ExaML outperforms RAxML-Light by a factor of 2.0.

It may appear intriguing that the execution times of ExaML are actually higher for 50 than for 100 partitions under the Γ model. The reason for this is that, the heuristic search algorithm of RAxML [29] that is implemented in ExaML executes a higher number of tree search iterations (23 on 50 versus 17 on 100 partitions) for the 50 partition dataset until convergence.

E. Cost of the Traversal Descriptor

As outlined in Section III, one major change in the MPI communication pattern of ExaML is the absence of traversal descriptor broadcasts. For assessing the impact of the traversal descriptor on MPI communication patterns in RAxML-Light, we determined the frequency of the parallel regions that are triggered during a typical RAxML-Light run on the 10 partition alignment. For each region, we also determined the theoretical number of bytes that need to be transferred (e.g., a `MPI_Allreduce` on 3 `MPI_DOUBLE` values is counted as $3 \times 8 = 24$ Bytes). Thus, the amount of data that needs to be communicated is measured independently of the number of MPI processes used. Of course, we cannot assume that a reduction in communication cost is directly proportional to a reduction in execution time. We grouped the parallel regions of RAxML-Light into four sets related to branch length optimization, communication of the per-site/per-partition likelihoods at the virtual root (to obtain the overall log likelihood), broadcasting of new model parameters (new α, GTR, or PSR values, for instance), and broadcasting the traversal descriptor.

Table I lists the contribution of each parallel region to the total MPI communication cost in Bytes (see above). Overall, broadcasting the traversal descriptor accounts for between 30% up to 97% of total MPI communication cost. The communication effort required for branch length optimization, evidently, strongly depends on the branch length optimization mode (per-partition versus joint branch length estimate). Because this parallel region is invoked substantially more frequently than the exchange of per-site/per-partition log likelihood scores (reduction at the virtual root), it has a



(a) joint branch length optimization

(b) individual branch length optimization

Figure 4: Execution times of ExaML (black) and RAxML-Light (gray) on alignments with an increasing number of partitions under the PSR (solid) and Γ (dashed) models of rate heterogeneity. For 10, 50 and 100 partitions MPS was disabled, for 500 and 1,000 partitions MPS option was enabled.

more pronounced impact on MPI communication cost, even under joint branch length optimization. Broadcasting new model parameters represents a rarely invoked parallel region (typically called between 10 and 15 times per run), where typically large messages are exchanged among processes. For runs with the MPS option enabled, the corresponding values in Table I only vary negligibly.

The substantial proportion of MPI communication that is required for communicating the traversal descriptor underlines how MPI communication effort can be drastically reduced by the the de-centralized parallelization scheme of ExaML.

V. Conclusion & Future Work

We have presented and made available ExaML, a new MPI-based code for large-scale phylogenomic inference under maximum likelihood for distributed memory systems. The code deploys a generally applicable new parallelization scheme that does not require a dedicated master process any more. On large, partitioned datasets that represent the typical current biological use case, it runs up to 3 times faster, than RAxML-Light which relies on the standard fork-join approach that is, to the best of our knowledge, being used for all current fine-grain parallel implementations of the likelihood function.

Apart from substantially increased parallel efficiency, the code complexity of ExaML is lower than that of RAxML-Light, requiring less MPI invocations. Finally, the concept of traversal descriptors (and analogous data structures as used in the BEAGLE library) for communicating the traversal order (the order by which conditional likelihood arrays are computed recursively according to the Felsenstein pruning algorithm) is not required any more. This further decreases code complexity as well as the amount of data that needs to be communicated.

Initially, we will review the parallelization of the PSR model to assess if better parallel efficiency can be achieved.

While the code is optimistically named Exascale Maximum Likelihood, achieving actual scalability to Exascale systems still requires a substantial amount of work. We intend to implement mechanisms for core fault tolerance. This should be relatively straight-forward, because the de-centralized scheme replicates the state of the search algorithm across all processes. Hence, unlike for the fork-join approach where a failure of the master process would be catastrophic, ExaML offers maximum state redundancy. When one or more cores fail, the data will merely have to be re-distributed to the remaining processes/cores such that computations can continue.

We have already developed a binary data format for storing input alignments and plan to use MPI parallel I/O routines to further accelerate data (re-) distribution. We will also address issues pertaining to energy efficiency. Since

	Γ, per-partition branches	Γ, joint branches	PSR, per-partition branches	PSR, joint branches
branch length optimization [%]	29.22	1.17	68.16	1.11
per-site/per-partition likelihoods [%]	0.25	0.40	0.51	0.39
model parameters [%]	0.33	0.52	0.99	2.78
traversal descriptor [%]	70.20	97.91	30.34	95.72
# parallel regions (in millions)	5.8	1.7	8.3	0.6
# bytes communicated (in MB)	2841	1809	1763	626

Table I: Relative contribution of parallel regions (lines 1 to 4) to the total number of bytes communicated (last line). Second line from bottom depicts the total number of parallel regions triggered. Four parameter combinations were run on the 10-partition dataset (joint versus per-partition branch length optimization, Γ versus PSR model of rate heterogeneity).

likelihood computations are mostly memory bandwidth-bound (they execute just a few floating point operations per conditional likelihood array entry and then process the next entry) we will assess if the CPU clock speed can be reduced to match RAM access speeds without inducing run time penalties.

We also plan to evaluate whether a hybrid MPI/PThreads approach can be used for accelerating the performance-critical `MPI_Allreduce()` calls. In a hybrid setting the number of processes participating at an `MPI_Allreduce()` will be significantly reduced. We will also explore classical techniques such as overlapping computation with computation which are applicable for partitioned datasets. If a process has finished computing the likelihood for one partition, it can already start sending this to all other processes while computing the likelihood of the next data partition.

Testing alternatives to MPI that may potentially be more scalable such as, for instance, GPI (Global address space Programming Interface; see, e.g., [30]) is also planned.

Finally, following the tradition of our lab, we will provide user support, maintenance, and keep updating ExaML with the help of the user community.

ACKNOWLEDGMENT

This work is funded via institutional funding of the Heidelberg Institute for Theoretical Studies.

REFERENCES

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

[2] Z. Yang and B. Rannala, "Bayesian phylogenetic inference using dna sequences: a markov chain monte carlo method." *Molecular Biology and Evolution*, vol. 14, no. 7, pp. 717–724, 1997.

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

[4] A. Stamatakis, A. Aberer, C. Goll, S. Smith, S. Berger, and F. Izquierdo-Carrasco, "Raxml-light: a tool for computing terabyte phylogenies," *Bioinformatics*, vol. 28, no. 15, pp. 2064–2066, 2012.

[5] S. Tavaré, "Some probabilistic and statistical problems in the analysis of DNA sequences," *Lect. Math. Life Sci*, vol. 17, pp. 57–86, 1986.

[6] R. Lanfear, B. Calcott, S. Ho, and S. Guindon, "PartitionFinder: combined selection of partitioning schemes and substitution models for phylogenetic analyses," *Molecular biology and evolution*, vol. 29, no. 6, pp. 1695–1701, 2012.

[7] D. Darriba, G. Taboada, R. Doallo, and D. Posada, "jmodeltest 2: more models, new heuristics and parallel computing," *Nature Methods*, vol. 9, no. 8, pp. 772–772, 2012.

[8] A. Stamatakis, "Orchestrating the phylogenetic likelihood function on emerging parallel architectures," *Bioinformatics–High Performance Parallel Computer Architectures, B. Schmidt, Ed. CRC Press*, pp. 85–115.

[9] X. Feng, K. Cameron, and D. Buell, "Pbpi: a high performance implementation of bayesian phylogenetic inference," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 75.

[10] 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.

[11] 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, M. Chetty, A. Ngom, and S. Ahmad, Eds., vol. 5265. Springer, 2008, pp. 424–435.

[12] M. Ott and A. Stamatakis, "Preparing raxml for the spec mpi benchmark suite," *High Performance Computing in Science and Engineering, Garching/Munich 2009*, pp. 757–768, 2010.

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

[14] 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.

[15] D. Ayres, A. Darling, D. Zwickl, P. Beerli, M. Holder, P. Lewis, J. Huelsenbeck, F. Ronquist, D. Swofford, M. Cummings *et al.*, "Beagle: an application programming interface and high-performance computing library for statistical phylogenetics," *Systematic Biology*, 2011.

[16] A. Stamatakis and M. Ott, "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] M. Charalambous, P. Trancoso, and A. Stamatakis, "Initial Experiences Porting a Bioinformatics Application to a Graphics Processor," in *Proc. of the 10th Panhellenic Conference on Informatics (PCI 2005)*, 2005, pp. 415–425.

[18] M. Suchard and A. Rambaut, "Many-core algorithms for statistical phylogenetics," *Bioinformatics*, vol. 25, no. 11, p. 1370, 2009.

[19] F. Pratas, P. Trancoso, A. Stamatakis, and L. Sousa, "Fine-grain Parallelism for the Phylogenetic Likelihood Functions on Multi-cores, Cell/BE, and GPUs," in *Proceedings of ICPP 2009*, 2009.

[20] F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Hhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck, "Mrbayes 3.2: Efficient bayesian phylogenetic inference and model choice across a large model space," *Systematic Biology*, 2012. [Online]. Available: http://sysbio.oxfordjournals.org/content/early/2012/03/02/sysbio.sys029.abstract

[21] S. Guindon, J. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel, "New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0," *Systematic biology*, vol. 59, no. 3, pp. 307–321, 2010.

[22] 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.

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

[24] J. Zhang and A. Stamatakis, "The multi-processor scheduling problem in phylogenetics," Heidelberg Institute for Theoretical Studies, Tech. Rep. Exelixis-RRDR-2011-12, November 2011.

[25] S. Berger and A. Stamatakis, "Assessment of barrier implementations for fine-grain parallel regions on current multi-core architectures," in *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–8.

[26] A. Stamatakis, "Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective," in *Proc. of IPDPS2006*, ser. HICOMB Workshop, Proceedings on CD, Rhodos, Greece, April 2006.

[27] N. Lartillot, T. Lepage, and S. Blanquart, "Phylobayes 3: a bayesian software package for phylogenetic reconstruction and molecular dating," *Bioinformatics*, vol. 25, no. 17, pp. 2286–2288, 2009.

[28] M. N. Price, P. S. Dehal, and A. P. Arkin, "Fasttree 2 approximately maximum-likelihood trees for large alignments," *PLoS ONE*, vol. 5, no. 3, p. e9490, 03 2010. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0009490

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

[30] D. Grunewald, "Bqcd with gpi: A case study," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*. IEEE, 2012, pp. 388–394.