Version dated: December 17, 2017

Fast Bayesian Inference of Phylogenetic Models Using Parallel Likelihood Calculation and Adaptive Metropolis Sampling

Venelin Mitov[1,2], Tanja Stadler[1,2]

[1]*Swiss Federal Institute of Technology in Zurich, Switzerland;*

[2]*Swiss Institute of Bioinformatics, Switzerland*

Corresponding authors: Venelin Mitov, Department of Biosystem Sciences and Engineering, Swiss Federal Institute of Technology, Mattenstrasse 26, CH-4058 Basel, Switzerland; E-mail: `vmitov@gmail.com`.

Tanja Stadler, Department of Biosystem Sciences and Engineering, Swiss Federal Institute of Technology, Mattenstrasse 26, CH-4058 Basel, Switzerland; E-mail: `tanja.stadler@bsse.ethz.ch`.

Abstract

Phylogenetic comparative methods have been used to model trait evolution, to test selection versus neutral hypotheses, to estimate optimal trait-values, and to quantify the rate of adaptation towards these optima. Several authors have proposed algorithms calculating the likelihood for trait evolution models, such as the Ornstein-Uhlenbeck (OU) process, in time proportional to the number of tips in the tree. Combined with gradient-based optimization, these algorithms enable maximum likelihood (ML) inference within seconds, even for trees exceeding 10,000 tips. Despite its useful statistical properties, ML has been criticised for being a point estimator prone to getting stuck in local optima. As an elegant alternative, Bayesian inference explores the entire information in the data and compares it to prior knowledge but, usually, needs much longer time, even on small trees. Here, we propose an approach to use the full potential of ML and Bayesian inference, while keeping the runtime within minutes. Our approach combines (i) a new algorithm for parallel traversal of the lineages in the tree, enabling parallel calculation of the likelihood; (ii) a previously published method for adaptive Metropolis sampling. In principle, the strategy of (i) and (ii) can be applied to any likelihood calculation on a tree which proceeds in a pruning-like fashion, leading to enormous speed improvements. We implement several variants of the parallel algorithm in the form of a generic C++ library, "SPLiTTree", capable to choose automatically the optimal algorithm for a given task and computing platform. We give examples of models of discrete and continuous trait evolution that are amenable to parallel likelihood calculation. As a complete showcase, we implement the phylogenetic Ornstein-Uhlenbeck mixed model (POUMM) in the form of an easy-to-use and highly configurable R-package that calls the library as a back-end. In addition to the above-mentioned usage of comparative methods, POUMM allows to estimate non-heritable variance and phylogenetic heritability. Using SPLiTTree, calculating the POUMM likelihood on a 4-core SIMD-enabled processor is up to 10× faster than serial implementations written in C and hundreds of times faster than serial implementations written in R. By combining SPLiTTree likelihood calculation with adaptive Metropolis sampling, the time for Bayesian POUMM inference on a tree of ten thousand tips is reduced from several days to a few minutes.

Keywords: Parallel tree traversal, post-order traversal, pre-order traversal, pruning, discrete character, continuous trait, phylogenetic comparative models, Brownian motion, measurement error, stabilizing selection, environmental contribution

Introduction

The past decades have seen active developement of phylogenetic comparative models of trait evolution, progressing from null neutral models, such as single-trait Brownian motion (BM), to complex multi-trait models incorporating selection, interaction between trait values and diversification, and co-evolution of multiple traits (O'Meara 2012; Manceau et al. 2016). Recent works have shown that, for a broad family of phylogenetic comparative models, the likelihood of an observed tree and data conditioned on the model parameters can be computed in time proportional to the size of the tree (FitzJohn 2012; Ho and Ané 2014; Goolsby et al. 2016; Manceau et al. 2016). This family includes Gaussian models like Brownian motion and Ornstein-Uhlenbeck phylogenetic models as well as some non-Gaussian models like phylogenetic logistic regression (Paradis and Claude 2002; Ives and Garland 2010; Ho and Ané 2014). All of these likelihood calculation techniques rely on post-order tree traversal also known as *"pruning"* (Felsenstein 1973, 1981, 1983). Using pruning algorithms for likelihood calculation in combination with a gradient-based optimization method (Boyd and Vandenberghe 2004), maximum likelihood model inference runs within seconds on contemporary computers, even for phylogenies containing many thousands of tips (Ho and Ané 2014). Other important features of the maximum likelihood estimate (MLE) are its simple interpretation as the point in parameter space maximizing the probability of the observed data under the assumed model, and its theoretical properties making it ideal for hypothesis testing and for model selection via likelihood ratio tests and information criteria. However, major disadvantages of MLE are that, being a point estimate, it does not allow to explore the likelihood surface; gradient based optimization, while fast, is prone to getting stuck in local optima.

As an elegant alternative, Bayesian approaches such as Markov Chain Monte Carlo (MCMC) allow to incorporate prior knowledge in the model inference and provide posterior samples and high posterior density (HPD) intervals for the model parameters (Slater et al. 2012a; FitzJohn 2012). In contrast with ML inference, though, Bayesian inference methods require many orders of magnitude more likelihood evaluations, in order to obtain a valuable posterior sample. This presents a bottleneck in Bayesian analysis, in particular, when faced with large phylogenies of many thousands of tips, such as transmission trees from large-scale epidemiological studies, e.g. Hodcroft et al. (2014). While big data provides sufficient statistical power to fit a complex model, the time needed to perform a full scale Bayesian inference often limits the choice to a faster but less informative ML-inference, or a Bayesian inference on a simplified model.

In this article, we propose a general approach allowing to use ML and Bayesian inference to their full potential, even for complex models and for very large trees exceeding millions of tips. To achieve this goal, our approach combines two ideas: (i) the pruning algorithm for likelihood calculation can be accelerated by orders of magnitude through parallel processing (traversal) of the independent lineages in the tree; (ii) the number of iterations needed for MCMC convergence can be reduced several times by the use of adaptive Metropolis sampling (Vihola 2012; Scheidegger 2017). A nice property of the parallel pruning algorithm is that its parallel efficiency increases with the number of tips in the tree as well as with the complexity of the pruning operation. Thus, for large trees and complex models, the parallel speed-up is practically limited by the number of available processing cores and other hardware resources such as the memory bandwidth.

We provide SPLiTTree: a C++ library for Serial and Parallel Lineage Traversal of Trees. The library is designed to be generic with respect to the tree topology, the type of data, the model parameters and the specific visit-node operation. It has been tested on large ($N > 100,000$) balanced and highly unbalanced trees. Noticing that there is no one-size-fits-all strategy for optimal parallel speed-up (e.g. the parallelization may be useless or even slowing-down the performance in the case of a ladder tree or in the case of a memory intensive visit-node operation), SPLiTTree implements several parallelization strategies (e.g. classical serial traversal versus queue-based and range-based parallel traversal) and is capable to automatically select the fastest strategy for a given tree and computing hardware. In this way, the library allows the user to neglect complex technical aspects of parallel programming and focus on implementing the visit-node operation specific for the application of interest.

We begin with a formulation of a general framework for parallel post-order tree traversal (pruning). Then, we give examples of models of discrete and continuous trait evolution, amenable to using the library for parallel likelihood calculation. These examples represent mere adaptations of previously published pruning algorithms to the terms of the framework (Felsenstein 1983; Pagel 1994; Ho and Ané 2014).

We go on with a showcase demonstrating the combined use of parallel pruning and adaptive Metropolis sampling on a univariate phylogenetic Ornstein-Uhlenbeck mixed model (POUMM). Previously, we and other authors have used the POUMM in heritability analysis of epidemiological data (Mitov and Stadler 2016; Bachmann et al. 2017; Bertels et al. 2017; Blanquart et al. 2017). Here, we describe the pruning formulation of the POUMM likelihood, validate the technical correctness of the implementation and compare the performance of this implementation to several other implementations of OU-models of evolution including FitzJohn (2012) and Pennell et al. (2014). We report significant speed-ups, both for the time for single likelihood calculation as well as for the overall Bayesian inference of the model.

Setup

Through the rest of the article we will use the following notation. Given is a rooted phylogenetic tree $\mathcal{T}$ with a total of M nodes, including $N < M$ tips denoted $1, ..., N$, $M - N - 1$ internal nodes denoted $N + 1, ..., M - 1$, and a root node denoted M (Fig. 1). Without restrictions on the tree topology, non-ultrametric trees (i.e. tips have different heights) and polytomies (i.e. nodes with any finite number of descendants) are accepted.

We denote by $\mathcal{T}_i$ the subtree rooted at node i. For any tip or internal node i, we denote its parent node by $Parent(i)$. For any node j, we denote by $Desc(j)$ the set of its direct descendants ($Desc(j) = \phi$ if j denotes a tip). Furthermore, for any $i \in Desc(j)$, we denote by t_i the length of the branch leading to i and by h_i the height of i, i.e. the sum of branch lengths from the root to i. The mean height of all tips in the tree is denoted by $\bar{h}$. For two tips, i and k, we denote by $h_{(ik)}$ the height to their most recent common ancestor (mrca), and by d_{ik} the sum of branch lengths on the path from i to k (also called phylogenetic or patristic distance between i and k). Associated with each node i there is an input data in the form of a single or multivariate categorical or numerical value denoted z_i. For tips, z_i can be partially unobserved (having NA entries), while for internal nodes or the root it can also be fully absent (NULL). We denote by $\mathbf{z}_i$ the sub-vector of input data for the nodes in $\mathcal{T}_i$. Associated with each node, i, there is a vector of model parameters, Θ_i. We use bold style $\mathbf{t}$, $\mathbf{z}$ and $\mathbf{\Theta}$ when denoting the vectors of all branch lengths, input data and parameters.

A general framework for parallel tree traversal

Let $F_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$ be a function of the branch lengths, the input data and the parameters. A post-order tree traversal algorithm can be used to calculate $F_{\mathcal{T}}$ if, for all subtrees $\mathcal{T}_j$ of $\mathcal{T}$, there exist functions $S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$, hereby called "states", satisfying the following rules:

(1) $F_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$ can be calculated from $S_M(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$;

(2) For each node $j \in \{1, ..., M\}$, there exists a (recursive) relationship R_j between S_j and the set of states at j's descendants, such that:

$$S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = R_j\Big(\big\{S_i(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) : i \in Desc(j)\big\}, \mathbf{t}, \mathbf{z}, \mathbf{\Theta}\Big). \quad (1)$$

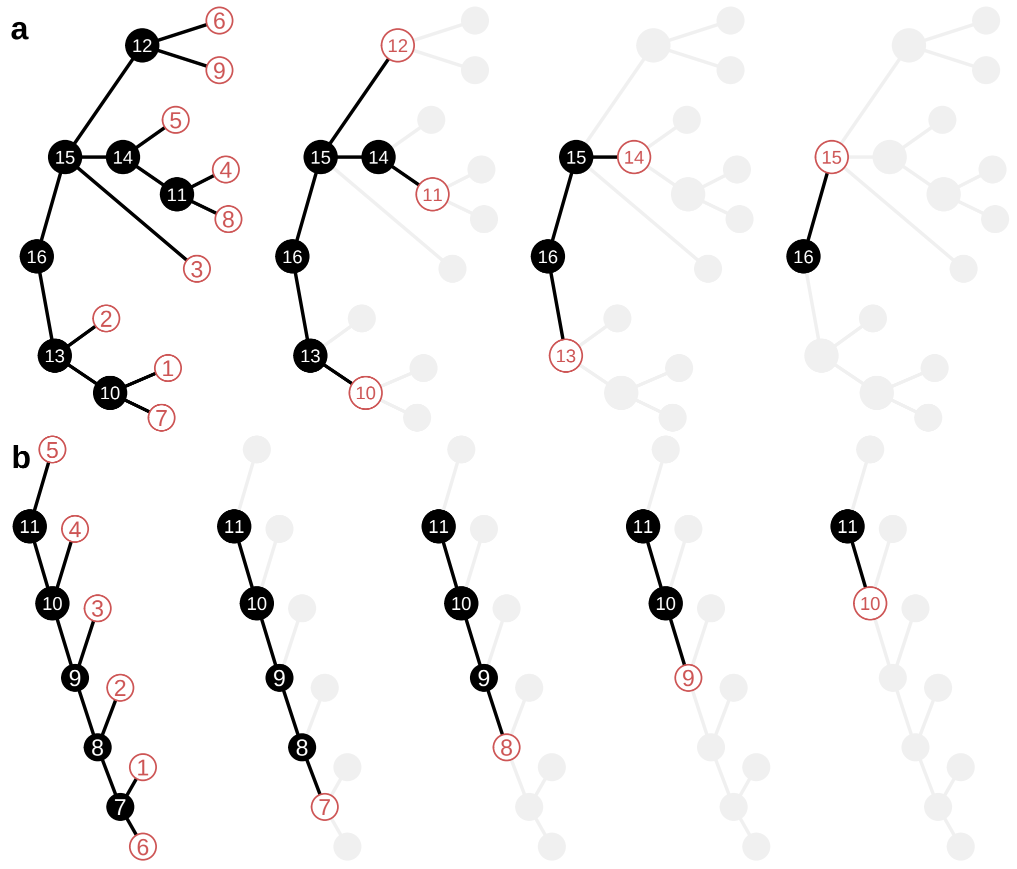

Figure 1: **Parallel pruning.** The trees from left to right depict generations of nodes that can be processed in parallel. The processing of a node consists in calculating its state based on the input data, the branch lengths and the states of the node's direct descendants (eq. 1). black: nodes having one or more non-processed descendants; red: nodes ready to be processed; grey: nodes processed in a previous generation. a) a balanced tree; b) a ladder.

We note that analogical terms can be defined for pre-order tree traversal. In this case the target functions are values $Z_{\mathcal{T},j}(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$ corresponding to the nodes $j \in \{1, ..., M\}$, and rule (2) is updated to:

(2') Z_M can be calculated from the input data. For each node $j \in \{1, ..., M-1\}$, there exists a (recursive) relationship R'_j between Z_j and $Z_{Parent(j)}$, such that:

$$Z_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = R'_j\big(Z_{Parent(j)}(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}), \mathbf{t}, \mathbf{z}, \mathbf{\Theta}\big). \tag{2}$$

The states, i.e. the values of the functions S_j and Z_j, may be deterministic or stochastic functions of the input tree and data. They can be real numbers, vectors, matrices or higher order combinations thereof.

For the rest of the article, we focus on parallel post-order tree traversal or "pruning", noting that the algorithms for parallel pre-order traversal are simple analogies. The SPLiTTree library implements both traversal types.

Rule (2) ensures that calculating the state of a node j can be done independently from the calculation of any other node k, provided that neither j is an ancestor of k, nor k is an ancestor of j. Based on this observation, we develop two alternative parallel algorithms for calculating the root state S_M described in the following subsections.

Queue-based parallel pruning

It is possible to parallelize the computation of the states S_j across multiple computing threads using a first-in-first-out list (queue) of the nodes in the tree (algorithm 1). Initially, the queue is filled with all tips in the tree and a counter with the number of direct descendants is set for each internal or root node. Then, each thread takes a node i from the front of the queue, calculates its state and decrements the counter of $Parent(i)$. If the counter of $Parent(i)$ has become zero, $Parent(i)$ is added to the queue, so that it will be processed as soon as a free thread picks it from the queue. Assuming an unlimited number of threads and a negligible cost of the queue- and the counter- operations, algorithm 1 guarantees that a node will be processed immediately after all of its direct descendants have been processed. Thus, in theory, algorithm 1 maximizes the parallel execution. However, an implementation of the atomic operations on the queue and the counters would have to rely on a thread synchonization mechanism such as a mutex, which can be slow on some systems. Thus, a decent parallelization speed-up would only be possible if the overall cost of synchronization is insignificant compared to the functions R_j.

Range-based parallel pruning

We consider an alternative of algorithm 1 minimizing the synchronization overhead. This approach consists in splitting the tree into ”generations” of nodes, such that nodes within a generation can be processed in random order and in parallel, but only if all generations containing descendants of these nodes have already been processed (fig. 1). A “master” thread is responsible for launching a team of “worker” threads on each generation, starting from a generation of all tips, then taking their parents, and so on until reaching the root of the tree. To be efficient, this procedure requires that the data associated with the nodes in a generation occupy a consecutive region in the address-space. This eliminates the need for synchronization between the worker threads, because each worker thread can deduce its own portion based on its thread-id and the address-range of the generation. To orchestrate the worker teams, the master thread only needs to keep account of the address-ranges. Technically, this is accomplished by iterating over a vector of offsets (algorithm 2).

In algorithm 2, the number of synchronization points is reduced to the number of generations, K. In balanced trees, K would increase logarithmically with N and, for big N, the tree would be split into a few generations of many nodes (fig. 1a). If there are no

Algorithm 1: Queue-based parallel pruning

```
   Input: $\mathcal{T}$, $\mathbf{t}$, $\mathbf{z}$, $\boldsymbol{\Theta}$
   Output: $S_M(\mathbf{t}, \mathbf{z}, \boldsymbol{\Theta})$
   /* a vector of $M$ states                                                   */
1  $State \longleftarrow [...]_M$;
   /* a vector of the numbers of remaining descendants for each node          */
2  $NumDesc \longleftarrow \big[|Desc(i)| : i \in \{1, ..., M\}\big]$;
   /* initiate $Queue$ with all tips:                                          */
3  $Queue \longleftarrow [1, ..., N]$;
4  begin Parallel block
5      while $(TRUE)$ do
           /* if $Queue$ is empty, thread waits.                               */
6          $j \longleftarrow$PopFirst$(Queue)$;
7          $State[j] \longleftarrow R_j\Big(\big\{State[i] : i \in Desc(j)\big\}, \mathbf{t}, \mathbf{z}, \boldsymbol{\Theta}\Big)$;
8          if $(j < M)$ then
               /* the root has not been processed yet.                         */
9              $NumDesc[Parent(j)] \longleftarrow NumDesc[Parent(j)] - 1$;
10             if $(NumDesc[Parent(j)] == 0)$ then
                   /* If $Queue$ is currently empty a waiting thread will be
                      notified.                                                */
11                 AddLast$(Queue, Parent(j))$;
12         else
               /* the root has been processed.                                 */
               /* Notify waiting threads by adding a stopping node-id to
                  $Queue$.                                                     */
13             AddLast$(Queue, M + 1)$;
               /* All work done, exit the loop.                                */
14             break;
15 return $State[M]$;
```

Algorithm 2: Range-based parallel pruning

```
  Input: $\mathcal{T}$, $\mathbf{t}$, $\mathbf{z}$, $\mathbf{\Theta}$
  Output: $S_M(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$
  Data:
  /* A pre-calculated vector with starting offsets for each generation:
     */
1 $Range = \left[0, N, N + |G_1|, N + |G_1| + |G_2|, ..., M - 1, M\right]_{K+1}$;
  /* a vector of $M$ elements                                        */
2 $State \longleftarrow [0, ..., 0]_M$;
  /* The master thread iterates over the generations:                */
3 foreach $k \in \{1, ..., K\}$ do
      /* The master thread starts a team of worker threads running equal
         portions of the following loop:                             */
4     foreach $j \in \{Range[k] + 1, ..., Range[k + 1]\}$ do
5         $State[j] \longleftarrow R_j\Big(\big\{State[i] : i \in Desc(j)\big\}, \mathbf{t}, \mathbf{z}, \mathbf{\Theta}\Big)$;
6 return $State[M]$;
```

other blocking resources, such as memory bandwidth, the parallel speed-up would be nearly as high as the number of processing cores and the synchronization overhead would be negligible. Conversely, in strongly unbalanced trees, K would tend to increase linearly with N and the tree would be split into many generations of a few nodes (fig. 1b). This would result in low parallel speed-up and excessive synchronization cost. Also noteworthy is the fact that algorithm 2 reduces the number of synchronization points at the cost of some parallelization. If each worker thread gets assigned to an approximately equal number of nodes in a generation and if a few of the nodes take much longer time to process than the rest, then most of the worker threads would have to wait until the last node in the generation has been processed. These subtleties of the tree and input data indicate that there is no "one size fits all" strategy when it comes to optimizing the parallelization speed-up. The parallel pruning framework provides two ways to deal with these: (a) allowing the user to choose a parallelization mode before executing a pruning operation on a given tree and data; (b) providing a mode "auto", in which the framework compares the execution time of different pruning algorithms during the first several calls on a given tree and data, choosing the fastest one for all subsequent calls.

The SPLiTTree library

We provide SPLiTTree in the form of an open source C++ library licensed under version 3.0 of the GNU Lesser General Public License (LGPL v3.0) and available on https://github.com/venelin/SPLiTTree.git. In its current implementation, the library uses the C++11 language standard, the standard template library (STL) and the OpenMP standard for parallel processing. The library is designed as a set of C++ template classes, generic with respect to the application specific details, such as the types of input data, model parameters and definitions of the node states, S_i, and visit-node functions, R_i. The library defines two layers (fig. 2):

- a framework layer defining the main logical and data structures. These include a linear algorithm for initial reordering and splitting of the input tree into generations of nodes, which can be visited in parallel, both during a post-order as well as pre-order traversal, and a growing collection of pre-order and post-order traversal algorithms, targeting different parallelization modes (e.g. queue-based versus range-based parallelization) on different computing devices (currently implemented for CPUs only).

- a user layer at which the user of the library must write a `CustomTraversalSpecification` class defining all typedefs and methods of the interface `TraversalSpecification`. The most important of entity to define is the method `VisitNode` specifying the function R_i for each tip or internal node, i.

The bridge between the two layers is provided by an object of the `TraversalTask` template class. Once the `TraversalSpecification` implementation has been written, the

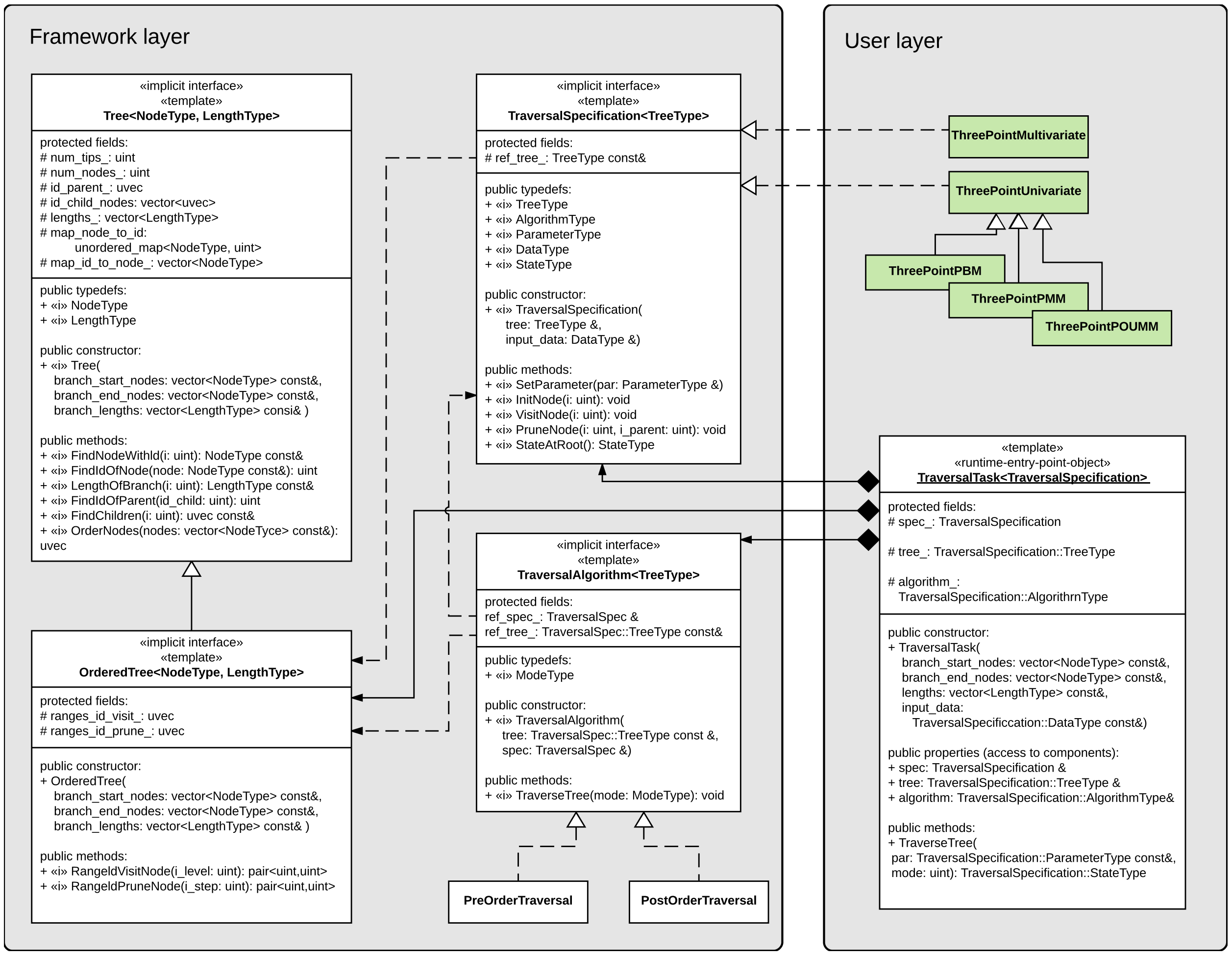
Framework layer
«implicit interface»
«template»
Tree<NodeType, LengthType>
protected fields:
# num_tips_: uint
# num_nodes_: uint
# id_parent_: uvec
# id_child_nodes: vector<uvec>
# lengths_: vector<LengthType>
# map_node_to_id:
unordered_map<NodeType, uint>
# map_id_to_node_: vector<NodeType>
public typedefs:
+ «i» NodeType
+ «i» LengthType
public constructor:
+ «i» Tree(
branch_start_nodes: vector<NodeType> const&,
branch_end_nodes: vector<NodeType> const&,
branch_lengths: vector<LengthType> consi&)
public methods:
+ «i» FindNodeWithId(i: uint): NodeType const&
+ «i» FindIdOfNode(node: NodeType const&): uint
+ «i» LengthOfBranch(i: uint): LengthType const&
+ «i» FindIdOfParent(id_child: uint): uint
+ «i» FindChildren(i: uint): uvec const&
+ «i» OrderNodes(nodes: vector<NodeTyce> const&): uvec
«implicit interface»
«template»
OrderedTree<NodeType, LengthType>
protected fields:
# ranges_id_visit_: uvec
# ranges_id_prune_: uvec
public constructor:
+ OrderedTree(
branch_start_nodes: vector<NodeType> const&,
branch_end_nodes: vector<NodeType> const&,
branch_lengths: vector<LengthType> const&)
public methods:
+ «i» RangeIdVisitNode(i_level: uint): pair<uint,uint>
+ «i» RangeIdPruneNode(i_step: uint): pair<uint,uint>
«implicit interface»
«template»
TraversalSpecification<TreeType>
protected fields:
# ref_tree_: TreeType const&
public typedefs:
+ «i» TreeType
+ «i» AlgorithmType
+ «i» ParameterType
+ «i» DataType
+ «i» StateType
public constructor:
+ «i» TraversalSpecification(
tree: TreeType &,
input_data: DataType &)
public methods:
+ «i» SetParameter(par: ParameterType &)
+ «i» InitNode(i: uint): void
+ «i» VisitNode(i: uint): void
+ «i» PruneNode(i: uint, i_parent: uint): void
+ «i» StateAtRoot(): StateType
«implicit interface»
«template»
TraversalAlgorithm<TreeType>
protected fields:
ref_spec_: TraversalSpec &
ref_tree_: TraversalSpec::TreeType const&
public typedefs:
+ «i» ModeType
public constructor:
+ «i» TraversalAlgorithm(
tree: TraversalSpec::TreeType const &,
spec: TraversalSpec &)
public methods:
+ «i» TraverseTree(mode: ModeType): void
PreOrderTraversal
PostOrderTraversal
User layer
ThreePointMultivariate
ThreePointUnivariate
ThreePointPBM
ThreePointPMM
ThreePointPOUMM
«template»
«runtime-entry-point-object»
TraversalTask<TraversalSpecification>
protected fields:
# spec_: TraversalSpecification
# tree_: TraversalSpecification::TreeType
# algorithm_:
TraversalSpecification::AlgorithrnType
public constructor:
+ TraversalTask(
branch_start_nodes: vector<NodeType> const&,
branch_end_nodes: vector<NodeType> const&,
lengths: vector<LengthType> const&,
input_data:
TraversalSpecificcation::DataType const&)
public properties (access to components):
+ spec: TraversalSpecification &
+ tree: TraversalSpecification::TreeType &
+ algorithm: TraversalSpecification::AlgorithmType&
public methods:
+ TraverseTree(
par: TraversalSpecification::ParameterType const&,
mode: uint): TraversalSpecification::StateType

Figure 2: **A UML class diagram of the SPLiTTree library** In the framework layer, the class `TraversalSpecification` defines the application-specific data types and logic; the class `Tree` serves as a base-class implementing common tree operations, such as constructing a tree from a list of branches, checking the validity of the input (e.g. lack of cycles or isolated branches), finding the parent and the descendants of a node, etc; the class `OrderedTree` maintains the order of the nodes in a tree so that they can be split in contiguous generations for parallel post-order or pre-order traversal; the class `TraversalAlgorithm` serves as a base class and an implicit interface for its two subclasses implementing the two supported types of tree traversal: `PreOrderTraversal` and `PostOrderTraversal`. At the user layer are the user defined implementations of the `TraversalSpecification` interface (shown in green) and an instance of the `TraversalTask` class, which constructs all necessary internal objects and serves as a runtime entry point to the framework.

user instantiates a `TraversalTask` object passing the tree and the input data as arguments. This triggers the creation of the internal objects of the framework, i.e. an `OrderedTree` object maintaining the order in which the nodes are processed and a `PreOrderTraversal` or a `PostOrderTraversal` object implementing different parallelization modes of the two traversal types. In the ideal use-case, the `TraversalTask`'s `TraverseTree()` method will be called repeatedly, varying the model parameters, the input data and branch lengths on a fixed tree topology. This encompasses all scenarios where a model is fitted to a fixed tree and data, e.g. ML or Bayesian PCM inference. In the current version of the framework any change in the tree topology invalidates the associated `TraversalTask` object, so a new one has to be created. Thus, in the case of inferring a tree topology, the overall performance would depend on the frequency of changes in the topology per traversal and the time for re-creating the `TraversalTask` object relative to the total time of one traversal.

Examples

In this section, we show example usages of the parallel traversal framework. In each of these examples, we solve a particular problem, such as calculating the likelihood of a continuous Markov model for a categorical or a continuous trait. In terms of the framework, the task boils down to formulating the node states $S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$ and the recursive functions R_j satisfying rules (1) and (2).

Example 1: Models of categorical trait evolution.—

Consider a trait taking values in $\{0, 1\}$ evolving independently along the lineages of a phylogenetic tree, $\mathcal{T}$ with branch lengths $\mathbf{t}$. A continuous-time Markov model can be used to characterize the transitions of the trait value along each branch (Felsenstein 1983; Pagel 1994). This model assumes constant rates of change from 0 to 1, q_{01} and from 1 to 0, q_{10}, representing the probability that the change has occurred during an infinitesimal interval of time. These rates are used to define a rate matrix:

$$\mathbf{Q} = \begin{bmatrix} -q_{01} & q_{01} \\ q_{10} & -q_{10} \end{bmatrix}. \tag{3}$$

Given $\mathbf{Q}$, the transition probability matrix $\mathbf{P}(t)$ for an arbitrary long period t is given by

$$\mathbf{P}(t) = \begin{bmatrix} P_{00}(t) & P_{01}(t) \\ P_{10}(t) & P_{11}(t) \end{bmatrix} = \mathbf{C} \begin{bmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{bmatrix} \mathbf{C}^{-1} \tag{4}$$

where λ_i are the eigenvalues of $\mathbf{Q}$ and $\mathbf{C}$ is a matrix, which's i^{th} column represents the i^{th} eigenvector of $\mathbf{Q}$ (Pagel 1994). Assuming that the value at the root is known to be z_M, we want to find the probability with which the model specified by the parameters $\mathbf{\Theta} = (q_{01}, q_{10})$ generates an N-vector of values, $\mathbf{z}$ observed at the tips. This represents the conditional likelihood $L_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}, z_M)$. The pruning algorithm for calclulating L relies on calculating the "fragmentary" likelihood $L_i(b) = P(\mathbf{z}_i | z_i = b; \mathbf{\Theta})$ for each node i and each $b \in \{0, 1\}$ (Felsenstein 1983). In terms of the framework, we define the state $S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta})$ of a node j as the pair $< L_j(0), L_j(1) >$. Following eq. 4 in (Felsenstein 1983), the recursive R_j are given by:

$$S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = \begin{cases} \langle \delta(z_j = 0), \delta(z_j = 1) \rangle & \text{if } j \text{ is a tip} \\ \left\langle \prod_{i \in Desc(j)} \left[\sum_{z_i} P_{0 z_i}(t_i) L_i(z_i) \right], \prod_{i \in Desc(j)} \left[\sum_{z_i} P_{1 z_i}(t_i) L_i(z_i) \right] \right\rangle & \text{if } j \text{ is internal,} \end{cases} \tag{5}$$

where we use the Kronecker delta function $\delta(x = y)$ equalling to 1 if $x = y$ and 0, otherwise. In the above eqation 5, the values $L_i(z_i)$ are available from the descendants' states S_i. Finally, the conditional likelihood $L_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}, z_M)$ is given by $L_M(z_M)$, which is one of the two members in S_M.

The above model can be extended to a multivariate case, such as calculating the probability of a nucleotide or aminoacid sequence alignment as is the case in (Felsenstein 1983). Suppose that there are p nucleotide sites, which are evolving independently. Then, the state for a node j would represent a $p \times 4$ matrix

$$S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = \begin{bmatrix} L_j^{(1)}(A) & L_j^{(1)}(C) & L_j^{(1)}(T) & L_j^{(1)}(G) \\ \vdots & \vdots & \vdots & \vdots \\ L_j^{(p)}(A) & L_j^{(p)}(C) & L_j^{(p)}(T) & L_j^{(p)}(G) \end{bmatrix}, \tag{6}$$

where the letters A, C, T and G denote the nucleotides and the superscript in parentheses denotes a site in the alignment. To define the recursive functions R_j, equation 5 can be extended to accomodate one row of S_j (four possible values instead of two) and evaluated p times to obtain the full state.

The model can also be extended to support correlated evolution between the sites. As shown in (Pagel 1994), this involves extending the rate matrix $\mathbf{Q}$ to embed transition rates between pairs, triplets or higher order combinations of sites in the sequence. Accounting for correlated evolution between combinations of sites dramatically increases the computational complexity, but does not present a conceptual change from the point of view of the pruning operation. Thus, accommodating such models in the framework, although involved technically, should not present a conceptual challenge.

We should not omit mentioning other software libraries implementing parallel likelihood computation of different Markov models of sequence evolution. For example, several high level tools for ML and Bayesian tree inference, e.g. Drummond et al. (2012); Bouckaert et al. (2014); Ronquist and Huelsenbeck (2003), use the library BEAGLE which distributes the computation for the independent sites of the sequence alignment among multiple CPU or GPU cores (Ayres et al. 2012). SPLiTTree operates on a different level, namely, it parallelizes the computation for independent lineages in the tree. Both approaches are interesting because they fit well to different sizes of the input data - while BEAGLE achieves significant parallel speed-ups in long alignments comprising many thousands nucleotide or codon columns (Ayres et al. 2012), SPLiTTree is better suited to shorter alignments of potentially many thousands of species.

Example 2: Models of continuous trait evolution.—

Ho and Ané (2014) noticed that the computational complexity in multivariate Gaussian and some non-Gaussian models concentrates in the calculation of determinants $|\mathbf{V}_{\mathbf{\Theta}}|$ and quadratic quantities of the form $\mathbf{Q}_{\mathbf{\Theta}} = \mathbf{X}'_{\mathbf{\Theta}}\mathbf{V}_{\mathbf{\Theta}}^{-1}\mathbf{Y}_{\mathbf{\Theta}}$, where $\mathbf{V}_{\mathbf{\Theta}}$ represents the variance covariance matrix expected under the model specified by $\mathbf{\Theta}$ and the matrices $\mathbf{X}_{\mathbf{\Theta}}$ and $\mathbf{Y}_{\mathbf{\Theta}}$ represent centered observed data at the tips in the tree. For example, in the case of Brownian motion and Ornestein-Uhlenbeck models, the log-likelihood function is equal to the log-density of a multivariate Gaussian distribution:

$$\ln f(\mathbf{z}|\mathbf{\Theta}) = -\frac{1}{2}\left(N\ln(2\pi) + \ln|\mathbf{V}_{\mathbf{\Theta}}| + (\mathbf{z}-\mu_{\mathbf{\Theta}})'\mathbf{V}_{\mathbf{\Theta}}^{-1}(\mathbf{z}-\mu_{\mathbf{\Theta}})\right), \tag{7}$$

where $\mathbf{V}_{\mathbf{\Theta}} = \mathbf{\Sigma}$ and $\mathbf{V}_{\mathbf{\Theta}} = \mu$ (table 1).

Ho and Ané (2014) developed a pruning algorithm which allows to calculate $|\mathbf{V}_{\mathbf{\Theta}}|$ and $\mathbf{Q}_{\mathbf{\Theta}}$ simultaneously and without constructing or allocating the matrix $\mathbf{V}_{\mathbf{\Theta}}$ in memory, provided $\mathbf{V}_{\mathbf{\Theta}}$ has a "3-point structure". Then, they showed several examples of Gaussian models such as Brownian motion and Ornstein-Uhlenbeck, as well as non-Gaussian models, such as phylogenetic logistic and Poisson regression, where $\mathbf{V}_{\mathbf{\Theta}}$ is or can be "converted" to a 3-point structured matrix (discussed later). Adapting the notation from (Ho and Ané 2014, p. 399), we define the node states as $S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = \left\langle p_{A,j}, p_j, \hat{\mu}_{Y,j}, \tilde{\mu}'_{X,j}, \ln|\mathbf{V}|_j, \mathbf{Q}_j \right\rangle$. The recursive functions R_j follow immediately from points 1 and 2 of the algorithm (Ho

Table 1: Population properties at the tips of the phylogeny under BM and OU models and their mixed counterparts The acronyms are: PBM - Phylogenetic Brownian motion (without non-heritable component); PMM - Phylogenetic Mixed Model (adding a non-heritable component to PBM); POU - Phylogenetic Ornstein-Uhlenbeck (without non-heritable component), also known as "Hansen's model" or Single Stationary Peak (SSP); POUMM - Phylogenetic Ornstein-Uhlenbeck Mixed Model (adding a non-heritable component to the POU model. Expressions for the OU-models were adapted from (Hansen 1997). $\mu_{\mathbf{\Theta},i}$: expected value at tip i; $\mathbf{V}_{\mathbf{\Theta},ii}$: expected variance for tip i; $\mathbf{V}_{\mathbf{\Theta},ij}$: expected covariance of the values of tips i and j.

	PBM	PMM	POU	POUMM
$\mathbf{\Theta}$:	$< g_M, \sigma >$	$< g_M, \sigma, \sigma_e >$	$< g_M, \alpha, \theta, \sigma >$	$< g_M, \alpha, \theta, \sigma, \sigma_e >$
$\mu_{\mathbf{\Theta},i}$:	g_M	g_M	$e^{-\alpha h_i} g_M + \left(1 - e^{-\alpha h_i}\right) \theta$	$e^{-\alpha h_i} g_M + \left(1 - e^{-\alpha h_i}\right) \theta$
$\mathbf{V}_{\mathbf{\Theta},ii}$:	$\sigma^2 h_i$	$\sigma^2 h_i + \sigma_e^2$	$\frac{\sigma^2}{2\alpha} \left(1 - e^{-2\alpha h_i}\right)$	$\frac{\sigma^2}{2\alpha} \left(1 - e^{-2\alpha h_i}\right) + \sigma_e^2$
$\mathbf{V}_{\mathbf{\Theta},ij}$:	$\sigma^2 h_{(ij)}$	$\sigma^2 h_{(ij)}$	$\frac{\sigma^2}{2\alpha} e^{-\alpha d_{ij}} \left(1 - e^{-2\alpha h_{(ij)}}\right)$	$\frac{\sigma^2}{2\alpha} e^{-\alpha d_{ij}} \left(1 - e^{-2\alpha h_{(ij)}}\right)$

316 and Ané 2014):

$$
\begin{cases}
S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = \langle \; p_{A,j} = 0, \\
\quad p_j = \frac{1}{t_j}, \\
\quad \hat{\mu}_{Y,j} = \mathbf{y}_{\mathbf{\Theta},j}, \\
\quad \tilde{\mu}'_{X,j} = \mathbf{x}'_{\mathbf{\Theta},j}, & \text{if } j \leq N \\
\quad \ln|\mathbf{V}|_j = \ln t_j, \\
\quad \mathbf{Q}_j = \mathbf{x}'_{\mathbf{\Theta},j} \mathbf{y}_{\mathbf{\Theta},j} \; \rangle \\
S_j(\mathbf{t}, \mathbf{z}, \mathbf{\Theta}) = \langle \; p_{A,j} = \sum_{i \in Desc(j)} p_i, \\
\quad p_j = \frac{p_{A,j}}{1 + t_j p_{A,j}}, \\
\quad \hat{\mu}_{Y,j} = \sum_{i \in Desc(j)} \frac{p_i}{p_A} \hat{\mu}_{Y,i}, & \text{otherwise.} \\
\quad \tilde{\mu}'_{X,j} = \sum_{i \in Desc(j)} \frac{p_i}{p_A} \tilde{\mu}'_{X,i}, \\
\quad \ln|\mathbf{V}|_j = \sum_{i \in Desc(j)} \ln|\mathbf{V}|_i + \ln(1 + t_j p_{A,j}), \\
\quad \mathbf{Q}_j = \sum_{i \in Desc(j)} \mathbf{Q}_i + \ln(1 + t_j p_{A,j}) \; \rangle
\end{cases}
\tag{8}
$$

The caveat in applying the 3-point algorithm is that except for BM models and OU models (ultrametric trees only), the matrix $\mathbf{V}_{\boldsymbol{\Theta}}$ does not necessarily satisfy the 3-point condition (Ho and Ané 2014). As the authors show, it is still possible to use the algorithm in that case, provided that $\mathbf{V}_{\boldsymbol{\Theta}}$ satisfies a "generalized 3-point condition" (Ho and Ané 2014). More precisely, in most of their examples, the authors showed that there exist a transformation of the branch lengths, $\tilde{\mathbf{t}}$, diagonal matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ with non-zero diagonal elements and a 3-point structured matrix $\tilde{\mathbf{V}}_{\boldsymbol{\Theta}}$, such that $\tilde{\mathbf{V}}_{\boldsymbol{\Theta}}$ is equal to the variance-covariance on the tree $\tilde{\mathcal{T}}$ with the transformed branch lengths and $\mathbf{V}_{\boldsymbol{\Theta}} = \mathbf{D}_1 \tilde{\mathbf{V}}_{\boldsymbol{\Theta}} \mathbf{D}_2$. If so, the algorithm is applied to $\tilde{\mathbf{V}}_{\boldsymbol{\Theta}}$ using $\tilde{\mathbf{t}}$ and transformed data $\tilde{\mathbf{X}} = \mathbf{D}_2^{-1}\mathbf{X}$, $\tilde{\mathbf{Y}} = \mathbf{D}_1^{-1}\mathbf{Y}$. Then the quadratic form of interest, $\mathbf{Q}_{\boldsymbol{\Theta}}$, would be equal to the resulting quadratic form at the root, $\mathbf{Q}_M$ and the determinant $|\mathbf{V}_{\boldsymbol{\Theta}}|$ is obtained by the formula:

$$|\mathbf{V}_{\boldsymbol{\Theta}}| = |\mathbf{D}_1||\tilde{\mathbf{V}}_{\boldsymbol{\Theta}}||\mathbf{D}_2| \tag{9}$$

We note that finding a suitable transformation of the branch lengths remains a model specific task. We give an example of such branch transformation in the next showcase, referring the reader to (Ho and Ané 2014) for further examples.

A showcase: The phylogenetic Ornstein-Uhlenbeck mixed model

Here, we describe a phylogenetic Ornstein-Uhlenbeck mixed model of continuous trait evolution, which we and other authors have used previously to analyze the evolution of set-point viral load in HIV patients (Mitov and Stadler 2016; Bachmann et al. 2017; Bertels et al. 2017; Blanquart et al. 2017). To calculate the likelihood, we will use the fact that the variance covariance matrix of the POUMM has a generalized 3-point structure.

We will use a simulation based method to validate the technical correctness of the model. Then, we will report a benchmark comparing the times for likelihood calculation between different serial and parallel pruning implementations in the SPLiTTree library, as well as two third-party serial pruning implementations of the same likelihood calculation. Finally, we will report the performance gain for the Bayesian inference of the model upon combining parallel pruning with adaptive Metropolis sampling on a real dataset.

The model

Consider a real valued trait evolving independently along the lineages of a phylogenetic tree, $\mathcal{T}$ with branch lengths $\mathbf{t}$. The phylogenetic Ornstein-Uhlenbeck mixed model (POUMM) decomposes the trait value as a sum of a non-heritable component, e, and a genetic component, g, which (i) evolves continuously according to an Ornstein-Uhlenbeck (OU) process along branches; (ii) gets inherited by the branches descending from each internal node. In biological terms, g is a genotypic value (Lynch and Walsh 1998) that evolves according to random drift with stabilizing selection towards a global optimum; e is a non-heritable component, which can be interpreted in different ways, depending on the application, i.e. a measurement error, an environmental contribution, a residual with respect to a model prediction, or the sum of all these. The OU-process acting on g is parameterized by an initial genotypic value at the root, g_M, a global optimum, θ, a selection strength, $\alpha>0$, and a random drift unit-time standard deviation, σ. Denoting by W_t the standard Wiener process (Grimmett and Stirzaker 2001), the evolution of the trait-value, $z(t)$, along a given lineage of the tree is described by the equations:

$$z(t) = g(t) + e \tag{10}$$

$$dg(t) = \alpha[\theta - g(t)]dt + \sigma dW_t \tag{11}$$

$$g(0) = g_M, \tag{12}$$

The stochastic differential equation 11 defines the OU-process, which represents a random walk tending towards the global optimum θ with stronger attraction for bigger difference between $g(t)$ and θ (Ornstein and Zernike 1919; Uhlenbeck and Ornstein 1930). The model assumptions for e are that they are independent and identically distributed (i.i.d.) normal with mean 0 and standard deviation σ_e at the tips. Any process along the tree that gives rise to this distribution at the tips may be assumed for e. For example, in the case of epidemics, a newly infected individual is assigned a new e-value which represents the contribution from its immune system and this value can change or remain constant throughout the course of infection. In particular, the non-heritable component e does not influence the behavior of the OU-process $g(t)$. Thus, if we were to simulate trait values z on the tips of a phylogenetic tree $\mathcal{T}$, we could first simulate the OU-process from the root to the tips to obtain g, and then add the white noise e (i.e. an i.i.d. draw from a normal distribution) to each simulated g value at the tips. The POUMM represents an extension of the phylogenetic mixed model (PMM) (Lynch 1991; Housworth et al. 2004), since, in the limit $\alpha \to 0$, the OU-process converges to a Brownian motion (BM) with unit-time standard deviation σ. Both, the POUMM and the PMM, define an expected multivariate normal distribution for the trait values at the tips. The mean vectors and the variance-covariance matrices of these distributions are written in table 1. Note that the trait expectation and variance for a tip i depends on its height (h_i), and the trait covariance for a pair of tips (ij) depends on the height of their mrca ($h_{(ij)}$), and, in the case of POUMM, on their patristic distance (d_{ij}) (table 1).

Calculating the likelihood

The POUMM likelihood is defined as the multivariate probability density of an observed vector $\mathbf{z}$ at the tips of $\mathcal{T}$ for given model parameters $\mathbf{\Theta} =< g_M, \alpha, \theta, \sigma, \sigma_e >$:

$$\ell\ell(\mathbf{\Theta}) = \ln(f(\mathbf{z}|\mathcal{T}, \mathbf{t}, \mathbf{\Theta})). \tag{13}$$

The probability density function, f is multivariate Gaussian with mean vector $\mu_{\mathbf{\Theta}}$ and variance-covariance matrix $\mathbf{V}_{\mathbf{\Theta}}$ written in table 1. Since $\mathbf{V}_{\mathbf{\Theta}}$ has a generalized 3-point structure (Ho and Ané 2014), we can apply the recursion in eq. 8, upon a transformation of the branch lengths and the data. This is obtained through adapting the transformation for an non-mixed OU-model in a ultrametric tree (Ho and Ané 2014) to accommodate the non-heritable variance:

$$\tilde{t}_i = \frac{\sigma^2}{2\alpha}\left[e^{2\alpha T}\left(e^{2\alpha h_i} - e^{2\alpha h_{Parent(i)}}\right)\right] + \frac{\sigma_e^2}{e^{2\alpha u_i}}\delta(i \leq N) \quad \text{for } i \in \{1, ..., M-1\} \tag{14}$$

$$\tilde{\mathbf{X}}_i = \tilde{\mathbf{Y}}_i = \frac{z_i - \mu_i}{e^{\alpha u_i}} \quad \text{for } i \in \{1, ..., N\}, \tag{15}$$

where $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are identical N-vectors, T is the maximum tip-height in the tree and $u_i = T - h_i$ for $i \in \{1, ..., N\}$. After running the post-order traversal, using eq. 8 as a visit-node operation, we apply eq. 9, to obtain $|\mathbf{V}_{\mathbf{\Theta}}|$ and eq. 7 to obtain the log-likelihood.

We note that the branch transformation (eq. 14) can be done "locally" on every branch, using pre-calculated heights of the parent and daughter nodes connected by the branch. Thus, it is safe to include the transformation in the visit-node operation and the parallelization of pruning would not suffer. Otherwise, the transformation would have had to be done in a preprocessing step. Again, this is a model specific consideration.

Combined maximum likelihood and Bayesian inference of the model

We implement maximum likelihood and Bayesian inference of the POUMM parameters, $\mathbf{\Theta}$, using the L-BFGS-R convex optimization algorithm (R-function `optim`) and a variant of the Random Walk Metropolis (RWM) Markov Chain Monte Carlo (MCMC) sampling (Metropolis et al. 1953). This combined inference capitalizes on two practical ideas:

- A MCMC has higher chance to converge to the target posterior distribution faster if it has been started from a previously estimated MLE;
- If an MCMC encounters a point in the parameter space that has higher likelihood than a previously inferred MLE, running maximum likelihood optimization from that point is more likely to find the global likelihood optimum.

An important step in RWM is the choice of a proposal (jump) distribution shape matrix used as a scaling factor on each next proposal in the Metropolis algorithm. Choosing the shape matrix with respect to the scale and the correlation structure of the parameter space minimizes the number of iterations needed for MCMC convergence and mixing. Thus, numerous variants of the RWM have been proposed, performing ”on-the-fly” adaptation of the shape matrix based on what has been ”learned” about the parameter space from the past RWM iterations (Haario et al. 2001; Vihola 2012). Of these variants, we chose the adaptive Metropolis sampling with coerced acceptance rate, because it is shown to be robust with respect to the posterior distribution, it performs a relatively cheap adaptation of the shape (Vihola 2012) and it has an implementation in the R within the package `adaptMCMC` (Scheidegger 2017).

The fitting of the POUMM model was implemented as a pipeline including the following steps:

1. Perform three MLE searches using the R-function `optim` and the L-BFGS-B method (Byrd et al. 1995), starting from three randomly chosen points in parameter space;
2. Run three MCMC chains as follows: (i) a chain sampling from the prior distribution; (ii) a chain sampling from the posterior distribution and started from the MLE found in step 1; (iii) a chain sampling from the posterior distribution and started from a random point in parameters space.
3. If the parameter tuple of highest likelihood sampled in the MCMC has a likelihood higher than the MLE found in step 1, repeat the MLE search starting from that parameter tuple;

By running MLE first and starting an MCMC chain from the MLE candidate, we increase the chance that at least one of the MCMCs would converge faster to the posterior distribution. By comparing the posterior samples from two MCMCs initiated from different starting points, it can be assessed whether the MCMCs have converged to the true posterior. We do this quantitatively by the use of the Gelman-Rubin convergence diagnostic (Brooks and Gelman 1998) implemented in the R-package coda (Plummer et al. 2006). Values of the Gelman-Rubin (G.R.) statistic significantly different from 1 indicate that at least one of the two posterior samples deviates significantly from the true posterior distribution. By visual comparison of posterior density with prior desnity plots, it is possible to assess whether the data contains information differing from the prior knowledge for a given parameter. In step 3, we capitalize on the chance that the MCMCs have explored a wider region of the parameter space than the likelihood optimization.

The POUMM R-package

We implement the model in the form of an R-package called POUMM, which embeds the SPLiTTree library as an Rcpp module. Before model fitting, the user can choose from different POUMM parametrizations and prior settings (function `specifyPOUMM`). A set of standard generic functions, such as `plot`, `summary`, `logLik`, `coef`, etc., provide means to assess the quality of a fit (i.e. MCMC convergence, consistence between ML and MCMC fits) as well as various inferred properties, such as high posterior density (HPD) intervals (more details in the package user guide).

Technical correctness

To validate the correctness of the Bayesian POUMM implmentation, we used the method of posterior quantiles (Cook et al. 2006). In this method, the idea is to generate samples from the posterior quantile distributions of selected model parameters (or functions thereof) by means of numerous "replications" of simulation followed by Bayesian parameter inference. In each replication, "true" values of the model parameters are drawn from a fixed prior distribution and trait-data is simulated under the model specified by these parameter values. We perform these simulations on a fixed tree of size $N = 4000$. Then, the to-be-tested software is used to produce a posterior distribution of parameters based on the simulated trait-data. Next, the posterior quantiles of the "true" parameter values (or functions thereof) are calculated from the corresponding posterior samples generated by the to-be-tested software. By running multiple independent replications on a fixed prior, it is possible to generate large samples from the posterior quantile distributions of the individual model parameters, as well as any derived quantities. Assuming correctness of the simulations, any statistically significant deviation from uniformity of these posterior quantile samples indicates an error in the to-be-tested software (Cook et al. 2006).

Two phylogenetic trees were used for the simulations:

- Ultrametric (BD, $N = 4000$) - an ultrametric birth-death tree of 4000 tips generated using the TreeSim R-package (Stadler et al. 2013; Boskova et al. 2014) (function call: `sim.bd.taxa(4000, lambda = 2, mu = 1, frac = 1, complete = FALSE)`);
- Non-ultrametric (BD, $N = 4000$) - a non-ultrametric birth-death tree of 4000 tips generated using the TreeSim R-package (Stadler et al. 2013; Boskova et al. 2014) (function call: `sim.bdsky.stt(4000, lambdasky = 2, deathsky = 1, timesky=0)`).

Simulation scenarios of 2000 replications were run using the prior distribution $< g_M, \alpha, \theta, \sigma, \sigma_e > \sim \mathcal{N}(5, 25) \times \text{Exp}(0.1) \times \mathcal{U}(2, 8) \times \text{Exp}(0.4) \times \text{Exp}(1)$. The goal of using this prior was to explore a large enough subspace of the POUMM parameter space, while keeping MCMC convergence and mixing within reasonable time (runtime up to 30 minutes for two MCMCs of 10^6 adaptive Metropolis iterations at target acceptance rate of 1%). From the above prior, we drew a sample of $n = 2000$ parameter tuples, $\{\mathbf{\Theta}^{(1)}, ..., \mathbf{\Theta}^{(n)}\}$, which were used as replication seeds. For a given $\mathbf{\Theta}^{(i)}$, we simulate genotypic values $\mathbf{g}^{(i)}(\mathcal{T}, \mathbf{\Theta}^{(i)})$ according to an OU-branching process with initial value $g_M^{(i)}$ and parameters $\alpha^{(i)}$, $\theta^{(i)}$, $\sigma^{(i)}$. Then, we add random white noise $(\sim \mathcal{N}(0, \sigma_e^{2(i)}))$ to the genotypic values at the tips, to obtain the final trait values $\mathbf{z}^{(i)}$.

For the two simulated trees, we executed a total of $2 \times 2000 = 4000$ replications. The resulting posterior quantile distributions for the each tree are shown on Fig. 3. We notice that the posterior quantiles for all relevant parameters are uniformly distributed. This is confirmed visually by the corresponding histograms (fig. 3), as well as statistically, by a non-significant p-value from a Kolmogorov-Smirnov uniformity test at the 0.01 level. This observation validates the technical correctness of the software.

Performance on simulated data

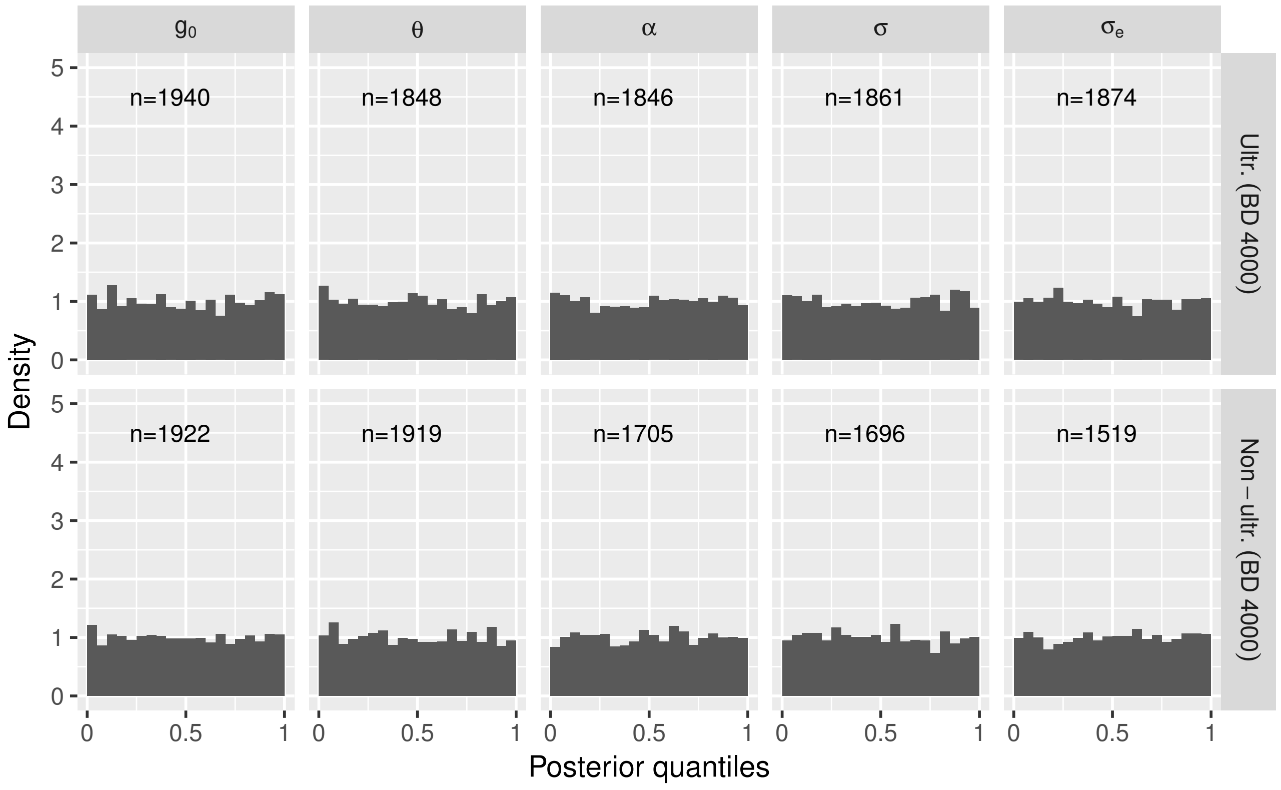

Figure 3: Posterior quantiles from simulations on a ultrametric and a non-ultrametric tree ($N = 4000$). The number n at the top of each histogram denotes the number of replications out of 2000 which reached acceptable MCMC convergence and mixing after one million iterations. Uniformity was confirmed using a Kolmogorov-Smirnov test which was insignificant for all parameters (P-value above 0.1).

We ran a performance benchmark on a MacBook Pro laptop (Retina, 15-inch, Late 2013) running a 2.3GHz Intel(R) Core i7 processor with 4 physical cores. We used the R-package `apTreeshape` (Bortolussi et al. 2012) to generate tree topologies of sizes $N \in \{100, 1000, 10,000, 100,000\}$. To generate the trees, we used the function `rtreeshape()` with a `biased` model. A parameter p in this model controls the disproportion of branching rates for the left and right lineages starting from a given parent node. For each N, we used four settings for p as follows:

1. $p = 0.5$ corresponding to equal left and right branching rates and resulting in balanced trees;
2. $p = 0.1$ corresponding to unbalanced trees in which one of any two sibling branches (sharing the same parent node) splits at rate $p = 0.1$, while the other splits at rate $p' = 1 - p = 0.9$ (time units are arbitrary, so we can assume that the rates correspond to splitting probabilities per unit time).
3. $p = 0.01$ corresponding to very unbalanced trees (splitting rates of $p = 0.01$ and $p' = 0.99$ for any couple of sibling branches;
4. $p = 0.01/N$ corresponding to a ladder-like tree (see fig. 1b).

This resulted in a total of 16 tree topologies. For each topology, random branch lengths were assigned overwriting the default branch lengths of 1 assigned by `rtreeshape()`. For each tree, we generated random trait-values using random parameters of the POUMM model.

We compared serial and parallel likelihood calculation within the POUMM package to serial pruning implementations provided in the R-packages geiger (Pennell et al. 2014), and diversitree (FitzJohn 2012). All packages including C or C++ code were compiled from source-code using the R-command `install.packages('package-directory',` `repos=NULL, type='source')`, and the same C++ compiler and compiler arguments (version 16.0.0 of the Intel compiler, command `icpc` with options `-O2 -march=native`).

Time for preprocessing the tree.—

Each of the tested packages implements a preprocessing step initializing cached data-structures that are re-used during likelihood calculation. In the case of `POUMM`, this is the constructor of the class `TraversalTask` (fig. 2); in the case of `diversitree`, this is the function `make.ou`; in the case of `geiger`, this is the internal function `bm.lik`. We note that the time for creating the cache structure is not important in scenarios of fitting comparative models to a fixed tree and data (created once, at the beginning of the inference process). These times become important in the case when the tree topology is inferred together with the model parameters from trait and sequence alignment data.

We measured the preprocessing time on the 16 trees (table 2). The times scaled linearly with the size of the tree for the packages `POUMM` and `diversitree`. For these two packages the time was not affected by the unbalancedness of the tree. For `geiger`, we observed higher time complexity, both in N as well as in the unbalancedness (longer times for unbalanced trees). For $N = 100,000$ and $p = 0.01/N$, both, `diversitree` and `geiger` failed with a `stack-overflow` error. The relatively short `POUMM` times indicate that `SPLiTTree` could potentially be used for phylogenetic inference models.

Time for one POUMM likelihood calculation.—

We distinguish the different implementations according to three criteria:

- Mode: denotes whether the implementation is single threaded using one physical core of the CPU (serial) or multi-threaded, running 8 virtual threads on 4 physical cores (parallel);
- Order: denotes the order in which the prune-able nodes are processed. We tested

Table 2: Times for tree-preprocessing in milliseconds.

N	Implementation	p=0.5	p=0.1	p=0.01	p=0.01/N
100	geiger	5	6	9	9
100	diversitree	4	4	4	4
100	POUMM	2	2	2	1
1,000	geiger	18	26	78	414
1,000	diversitree	20	20	22	30
1,000	POUMM	3	2	3	3
10,000	geiger	358	449	1,345	355,396
10,000	diversitree	207	211	227	1,338
10,000	POUMM	14	13	13	15
100,000	geiger	20,215	21,629	36,349	-
100,000	diversitree	2,421	2,619	2,883	-
100,000	POUMM	130	131	131	140

three possible orders: postorder (Mode=serial only) - the nodes are processed sequentially without paying attention to their allocation in the memory - no SIMD operations are possible; queue-based (Mode=parallel only) - the nodes are processed according to their entering order in the queue (see algorithm 1) - no SIMD operations are possible, synchronized access to the queue; range-based (Mode=parallel only) - the nodes in each pruning generation are processed in order of their allocation in memory - SIMD operations are possible, no need for a synchronized access to a queue (see algorithm 2).

- Implementation: the R-package and the back-end used (R or C++).

To measure the likelihood calculation times we used the R-package microbenchmark (Mersmann 2015) with argument `times` set to 100. The resulting average times in milliseconds are shown on fig. 4.

On small trees of 100 tips, the fastest implementations were the serial C++ implementations from the packages POUMM and diversitree (about 0.03 ms); the POUMM range-based parallel implementation was nearly as fast on balanced trees ($p = 0.5$) but was progressively slower on unbalanced trees. The geiger implementation was

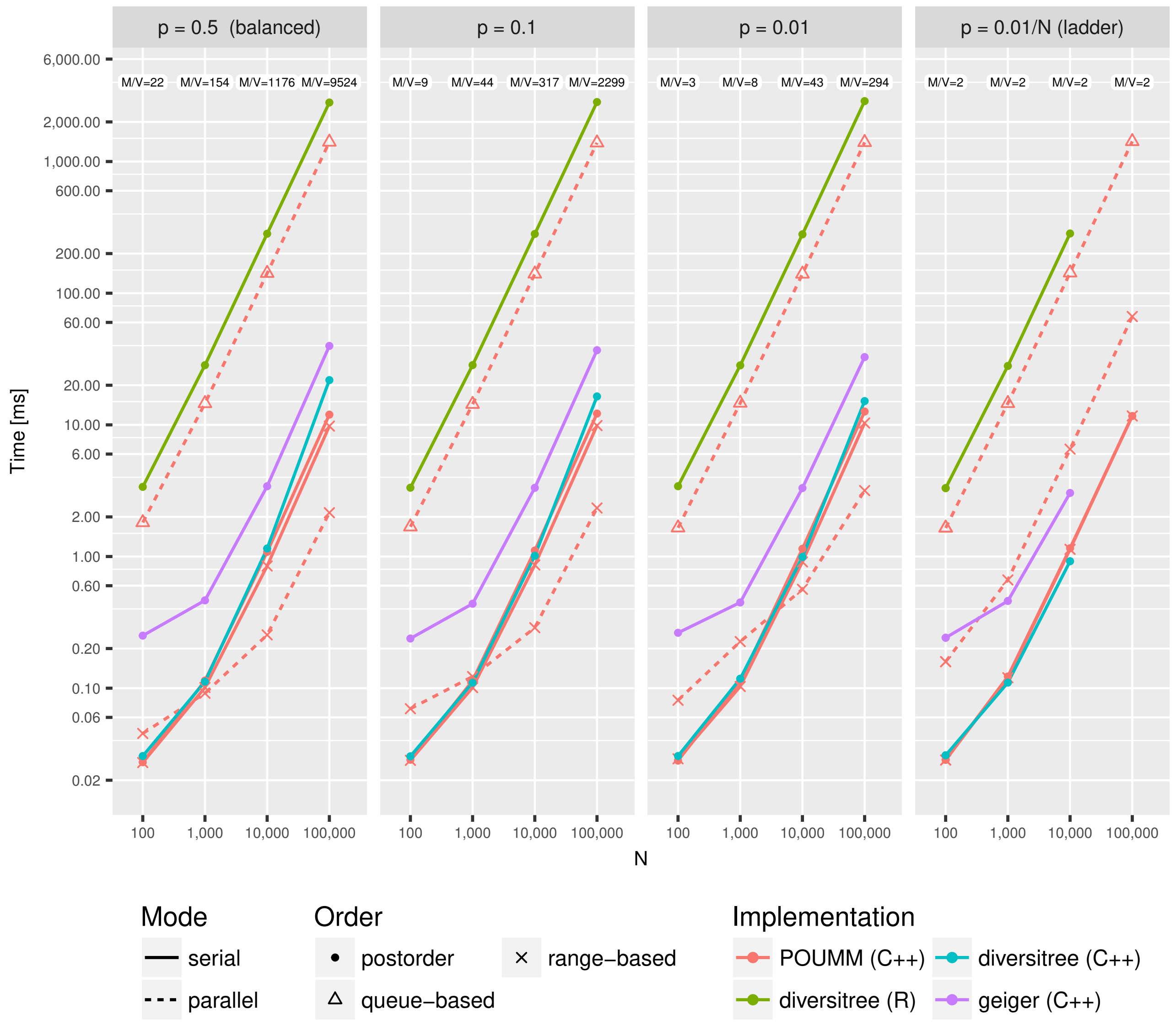

Figure 4: Likelihood calculation times for R and C++ implementations of the pruning algorithm. The labels M/V on top denote the average number of nodes per generation (Visit-range) (see algorithm 2).

nearly an order of magnitude slower (0.2 ms). The POUMM queue-based parallel implementation was nearly 100 times slower (nearly 2 ms), presumably due to the excessive synchronization overhead. The serial R implementation from the diversitree package was the slowest (above 2 ms), which was expected, since the R interpreter is notorious for its slow speed compared to compiled languages like C++.

On bigger balanced trees ($N > 100$, $p = 0.5$), the POUMM range-based parallel implementation took over, reaching up to 4× speed-up with respect to the POUMM range-based serial implementation, up to 6× speed-up with respect to the POUMM postorder serial implementation and up to 10× speed-up with respect to the diversitree serial C++ implementation. This reveals a consistent speed-up from SIMD operations for all trees except the ladder tree, where parallelization of the internal nodes is not possible (see fig. 1b). The time for the other serial implementations and the POUMM queue-based parallel implementation scaled up by a factor of 10 for each higher value of N.

There was no significant difference in the times for the serial implementations and the queue-based parallel implementation when comparing their performance on balanced versus unbalanced trees. For the POUMM range-based parallel implementation, though, the parallel speed-up was progressively less pronounced, in particular, for $N < 1000$ and for ladder trees. Still, the speed-up was very good for $N \geq 10,000$ and $p \geq 0.01$.

Improved MCMC convergence and MLE inference through adaptive Metropolis sampling.—

To measure the MCMC convergence speed-up from the adaptive Metropolis sampling, we reran one simulation scenario (2000 replications on a non-ultrametric tree of 4000 tips) with disabled adaptation. As a criterion for convergence, we used the absolute difference from 1 of the Gelman-Rubin convergence diagnostic (Brooks and Gelman 1998) (the closer $|G.R. - 1|$ is to 0, the better the convergence). When enabling adaptive Metropolis sampling, more than 1600 (80%) of the 2000 replications had reached

$|G.R. - 1| < 0.01$ after a million iterations. Conversely, when disabling adaptive Metropolis sampling, less than 300 (15%) of the replications had reached $|G.R. - 1| < 0.01$ after a million iterations (the 80% quantile of $|G.R. - 1|$ was equal to 0.57, indicating very power convergence). We also noticed that 1455 out of 2000 replications (73%) of the POUMM inferences with enabled adaptative Metropolis sampling resulted in an improved MLE after running the MCMC chains, compared to 1045 (50%) when disabling adaptation. These observations show that adaptive Metropolis sampling considerably accelerates the MCMC convergence towards the posterior distribution and can be used to improve the MLE inference when using a weak prior or a prior that does not strongly contradict with the evidence (likelihood).

Performance on real data

This showcase would be incomplete if we don't provide an assessment of the performance of the combined parallel likelihood and adaptive MCMC approach on a real dataset. We have used the POUMM package to estimate the heritability of set-point viral load in a data-set of 8,483 HIV patients. While the results of this analysis have been reported elsewhere (Mitov and Stadler 2016), here, we briefly report the times and the quality statistics for the MCMC inference of the model with and without adaptation.

First, we ran the classical RWM Metropolis sampler with a default identity shape matrix for two MCMCs of ten million iterations on the above-mentioned hardware (2.3GHz Intel(R) Core i7 processor with 4 cores), using the fastest (range-based) parallel likelihood calculation. The total time for the two MCMCs was 3:18 hours. The run resulted in poor mixing and very low effective posterior sample size for most of the inferred parameters of the model (fig. 5a,b). The Gelman-Rubin statistic was greater than 1.1 for all parameters and the effective sample size was below 400 for all parameters, falling below 50 for α and σ.

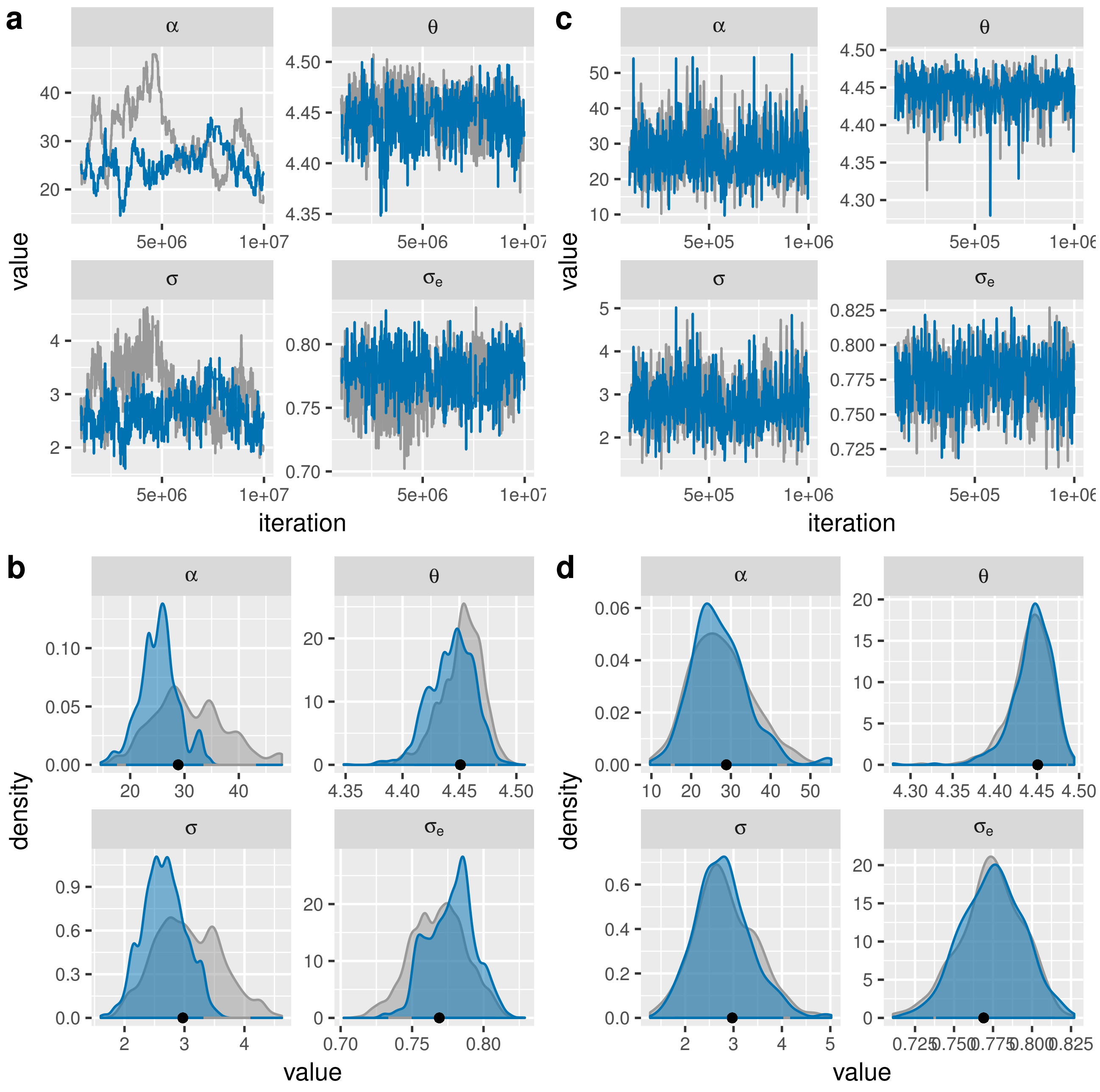

Figure 5: Sample trace- and density plots from a POUMM fit to a tree and virulence data 8483 HIV patients (Mitov and Stadler 2016) a,b: no adaptation of the proposal shape matrix (ten million iterations); c,d: on-the-fly adaptation of the proposal shape matrix from the first 100,000 out of one million iterations. The colors correspond to the different chains.

Next, we ran the adaptive Metropolis sampler for two MCMCs of one million iterations. Adaptations has been enabled only for the first 100,000 iterations in each MCMC. The total runtime was 25 minutes. The two chains mixed very well and the effective sample size for all parameters exceeded 1200 (fig. 5c,d). The difference $|G.R. - 1|$ was below 0.01 for all parameters, proving that the MCMCs have converged to the same distribution, which is very likely the true posterior distribution for the model parameters.

Discussion

SPLiTTree can in principle be used for any algorithm that runs a pre-order or post-order tree traversal. However, our benchmarks have shown that a performance gain from parallelization will in most cases depend strongly on the application and the topology of the tree. To cope with this issue, SPLiTTree implements both, serial as well as different parallel modes of the tree traversal and can chose between these implementations during the initial steps of the inference procedure.

The examples in this article focused on models of discrete and continuous trait evolution. Another family of models where SPLiTTree could be used are population models, e.g. models with structured populations. For example, when calculating the likelihood for a phylogenetic tree under a structured birth-death model, the calculations proceed in a pruning fashion (Kühnert et al. 2016) and may be improved with respect to speed using our approach. However, the structured coalescent likelihood for a tree is a function of all co-existing lineages even for approximate methods (Müller et al. 2017), and thus a pruning formulation is not available.

We did not develop examples of pre-order traversal. One such example is the simulation of traits evolving along the tree, which can be used for validation and approximate inference of phylogenetic models. In complex phylogenetic comparative models, where an exact calculation of the likelihood is elusive or computationally intractable, it is possible to use simulations of trait evolution along the tree for approximate likelihood calculation (Kutsukake and Innan 2013) or approximate Bayesian computation (ABC) (Slater et al. 2012b). Both approaches are computationally intensive and could benefit from parallel execution using our framework.

The `POUMM` R-package joins a growing collection of tools implementing phylogenetic OU inference. Among others, these include the R-packages `ape v4.0` (Paradis et al. 2004), `ouch v2.9-2` (Butler and King 2004), `GLSME v1.0.3` (Hansen and Bartoszek 2012), `diversitree v0.9-9` (FitzJohn 2012), `geiger v2.0.6` (Pennell et al. 2014), `surface v0.4-1` (Ingram and Mahler 2013), `mvMORPH v1.0.8` (Clavel et al. 2015), `bayou v1.0.0` (Uyeda et al. 2015), `OUwie v1.50` (Beaulieu and OMeara 2016), `phylolm v2.5` (Ho and Ané 2014), `RPANDA v1.1` (Manceau et al. 2016). Compared to these packages POUMM provides fast Bayesian inference using the combined parallel likelihood and adaptive sampling approach. Another feature of POUMM is that it implements different re-parametrizations of $\mathbf{\Theta} =< g_M, \alpha, \theta, \sigma, \sigma_e >$. This is helpful in Bayesian inference, because it allows to express the prior distribution in application-specific terms, such as phylogenetic heritability (Lynch 1991; Housworth et al. 2004).

Outlook

The past decade has seen a rapid advance in the production of multi-core and SIMD processors. At the same time, it appears that the maximum clock frequency of a single processing unit is approaching the maximum achievable for semi-conductor based architectures. This brings the need for development of novel parallel algorithms capitalizing on the multi-core technology. The parallel tree traversal library should enable parallel computation for a vast set of applications facing the challenges of increasing model complexity and volumes of data in phylogenetic analysis.

Supplementary Material

Data from the performance benchmarks and simulations for technical correctness is available on the dryad database. The POUMM package and user guide is available at `https://CRAN.R-project.org/package=POUMM`. The package also includes the source code of the SPLiTTree framework.

Funding

V.M. and T.S. thank ETH Zürich for funding. T.S. is supported in part by the European Research Council under the 7th Framework Programme of the European Commission (PhyPD: Grant Agreement Number 335529).

Acknowledgements

We thank Dr. Krzysztof Bartoszek for valuable insights on the Ornstein-Uhlenbeck process.

*

References

Ayres, D. L., A. Darling, D. J. Zwickl, P. Beerli, M. T. Holder, P. O. Lewis, J. P. Huelsenbeck, F. Ronquist, D. L. Swofford, M. P. Cummings, A. Rambaut, and M. A. Suchard. 2012. BEAGLE: An Application Programming Interface and High-Performance Computing Library for Statistical Phylogenetics. Systematic Biology 61:170–173.

Bachmann, N., T. Turk, C. Kadelka, A. Marzel, M. Shilaih, J. Böni, V. Aubert, T. Klimkait, G. E. Leventhal, H. F. Günthard, R. Kouyos, and Swiss HIV Cohort Study. 2017. Parent-offspring regression to estimate the heritability of an HIV-1 trait in a realistic setup. Retrovirology 14:33.

Beaulieu, J. M. and B. OMeara. 2016. OUwie: Analysis of Evolutionary Rates in an OU Framework .

Bertels, F., A. Marzel, G. Leventhal, V. Mitov, J. Fellay, H. F. Günthard, J. Böni, S. Yerly, T. Klimkait, V. Aubert, M. Battegay, A. Rauch, M. Cavassini, A. Calmy, E. Bernasconi, P. Schmid, A. U. Scherrer, V. Müller, S. Bonhoeffer, R. Kouyos, R. R. Regoes, and Swiss HIV Cohort Study. 2017. Dissecting HIV Virulence: Heritability of Setpoint Viral Load, CD4+ T Cell Decline and Per-Parasite Pathogenicity. Molecular biology and evolution .

Blanquart, F. o., C. Wymant, M. Cornelissen, A. Gall, M. Bakker, D. Bezemer, M. Hall, M. Hillebregt, S. H. Ong, J. Albert, N. Bannert, J. Fellay, K. Fransen, A. J. Gourlay, M. K. Grabowski, B. Gunsenheimer-Bartmeyer, H. F. G nthard, P. Kivel, R. Kouyos, O. Laeyendecker, K. Liitsola, L. Meyer, K. Porter, M. Ristola, A. van Sighem, G. Vanham, B. Berkhout, P. Kellam, P. Reiss, C. Fraser, and BEEHIVE collaboration. 2017. Viral genetic variation accounts for a third of variability in HIV-1 set-point viral load in Europe. Plos Biology 15:e2001855.

Bortolussi, N., E. Durand, M. Blum, and O. Francois. 2012. apTreeshape: Analyses of Phylogenetic Treeshape. R package .

Boskova, V., S. Bonhoeffer, and T. Stadler. 2014. Inference of Epidemiological Dynamics Based on Simulated Phylogenies Using Birth-Death and Coalescent Models. PLoS Computational Biology (PLOSCB) 10(4) 10:e1003913.

Bouckaert, R. R., J. Heled, D. Kühnert, T. G. Vaughan, C.-H. Wu, D. Xie, M. A. Suchard, A. Rambaut, and A. J. Drummond. 2014. BEAST 2 - A Software Platform for Bayesian Evolutionary Analysis. PLoS Computational Biology (PLOSCB) 10(4) 10:e1003537–.

Boyd, S. P. and L. Vandenberghe. 2004. Convex Optimization. Cambridge University Press.

Brooks, S. P. and A. Gelman. 1998. General methods for monitoring convergence of iterative simulations. Journal of Computational and Graphical Statistics 7:434–455.

Butler, M. A. and A. A. King. 2004. Phylogenetic comparative analysis: A modeling approach for adaptive evolution. American Naturalist 164:683–695.

Byrd, R. H., P. Lu, J. Nocedal, and C. Y. Zhu. 1995. A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing 16:1190–1208.

Clavel, J., G. Escarguel, and G. Merceron. 2015. mvmorph: an r package for fitting multivariate evolutionary models to morphometric data. Methods in Ecology and Evolution 6:1311–1319.

Cook, S. R., A. Gelman, and D. B. Rubin. 2006. Validation of Software for Bayesian Models Using Posterior Quantiles. Journal of Computational and Graphical Statistics 15:675–692.

Drummond, A. J., M. A. Suchard, D. Xie, and A. Rambaut. 2012. Bayesian phylogenetics with BEAUti and the BEAST 1.7. Molecular biology and evolution 29:1969–1973.

Felsenstein, J. 1973. Maximum-likelihood estimation of evolutionary trees from continuous characters. American Journal of Human Genetics 25:471–492.

Felsenstein, J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of molecular evolution 17:368–376.

Felsenstein, J. 1983. Statistical Inference of Phylogenies. Journal of the Royal Statistical Society. Series A (General) 146:246.

FitzJohn, R. G. 2012. Diversitree: comparative phylogenetic analyses of diversification in R. Methods in Ecology and Evolution 3:1084–1092.

Goolsby, E. W., J. Bruggeman, and C. Ané. 2016. Rphylopars: fast multivariate phylogenetic comparative methods for missing data and within-species variation. Methods in Ecology and Evolution 8:22–27.

Grimmett, G. and D. Stirzaker. 2001. Probability and Random Processes. Oxford University Press.

Haario, H., E. Saksman, and J. Tamminen. 2001. An adaptive metropolis algorithm. Bernoulli. Official Journal of the Bernoulli Society for Mathematical Statistics and Probability 7:223–242.

Hansen, T. F. 1997. Stabilizing Selection and the Comparative Analysis of Adaptation. Evolution; international journal of organic evolution 51:1341–1351.

Hansen, T. F. and K. Bartoszek. 2012. Interpreting the evolutionary regression: the interplay between observational and biological errors in phylogenetic comparative studies. Systematic Biology 61:413–425.

Ho, L. s. T. and C. Ané. 2014. A linear-time algorithm for Gaussian and non-Gaussian trait evolution models. Systematic Biology 63:397–408.

Hodcroft, E., J. D. Hadfield, E. Fearnhill, A. Phillips, D. Dunn, S. O'Shea, D. Pillay, A. J. L. Brown, o. b. o. t. U. H. D. R. Database, and t. U. C. Study. 2014. The Contribution of Viral Genotype to Plasma Viral Set-Point in HIV Infection. PLoS pathogens 10:e1004112.

Housworth, E. A., E. P. Martins, and M. Lynch. 2004. The phylogenetic mixed model. The American Naturalist 163:84–96.

Ingram, T. and D. L. Mahler. 2013. SURFACE: detecting convergent evolution from comparative data by fitting Ornstein-Uhlenbeck models with stepwise Akaike Information Criterion. Methods in Ecology and Evolution 4:416–425.

Ives, A. R. and T. J. Garland. 2010. Phylogenetic Logistic Regression for Binary Dependent Variables. Systematic Biology 59:9–26.

Kühnert, D., T. Stadler, T. G. Vaughan, and A. J. Drummond. 2016. Phylodynamics with Migration: A Computational Framework to Quantify Population Structure from Genomic Data. Molecular biology and evolution 33:msw064–2116.

Kutsukake, N. and H. Innan. 2013. Simulation-Based Likelihood Approach for Evolutionary Models of Phenotypic Traits on Phylogeny. Evolution; international journal of organic evolution 67:355–367.

Lynch, M. 1991. Methods for the Analysis of Comparative Data in Evolutionary Biology. Evolution; international journal of organic evolution 45:1065–1080.

Lynch, M. and B. Walsh. 1998. Genetics and Analysis of Quantitative Traits. Sinauer Associates Incorporated.

Manceau, M., A. Lambert, and H. Morlon. 2016. A unifying comparative phylogenetic framework including traits coevolving across interacting lineages. Systematic Biology 66:syw115–568.

Mersmann, O. 2015. Accurate Timing Functions [R package microbenchmark version 1.4-2.1] .

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. 1953. Equation of State Calculations by Fast Computing Machines. The Journal of Chemical Physics 21:1087–1092.

Mitov, V. and T. Stadler. 2016. The heritability of pathogen traits - definitions and estimators. bioRxiv Page 058503.

Müller, N. F., D. A. Rasmussen, and T. Stadler. 2017. The Structured Coalescent and Its Approximations. Molecular biology and evolution 34:2970–2981.

O'Meara, B. C. 2012. Evolutionary Inferences from Phylogenies: A Review of Methods. Annual Review of Ecology, Evolution, and Systematics 43:267–285.

Ornstein, L. S. and F. Zernike. 1919. The theory of the Brownian Motion and statistical mechanics. Proceedings of the Koninklijke Akademie Van Wetenschappen Te Amsterdam 21:109–114.

Pagel, M. 1994. Detecting Correlated Evolution on Phylogenies - a General-Method for the Comparative-Analysis of Discrete Characters. Proceedings of the Royal Society B-Biological Sciences 255:37–45.

Paradis, E. and J. Claude. 2002. Analysis of comparative data using generalized estimating equations. Journal of theoretical biology 218:175–185.

Paradis, E., J. Claude, and K. Strimmer. 2004. APE: Analyses of Phylogenetics and Evolution in R language. Bioinformatics 20:289–290.

Pennell, M. W., J. M. Eastman, G. J. Slater, J. W. Brown, J. C. Uyeda, R. G. FitzJohn, M. E. Alfaro, and L. J. Harmon. 2014. geiger v2.0: an expanded suite of methods for fitting macroevolutionary models to phylogenetic trees. Bioinformatics 30:2216–2218.

Plummer, M., N. Best, K. Cowles, and K. Vines. 2006. CODA: Convergence Diagnosis and Output Analysis for MCMC. R News 6:7–11.

Ronquist, F. and J. P. Huelsenbeck. 2003. MrBayes 3: Bayesian phylogenetic inference under mixed models. Bioinformatics 19:1572–1574.

Scheidegger, A. 2017. adaptMCMC. R package .

Slater, G. J., L. J. Harmon, and M. E. Alfaro. 2012a. Integrating Fossils With Molecular Phylogenies Improves Inference Of Trait Evolution. Evolution; international journal of organic evolution 66:3931–3944.

Slater, G. J., L. J. Harmon, D. Wegmann, P. Joyce, L. J. Revell, and M. E. Alfaro. 2012b. Fitting Models of Continuous Trait Evolution to Incompletely Sampled Comparative Data Using Approximate Bayesian Computation. Evolution; international journal of organic evolution 66:752–762.

Stadler, T., D. Kühnert, S. Bonhoeffer, and A. J. Drummond. 2013. Birth-death skyline plot reveals temporal changes of epidemic spread in HIV and hepatitis C virus (HCV). PNAS 110:228–233.

Uhlenbeck, G. E. and L. S. Ornstein. 1930. On the Theory of the Brownian Motion. Physical Review 36:823–841.

Uyeda, J. C., J. Eastman, and L. Harmon. 2015. bayou: Bayesian Fitting of Ornstein-Uhlenbeck Models to Phylogenies .

Vihola, M. 2012. Robust adaptive Metropolis algorithm with coerced acceptance rate. Statistics and Computing 22:997–1008.