

Parallel Likelihood Calculation for Phylogenetic Comparative Models: the SPLITT C++ Library - SUPPLEMENTARY INFORMATION

Venelin Mitov,^{*,1,2} Tanja Stadler,^{1,2}

¹Department of Biosystems, Science and Engineering (D-BSSE)

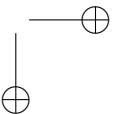
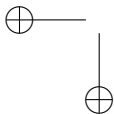
²Swiss Federal Institute of Technology (ETH), Zürich, Switzerland

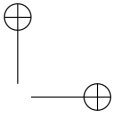
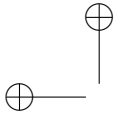
***Corresponding author:** E-mail: vmitov@gmail.com

Contents

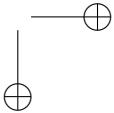
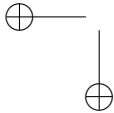
Parallel Likelihood Calculation for Phylogenetic Comparative Models: the SPLITT C++ Library - SUPPLEMENTARY INFORMATION	1
1 A general framework for parallel tree traversal	3
1.1 Strategies for parallel post-order tree traversal (pruning)	4
1.1.1 Queue-based parallel pruning	4
1.1.2 Range-based parallel pruning	4
1.1.3 Hybrid parallel/sequential strategies	6
1.1.4 Parallel pruning orders	7
2 Examples of using the parallel tree traversal framework	8
2.1 Example 1. The generalized 3-point structure pruning algorithm for Gaussian models of continuous trait evolution	8
2.2 Example 2: Calculating the POUMM log-likelihood	10
2.2.1 The model	10
2.2.2 Generalized 3-point structure of the POUMM variance-covariance matrix	12
2.2.3 A quadratic polynomial representation of the POUMM log-likelihood	12
2.3 Example 3: Models of categorical trait evolution	16
3 The POUMM R-package	17
3.1 Model inference	18
3.2 Technical correctness	19
4 Supplementary results from the performance benchmarks	20

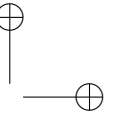
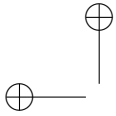
SI





5 Combined speed-up from parallel likelihood calculation and adaptive Metropolis sampling





1 A general framework for parallel tree traversal

Through the rest of this document, we use the following mathematical notation. Given is a rooted phylogenetic tree \mathcal{T} with a total of M nodes, including $N < M$ tips denoted $1, \dots, N$, $M - N - 1$ internal nodes denoted $N + 1, \dots, M - 1$, and a root node denoted M (Fig. 1a,d). 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 ($Desc(j) = \emptyset$ 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 . 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 Θ when denoting the vectors of all branch lengths, input data and parameters.

Let $F_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \Theta)$ 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 $S_j(\mathbf{t}, \mathbf{z}, \Theta)$, hereby called "states", satisfying the following rules:

- (1) $F_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \Theta)$ can be calculated from $S_M(\mathbf{t}, \mathbf{z}, \Theta)$;
- (2) For each node $j \in \{1, \dots, M\}$, there exists a (recursive) relationship R_j between S_j and the set of states at j 's descendants, such that:

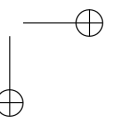
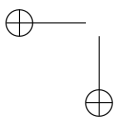
$$S_j(\mathbf{t}, \mathbf{z}, \Theta) = R_j \left(\{S_i(\mathbf{t}, \mathbf{z}, \Theta) : i \in Desc(j)\}, \mathbf{t}, \mathbf{z}, \Theta \right). \quad (\text{S1})$$

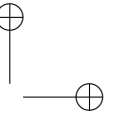
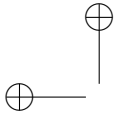
We note that analogical terms can be defined for pre-order tree traversal. In this case the target functions are values $Z_{\mathcal{T},j}(\mathbf{t}, \mathbf{z}, \Theta)$ corresponding to the nodes $j \in \{1, \dots, M\}$, and rule (2) is updated to:

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

$$Z_j(\mathbf{t}, \mathbf{z}, \Theta) = R'_j(Z_{Parent(j)}(\mathbf{t}, \mathbf{z}, \Theta), \mathbf{t}, \mathbf{z}, \Theta). \quad (\text{S2})$$

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.





In Supplementary information, we provide 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 time Markov model for a categorical or a continuous trait. In terms of the framework, the task boils down to formulating the node states $S_j(\mathbf{t}, \mathbf{z}, \Theta)$ and the recursive functions R_j satisfying rules (1) and (2).

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

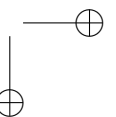
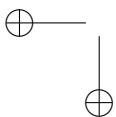
1.1 Strategies for parallel post-order tree traversal (pruning)

1.1.1 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 S1). 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 S1 guarantees that a node will be processed immediately after all of its direct descendants have been processed. Thus, in theory, Algorithm S1 maximizes the parallel execution. However, an implementation of the atomic operations on the queue and the counters would have to rely on a thread synchronization 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 .

1.1.2 Range-based parallel pruning

We consider an alternative of Algorithm S1 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 S2).



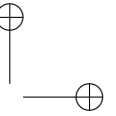
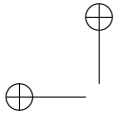
Algorithm S1 Queue-based parallel pruning**Input:** \mathcal{T} , \mathbf{t} , \mathbf{z} , Θ **Output:** $S_M(\mathbf{t}, \mathbf{z}, \Theta)$

```

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

```

In Algorithm S2, 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 (panel $p=0.5$ on Fig. 2). 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 (rightmost panels on Fig. 2). This would result in low parallel speed-up and excessive synchronization cost for both, the queue-based and the range-based algorithms. Also noteworthy is the fact that Algorithm S2 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.



Algorithm S2 Range-based parallel pruning

Input: \mathcal{T} , \mathbf{t} , \mathbf{z} , Θ
Output: $S_M(\mathbf{t}, \mathbf{z}, \Theta)$
Data:

```

/* A pre-calculated vector with starting offsets for each generation: */
12  $Range = [0, N, N + |G_1|, N + |G_1| + |G_2|, \dots, M - 1, M]_{K+1}$  /* a vector of  $M$  elements */
13  $State \leftarrow [0, \dots, 0]_M$ 

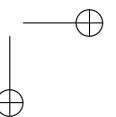
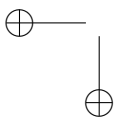
/* The master thread iterates over the generations: */
14 foreach  $k \in \{1, \dots, K\}$  do
    /* The master thread starts a team of worker threads running equal portions of the
       following loop: */
    15 foreach  $j \in \{Range[k] + 1, \dots, Range[k + 1]\}$  do
    16      $State[j] \leftarrow R_j\left(\{State[i] : i \in Desc(j)\}, \mathbf{t}, \mathbf{z}, \Theta\right)$ 
17 return  $State[M]$ 

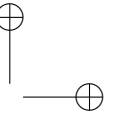
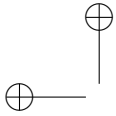
```

These and other subtleties indicate that there is no “one size fits all” strategy when it comes to maximizing parallel speed-up. The framework provides two ways to deal with these: (a) allowing the user to choose a parallelization mode before executing a pruning procedure 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.

1.1.3 Hybrid parallel/sequential strategies

An important problem occurring with all parallel pruning strategies is that the number of lineages tends to decrease exponentially towards the root of the tree. As a result, if the original thread-team consisted of numerous threads each one reserving one of multiple processing cores, there will be many idle threads/cores as the pruning approaches the root. While this issue could potentially be solved at the level of the multi-threading back-end, it is possible to implement a hybrid pruning strategy similar to the rake-compression algorithm described in (Reif, 1989). Reif (1989) introduce two operations: rake (could be seen as the parallel calculation on the nodes in one generation) and compress (compression of chains). For example, on a ladder tree (see panel $p=0.01/N$ (ladder) on Fig. 2), after visiting the generation of tips, one obtains a chain: the chain should be processed sequentially on one thread (compressed), thus, reducing the synchronization thread-starvation issues.





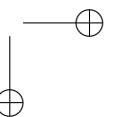
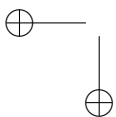
1.1.4 Parallel pruning orders

As described in the main text, to maximize the potential for parallel execution SPLITT divides the evaluation of the recursive functions R_j in a *InitNode*, a *VisitNode*, and a *PruneNode* operation (see Fig. 1c,d). Further, SPLITT rearranges the node-ids (and, hence the data) associated with the nodes in the computer memory, so that the *InitNode* and the *VisitNode* operations can be performed on ranges of consecutive node-ids (Fig. 1c). Performing these operations on a range of consecutive addresses in the memory has a potential to increase the efficiency of computation, because modern CPUs provide vectorized instruction sets, i.e. low-level processor instructions, such as (addition and multiplication) that can be executed simultaneously on a group of 2, 4 or more consecutive words in the memory. Another benefit from ordering the *VisitNode* operations over such ranges is that the data associated with these nodes will tend to be less fragmented and, therefore, it will be found more often in the processor cache.

In general, there is no arrangement of the nodes in the memory, in which both, the *VisitNode* and the *PruneNode* operations will be executed on ranges of consecutive nodes. For example, the range of daughter-nodes 1, 2, 3 (see step 1. Prune-range [1-3], Fig. 1d) corresponds to the parent nodes 6, 8 and 7. In a bigger tree, the parents could well be nodes 6, 25, 12 or any other non-consecutive ids. Thus, the efficiency of the calculation can, in principle, be affected by the order in which the ranges of *VisitNode*- and *PruneNode*-operations are processed. SPLITT provides several possible arrangements (called “orders”) of the execution cycle of these operations, which are all equivalent with respect to the result of the calculation, but may have different parallel efficiency, depending on the tree and the application-specific traversal operations. In the performance benchmark, we denote by “parallel range” the order which is illustrated on Fig. 1d. Specifically, this reflects the following cycle:

1. First, execute in parallel $InitNode(i)$ for each $1 \leq i \leq M$;
2. Then, for each prune-range $[r_b, r_e]$, execute in parallel $\{ VisitNode(i); PruneNode(i, Parent(i)); \}$ for each $r_b \leq i \leq r_e$. The prune-ranges are formed as subsets of “Visit”-able nodes with contiguous ids that are not siblings, i.e. do not have a shared parent. This guarantees that the *PruneNode* operation is synchronized between siblings.

Other orders are documented in the SPLITT online manual. Except for the noticeably slower performance of the queue-based order, no significant difference in the execution times has been observed between the other parallel orders (not reported).



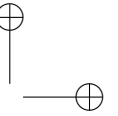
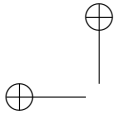


Table S1. Population properties at the tips of the phylogeny under BM and OU models and their mixed counterparts. The acronyms are: PBM - Phylogenetic Brownian motion (without non-heritable component); PMM - Phylogenetic Mixed Model (adding a non-heritable component to PBM); POU - Phylogenetic Ornstein-Uhlenbeck (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). $\mu_{\Theta,i}$: expected value at tip i ; $\mathbf{V}_{\Theta,ii}$: expected variance for tip i ; $\mathbf{V}_{\Theta,ij}$: expected covariance of the values of tips i

and j . 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}).

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

2 Examples of using the parallel tree traversal framework

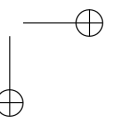
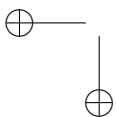
In this section, we show example usages of the parallel traversal framework. In each of these examples, we solve a particular problem, such as calculating the likelihood of a continuous Markov model for a categorical or a continuous trait. In terms of the framework, the task boils down to formulating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

2.1 Example 1. The generalized 3-point structure pruning algorithm for Gaussian 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 $|\mathbf{V}_{\Theta}|$ and quadratic quantities of the form $\mathbf{Q}_{\Theta} = \mathbf{X}_{\Theta}' \mathbf{V}_{\Theta}^{-1} \mathbf{Y}_{\Theta}$, where \mathbf{V}_{Θ} represents the variance covariance matrix expected under the model specified by Θ and the matrices \mathbf{X}_{Θ} and \mathbf{Y}_{Θ} represent centered observed data at the tips in the tree. For example, in the case of Brownian motion and Ornstein-Uhlenbeck models, the log-likelihood function is equal to the log-density of a multivariate Gaussian distribution:

$$\ln f(\mathbf{z}|\Theta) = -\frac{1}{2} (N \ln(2\pi) + \ln |\mathbf{V}_{\Theta}| + (\mathbf{z} - \mu_{\Theta})' \mathbf{V}_{\Theta}^{-1} (\mathbf{z} - \mu_{\Theta})), \quad (\text{S3})$$

where μ_{Θ} and \mathbf{V}_{Θ} denote functions of the tree and the model parameters Θ representing the expectation under the model for the mean-vector and the variance-covariance matrix of the trait-values at the tips in the tree. The mathematical formulas of each element of μ_{Θ} and \mathbf{V}_{Θ} for four single-trait phylogenetic comparative models are given in Table S1.

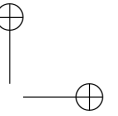
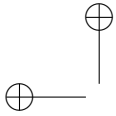


Ho and Ané (2014) developed a pruning algorithm which allows to calculate $|\mathbf{V}_\Theta|$ and \mathbf{Q}_Θ simultaneously and without constructing or allocating the matrix \mathbf{V}_Θ in memory, provided \mathbf{V}_Θ has a "3-point structure". Then, they showed several examples of Gaussian models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

$$\left\{ \begin{array}{ll} S_j(\mathbf{t}, \mathbf{z}, \Theta) = \langle p_{A,j}=0, & \\ & p_j = \frac{1}{t_j}, \\ & \hat{\mu}_{Y,j} = \mathbf{y}_{\Theta,j}, \\ & \tilde{\mu}'_{X,j} = \mathbf{x}'_{\Theta,j}, \\ & \ln|\mathbf{V}|_j = \ln t_j, \\ & \mathbf{Q}_j = \mathbf{x}'_{\Theta,j} \mathbf{y}_{\Theta,j} \rangle & \text{if } j \leq N \\ \\ S_j(\mathbf{t}, \mathbf{z}, \Theta) = \langle p_{A,j} = \sum_{i \in Desc(j)} p_i, & \\ & p_j = \frac{p_{A,j}}{1 + t_j p_{A,j}}, \\ & \hat{\mu}_{Y,j} = \sum_{i \in Desc(j)} \frac{p_i}{p_A} \hat{\mu}_{Y,i}, \\ & \tilde{\mu}'_{X,j} = \sum_{i \in Desc(j)} \frac{p_i}{p_A} \tilde{\mu}'_{X,i}, \\ & \ln|\mathbf{V}|_j = \sum_{i \in Desc(j)} \ln|\mathbf{V}|_i + \ln(1 + t_j p_{A,j}), \\ & \mathbf{Q}_j = \sum_{i \in Desc(j)} \mathbf{Q}_i + \ln(1 + t_j p_{A,j}) \rangle & \text{otherwise.} \end{array} \right. \quad (\text{S4})$$

A SPLITT-based implementation of the above pruning scheme is available at <https://github.com/venelin/ThreePointUsingSPLITT.git>.

The caveat in applying the 3-point algorithm is that except for BM models, the matrix \mathbf{V}_Θ 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}_Θ 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 $\tilde{\mathbf{V}}_\Theta$, such that $\tilde{\mathbf{V}}_\Theta$ is equal to the variance-covariance on the tree $\tilde{\mathcal{T}}$ with the transformed branch lengths and $\mathbf{V}_\Theta = \mathbf{D}_1 \tilde{\mathbf{V}}_\Theta \mathbf{D}_2$. If so, the algorithm is applied to $\tilde{\mathbf{V}}_\Theta$ 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}_{Θ} , would be equal to the resulting quadratic form at the root, \mathbf{Q}_M and the determinant $|\mathbf{V}_{\Theta}|$ is obtained by the formula:

$$|\mathbf{V}_{\Theta}| = |\mathbf{D}_1| |\tilde{\mathbf{V}}_{\Theta}| |\mathbf{D}_2| \quad (\text{S5})$$

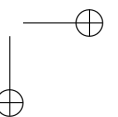
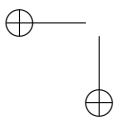
Nevertheless, the branch transformation required for a model to satisfy the 3-point condition is specific for every model. This makes it hard to apply the 3-point algorithm in general for single-trait and multi-trait Gaussian models. An alternative approach is proposed in the next example.

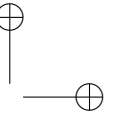
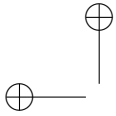
2.2 Example 2: Calculating the POUMM log-likelihood

In this section we describe two pruning algorithms for calculating the likelihood of the single-trait phylogenetic Ornstein-Uhlenbeck mixed model. First, we briefly introduce the model, its biological interpretation and its parameters. Then, we show how the likelihood can be calculated using the 3-point algorithm described in the previous example, and using a quadratic polynomial representation of the likelihood function in terms of the root value. The code for these two alternative implementations is available at <https://github.com/venelin/ThreePointUsingSPLITT.git> and at <https://github.com/venelin/POUMM.git>, respectively. For the example in the main text (Fig. 1), we have used the quadratic polynomial likelihood representation of the PMM model, which represents a simpler special case of the POUMM. For this example, the code is available at <https://github.com/venelin/PMMUsingSPLITT.git>.

2.2.1 The model

The P(OU)MM models the evolution of a continuous trait, z , along the lineages of a phylogenetic tree \mathcal{T} with branch lengths \mathbf{t} . The trait value is modeled as a sum of a non-heritable component, e , and a heritable component, g , which (i) evolves continuously according to a Brownian motion (BM) or an Ornstein-Uhlenbeck (OU) process along each branch; (ii) gets inherited by the branches descending from each internal node. In biological terms, g is interpreted as the contribution to the phenotype that has evolved in association with species' genetic sequence used to build the phylogeny; 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 BM/OU stochastic process modeling the evolution of g is defined as a system of equations:





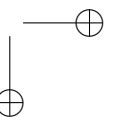
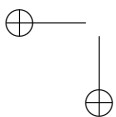
$$z(t) = g(t) + e \quad (\text{S6})$$

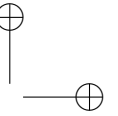
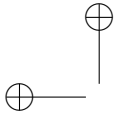
$$dg(t) = \alpha[\theta - g(t)]dt + \sigma dW_t \quad (\text{S7})$$

$$g(0) = g_M, \quad (\text{S8})$$

where g_M denotes the initial value of g at the root of \mathcal{T} , θ denotes a long-term optimum value for the trait, $\alpha > 0$ denotes the strength of selection (i.e. long-term tendency of g towards θ), $\sigma > 0$ denotes the unit-time standard deviation of fluctuations in g due to random drift and W_t denotes the standard Wiener process (Grimmett and Stirzaker, 2001). The stochastic differential equation S7 defines an OU-process, representing a random walk tending towards θ with stronger attraction for bigger difference between $g(t)$ and θ and/or bigger selection strength α (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 $\sigma_e > 0$ 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 behaviour of the stochastic process $g(t)$. Thus, if we were to simulate trait values z on the tips of \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. In the limit $\alpha \rightarrow 0$, the OU-process converges to a BM-process with unit-time standard deviation σ (Fig. 1b).

Here we describe two ways to calculate the POUMM likelihood using a post-order traversal of the tree, which can be easily incorporated with the framework. The first approach is based on the generalized 3-point structure algorithm (Ho and Ané, 2014). This approach has the caveat that it requires a model-specific branch-length transformation. The second approach is based on direct integration over the ancestor genotypic values at the internal nodes, capitalizing on a recurrent quadratic polynomial formula. Previously, a similar integration technique has been described in (FitzJohn, 2012). The advantage of the quadratic polynomial representation described here is that it can be generalized to multivariate OU models as well as a more general class of Gaussian models (Mitov *et al.*, 2018).





2.2.2 Generalized 3-point structure of the POUMM variance-covariance matrix

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 $\Theta = \langle g_M, \alpha, \theta, \sigma, \sigma_e \rangle$:

$$\ell\ell(\Theta) = \ln(f(\mathbf{z}|\mathcal{T}, \mathbf{t}, \Theta)). \quad (\text{S9})$$

The probability density function, f is multivariate Gaussian with mean vector μ_Θ and variance-covariance matrix \mathbf{V}_Θ written in Table S1. Since \mathbf{V}_Θ has a generalized 3-point structure (Ho and Ané, 2014), we can apply the recursion in eq. S4, 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:

$$\tilde{t}_i = \frac{\sigma^2}{2\alpha} \left[e^{2\alpha T} (e^{2\alpha h_i} - e^{2\alpha h_{\text{Parent}(i)}}) \right] + \frac{\sigma_e^2}{e^{2\alpha u_i}} \delta(i \leq N) \text{ for } i \in \{1, \dots, M-1\} \quad (\text{S10})$$

$$\tilde{\mathbf{X}}_i = \tilde{\mathbf{Y}}_i = \frac{z_i - \mu_i}{e^{\alpha u_i}} \text{ for } i \in \{1, \dots, N\}, \quad (\text{S11})$$

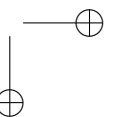
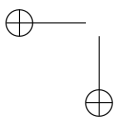
where $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are identical N -vectors, T is the maximum tip-height in the tree and $u_i = T - h_i$ for $i \in \{1, \dots, N\}$. After running the post-order traversal, using eq. S4 as a VisitNode operation, we apply eq. S5, to obtain $|\mathbf{V}_\Theta|$ and eq. S3 to obtain the log-likelihood.

We note that the branch transformation (eq. S10) 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 VisitNode 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.

2.2.3 A quadratic polynomial representation of the POUMM log-likelihood

We begin by defining for each nodes j states, $S_j(\mathbf{t}, \mathbf{z}, \Theta)$, and recursive functions R_j that allow the calculation of the likelihood, $\ell\ell(\Theta)$, from S_M . It turns out that $\ell\ell(\Theta)$ has a simple representation as a quadratic polynomial of g_M (root state), which can be obtained by pruning-wise integration over the unobserved internal node states, g_i , progressing from the tips to the root. We formalize this idea in the following theorem:

THEOREM S1 (Recurrent quadratic polynomial representation of the POUMM log-likelihood). *For $\alpha \geq 0$, a real θ and positive σ and σ_e , the POUMM log-likelihood can be expressed as a quadratic polynomial of*



g_M :

$$\ell\ell(\Theta) = a_M g_M^2 + b_M g_M + c_M, \quad (\text{S12})$$

where $a_M < 0$, b_M and c_M are real coefficients. We denote by $u(\alpha, t)$ the function:

$$u(\alpha, t) := \begin{cases} \alpha/(1 - e^{\alpha t}), & \text{for } \alpha > 0 \\ -1/t, & \text{for } \alpha = 0 \end{cases} \quad (\text{S13})$$

Then, the coefficients in eq. S12 can be expressed with the following recurrent relation:

1. For $j \in \{1, \dots, N\}$ (tips):

$$a_j = -\frac{1}{2\sigma_e^2}; b_j = \frac{z_j}{\sigma_e^2}; c_j = -\frac{z_j^2}{2\sigma_e^2} - \ln \sqrt{2\pi\sigma_e^2} \quad (\text{S14})$$

2. For $j > N$ (internal nodes) or $j = M$ (root):

$$\begin{aligned} a_j &= \sum_{i \in \text{Desc}(j)} \frac{a_i u(\alpha, 2t_i)}{u(\alpha, 2t_i) - \alpha + \sigma^2 a_i} \\ b_j &= \sum_{i \in \text{Desc}(j)} \frac{u(\alpha, 2t_i) [2\theta a_i (e^{\alpha t_i} - 1) + b_i e^{\alpha t_i}]}{u(\alpha, 2t_i) - \alpha + \sigma^2 a_i} \\ c_j &= \sum_{i \in \text{Desc}(j)} \left\{ c_i + \alpha t_i - \frac{0.25 b_i^2 \sigma^2}{-\alpha + a_i \sigma^2 + u(\alpha, 2t_i)} - \right. \\ &\quad \left. 0.5 \ln \left(\frac{-\alpha + a_i \sigma^2 + u(\alpha, 2t_i)}{u(\alpha, 2t_i)} \right) + \right. \\ &\quad \left. \frac{\alpha \theta [a_i \theta - (b_i + a_i \theta) e^{\alpha t_i}]}{u(\alpha, t_i) + (-\alpha + a_i \sigma^2)(1 + e^{\alpha t_i})} \right\}. \end{aligned} \quad (\text{S15})$$

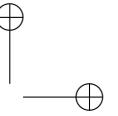
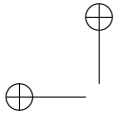
PROOF. Induction from the tips to the root of the tree.

- *Basis*: For a tip-node i , \mathcal{T}_i is the trivial tree consisting of this tip-node only and the pdf of \mathbf{z}_i , conditioned on the unobservable genotypic value g_i , is given by the normal pdf with mean g_i and variance σ_e^2 . This pdf can be written as:

$$\begin{aligned} f(\mathbf{z}_i | g_i; \sigma_e) &= \mathcal{N}(z_i; g_i, \sigma_e^2) \\ &= \frac{1}{\sqrt{2\pi\sigma_e^2}} e^{-\frac{(z_i - g_i)^2}{2\sigma_e^2}} \\ &= e^{-\frac{1}{2\sigma_e^2} g_i^2 + \frac{z_i}{\sigma_e^2} g_i - \frac{z_i^2}{2\sigma_e^2} - 0.5 \ln(2\pi\sigma_e^2)} \end{aligned} \quad (\text{S16})$$

By defining $a_i = -\frac{1}{2\sigma_e^2}$, $b_i = \frac{z_i}{\sigma_e^2}$ and $c_i = -\frac{z_i^2}{2\sigma_e^2} - 0.5 \ln(2\pi\sigma_e^2)$ and taking the natural logarithm of the pdf we obtain the log-likelihood representation from eq. S12, where $a_M < 0$, b_M and c_M can be calculated from the observed value z_i and the model parameter σ_e .

- *Inductive hypothesis*: Assume that for an internal node j , the statement of the theorem has been proven for all subtrees \mathcal{T}_i , $i \in \text{Desc}(j)$.
- *Inductive step*: Assuming that g_j is known, we consider the OU process starting from g_j and parametrized by α and σ . Under this process, the expected distribution at time t_i is normal with



mean $\mu_{ji} = e^{-\alpha t_i} g_j + (1 - e^{-\alpha t_i}) \theta$ and variance $\sigma_{ji}^2 = (1 - e^{-2\alpha t_i}) \frac{\sigma^2}{2\alpha}$. Then, the probability of \mathbf{z}_i given g_j is given by the integral

$$\begin{aligned}
 f(\mathbf{z}_i | \Theta, t_i, g_j) &= \int_{-\infty}^{\infty} f(g_i | \Theta, t_i, g_j) \times e^{a_i g_i^2 + b_i g_i + c_i} dg_i \\
 &= \int_{-\infty}^{\infty} \mathcal{N}[g_i; \mu_{ji}, \sigma_{ji}^2] \times e^{a_i g_i^2 + b_i g_i + c_i} dg_i \\
 &= \int_{-\infty}^{\infty} e^{(p_{ji} + a_i) g_i^2 + (q_{ji} + b_i) g_i + (r_{ji} + c_i)} dg_i, \text{ where} \\
 p_{ji} &= -\frac{1}{2\sigma_{ji}^2} = -\frac{\alpha e^{2\alpha t_i}}{\sigma^2(e^{2\alpha t_i} - 1)} \\
 q_{ji} &= \frac{\mu_{ji}}{\sigma_{ji}^2} = \frac{2\alpha e^{\alpha t_i} [g_j + \theta(e^{\alpha t_i} - 1)]}{\sigma^2(e^{2\alpha t_i} - 1)} \\
 r_{ji} &= -\frac{\mu_{ji}^2}{2\sigma_{ji}^2} - \frac{1}{2} \ln(2\pi\sigma_{ji}^2) \\
 &= -\frac{\alpha [g_j + \theta(e^{\alpha t_i} - 1)]^2}{\sigma^2(e^{2\alpha t_i} - 1)} - \frac{1}{2} \ln\left(\frac{\pi\sigma^2(1 - e^{-2\alpha t_i})}{\alpha}\right)
 \end{aligned} \tag{S17}$$

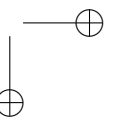
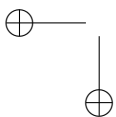
We notice that p_{ji} , q_{ji} and r_{ji} in eq. S17 are not defined in the case of BM ($\alpha=0$). In this case, we take the limit for $\alpha \rightarrow 0$ represented by the case $\alpha=0$ of function $u(\alpha, t)$ (eq. S13). By substituting $u(\alpha, t)$ in the expressions for p_{ji} , q_{ji} , r_{ji} (eq. S17) we obtain:

$$\begin{aligned}
 p_{ji} &= \frac{e^{2\alpha t_i} u(\alpha, 2t_i)}{\sigma^2} \\
 q_{ji} &= -\frac{u(\alpha, 2t_i) [g_j + \theta(e^{\alpha t_i} - 1)]}{\sigma^2} \\
 r_{ji} &= \frac{u(\alpha, 2t_i) [g_j + \theta(e^{\alpha t_i} - 1)]^2}{\sigma^2} - \frac{1}{2} \ln\left(-\frac{\pi\sigma^2}{u(\alpha, 2t_i)e^{2\alpha t_i}}\right).
 \end{aligned} \tag{S18}$$

Since $a_i < 0$ and, for positive t and $\alpha \in [0, \infty)$, $u(\alpha, t)$ accepts strictly negative values in the interval $[-1/t, 0)$, the integral in eq. S17 has a closed form solution:

$$\begin{aligned}
 &\int_{-\infty}^{\infty} e^{(p_{ji} + a_i) g_i^2 + (q_{ji} + b_i) g_i + (r_{ji} + c_i)} dg_i \\
 &= \exp\left[\frac{-(q_{ji} + b_i)^2}{4(p_{ji} + a_i)} + (r_{ji} + c_i) + \ln\left(\sqrt{\frac{\pi}{-(p_{ji} + a_i)}}\right)\right] \\
 &= e^{a_{ji} g_j^2 + b_{ji} g_j + c_{ji}}, \text{ where} \\
 a_{ji} &= \frac{a_i u(\alpha, 2t_i)}{u(\alpha, 2t_i) - \alpha + \sigma^2 a_i} \\
 b_{ji} &= \frac{u(\alpha, 2t_i)(e^{\alpha t_i}(2\theta a_i + b_i) - 2\theta a_i)}{u(\alpha, 2t_i) - \alpha + \sigma^2 a_i} \\
 c_{ji} &= c_i + \alpha t_i - \frac{0.25 b_i^2 \sigma^2}{-\alpha + a_i \sigma^2 + u(\alpha, 2t_i)} - \\
 &\quad 0.5 \ln\left(\frac{-\alpha + a_i \sigma^2 + u(\alpha, 2t_i)}{u(\alpha, 2t_i)}\right) + \frac{\alpha \theta [a_i \theta - (b_i + a_i \theta) e^{\alpha t_i}]}{u(\alpha, t_i) + (-\alpha + a_i \sigma^2)(1 + e^{\alpha t_i})}
 \end{aligned} \tag{S19}$$

In eq. S19 above, $a_{ji} < 0$ because it is a fraction with a positive nominator and a negative denominator (note that $a_i < 0$ by the inductive hypothesis and $u(\alpha, 2t_i) < 0$ by definition). Since the vectors \mathbf{z}_i , $i \in Desc(j)$, are conditionally independent given g_j , the conditional pdf of \mathbf{z}_j factorizes as:



$$\begin{aligned}
 f(\mathbf{z}_j | \Theta, g_j, \mathcal{T}_j) &= \prod_{i \in Desc(j)} f(\mathbf{z}_i | \Theta, t_i, g_j) \\
 &= \prod_{i \in Desc(j)} e^{a_{ji}g_j^2 + b_{ji}g_j + c_{ji}} \\
 &= \exp \left[\left(\sum_{i \in Desc(j)} a_{ji} \right) g_j^2 + \left(\sum_{i \in Desc(j)} b_{ji} \right) g_j + \sum_{i \in Desc(j)} c_{ji} \right].
 \end{aligned} \tag{S20}$$

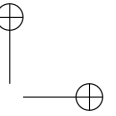
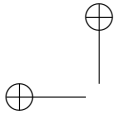
By denoting $a_j = \sum_{i \in Desc(j)} a_{ji}$, $b_j = \sum_{i \in Desc(j)} b_{ji}$ and $c_j = \sum_{i \in Desc(j)} c_{ji}$ and noticing that $a_j < 0$ as a sum of negative terms, we have proven the inductive step and, thus, the theorem.

□

The following two corollaries give the recursive functions R_j for the POUMM and the PMM model, respectively. These follow from the eqs. S14 and S15, theorem S1.

COROLLARY S1. *The recursive functions R_j for obtaining the quadratic polynomial representation of the POUMM log-likelihood (eq. S12) are given by the equation:*

$$\left\{ \begin{array}{l} S_j(\mathbf{t}, \mathbf{z}, \Theta) = \langle a_j = -0.5/\sigma_e^2, \\ \quad b_j = z_j/\sigma_e^2, \\ \quad c_j = -0.5[z_j^2/\sigma_e^2 + \ln(2\pi\sigma_e^2)] \rangle \\ S_j(\mathbf{t}, \mathbf{z}, \Theta) = \langle a_j = \sum_{i \in Desc(j)} \frac{a_i u(\alpha, 2t_i)}{u(\alpha, 2t_i) - \alpha + \sigma^2 a_i}, \\ \quad b_j = \sum_{i \in Desc(j)} \frac{u(\alpha, 2t_i)[2\theta a_i(e^{\alpha t_i} - 1) + b_i e^{\alpha t_i}]}{u(\alpha, 2t_i) - \alpha + \sigma^2 a_i}, \\ \quad c_j = \sum_{i \in Desc(j)} \left[c_i + \alpha t_i - \frac{0.25 b_i^2 \sigma^2}{-\alpha + a_i \sigma^2 + u(\alpha, 2t_i)} - \right. \\ \quad \quad \left. 0.5 \ln \left(\frac{-\alpha + a_i \sigma^2 + u(\alpha, 2t_i)}{u(\alpha, 2t_i)} \right) + \right. \\ \quad \quad \left. \frac{\alpha \theta [a_i \theta - (b_i + a_i \theta) e^{\alpha t_i}]}{u(\alpha, t_i) + (-\alpha + a_i \sigma^2)(1 + e^{\alpha t_i})} \right] \rangle \end{array} \right. \quad \begin{array}{l} \text{if } j \leq N \\ \\ \\ \text{otherwise.} \end{array} \tag{S21}$$



COROLLARY S2. *The recursive functions R_j for obtaining the quadratic polynomial representation of the PMM log-likelihood (eq. S12) are given by the equation:*

$$\left\{ \begin{array}{ll} S_j(\mathbf{t}, \mathbf{z}, \Theta) = \langle \begin{array}{l} a_j = -0.5/\sigma_e^2, \\ b_j = z_j/\sigma_e^2, \\ c_j = -0.5[z_j^2/\sigma_e^2 + \ln(2\pi\sigma_e^2)] \end{array} \rangle & \text{if } j \leq N \\ S_j(\mathbf{t}, \mathbf{z}, \Theta) = \langle \begin{array}{l} a_j = \sum_{i \in Desc(j)} a_i / (1 - 2a_i\sigma^2 t_i), \\ b_j = \sum_{i \in Desc(j)} b_i / (1 - 2a_i\sigma^2 t_i), \\ c_j = \sum_{i \in Desc(j)} \left[c_i - \ln \sqrt{1 - 2a_i\sigma^2 t_i} + b_i^2\sigma^2 t_i / (2 - 4a_i\sigma^2 t_i) \right] \end{array} \rangle & \text{otherwise.} \end{array} \right. \quad (\text{S22})$$

2.3 Example 3: Models of categorical trait evolution

It is worthy mentioning that SPLITT can be readily applied to any pruning-wise calculation, including calculating the likelihoods of categorical trait models. Here, we show how this can be done for a binary trait substitution model. The code for this example is available at <https://github.com/venelin/BinaryPoissonUsingSPLITT.git>.

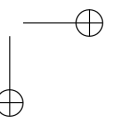
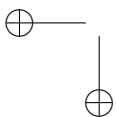
Consider a trait taking values in $\{0,1\}$ evolving independently along the lineages of a phylogenetic tree, \mathcal{T} with branch lengths \mathbf{t} . A continuous-time Markov model can be used to characterize the transitions of the trait value along each branch (Felsenstein, 1983; Pagel, 1994). 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:

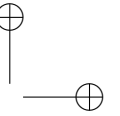
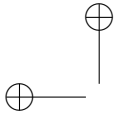
$$\mathbf{Q} = \begin{bmatrix} -q_{01} & q_{01} \\ q_{10} & -q_{10} \end{bmatrix}. \quad (\text{S23})$$

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

$$\mathbf{P}(t) = \begin{bmatrix} P_{00}(t) & P_{01}(t) \\ P_{10}(t) & P_{11}(t) \end{bmatrix} = \mathbf{C} \begin{bmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{bmatrix} \mathbf{C}^{-1} \quad (\text{S24})$$

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 $\Theta = (q_{01}, q_{10})$ generates an N -vector of values, \mathbf{z} observed at the tips. This represents the conditional likelihood $L_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \Theta, z_M)$. The pruning algorithm for calculating L relies on calculating the “fragmentary” likelihood $L_i(b) = P(\mathbf{z}_i | z_i = b; \Theta)$ for each node i and each





$b \in \{0, 1\}$ (Felsenstein, 1983). In terms of the framework, we define the state $S_j(\mathbf{t}, \mathbf{z}, \Theta)$ of a node j as the pair $\langle L_j(0), L_j(1) \rangle$. Following eq. 4 in (Felsenstein, 1983), the recursive R_j are given by:

$$S_j(\mathbf{t}, \mathbf{z}, \Theta) = \begin{cases} \langle \delta(z_j=0), \delta(z_j=1) \rangle & \text{if } j \text{ is a tip} \\ \left\langle \prod_{i \in \text{Desc}(j)} [\sum_{z_i} P_{0z_i}(t_i) L_i(z_i)], \prod_{i \in \text{Desc}(j)} [\sum_{z_i} P_{1z_i}(t_i) L_i(z_i)] \right\rangle & \text{if } j \text{ is internal,} \end{cases} \quad (\text{S25})$$

where we use the Kronecker delta function $\delta(x=y)$ equalling to 1 if $x=y$ and 0, otherwise. In the above equation S25, the values $L_i(z_i)$ are available from the descendants' states S_i . Finally, the conditional likelihood $L_{\mathcal{T}}(\mathbf{t}, \mathbf{z}, \Theta, z_M)$ 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

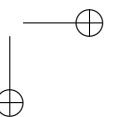
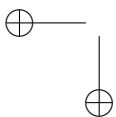
$$S_j(\mathbf{t}, \mathbf{z}, \Theta) = \begin{bmatrix} L_j^{(1)}(A) & L_j^{(1)}(C) & L_j^{(1)}(T) & L_j^{(1)}(G) \\ \vdots & \vdots & \vdots & \vdots \\ L_j^{(p)}(A) & L_j^{(p)}(C) & L_j^{(p)}(T) & L_j^{(p)}(G) \end{bmatrix}, \quad (\text{S26})$$

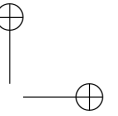
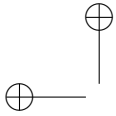
where the letters A , C , T and G denote the nucleotides and the superscript in parentheses denotes a site in the alignment. To define the recursive functions R_j , equation S25 can be extended to accommodate one row of S_j (four possible values instead of two) and evaluated p times to obtain the full 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 procedure. Thus, accommodating such models in the framework, although involved technically, should not present a conceptual challenge.

3 The POUMM R-package

We implement the POUMM model in the form of an R-package called `POUMM`, which embeds the `SPLITT` library as an Rcpp module (Eddelbuettel and Sanderson, 2014). 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).





3.1 Model inference

We implement maximum likelihood and Bayesian inference of the POUMM parameters, Θ , 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:

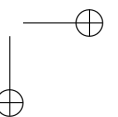
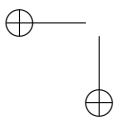
- A MCMC has higher chance to converge to the target posterior distribution faster if it has been started from a previously estimated MLE;
- If an MCMC encounters a point in the parameter space that has higher likelihood than a previously inferred MLE, running maximum likelihood optimization from that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

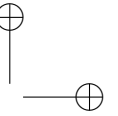
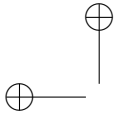
An important step in RWM is the choice of a proposal (jump) distribution shape matrix used as a scaling factor on each next proposal in the Metropolis algorithm. Choosing the shape matrix with respect to the scale and the correlation structure of the parameter space minimizes the number of iterations 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 density 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.

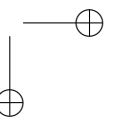
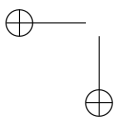
3.2 Technical correctness

To validate the correctness of the Bayesian POUMM implementation, 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 $\langle g_M, \alpha, \theta, \sigma, \sigma_e \rangle \sim \mathcal{N}(5, 25) \times \text{Exp}(0.1) \times \mathcal{U}(2, 8) \times \text{Exp}(0.4) \times \text{Exp}(1)$. The goal of using this prior was to explore a large



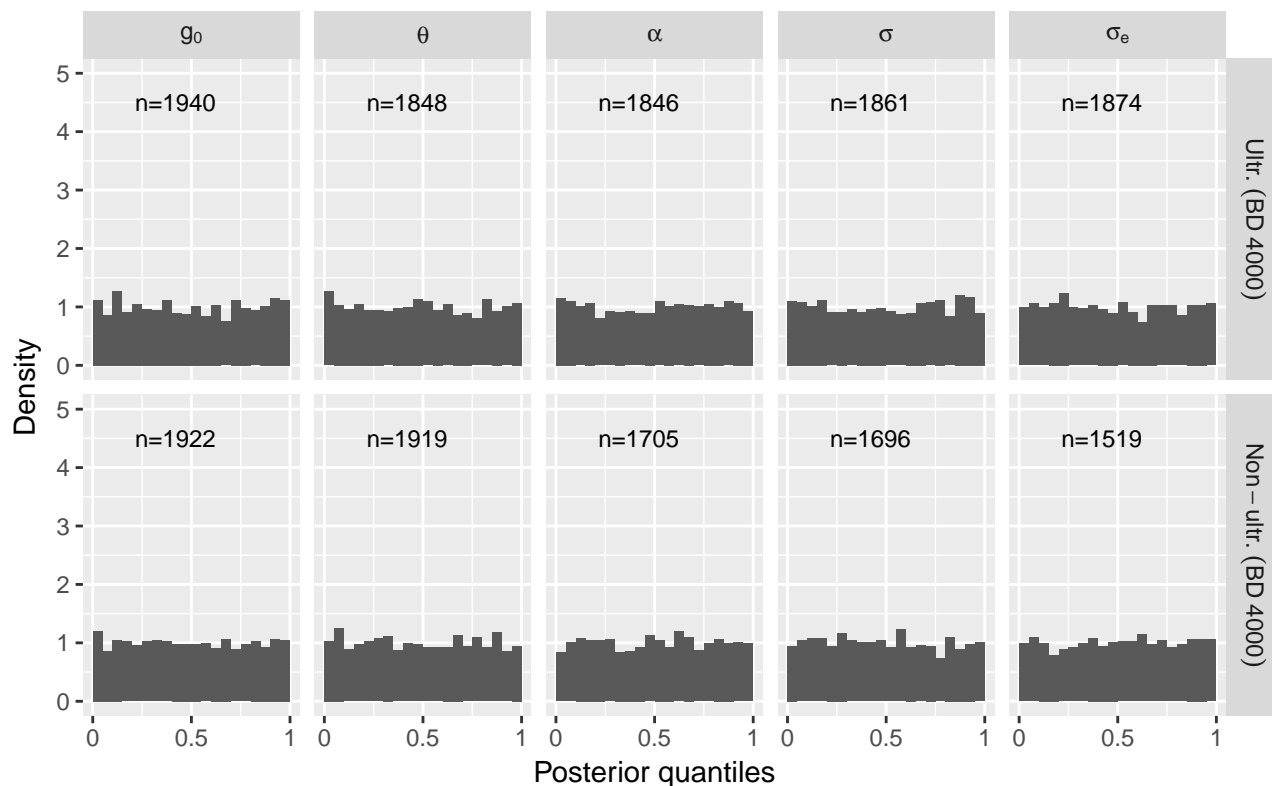


FIG. S1. Posterior quantiles from simulations on a ultrametric and a non-ultrametric tree ($N=4000$). The number n at the top of each histogram denotes the number of replications out of 2000 which reached acceptable MCMC convergence and mixing after one million iterations. Uniformity was confirmed using a Kolmogorov-Smirnov test which was insignificant for all parameters (P-value above 0.1).

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, $\{\Theta^{(1)}, \dots, \Theta^{(n)}\}$, which were used as replication seeds. For a given $\Theta^{(i)}$, we simulate genotypic values $\mathbf{g}^{(i)}(\mathcal{T}, \Theta^{(i)})$ 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. S1. We notice that the posterior quantiles for all relevant parameters are uniformly distributed. This is confirmed visually by the corresponding histograms (Fig. S1), 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.

4 Supplementary results from the performance benchmarks

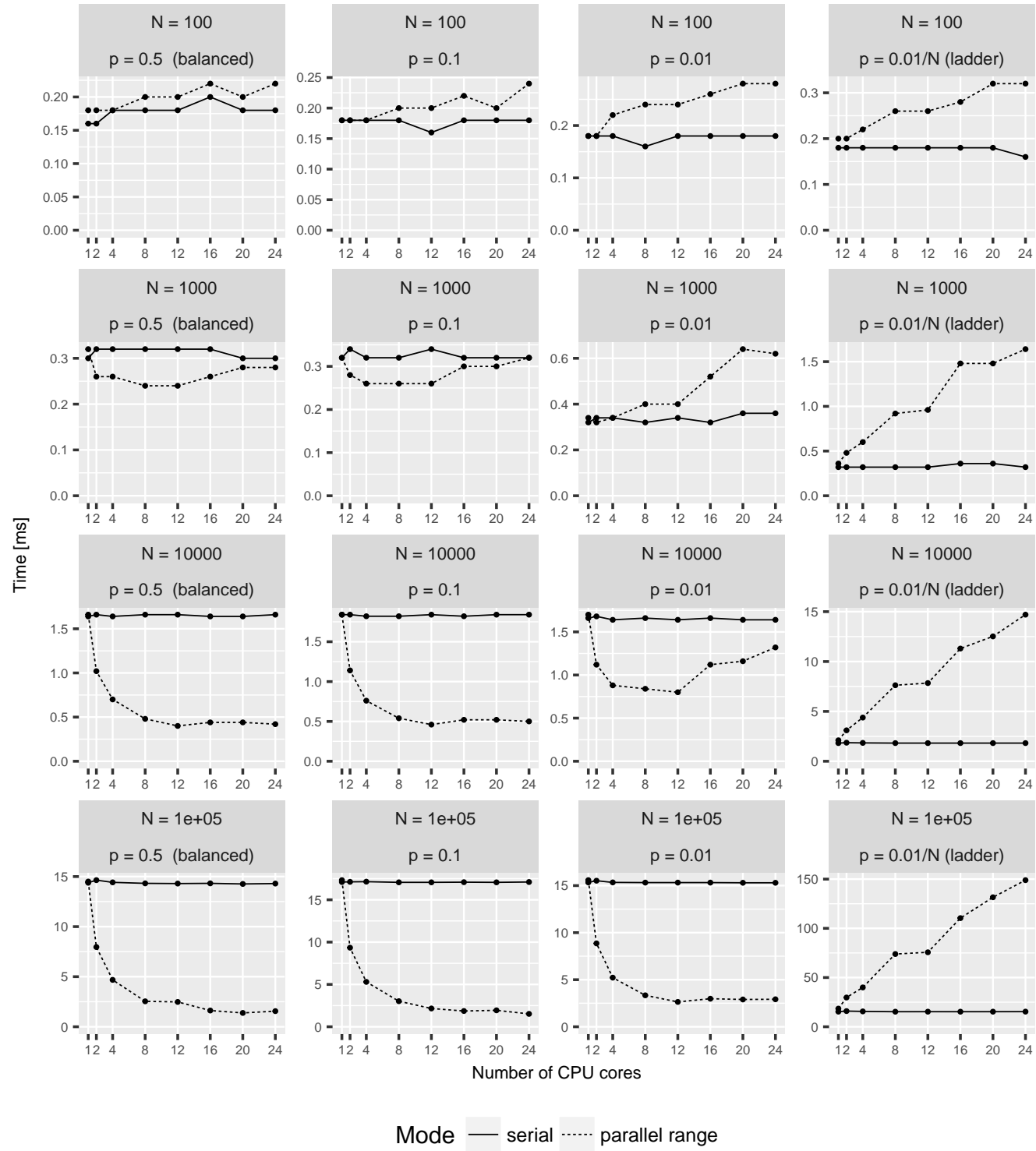


FIG. S2. Likelihood calculation times for the single-trait POUMM implementation (package POUMM) on Euler cluster (a single shared memory node with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz running 24 physical cores). Both, the x -axis denoting the number of cores, and the y -axis denoting the calculation time in milliseconds are on the linear scale. Horizontally, the panels correspond to the different tree topologies, see also Fig. 2. Vertically, the panels correspond to the different tree-sizes. For visualization purpose, only the times for the serial postorder and the fastest parallel algorithm (parallel range-based) are shown. The times for the parallel queue-based implementation were significantly higher.

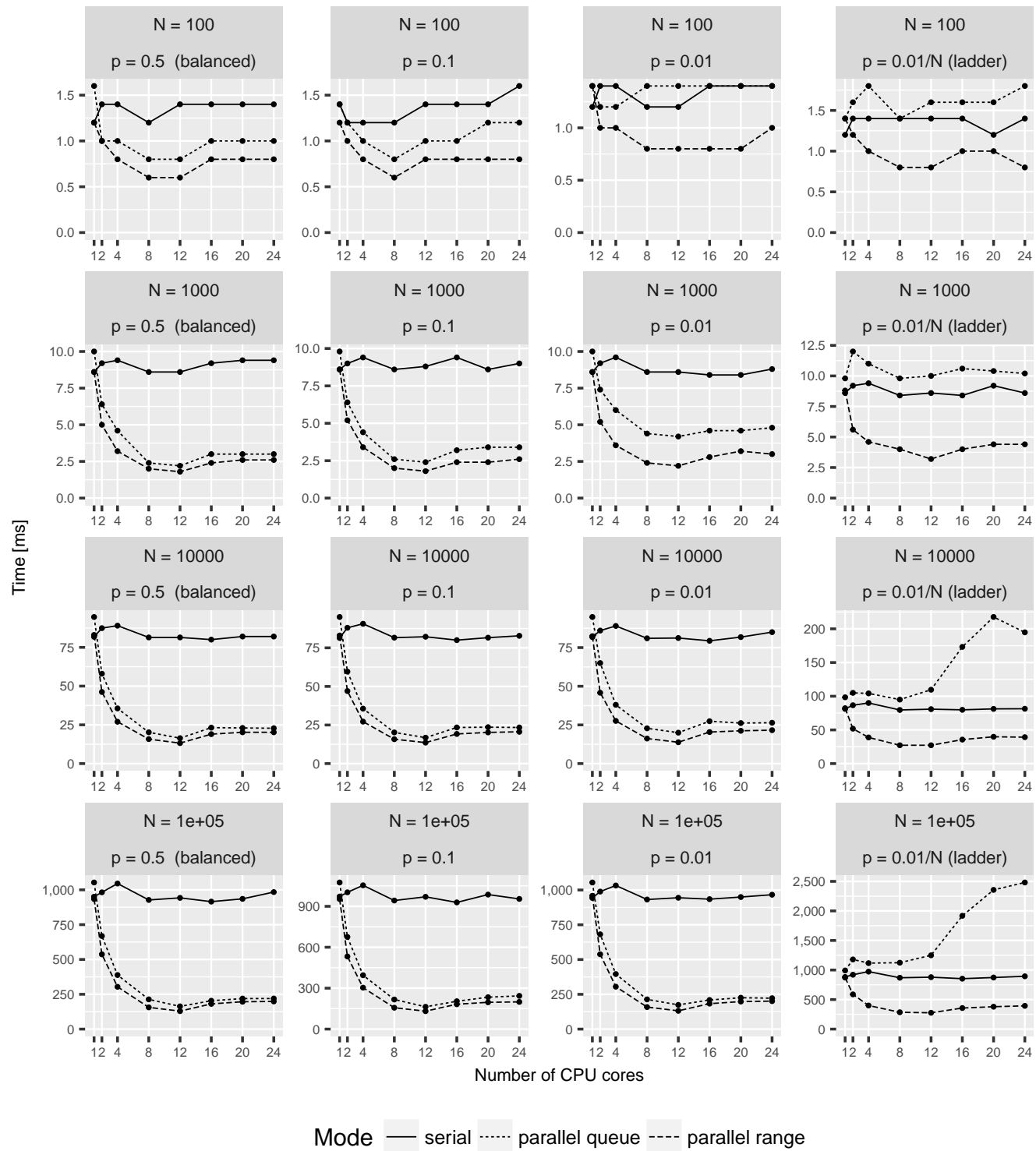


FIG. S3. Likelihood calculation times for the multi-trait POUMM implementation (package `PCMBaseCpp`) with 1 trait on Euler cluster (a single shared memory node with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz running 24 physical cores). Both, the x -axis denoting the number of cores, and the y -axis denoting the calculation time in milliseconds are on the linear scale. Horizontally, the panels correspond to the different tree topologies, see also Fig. 2. Vertically, the panels correspond to the different tree-sizes.

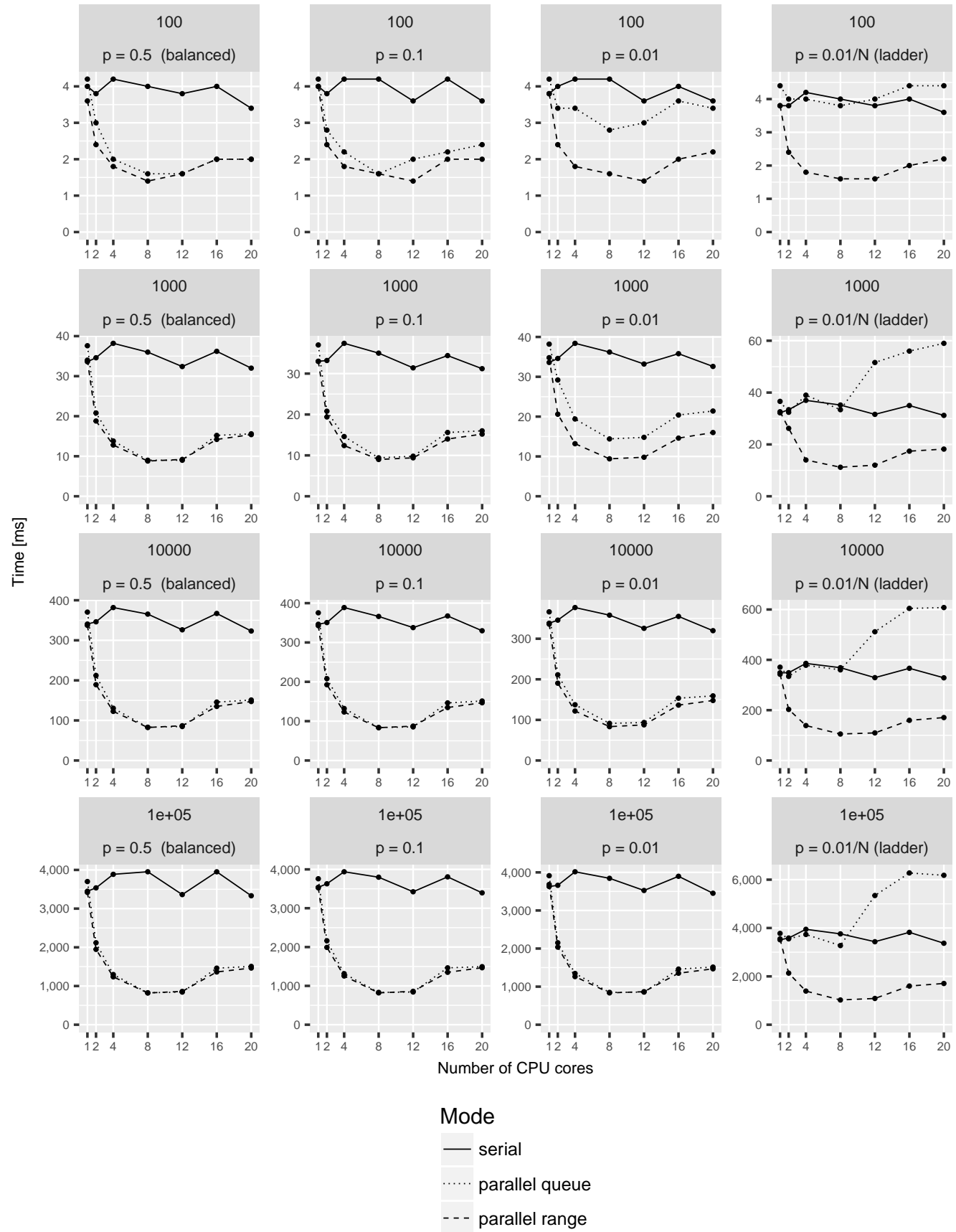


FIG. S4. Likelihood calculation times for the multi-trait POUMM implementation (PCMBaseCpp) with 4 traits on Euler cluster (a single shared memory node with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz running 24 physical cores). Both, the x -axis denoting the number of cores, and the y -axis denoting the calculation time in milliseconds are on the linear scale. Horizontally, the panels correspond to the different tree topologies, see also Fig. 2. Vertically, the panels correspond to the different tree-sizes.

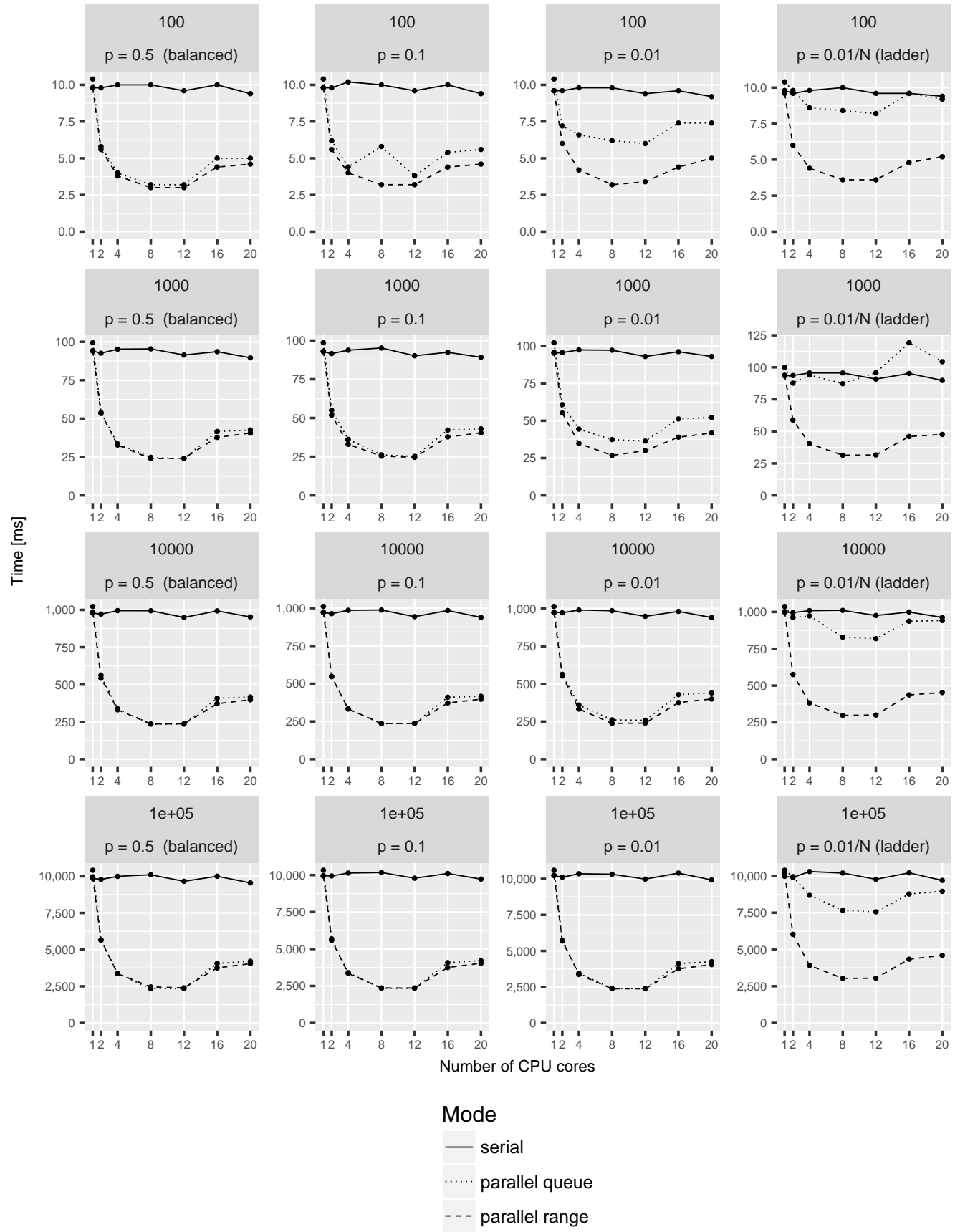


FIG. S5. Likelihood calculation times for the multi-trait POUMM implementation (PCMBaseCpp) with 8 traits on Euler cluster (a single shared memory node with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz running 24 physical cores). Both, the x -axis denoting the number of cores, and the y -axis denoting the calculation time in milliseconds are on the linear scale. Horizontally, the panels correspond to the different tree topologies, see also Fig. 2. Vertically, the panels correspond to the different tree-sizes.

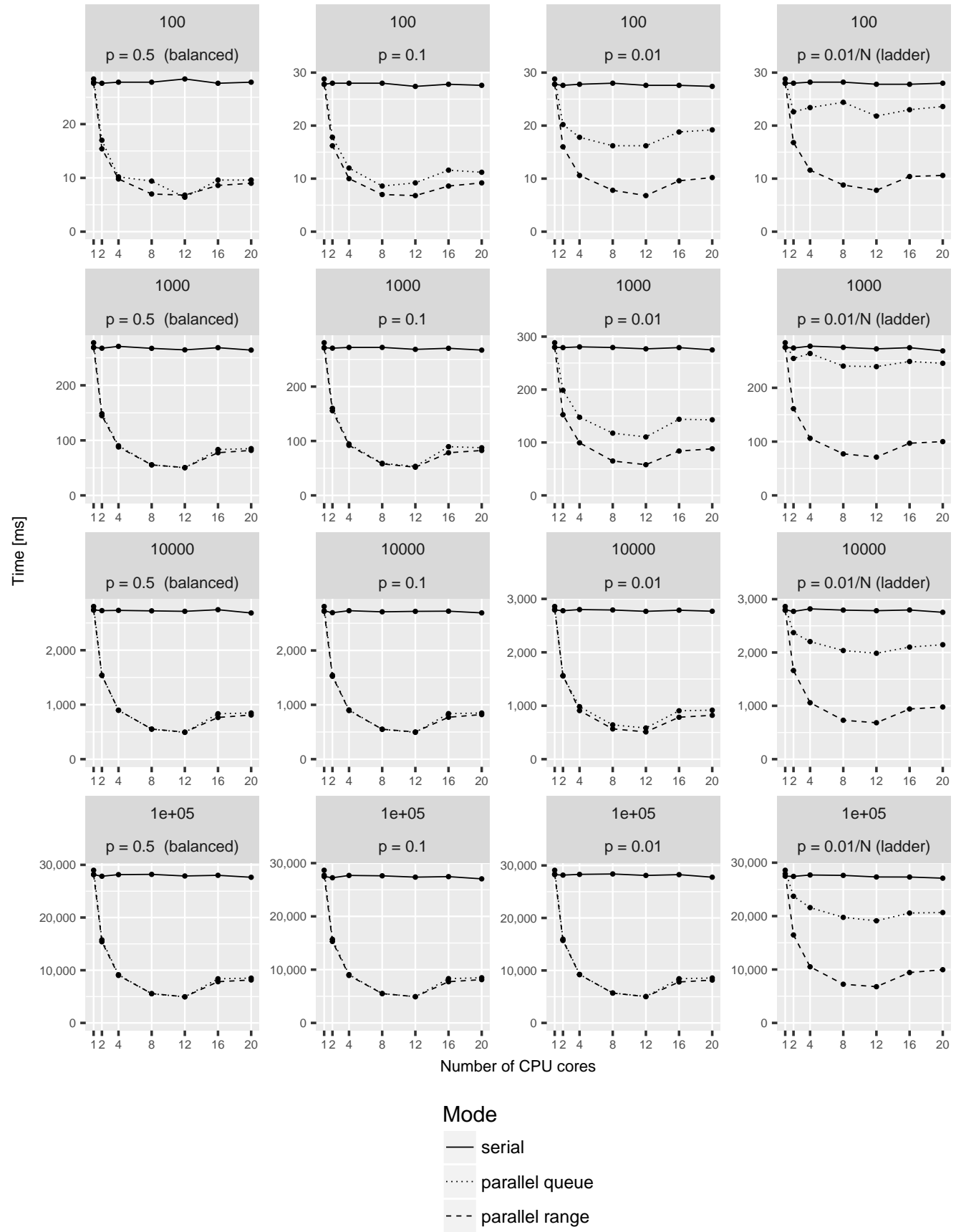


FIG. S6. Likelihood calculation times for the multi-trait POUMM implementation (PCMBaseCpp) with 16 traits on Euler cluster (a single shared memory node with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz running 24 physical cores). Both, the x -axis denoting the number of cores, and the y -axis denoting the calculation time in milliseconds are on the linear scale. Horizontally, the panels correspond to the different tree topologies, see also Fig. 2. Vertically, the panels correspond to the different tree-sizes.

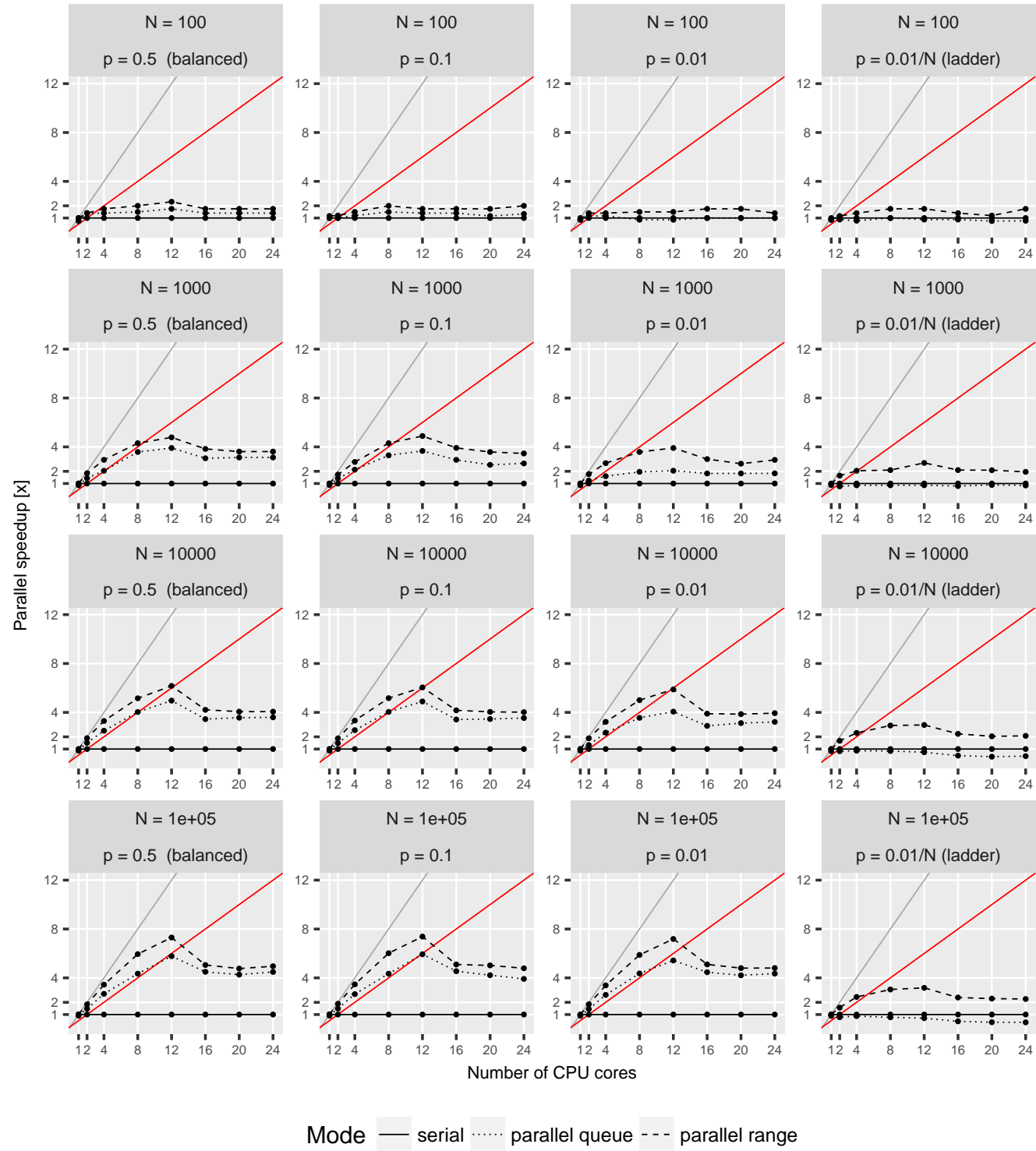


FIG. S7. Parallel speed-up for the multi-trait POUMM implementation (PCMBaseCpp) with 1 trait on Euler cluster. The grey and red lines denote, the expected speed-up at 100% and 50% parallel efficiency, respectively. Horizontally, the panels correspond to the different tree topologies. Vertically, the panels correspond to the different tree-sizes.

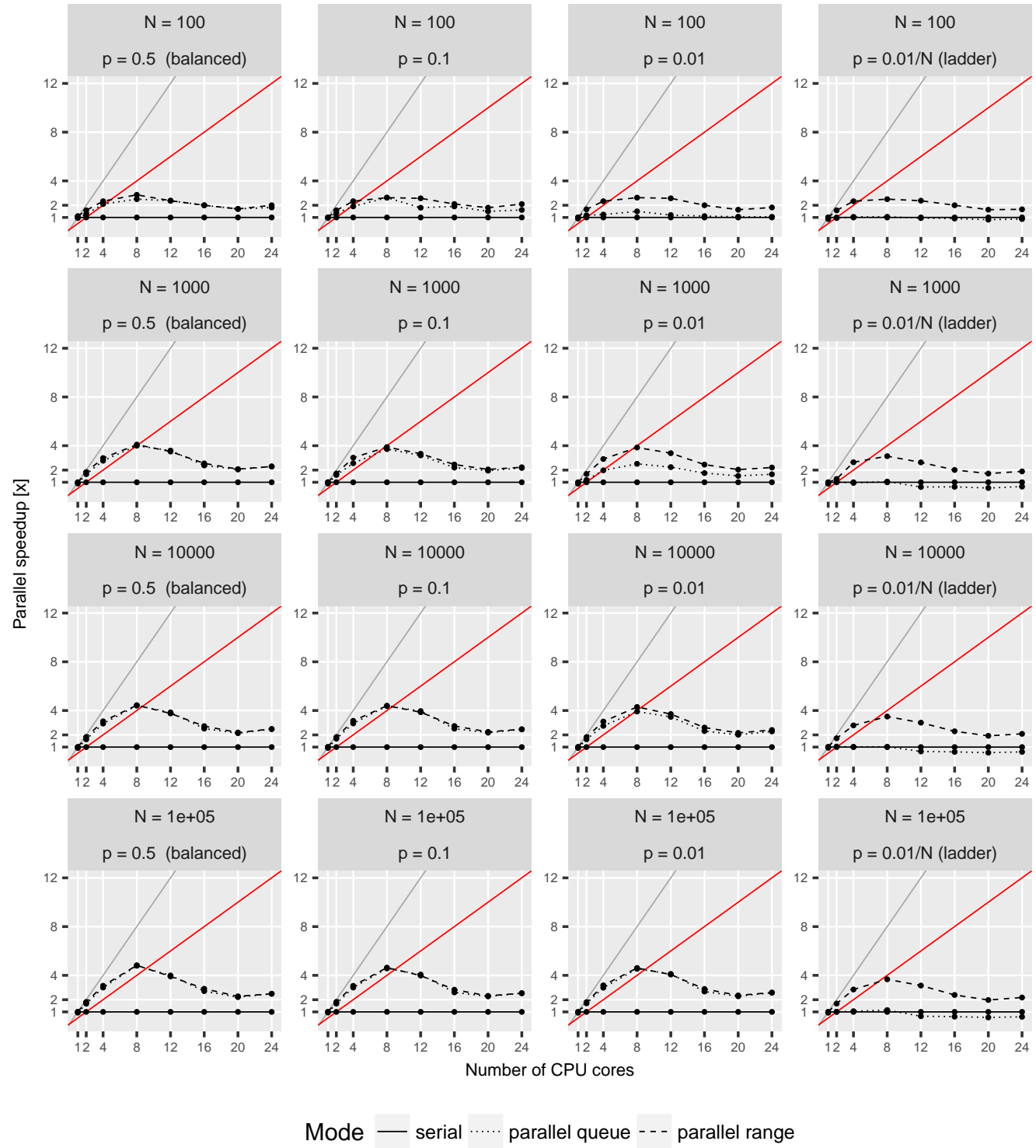


FIG. S8. Parallel speed-up for the multi-trait POUMM implementation (PCMBaseCpp) with 4 traits on Euler cluster. The grey and red lines denote, the expected speed-up at 100% and 50% parallel efficiency, respectively. Horizontally, the panels correspond to the different tree topologies. Vertically, the panels correspond to the different tree-sizes.

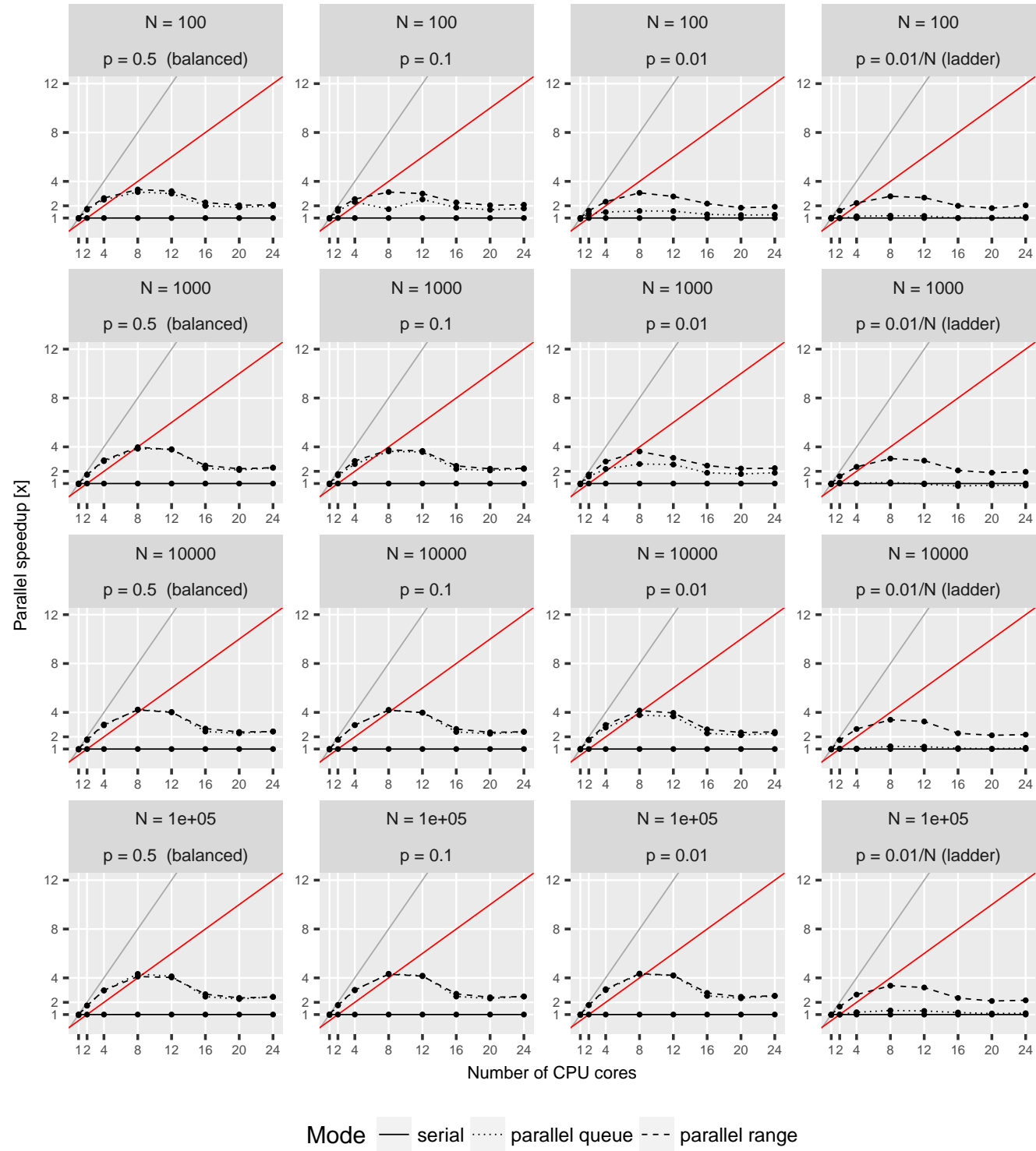


FIG. S9. Parallel speed-up for the multi-trait POUMM implementation (PCMBaseCpp) with 8 traits on Euler cluster. The grey and red lines denote, the expected speed-up at 100% and 50% parallel efficiency, respectively. Horizontally, the panels correspond to the different tree topologies. Vertically, the panels correspond to the different tree-sizes.

5 Combined speed-up from parallel likelihood calculation and adaptive Metropolis sampling

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, 2018), here, we briefly report the times and the quality statistics for the MCMC inference of the model with and without adaptive Metropolis sampling.

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. S10a,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 .

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. S10c,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.

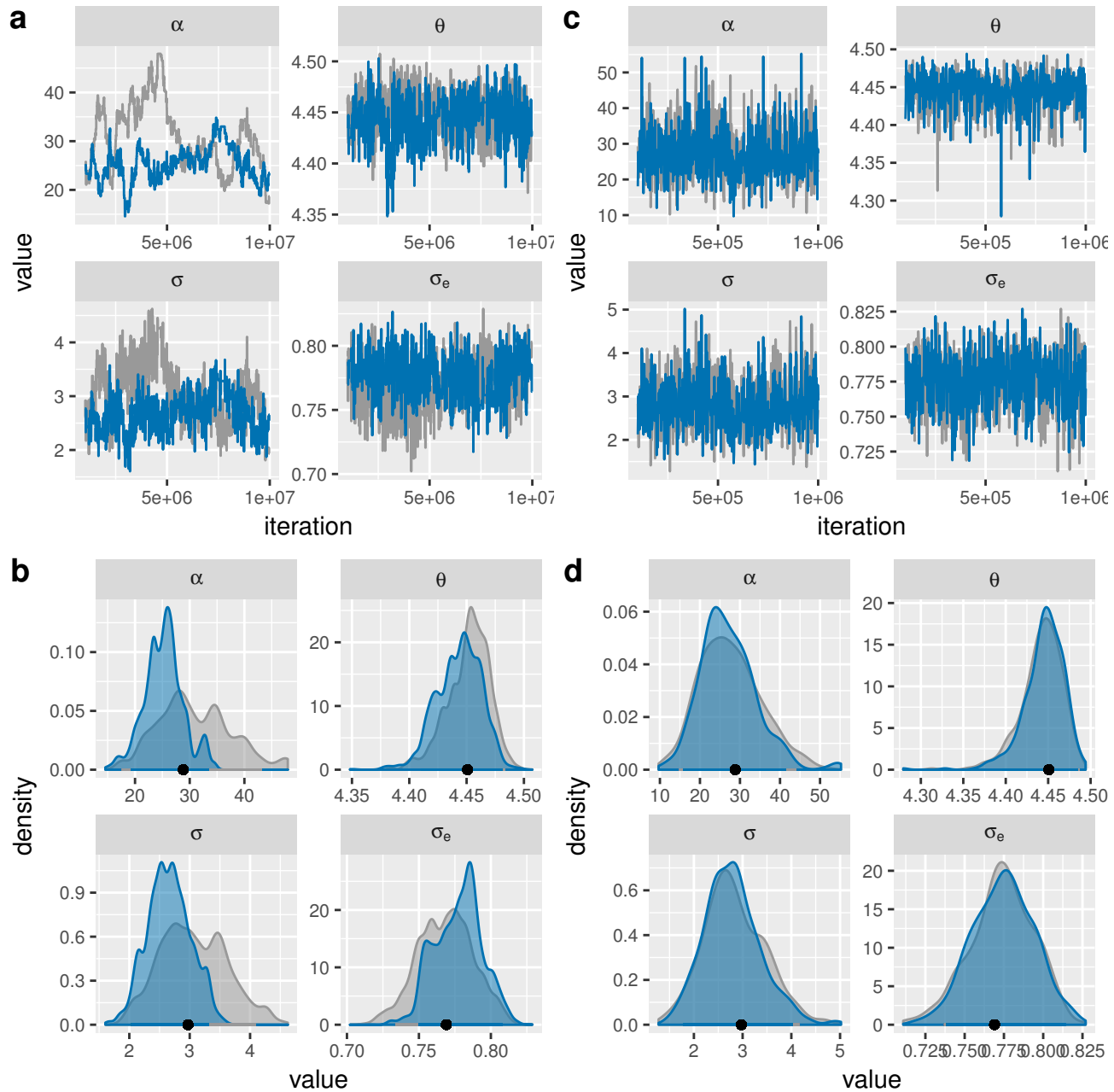


FIG. S10. Sample trace- and density plots from a POUMM fit to a tree and virulence data 8483 HIV patients (Mitov and Stadler, 2018) a,b: no adaptation of the proposal shape matrix (ten million iterations); c,d: on-the-fly adaptation of the proposal shape matrix from the first 100,000 out of one million iterations. The colors correspond to the different chains.

References

- Boskova, V., Bonhoeffer, S., and Stadler, T. 2014. Inference of Epidemiological Dynamics Based on Simulated Phylogenies Using Birth-Death and Coalescent Models. *PLoS Computational Biology (PLOS CB)* 10(4), 10(11): e1003913.
- Brooks, S. P. and Gelman, A. 1998. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4): 434–455.
- Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. Y. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5): 1190–1208.
- Cook, S. R., Gelman, A., and Rubin, D. B. 2006. Validation of Software for Bayesian Models Using Posterior Quantiles. *Journal of Computational and Graphical Statistics*, 15(3): 675–692.

- Eddelbuettel, D. and Sanderson, C. 2014. RcppArmadillo - Accelerating R with high-performance C++ linear algebra. *Computational Statistics & Data Analysis*, 71: 1054–1063.
- Felsenstein, J. 1983. Statistical Inference of Phylogenies. *Journal of the Royal Statistical Society. Series A (General)*, 146(3): 246.
- FitzJohn, R. G. 2012. Diversitree: comparative phylogenetic analyses of diversification in R. *Methods in Ecology and Evolution*, 3(6): 1084–1092.
- Grimmett, G. and Stirzaker, D. 2001. *Probability and Random Processes*. Oxford University Press.
- Haario, H., Saksman, E., and Tamminen, J. 2001. An adaptive metropolis algorithm. *Bernoulli. Official Journal of the Bernoulli Society for Mathematical Statistics and Probability*, 7(2): 223–242.
- Hansen, T. F. 1997. Stabilizing Selection and the Comparative Analysis of Adaptation. *Evolution*, 51(5): 1341–1351.
- Ho, L. s. T. and Ané, C. 2014. A linear-time algorithm for Gaussian and non-Gaussian trait evolution models. *Systematic Biology*, 63(3): 397–408.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. 1953. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6): 1087–1092.
- Mitov, V. and Stadler, T. 2018. A Practical Guide to Estimating the Heritability of Pathogen Traits. *Molecular biology and evolution*, 6(9): e1001123.–msx328 VL – IS –.
- Mitov, V., Bartoszek, K., Asimomitis, G., and Stadler, T. 2018. Fast likelihood evaluation for multivariate phylogenetic comparative methods: the PCMBase R package. *arXiv.org*, page arXiv:1809.09014.
- Ornstein, L. S. and Zernike, F. 1919. The theory of the Brownian Motion and statistical mechanics. *Proceedings of the Koninklijke Akademie Van Wetenschappen Te Amsterdam*, 21(1/5): 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(1342): 37–45.
- Plummer, M., Best, N., Cowles, K., and Vines, K. 2006. CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, 6: 7–11.
- Reif, J. H. 1989. Parallel Algorithms Derivation. Technical report, US Dept of the Navy, Funding, Fort Belvoir, VA.
- Scheidegger, A. 2017. adaptMCMC. *R package*.
- Stadler, T., Kühnert, D., Bonhoeffer, S., and Drummond, A. J. 2013. Birth-death skyline plot reveals temporal changes of epidemic spread in HIV and hepatitis C virus (HCV). *PNAS*, 110(1): 228–233.
- Uhlenbeck, G. E. and Ornstein, L. S. 1930. On the Theory of the Brownian Motion. *Physical Review*, 36(5): 823–841.
- Vihola, M. 2012. Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, 22(5): 997–1008.