Subject Section

Parallel power posterior analyses for fast computation of marginal likelihoods in phylogenetics

Sebastian Höhna1,2,*, Michael J. Landis1,3 and John P. Huelsenbeck1

1Department of Integrative Biology, University of California, Berkeley, CA, 94720, USA
2Department of Statistics, University of California, Berkeley, CA, 94720, USA
3Department of Ecology & Evolutionary Biology, Yale University, New Haven, CT, 06520, USA

*To whom correspondence should be addressed.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: In Bayesian phylogenetic inference, marginal likelihoods are estimated using either the path-sampling or stepping-stone-sampling algorithms. Both algorithms are computationally demanding because they require a series of power posterior Markov chain Monte Carlo (MCMC) simulations. Here we introduce a general parallelization strategy that distributes the power posterior MCMC simulations and the likelihood computations over available CPUs. Our parallelization strategy can easily be applied to any statistical model despite our primary focus on molecular substitution models in this study.

Results: Using two phylogenetic example datasets, we demonstrate that the runtime of the marginal likelihood estimation can be reduced significantly even if only two CPUs are available (an average performance increase of 1.96x). The performance increase is nearly linear with the number of available CPUs. We record a performance increase of 11.4x for cluster nodes with 16 CPUs, representing a substantial reduction to the runtime of marginal likelihood estimations. Hence, our parallelization strategy enables the estimation of marginal likelihoods to complete in a feasible amount of time which previously needed days, weeks or even months.

Availability: The methods described here are implemented in our open-source software RevBayes which is available from http://www.RevBayes.com.

Contact: Sebastian.Hoehna@gmail.com

Supplementary information: Supplementary data are available at Bioinformatics online.

1 Introduction

Model selection in Bayesian phylogenetic inference is performed by computing Bayes factors, which are ratios of the marginal likelihoods for two alternative models (Kass and Raftery, 1995; Sullivan and Joyce, 2005). The Bayes factor indicates support for a model when the ratio of the marginal likelihoods is greater than one. This procedure is very similar to likelihood ratio tests with the difference being that one averages the likelihood over all possible parameter values weighted by the prior probability rather than maximizing the likelihood with respect to the parameters (Posada and Crandall, 2001; Holder and Lewis, 2003). More specifically, the marginal likelihood of a model, \(f(D|M)\), is calculated as the product of the likelihood, \(f(D|\theta, M)\), and the prior, \(f(\theta|M)\), integrated (or marginalized) over all possible parameter combinations,

\[
f(D|M) = \int f(D|\theta, M)f(\theta|M)d\theta .\] (1)

In the context of Bayesian phylogenetic inference, this quantity is computed by marginalizing over the entire parameter space, namely over all possible tree topologies, branch lengths, substitution model parameters and other model parameters (Huelsenbeck et al., 2001; Suchard et al., 2001). The computation of the marginal likelihood is intrinsically difficult because the dimension-rich integral is impossible to compute analytically. Monte Carlo sampling methods have been proposed to circumvent the
analytical computation of the marginal likelihood (Gelman and Meng, 1998; Neal, 2000). Lartillot and Philippe (2006) introduced a technique called thermodynamic integration, (also called path-sampling; Baele et al., 2012), to approximate the marginal likelihood. A similar method, stepping-stone-sampling (Xie et al., 2011; Fan et al., 2011), has more recently been proposed (see also Baele et al., 2012; Baele and Lemey, 2013; Friel et al., 2014, for a summary and comparison of these methods). The fundamental idea of path-sampling and stepping-stone-sampling is to use a set of \( K \) importance distributions, or power posterior distributions, from which likelihood samples are taken (Gelman and Meng, 1998; Neal, 2000; Lartillot and Philippe, 2006; Friel and Pettitt, 2008). The sampling procedure for each importance distribution is performed by a Markov chain Monte Carlo (MCMC) algorithm. That is, instead of running a single MCMC simulation as commonly done to estimate posterior probabilities (Huelsenbeck et al., 2001), selecting between complex diversification rate models (FitzJohn, 2012), and evaluating competing trait processes (e.g., Uyeda and Harmon, 2014).

In the present article we demonstrate how power posterior simulations can be performed on parallel computer architectures and report the achieved computational gain. The idea of parallel power posterior simulations is similar to parallel Metropolis coupled MCMC algorithm (Altekar et al., 2004), with the important difference that power posterior simulations can be parallelized even more easily because no communication between processes is necessary. Additionally we show how our parallelization scheme can combined with existing parallelization techniques for distributed likelihood computation (e.g., Aberer et al., 2014) to maximize usage of available CPUs.

## 2 Methods

The algorithm underlying path-sampling and stepping-stone-sampling can be separated into two steps: (1) likelihood samples are obtained from a set of \( K \) power posterior simulations; and (2) the marginal likelihood is approximated either by numerical integration of the likelihood samples over the powers (path-sampling) or by the likelihood ratio between powers (stepping-stone-sampling). The first step is the same for both methods and is the computationally expensive part. Thus, once samples from the power posterior distributions are obtained, it is possible to rapidly compute both the path-sampling and stepping-stone-sampling marginal likelihood estimates.

### 2.1 Power posterior sampling

Both, stepping-stone-sampling and path-sampling, construct and sample from a series of importance distributions. Lartillot and Philippe (2006) define the importance distributions as power posterior distributions, which are obtained by modifying the posterior probability density as

\[
 f_p(\theta) = f(Y|\theta, M)^{\beta_p} f(\theta|M) .
\]

Here, \( \beta_p \) represent a vector of powers between 0 and 1. Then, for every value of \( \beta_p \), a draw from the power posterior distribution is needed and its likelihood score, \( l_p \), is recorded (Lartillot and Philippe, 2006; Friel and Pettitt, 2008). In principle, one such likelihood sample per power posterior distribution is sufficient, although multiple samples improve the accuracy of the estimated marginal likelihood considerably (Baele et al., 2012). We will use the notation \( l_{ij} \) to represent the \( j^{th} \) likelihood sample from the \( i^{th} \) power posterior distribution.

We illustrate the mean log-likelihood over different values of \( \beta \) in Figure 1. Commonly, the values of the powers \( \beta \) are set to the \( \beta \) quantile of a beta(0.3, 1.0) distribution (Xie et al., 2011; Baele et al., 2012). The rationale is that more narrowly spaced intervals are needed for the range of \( \beta \) where the expected likelihood changes most rapidly, i.e., for \( \beta \) values close to 0 (Figure 1).

Draws from the power posterior distribution are obtained by running a modified Markov chain Monte Carlo (MCMC) algorithm (Altekar et al., 1998; Neal, 2000). Lartillot and Philippe (2006) introduced a technique to maximize usage of available CPUs.

As can be seen from this brief description of the modified MCMC algorithm, only the likelihood values need to be raised to the power \( \beta \). All remaining aspects of the MCMC algorithm stay the same as the standard implementations in Bayesian phylogenetics (Huelsenbeck and Ronquist, 2001; Drummond and Rambaut, 2007; Lakner et al., 2008; Lartillot et al., 2009; Höhna and Drummond, 2012).

It is important to note that every MCMC simulation for each power \( \beta_j \in \beta \) necessarily includes its own burn-in period before the first sample can be taken. The power posterior analysis can be ordered to start from the full posterior (\( K = 1 \)) and then to use monotonically decreasing powers until the prior (\( \beta_0 = 0.0 \)) has been reached. Thus, the last sample...
Parallel power posterior analyses

of the previous power posterior run can be used as the new starting state. This strategy has been shown to be more efficient because it is easier to disperse from the (concentrated) posterior distribution to the (vague) prior distribution thereby reducing the burn-in period significantly (Baele et al., 2012).

2.2 Parallel power posterior analyses

The sequential algorithm of a power posterior analysis starts with a pre-burnin phase to converge to the posterior distribution. Then, consecutive power posterior simulations are performed sequentially, starting with $\beta_{K-1} = 1.0$ (i.e., the posterior) to $\beta_0 = 0.0$ (i.e., the prior). Each power posterior simulation contains $L$ iterations, with the likelihood of the current state recorded every $T$th iteration. These ‘thinned’ samples are less correlated than the original draws from the MCMC simulation. The number of samples taken per power is $n = L/T$. At the beginning of each run a short burn-in phase is conducted, for example 10% or 25% of the run length.

The parallel algorithm for a power posterior analysis is set up almost identically to the sequential algorithm (see Figure 2). Let us assume we have $M$ CPUs available. Then, we split the set of powers into $M$ consecutive blocks; the $m^{th}$ block contains the powers from $\beta_{K-1} \cdot \frac{\prod l_{i+1} \cdot \prod k_{i+1}}{\prod l_i \cdot \prod k_i}$ to $\beta_{(m-1)K/M}$, e.g., the first out of four blocks for 128 analyses contains $\{\beta_{127}, \ldots, \beta_4\}$, the second block contains $\{\beta_9, \ldots, \beta_4\}$ etc. If the set of $\beta$ cannot be split evenly into blocks then some blocks have one additional simulation, which is enforced by using only the integer part of the index. This block-strategy ensures that each CPU works on a set of consecutive powers which has the advantage of a shorter burn-in between simulations because the importance distributions are more similar to one another.

Regardless, each parallel sampler needs to start with an independent pre-burnin phase which creates an additional overhead. Thus, instead of running only one pre-burnin phase, as under the sequential power posterior analysis, we need to run $M$ pre-burnin phases. This overhead could be drawn only if it would be possible to draw initial values directly from the power posterior distribution. Figure 2 shows a schematic of our parallelization algorithm. After the initial pre-burnin phase, the workload is divided into blocks and equally distributed over the available CPUs. Note that CPUs can be combined for distributed likelihood computation. No synchronization or communication between samplers is necessary because each power posterior simulation is independent. The only parallelization barrier occurs at the end when all power posterior simulations have finished. Finally, the master CPU collects all likelihood samples, combines the results, and computes the marginal likelihood using one of equations given below. These equations are computationally cheap compared with obtaining the likelihood samples. We thus expect that the performance gain is close to linear with the number of available cores. The algorithm described here is implemented in the open-source software RevBayes (Höhna et al., 2014; Höhna et al., 2016), available at http://www.RevBayes.com.

2.3 Path-Sampling

Path-sampling was the first numerical approximation method developed for marginal likelihood computation in Bayesian phylogenetic inference (Lartillot and Philippe, 2006). Path-sampling uses the trapezoidal rule to compute the integral of the log-likelihood samples between the prior and the posterior (see Figure 1), which equals the marginal likelihood (Lartillot and Philippe, 2006). The equation of the trapezoidal rule for a single likelihood sample from each power posterior simulation is

$$\ln f(D|M) = \sum_{k=0}^{K-1} \frac{(\ln l_k + \ln l_{k+1})}{2} \cdot (\beta_{k+1} - \beta_k)$$

Samples of the log-likelihood have a large variance. Hence, it is more robust to take many log-likelihood samples and use the mean instead. This yields the equation to estimate the marginal log-likelihood,

$$\ln f(D|M) = \frac{\sum_{k=0}^{K-1} \frac{n}{n} \ln l_k}{n} + \frac{\sum_{k=0}^{K-1} \frac{n}{n} \ln l_{k+1}}{n} \cdot (\beta_{k+1} - \beta_k)$$

which was proposed by Baele et al. (2012).

2.4 Stepping-Stone-Sampling

Stepping-stone-sampling approximates the marginal likelihood by computing the ratio between the likelihood sampled from the posterior and the likelihood sampled from the prior. However, this ratio is unstable to compute and thus a series of intermediate ratios is computed: the stepping-stones (Xie et al., 2011; Fan et al., 2011). The stepping-stones can be chosen to be exactly the same powers as those used for path-sampling. The equation to approximate the marginal likelihood using stepping stone sampling is

$$f(D|M) = \prod_{k=0}^{K-1} \left( \frac{1}{n} \sum_{i=1}^{n} \frac{\beta_{k+1}}{\beta_k} \right) \cdot \left( \frac{1}{n} \sum_{i=1}^{n} \beta_{k+1} \right)$$

Numerical stability of the computed marginal likelihood can be improved by retrieving first the highest log-likelihood sample, denoted
The performance increase levels off quickly once 8 or 16 CPUs are used. This is simply due to the fact that twice as many CPUs are needed each time to roughly halve the computational time. Hence, the gain from 1 to 4 CPUs is approximately equivalent to the gain from 16 to 64 CPUs. Additionally, the overhead (i.e., the independently run pre-burnin for each chain) which each CPU needs to perform reduces the performance gain for larger number of CPUs.

We computed the expected runtime to assess whether our implementation achieved the largest possible performance gain. For example, we wanted to explore if there is an additional overhead for using parallelization that was possibly introduced by our specific implementation. Having $M$ CPUs available, each CPU needs to run at most $[K/M]$ power posterior simulations, which is the ratio of the total number of power posterior simulations to CPUs rounded upwards (ceiling).

Each analysis consisted of a set of $K = 128$ power posterior simulations (see Figure 2 for a schematic overview). The analyses started with a pre-burnin period of 10,000 iterations to converge to the posterior distribution. Then, each power posterior analysis was run for 10,000 iterations and samples of the likelihood were taken every 10 iterations. The 25% initial samples of each power posterior distribution were discarded as additional burnin. The marginal likelihood was estimated using both path-sampling and stepping-stone-sampling once all power posterior simulations had finished as they contribute to performance overhead in practice. We ran each analysis 10 times and measured the computation time on the San Diego Supercomputer (SDSC) Gordon. The experiment was executed using 1, 2, 4, 8, 16, 32 and 64 CPUs, respectively.
Additionally, each CPU runs its own pre-burnin phase, which had the same length as a single power posterior simulation in our tests. Therefore, we can compute the average runtime of a single power posterior simulation by dividing the runtime of the single CPU analysis by \(K + 1\). Then, the expected runtime for \(M\) CPUs, \(t_{\text{exp}}\), is given by

\[
E[t_{\text{exp}}] = t_1 \times \frac{[K/M] + 1}{K + 1}
\]

where \(t_1\) corresponds to the runtime when only one CPU was available.

In general, our implementation seems to perform close to the expected optimal performance (Figure 3). However, we observe an increasing discrepancy between the expected and the observed performance gain when many CPUs were used. This discrepancy is most likely due to bottlenecks in competing hardware allocations. For example, we noticed that I/O operations performed on a network filesystem, which are commonly used among large computer clusters, significantly influenced the performance, especially when many CPUs frequently wrote samples of the parameters to a file.

We performed an additional performance analysis where we omitted the pre-burnin phase. This scenario could be realistic when one has already performed a full posterior probability estimation and only wants to compute the marginal likelihoods for model selection. In this case, the samples from the posterior distribution can be used to specify starting values of the power posterior analysis. Here we see that the performance improvement becomes more linear with the number of CPUs (see Figure 4).

Although this case might not happen frequently in practice, we use this to demonstrate that only the pre-burnin phase prevents us from having an almost linear, and thus optimal, performance increase.

We also investigated whether the performance overhead (observed in Figure 3) is correlated with the number of stepping stones per CPU. For example, we observed the largest difference between the expected and actual runtime when 64 CPUs were used (each CPU ran only one or two power posterior simulations plus the pre-burnin phase). Thus, we tested if there was an effect of small numbers of power posterior simulations by running analysis with \(K \in \{2, 3, 5, 10, 20, 30, 40, 50\}\) on a single CPU. As the expected runtime, we computed the mean runtime per individual power posterior simulation when \(K = 50\). Our results, shown in Figure 5, demonstrate that there is an intrinsic overhead for small number of power posterior simulations. This overhead seemed to be the cause of the discrepancy between our expected and observed performance increase in the parallel power posterior algorithm (Figure 3). Part of the overhead is caused by the additional time to start the process, load the data, allocate memory, receive file handles and all other tasks that need to be performed before and after a power posterior analysis.

Finally, we compared the performance increase when parallelizing the power posterior analysis, the likelihood computation, or both. For this combined parallelization scheme we implemented a hierarchical parallelization structure as describe by Aberer et al. (2014). For example, when 4 CPUs are available we can divide the likelihood computation over 2 CPUs and divide the power posterior analysis into 2 blocks (see Figure 1). This text thus includes the parallelization approach over the likelihood function as suggested by Baele and Lemey (2013). We tested the performance difference using \(M = \{2, 4, 8, 16, 32, 64\}\) CPUs of which we assigned \(N\) to share the likelihood computation. We observed the best overall runtime reduction when we applied a combined likelihood and power posterior analysis parallelization (Table 1 and Table 2). Furthermore, the improvement of each parallelization yields diminishing returns when more CPUs are used, which additionally supports the utility of a combined parallelization scheme. We conclude that using \(N = \lceil \sqrt{\frac{M}{2}} \rceil\) will give the overall best performance and set this distribution of CPUs as the default option in RevBayes.
4 Conclusion

Modern phylogenetic analyses depend on increasingly complex models and increasingly large data set sizes. Even phylogenetic analyses which do not use molecular sequence data (for example, diversification rate analyses (FitzJohn, 2012), continuous trait analyses (Uyeda and Harmon, 2014), and historical biogeography analyses (Landis et al., 2013)) have grown more complex and use time-intensive likelihood calculations that are not always easily parallelizable. Both trends lead to longer runtimes, which is even more pronounced for Bayesian model selection exercises using marginal likelihoods; the path-sampling and stepping-stone-sampling algorithms used for approximating marginal likelihoods are inherently computationally demanding. In the present paper we have developed a simple parallel algorithm to speed up the computation of marginal likelihoods for Bayesian phylogenetic inference. In our simulation study, which serves mostly as a proof of concept, we showed that performance improvement is close to linear for few CPUs, i.e., between one and 16 CPUs. An analysis that previously took 8 weeks on a single CPU can now be completed in four days when 16 CPUs are available.

Our new parallel power posterior analysis can be more than an order of magnitude faster than ordinary, sequential algorithms. The presented parallel algorithm should be straightforward to be implemented in other software or applied to a variety of different model types. Finally, the described parallelization scheme should be applicable to alternative methods for computing marginal likelihood (e.g., Fan et al., 2011) and Bayes factors (Lartillot and Philippe, 2006; Baele et al., 2013) because all these approaches rely on a set of power posterior analyses.

Acknowledgements

We would like to thank David Posada and three anonymous reviewers for valuable comments on a previous version of this manuscript.

Funding

SH was supported by the Miller Institute for Basic Research in Science. MFL and JPH were supported by XXX. JPH was supported by XXX. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

References


Parallel power posterior analyses


