Statistics and Computing manuscript No.
(will be inserted by the editor)

New algorithms for parallelising Markov chain Monte Carlo

With applications to bioinformatics

Hákon Jónsson · John Lees · Tobias Madsen · Joe Herman · Jotun Hein · Geoff Nichols

Received: date / Accepted: date

Abstract This paper considers how to most efficiently reduce the run time of Markov Chain Monte Carlo simulations by using speculative computation over multiple processors. Current algorithms are reviewed, and a proof for an the optimal use of parallel processors is presented. Results for optimal jump size in Roberts and Rosenthal (2001) are extended to the case of parallel MCMC using speculative computation to develop an algorithm which gives the maximum possible speedup for an arbitrary target distribution.

A speedup of 3.13 is achieved on a four core shared memory system, and a speedup of 25.3 times is achieved on 64 core cluster. The algorithm is simple to implement and can provide speedups to any MCMC process with a likelihood evaluation time longer than $\approx 100\mu$s. An application to estimating mutation rate is considered, where a speedup of 2.70 is achieved using four cores.

Keywords Markov Chain Monte Carlo · Metropolis-Hastings · MPI · Optimal Scaling · Parallel MCMC · Parallel Models · Mutation Rate · Speculative Computation

Funding was provided by the Wellcome Trust (JL) and COGANGS – EU Grant (HJ, TM)

H. Jónsson
University of Copenhagen
E-mail: jonsson.hakon@gmail.com

J. Lees
Department of Physics, University of Oxford
E-mail: john.lees@physics.ox.ac.uk

T. Madsen
Aarhus University
E-mail: tobias.madsen@gmail.com

J. Herman
Department of Statistics, University of Oxford, 1 South Parks Road, OX1 3TG, United Kingdom
E-mail: herman@stats.ox.ac.uk

J. Hein
Department of Statistics, University of Oxford
E-mail: hein@stats.ox.ac.uk

G. Nicholls
Department of Statistics, University of Oxford
E-mail: nicholls@stats.ox.ac.uk

1 Introduction

Markov chain Monte Carlo (MCMC) is an important tool for fitting complex stochastic models to data, and remains the generic method of last resort for statistical inference from complex models. Application of the Metropolis-Hastings algorithm (Metropolis et al., 1953; Hastings, 1970) with some conditions on the proposal distribution will guarantee convergence of the simulated chain $\Theta_0, \Theta_1, \Theta_2, \ldots$ to the stationary distribution $\pi(\theta)$.

The method is however computationally intensive, and in cases where likelihood evaluations take a significant fraction of a second and many (potentially millions) of iterations are required to provide a representative sample this can cause severe problems. One approach to reducing computation time for long MCMC runs is the use of parallel processors.

The Metropolis-Hastings algorithm is sometimes considered to be 'embarrassingly parallel' as multiple chains can be run on different processors simultaneously, with no need for communication between threads (Rosenthal, 1999). However it is often the case that it is desirable to achieve large depth on a serial chain, as there is a long burn-in period before the chain converges to the target distribution. MCMC in this sense appears to be inherently serial, and new algorithms must be developed to achieve a speedup by using parallel computation.

Speculative computation was first applied to MCMC in the context of simulated annealing by Witte et al. (1991).

Here the concept of computing log-likelihoods in parallel before they are needed is introduced. The process at each step is visualised as a binary tree, where each available processor can be assigned to calculate the next log-likelihood assuming the previous state was either rejected or accepted. Any unused calculations due to the speculation being incorrect are 'thrown away'.

As the temperature is increased, the acceptance probability increases, and so the shape of the tree used for the speculative computation is dynamically altered to achieve the maximum expected depth at each step. This leads to unbalanced trees where all processors assume the previous update will be rejected at low temperatures, through balanced trees where there is an equal likelihood for the proposal to be rejected, to again unbalanced trees in the opposite direction at high temperatures.

The application of speculative computation to sampling distributions using MCMC has been considered by Byrd et al. (2008) and Brockwell (2006). These authors do not consider different tree shapes, and only consider the cases where all processors assume rejection and that of a balanced tree respectively. This will not in general give the optimal speedup.

The basic algorithm on K processors considered by Byrd et al. (2008) is as follows:

If $\Theta_t = \theta$ is the previously simulated state, the values of $\Theta_{t+1}, \ldots, \Theta_{t+k}$ with $k \in \{1, 2, \ldots, K\}$ random are determined as follows: The computation of all $\alpha_1 \cdots \alpha_K$ is done in parallel on the K processors. The first step of simulating from the proposal distribution may also be done in parallel.

Algorithm 1 Parallel MCMC using an unbalanced tree shape

for i from 1 to K **do**
  simulate iid $\theta_i' \sim q(\theta, \cdot)$;
end for
for i from 1 to K **do** ▷ in parallel
  calculate probabilities $\alpha_i \leftarrow \min\left\{1, \frac{\pi(\theta_i'|y)q(\theta_i|\theta_i')}{\pi(\theta_i|y)q(\theta_i'|\theta_i)}\right\}$;
  with probability α_i set $A_i = 1$, otherwise set $A_i = 0$;
end for
for i from 1 to K **do**
  if $A_i = 0$ **then**
    $\Theta_{t+i} \leftarrow \theta_t$;
  else
    $\Theta_{t+i} \leftarrow \theta_i'$;
    exit for;
  end if
end for

When working with a problem where the acceptance probability is roughly fixed, or difficult to tune (for example making rearrangements of a tree structure), this algorithm will certainly not always be optimal. Consider for example a case where 90% of updates are accepted, then the expected depth on such a tree will only be 1, whereas a balanced tree will always achieve $\log_2(K+1)$ updates per iteration. This has been addressed in the context of simulated annealing by Witte et al. (1991), but has not been previously considered in the context of sampling.

However, it is often the case that the acceptance probability can be changed. In this case a feature of algorithm 1 not previously considered is that it may become more efficient if the acceptance probability is lowered. This can be achieved if it is possible to propose more widely distributed candidate steps θ' (choosing $q(\theta, \theta')$ to have more heavily weighted tails at large values of $|\theta' - \theta|$).

Changing the acceptance probability also affects the mixing efficiency of the chain. For an infinite dimensional Gaussian target distribution Roberts, Gelman, et al. (1997) show that the fastest convergence is achieved by choosing an acceptance probability of 0.234. An analytic function for efficiency is also given, showing that anything between 0.15 and 0.4 will still give 80% of the maximum possible efficiency in this case. This effect must also be considered when choosing the acceptance probability if the aim is to minimise convergence time.

This paper extends the idea of using different tree shapes introduced by Witte et al. (1991) to the problem of sampling using MCMC by maximising the expected tree depth and mixing efficiency of the serial Metropolis-Hastings algorithm (Roberts and Rosenthal, 2001) together to give the maximum speedup.

With the ready availability of multi-core processors with fast inter-processor communication at relatively high bandwidth, this algorithm allows a speedup of up to around 3 times essentially for free on a standard 4-core laptop. Even larger speedups can be achieved with supercomputing resources. This has useful applications in bioinformatics, where log-likelihood evaluations take a significant fraction of a second and burn-in time is relatively long. Such an example is considered in Sect. 6.

The remainder of this article is arranged as follows. In Sect. 2 an extension to the basic algorithm (1) where the path through the algorithm is represented as a binary tree is introduced, and this tree is shaped to give the maximum expected depth (and hence movement through the chain) at each iteration given an acceptance probability. Sect. 3 exhibits a simple model for the processing times involved the the algorithm, allowing analysis of when the algorithm will be effective. In Sect. 4, the concept of effective sample size is introduced, and how the efficiency of the chain depends on acceptance probability in an idealised situation is analysed. This result is then extended to the case of the parallel algorithm by considering what the optimal acceptance probability is for K processors, bringing together the results of Roberts and Rosenthal (2001) and the optimal tree shape of Sect. 2. An implementation of the algorithm and the real-time speedups measured are presented in Sect. 5. An appli-

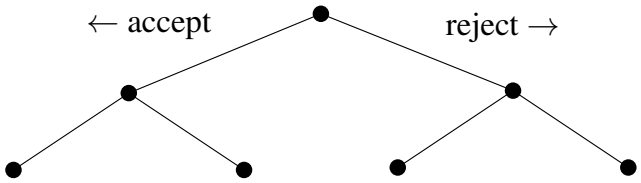

Fig. 1 Balanced accept-reject tree as considered by Brockwell (2006), giving speedup $\log_2(K+1)$

cation to estimating effective mutation rate from genomic data is shown in Sect. 6, showing measured speedups in a real use of MCMC. Sect. 7 concludes with a brief discussion of the results.

2 Optimal tree structure

Instead of using one specific tree shape in all cases (Brockwell, 2006; Byrd et al., 2008) it is possible to construct a tree, given an average acceptance probability, that maximises the speedup. In this section, by maximising the expected depth reached into the tree at each iteration, an algorithm for the optimal tree shape is developed. The optimal acceptance probability is then found in Sect. 4.

In appendix A this result is presented in a more general setting. Here it is assumed that there is a fixed accept probability and the time per iteration is constant. Although in actuality seeing a number of rejects in a row will decrease the probability of seeing an accept as the current state is likely to have a higher likelihood than the proposals given this state, it is a reasonable approximation to think of acceptance rate as constant over a long chain.

If I denotes the node at which the accept/reject path leaves the tree and d_i is the depth of node i (with the root node defined as being at a depth of one), our objective is then to maximize

$$\mathbb{E}_{T_K}[d_I] \tag{1}$$

over different treeshapes T_K with K nodes, where K is the number of available processors. This is efficiently achieved using a greedy choice algorithm.

Let P denote the stochastic accept/reject path. Then

$$\begin{aligned}\mathbb{E}_{T_K}[d_I] &= \mathbb{E}[\sum_{i\in T_K} \mathbb{1}(i \in P)] \\ &= \sum_{i\in T_K} \mathbb{P}(i \in P)\end{aligned}$$

This suggests that K nodes should be chosen greedily after their probability of being visited $\mathbb{P}(i \in P)$. Since the probability of visiting a parent is always greater than visiting one of its children, this guarantees a valid binary tree is formed, which is grown from the root as in algorithm 2.

Algorithm 2 Optimal tree shape by greedy choice

```
Create priority queue Q with priorities given by equation (13);
Push root into Q;
for i from 1 to K do
    Pop v from Q;
    Add v to T_K;
    Push children of v onto Q;
end for
return T_K
```

Given this tree shape, algorithm 1 must be extended to allow for speculative computation with an arbitrary processor tree shape. This is done for each iteration using a master/slave architecture in algorithm 2.

Algorithm 3 Parallel MCMC using an arbitrary tree shape

```
function GenerateProposals(node i, θ_i)
    if right child exists then                          ▷ reject node
        GenerateProposals(right child node, θ_i);
    end if
    simulate iid θ'_i ~ q(θ_i, ·);
    if left child exists then                           ▷ accept node
        GenerateProposals(left child node, θ'_i);
    end if
end function

GenerateProposals(root node, θ_0);
for i from 1 to K do
    send state θ'_i to processor i;
end for
calculate log-likelihood of received state;             ▷ in parallel
for i from 1 to K do
    receive log-likelihood from processor i;
end for
loop
    calculate Hastings ratio α_i with θ_i and θ'_i;
    with probability α_i accept update;        ▷ add θ_i or θ'_i to chain
    if accepted and a left child exists then
        i ← i_leftchild;
    else if right child exists then
        i ← i_rightchild;
    else
        exit loop;
    end if
end loop
```

Child nodes could also be spawned off recursively, allowing distribution of proposals. However this leads to longer communication times, potentially reducing the maximum speedup achievable.

3 Computation and communication times

There are four computation times (all in CPU seconds) of interest to the problem:

– t_p is the time taken to generate a proposal $\theta' \sim q(\theta, \cdot)$

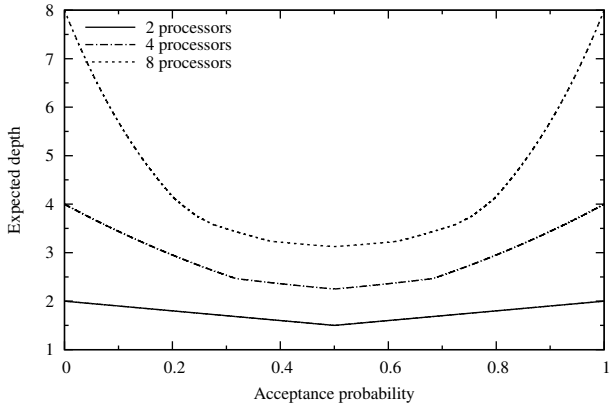

Fig. 2 Expected tree depth as a function of accept probability, for various different numbers of processors in the tree. It is maximal for the ladder trees chosen for low or high acceptance probabilities, and reaches a minimum bounded by $\mathcal{O}(\log_2(K+1))$ at an accept rate of 0.5

- t_e is the time taken to evaluate the log-likelihood $\log(L(\theta';y))$ of the proposal state θ' in the distribution π given data y.
- t_c is the time taken to communicate between separate processors - the time between calling MPI::Send and MPI::Recv unblocking.
- t_d is the time taken to calculate the Hastings ratio, and decide whether the update is accepted or not.

similar to those identified by Witte et al. (1991).

There will also be some fixed time to start the program, but this will be short compared to overall program run-time. A comparison between the parallel algorithm and the serial algorithm including initialisation time will be made in Sect. 5.

The proposal time t_p may in many cases be extremely short, for example symmetric random walks (which are commonly chosen for MCMC as they simplify the Hastings ratio) will have:

$$t_p \ll t_c \tag{2}$$

A random walk with Gaussian dispersion is considered as the proposal distribution in the example used in Sect. 6. This does obey equation 2.

More complex proposals such as rearranging tree structures can have longer t_p so that equation 2 no longer holds. In these cases the computation of proposals may be distributed among the parallel processors which may yield a greater speedup by reducing the sum of all the proposal times t_p at each iteration, though at an expense of increasing t_c.

For long likelihood computation times $t_e \gg t_c, t_p, t_d$ such that the overheads due to parallelisation can be neglected, the maximum speedup of E can be achieved. For shorter evaluation times the speedup will not be as great, and when $t_e \cdot E \approx t_c$ the parallel algorithm will start being less effective than the serial algorithm.

3.1 Communication time with MPI

The LogP model of parallel computation of Culler et al. (1993) introduces four parameters that characterise the communication times in parallel algorithms. In simple algorithms involving only point to point communication using MPI, communication time for a single send and receive t_c in µs can be described by a simple formula involving three parameters (Xu and Hwang, 1996; Girona et al., 2000):

$$t_c = L + \frac{m}{r_\infty} \tag{3}$$

Where L is the latency (or startup time) in µs, m is the message size in bytes and r_∞ is the asymptotic bandwidth for large messages in Mbytes per second. Therefore, due to the latency term, even small amounts of message passing will stop the maximum speedup from ever being reached.

Point to point communication latency and bandwidth are both expected to be $\mathcal{O}(1)$ (Xu and Hwang, 1996), though running Intel® MPI Benchmarks 3.2.3 on the system shows weak dependence on number of nodes K. The results of the run for up to 8 processes for a fixed message size are given in table 1.

Table 1 MPI point to point communication times in µs for a 16 byte message on system 1 (see Sect. 5 for details on the system)

Nodes	MPI::Send		MPI::Bcast
	Single send	Complete send	
2	0.47	0.47	0.37
4	0.65	1.95	0.85
8	0.79	5.53	1.44

Rather than using point-to-point communication it is also possible to use MPI::Bcast, which sends a message to all nodes. Broadcast latency is $\mathcal{O}(\log K)$, and bandwidth is $\mathcal{O}\left(\frac{1}{\log K}\right)$ (Xu and Hwang, 1996), though the constants of proportionality are slightly larger than for point-to-point send/recv. This also necessitates sending all the proposals to all processors, which gives a larger message size. Despite this, MPI::Bcast is very efficient compared to sequential point-to-point send and receives, and the benchmarks in table 1 suggest that for small message sizes m, as used in Sect. 5, MPI::Bcast should be used instead of MPI::Send and MPI::Recv for any number of cores.

Benchmarking t_d and t_p for a random proposal with Gaussian dispersion gave the empirically determined times of around 1µs each on system 1 (see Sect. 5 for details on the system).

3.2 Threshold for speedup

The execution times of the serial and parallel algorithms can be expressed in terms of the four times identified. The pro-

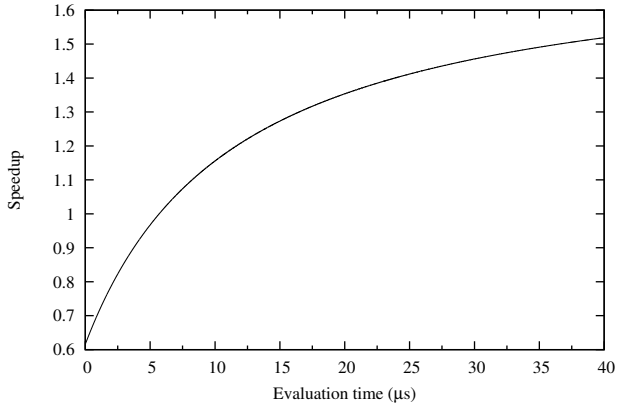

Fig. 3 Theoretical speedup (equation 6) against evaluation time t_e for $K = 2$ at the optimal acceptance probability. Note the curve asymptotes towards the expected depth for large evaluation times. Speedup greater than 1 is achieved when $t_e = 6\mu s$

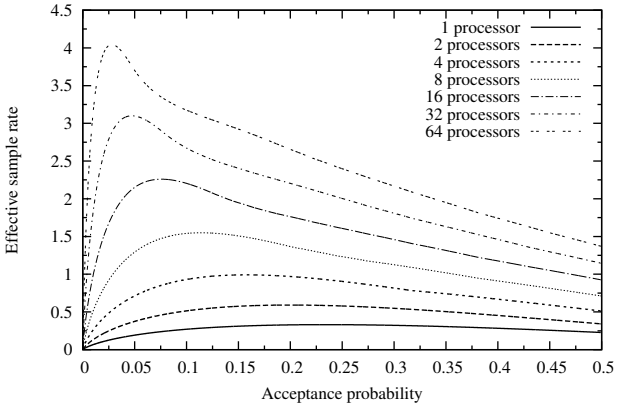

Fig. 4 Effective sample rate plotted against acceptance probability for a number of cores ranging from 1 to 64. While the curve has a shallow maximum for small number of cores, it becomes more pointed for larger number of cores and hence picking the optimal acceptance probability becomes more important

cessing time in the serial case to make E updates is therefore

$$t_S = E \cdot (t_p + t_e + t_d) \tag{4}$$

Whereas in the parallel case, following algorithm 2, the processing time to make the same number of updates requires just one iteration, which takes a time

$$t_P = K \cdot t_p + (K-1) \cdot t_c + t_e + (K-1) \cdot t_c + E \cdot t_p \tag{5}$$

Proposals must be made for all processors K, then sent to all slave processors. Evaluation of log-likelihoods is done in parallel, so as soon as the master has finished its computation after t_e the log-likelihoods are then received from all the processors slave taking a time $K \cdot t_c$. Finally, an average of E decisions must be made to traverse the tree.

The speedup, defined as the ratio of the processing time in serial to the processing time in parallel

$$S = \frac{t_S}{t_P}$$

can then be written as the ratio of equations 4 and 5:

$$S = \frac{E \cdot (t_p + t_e + t_d)}{(K-1) \cdot (t_p + 2t_c) + t_p + t_e + E \cdot t_d} \tag{6}$$

In the limit $t_e \gg t_c, t_p, t_d$ then $S \to E$ as expected. When $S > 1$ then a speedup can be achieved over the serial algorithm.

Using the times obtained in Sect. 3.1, the speedup versus t_e for two cores with an acceptance probability of 0.2 is plotted in figure 3. Note that speedup is achieved starting from very short evaluation times of the order of μs.

4 Effect of sampling efficiency

Under certain conditions, it is possible to calculate the optimum acceptance probability given the number of processors K, and from this use algorithm 2 to calculate the optimum tree shape.

Roberts, Gelman, et al. (1997) consider an n-dimensional distribution of n iid. components and normal distributed random walk proposals. Under some regularity conditions[1] on the 1-dimensional density, it is shown that in the limit of $n \to \infty$ the effective sample size as a function of the acceptance probability p is proportional to

$$p \cdot \Phi^{-1}\left(\frac{p}{2}\right)^2 \tag{7}$$

where Φ^{-1} is the inverse of the cummulative density function for a standard normal.

Combining this result with the result of Sect. 2 the effective sample size per unit time can be mapped (up to a constant) as a function of the accept probability (see fig. 4) and determine the acceptance probabilities for which maximum sampling efficiency is achieved for any number of cores. It turns out that these maxima are all obtained when the tree is shaped as the simple ladder tree shape as in Byrd et al. (2008). This will be shown numerically for a number of cores $K \leq 100$.

For a fixed number of cores K and a given p an easy criterion for determining if the ladder shape is optimal is checking if the RR...R-node ($K-1$ R's, at a depth of K), will be chosen before the A-node (at a depth of 2), that is if:

$$(1-p)^{k-1} \geq p \tag{8}$$

[1] TODO

Let p_{eq} be the point where the optimal tree structure switches from the ladder to some other shape, that is $(1-p_{eq})^{k-1} = p_{eq}$. The idea is to give a lower bound for the efficiency for $p \leq p_{eq}$ satisfying (8) and an upper bound for $p \geq p_{eq}$. Then proceed by observing that the lower bound for the ladder is higher than the upper bound for non-ladders[2].

Using the results in previous section the expected depth of the optimal tree can be numerically determined for a given probability p and a given number of cores k, $D_{p,k}$, for all p in the interval $[0;1]$ in steps of 10^{-4} and all $k \in \{1,2,\ldots,100\}$. The lower bound for the maximum efficiency for $p \leq p_{eq}$ consists of simply taking the maximum of

$$p \cdot \Phi^{-1}\left(\frac{p}{2}\right)^2 D_{p,k} \tag{9}$$

for the $p \in \{p = 0.0001z \mid z \in \mathbb{Z},\ p \leq p_{eq}\}$. This yields a lower bound as the maximum, and might be attained in the interior of an interval.

Consider now intervals of the form $I_z = [0.0001 \cdot z; 0.0001 \cdot (z+1)]$ containing $p \geq p_{eq}$. Knowing that $D_{p,k}$ is decreasing on $[0;0.5]$ and increasing on $[0.5;1]$ it can be seen that $\arg\max_{p \in I_z} D_{p,k}$ is in an endpoint of I_z. Similarly the maximum of $p \cdot \Phi^{-1}\left(\frac{p}{2}\right)^2$, attained is an endpoint except for the interval[3] $[0.2338;0.2339]$. For each interval the upper bound of the efficiency is the product of each of these maxima. Finally maximum is taken over the intervals. The result of this analysis is shown in appendix B.

The probabilities thus derived can be used to tune a parallel MCMC algorithm. TODO: emphasise and expand on this point

4.1 Counter example

The previous result depends on the efficiency function. It is not so that the ladder shape is the optimal shape in all generality. For example, if the target distribution is a one dimensional normal other tree shapes should be utilized to achieve the optimal speedup. The efficiency curve can be estimated with the inverse integrated autocorrelation times (inverse IACT) with different proposal variances:

$$[1 + 2(\rho_1 + \rho_2 + \rho_3 \ldots)]^{-1} \tag{10}$$

Using a straightforward linear search the optimal accept probabilities and the corresponding treeshapes can be found. These are depicted in figure 7.

[2] The wording might be a bit misleading, as it is only an upper bound for non-ladder efficiencies over probabilities where the ladder shape is not the optimal

[3] The maximum of equation 7 is found to be 0.3314332 for $p = 0.2338102$

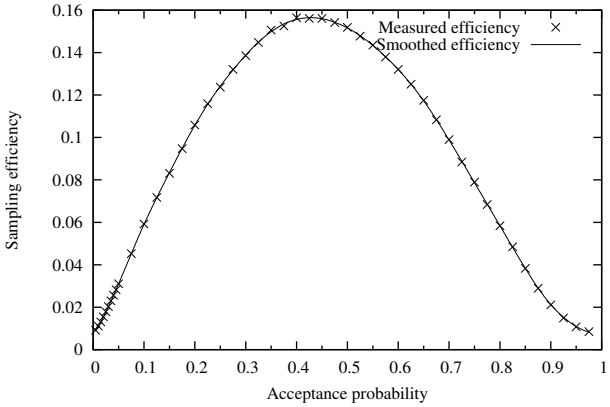

Fig. 5 Approximation of efficiency plotted against accept probability for a one dimensional standard normal target

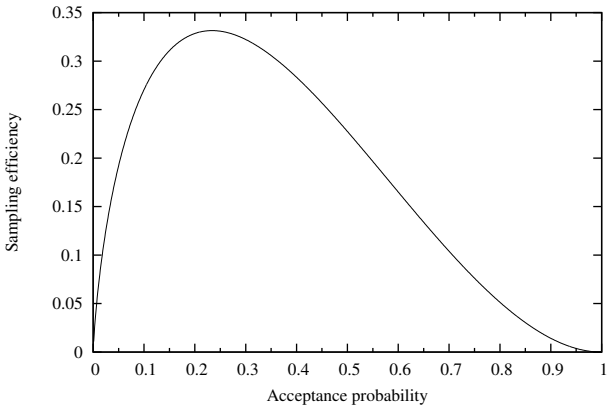

Fig. 6 Asymptotic efficiency curve derived from Roberts, Gelman, et al. (1997)

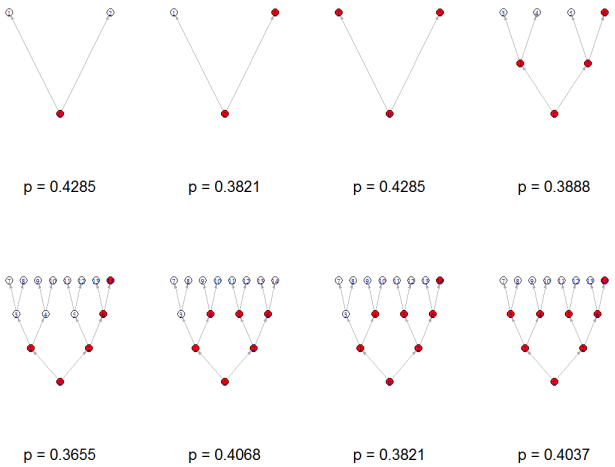

Fig. 7 Optimal tree shapes and corresponding acceptratios. Reject nodes are right accepts are left

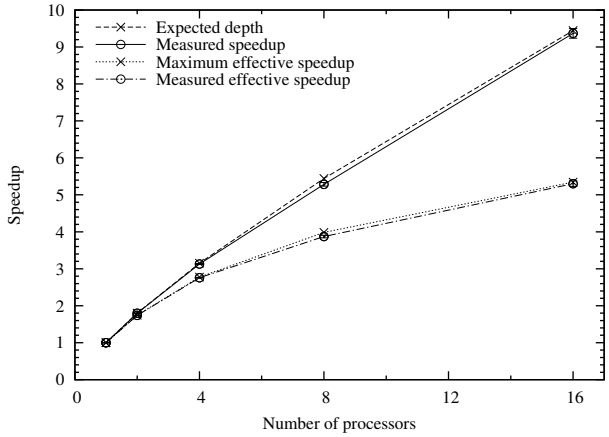

Fig. 8 Speedup compared to the serial algorithm on system 2. The maximum possible speedup is the expected depth; the lower two lines adjust for the decreased sampling efficiency caused by lowering the acceptance probability

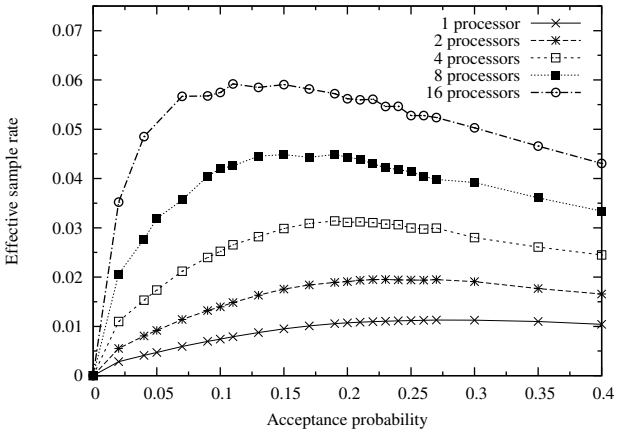

Fig. 9 Effective sample rate as measured on system 2, using the numerical approximation for the efficiency (equation 10). The shape is the same as figure 4

5 Simulations

The simulation run is of 4000 updates while sampling from a five-dimensional uncorrelated Gaussian, which is close to the infinite dimensional distribution considered by Roberts, Gelman, et al. (1997). Proposals are normally distributed with a variance chosen (empirically) to give the optimal accept probability for the given number of cores. The chain converges to the target distribution, as measured by eye, after only around 10 iterations so the results measured are applicable to the speedup of the algorithm after burn-in.

The simulations were run on the following systems:

- System 1 - 2.8GHz Xeon/Harpertown (8 cores per node) SGI ICE 8200 MPI Cluster, Linux 2.6.16.60
- System 2 - 2.67GHz Xeon/Westmere EX (8 cores per node) SGI UV 100 Shared memory system, Linux 2.6.32.54

Code was written in C++, using MPI for inter-process communication. Between 1 and 64 MPI processes were tested on system 1. 8 cores were available at each node, thus 2 nodes were used for 16 processes, 4 nodes for 32 processes and so on. This introduces extra communication time t_c for each extra node required, as it necessitates more costly inter-processor communication across the buses on the motherboard.

The run-time of 10 such simulations is averaged at each point. t_e is artificially made longer by generating large amounts of random numbers for each log-likelihood evaluation, allowing measurements over a range of values of t_e. The time recorded was between execution starting and the chain being generated, therefore including the extra time to initialise the MPI environment and calculate the optimal tree shape in the parallel case.

Table 2 shows the results for the speedup on both of the systems. The speedups of 1.80 in the two-core case, and 3.13 in the four-core case are slightly higher than the 1.54 and 2.22 achieved by Byrd et al. (2008). Efficient communication between large number of cores is now possible, and speedups are still increasing up to at least 64 cores.

Cores	System 1	System 2
2	1.79	1.80
4	2.97	3.13
8	5.17	5.28
16	8.67	9.37
32	14.4	–
64	25.3	–

Table 2 Measured speedups on systems 1 and 2

As seen in table 2 system two performed better than system one, particularly for larger numbers of processors. This is because the shared memory allows much faster communication between processes, and there is therefore no penalty for communication between multiple nodes. System two realises speedups 99% of the maximum achievable with the algorithm using 16 cores, whereas system one achieves 92% of the maximum. The shared memory scenario of system two is most similar to consumer multi-core systems, so those running the algorithm on standard hardware can expect to see similar speedups.

Effective speedup is defined as

$$S_{\text{eff}} = \frac{t_S}{t_P} \cdot \frac{\text{efficiency}_P}{\text{efficiency}_S} \tag{11}$$

where efficiency_P is the sampling efficiency for the parallel case given the acceptance probability used, and efficiency_S is the same for the serial case. It represents speedup of independent sampling rate. This is plotted for system 2 in figure 8, and as predicted in Sect. 4 is always increasing with number of cores, and our implementation successfully realises this.

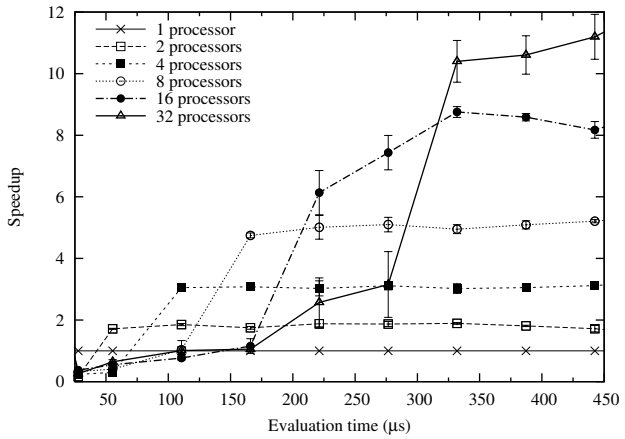

Fig. 10 Speedup at various values of t_e on system 1. The point at which each line crosses the one beneath it is the breakeven point, the minimum value of t_e for which additional speedup can be achieved by using double the previous number of cores. When the lines become horizontal, t_e is long enough to achieve the maximum speedup for the number of cores

Figure 9 shows a plot of effective sample rate against acceptance probability for different numbers of cores. Though this is for a five-dimensional Gaussian, it should be compared with the theoretical analysis of an infinite dimensional Gaussian presented in figure 4. The shape is very similar, and shows the important result that for increasing number of cores, acceptance probability should be lowered to maximise sampling rate.

To test the analysis of Sect. 3 and probe the efficacy of the algorithm in more extreme limits, figure 10 was produced. This shows speedup S plotted against log-likelihood evaluation time t_e for different numbers of cores. For two cores, the shape of the curve matches that of figure 3 which shows the theoretical analysis is qualitatively correct. Quantitatively there is a slight deviation, as the x-axis appears slightly stretched. That is, it actually takes longer evaluation times than predicted to achieve a given speedup. This is likely due to some extra time in initialising the parallel algorithm, and extra overheads in communication time that are not seen in a benchmarking setup. Note that the algorithm has still performed very well, with the log-likelihood evaluations times required for a large speedup to be seen very small, on the order of hundreds of µs.

Distribution of the proposals between processors was also implemented, and simulated on system 1 with the same parameters as in fig 8. It performed worse than the case where one processor makes all proposals then sends them out to have their log-likelihoods computed. In the 32-processor case it was 75% slower, and in the 4-processor case 10% slower. This is due an extra overhead in total communication time per iteration of $K \cdot (2L)$, much greater than the decrease in the extremely short proposal time $\frac{t_p}{E}$.

6 Application from population genetics

In population genetics its easy to simulate genealogies histories using the continuous time approximation in the Wright-Fisher (WF) model (Wright, 1931; Fisher, 1930). The infinite-site assumption makes a convenient constraint on the genealogical histories reducing the possible ancestral states needed to be considered (Kimura, 1969).

The effective mutation rate θ is an important parameter that quantifies genetic diversity in the WF model. For a given evolutionary relationship connecting the sequences it is relatively straight forward to calculate the likelihood of the effective mutation rate θ using the Ethier-Griffiths-Tavaré (EGT) recursive equation (Ethier and Griffiths, 1987; Griffiths and Simon, 1994).

$$\pi_n(\mathbf{n}) = \frac{n-1}{n-1+\theta} \sum_{n_j>1} \frac{n_j-1}{n-1} \pi_{n-1}(\mathbf{n}-\mathbf{e}_j) + \frac{\theta}{n-1+\theta} \sum_{\text{singletons}} \frac{n_i+1-\delta_{ij}}{n} \pi_n(\mathbf{n}-\mathbf{e}_j+\mathbf{e}_i)$$

The exact approach using the recursive equation is not computational viable for large datasets since the number of events grows quite rapidly. The other naive approach is to estimate the posterior distribution by sampling random genealogical histories.

$$\pi(\theta|y) \approx \sum_G \pi(\theta|y,G)$$

Since the number of possible histories grows quickly with number of sequences and sites, this approach is not viable for real datasets. One approach is by introducing an $I(G)$ importance distribution for the histories by simulating draws that make significant contributions to the posterior probability estimate using $I(G)$.

$$\hat{\pi}(\theta|y) = \sum_G \pi(G|y) \frac{\pi(G|\theta)}{I(G)}$$

By choosing histories that contribute significantly to the posterior distribution we can reduce the variance the estimate for the same number of iterations. This special use case of importance sampling was pioneered by Griffiths and Simon (1994) and there has been some research into the choice of the importance sampling distribution and for our purposes we use the proposal distribution from Hobolth et al. (2008).

6.1 GIMH (IS-MCMC)

A Markov chain was constructed to estimate the posterior probability of θ and in each step we use the estimate of $\pi(\theta|y)$. This special form of MCMC framework with a embedded Monte Carlo estimate is called grouped independence Metropolis Hastings sampler (GIMH) by Beaumont

(2003). If the estimate of the posterior distribution of θ is kept from the last iteration in the MCMC sampler (see algorithm 6.1) it guarantees that the correct posterior distribution is reached in the limit (Beaumont, 2003).

Algorithm 4 MCMC with importance sampling

for i from 1 to N **do**
  simulate $\theta' \sim q(\theta_i, \cdot)$;
  Generate unbiased estimate $\hat{\pi}(\theta')$ of $\pi(\theta')$ with IS;
  Set $\alpha \leftarrow \min\left\{1, \frac{\hat{\pi}(\theta'|y)}{\hat{\pi}(\theta_i|y)}\right\}$;
  with probability α set $\theta_{i+1} = \theta'$ and $\hat{\pi}(\theta_{i+1}|y) = \hat{\pi}(\theta'|y)$;
  otherwise set $\theta_{i+1} = \theta_i$ and $\hat{\pi}(\theta_{i+1}|y) = \hat{\pi}(\theta_i|y)$;
end for

The acceptance ratio over the whole chain is expected to be lower when using estimated posterior probabilities instead of the exact probabilities. Intuitively this is because if the posterior probability is overestimated for a particular value of the parameter and the Markov chain gets "stuck" in that state.

Proposition 1 in the appendix shows that this is more than an intuition and the usage of the estimated likelihood instead of the exact likelihood truly lowers the acceptance ratio. For the efficiency curve in the high dimensional case, the optimal tree shape was a ladder, this lowering of the acceptance ratio in the GIMH framework shifts the optimal tree shape even further towards a ladder.

6.2 Simulations

We got a working C++ implementation of the importance sampling scheme from Hobolth et al. (2008). To our surprise we didn't find any find suitable implementation of an importance sampler of genealogical histories in a high-performance language such as C and C++.

A dataset of segregating sites of ten samples and effective mutation rate of one was simulated using the program ms developed by Hudson (2002). Estimation was done using the parallel MCMC algorithm with computer system 1 from section 5.

Initial state was 1 for θ and Gaussian distribution on the log scale was used for a jumping distribution negating the need for the MH-proposal correction. The proposal variance was hand tuned until the acceptance ratio reached value near the optimal value for the one dimensional Gaussian case with reasonable high number of IS samples 50. The uninformative improper prior on the positive real half-line was used for computational convenience.

As described in Sect. 6.1, relationship between lower acceptance ratio and worse estimates of the likelihood is demonstrated in figure 11. The estimated posterior density

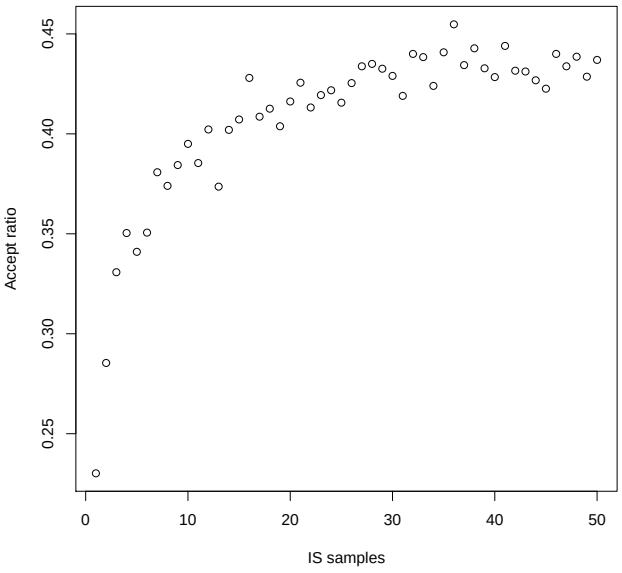

Fig. 11 Importance samples versus acceptance ratio, with higher number of samples the acceptance ratio converges to the acceptance ratio using the exact posterior probability.

for the effective mutation parameter from the MCMC iterations has the peak in $\theta = 1$ validating the implementation of the IS algorithm figure 12.

The main point here is the speedup gained by using the parallelization algorithm. Table 3 shows a significant speedup for a practical example from population genetics although the example is very simple further extension using the parallelization are trivially implemented. This demonstrates the practicality of this algorithm, gaining speedup without great modification to the typical MCMC estimation scenario.

Table 3 Running times for the parallelized GIMH algorithm for estimating the effective mutation rate using multiple cores.

Nr of cores	1	2	4	8
Time (s)	115.11	64.72	42.65	32.64

7 Concluding remarks

An algorithm has been developed which parallelises MCMC using speculative computation in the most efficient way possible. The algorithm is simple to implement and can provide speedups to any MCMC process with a likelihood evaluation time longer than 100μs. Speedups of 3.13 were achieved on a four core shared memory system, and a speedup of 25.3 times was achieved on 64 core cluster. A real-life application to estimating mutation rate was also considered, where a speedup of 2.70 was achieved using four cores. These are

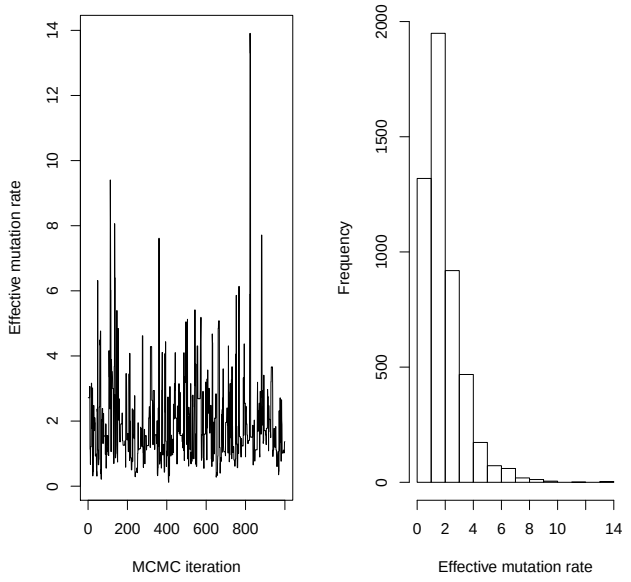

Fig. 12 The trace plot of the MCMC iterations on the left and the estimated posterior distribution on the right for the mutation rate parameter θ.

easily accessible speedups with modern hardware, and may be of interest to many users of MCMC.

The algorithm has the feature that lowering the acceptance rate increases its sampling efficiency. This is very useful in applications where it is difficult to create proposals that are ever accepted, for example rearranging phylogenetic trees. In these cases it is possible to achieve speedups which approach being linear with number of processors for very little extra effort.

The rule of thumb that should be taken away is that for most applications the best thing to do is use a tree consisting of entirely reject nodes, as proposed by Byrd et al. (2008). The acceptance probability should be lowered the more processors used, in line with Sect. 4 if possible.

Further work in this area may include investigation into the most effective acceptance probability and the tree shape this gives in the case of pseudomarginal MCMC, considered briefly in Sect. 6. Here, the importance sample size affects the efficiency of the MCMC sampling, so there are two parameters to optimise over to get the optimal parallelization. Including this in the theoretical framework would allow efficient application of the algorithm to this newer area of MCMC, and expand the potential user base.

The assumption that acceptance probability is constant and uncorrelated to the previous state could be relaxed. It is clearly not entirely correct, as when in a low likelihood state it will be higher than the average value, and when in a high likelihood state is will be lower than the average value. A potential idea for this is use of the beta distribution to estimate the acceptance probability given the state, and redraw the tree periodically.

Acknowledgements This work was carried out as part of the Oxford Summer School in Computational Biology, 2012, in conjunction with the Department of Plant Sciences, and with support from the Department of Zoology. The authors would like to acknowledge the use of the Oxford Supercomputing Centre (OSC) in carrying out this work.

References

Beaumont, M. A.: Estimation of population growth or decline in genetically monitored populations. Genetics **164**(3), 1139–1160 (July 2003).

Brockwell, A. E.: Parallel Markov Chain Monte Carlo Simulation by Pre-Fetching. Journal of Computational and Graphical Statistics **15**(1), 246–261 (2006). doi: 10.1198/106186006X100579.

Byrd, J., S. Jarvis, and A. Bhalerao: Reducing the run-time of MCMC programs by multithreading on SMP architectures. Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 1–8 (2008). doi: 10.1109/IPDPS.2008.4536354.

Culler, D. et al.: LogP: towards a realistic model of parallel computation. Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming - PPOPP '93 **28**(7), 1–12 (1993). doi: 10.1145/155332.155333.

Ethier, S. N. and R. C. Griffiths: The infinitely-many-sites model as a measure-valued diffusion. The Annals of Probability **15**, 515–545 (1987).

Fisher, R. A.: *The Genetical Theory of Natural Selection*. Clarendon press.

Girona, S., J. Labarta, and R. M. Badia: Validation of Dimemas Communication Model for MPI Collective Operations. Recent Advances in Parallel Virtual Machine and Message Passing Interface. Lecture Notes in Computer Science **1908**, 39–46 (2000).

Griffiths, R. C. and T. Simon: Simulating probability distributions in the coalescent. Theoretical Population Biology **46**(2), 131–159 (1994).

Hastings, W. K.: Monte Carlo Sampling Methods Using Markov Chains and Their Applications. Biometrika **57**(1), 97–109 (1970).

Hobolth, A., M. K. Uyenoyama, and C. Wiuf: Importance Sampling for the Infinite Sites Model. Statistical applications in genetics and molecular biology **7**(1), Article32 (Jan. 2008). doi: 10.2202/1544-6115.1400.

Hudson, R. R.: Generating samples under a WrightffffdfffdfffdFisher neutral model of genetic variation. Bioinformatics **18**(2), 337–338 (2002).

Kimura, M.: The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. Genetics **61**, 893–903 (1969).

Metropolis, N. et al.: Equation of State Calculations by Fast Computing Machines. The Journal of Chemical Physics **21**(6), 1087–1092 (1953). doi: 10.1063/1.1699114.

Roberts, G. O., A. Gelman, and W. R. Gilks: Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms. The Annals of Applied Probability **7**(1), 110–120 (1997).

Roberts, G. O. and J. S. Rosenthal: Optimal Scaling for Various Metropolis-Hastings Algorithms. Statistical Science **16**(4), 351–367 (2001).

Rosenthal, J. S.: Parallel computing and Monte Carlo algorithms. Far East Journal of Theoretical Statistics **4**, 207–236 (1999).

Witte, E., R. Chamberlain, and M. Franklin: Parallel simulated annealing using speculative computation. Parallel and Distributed Systems, IEEE Transactions on **2**(4), 483–494 (1991). doi: 10.1109/71.97904.

Wright, S.: Evolution in Mendelian Populations. Genetics **16**, 97–159 (1931).

Xu, Z. and K. Hwang: Modeling communication overhead: MPI and MPL performance on the IBM SP2. Parallel Distributed Technology: Systems Applications, IEEE **4**(1), 9–24 (1996). doi: 10.1109/88.481662.

A Optimal tree shape in generality

The greedy choice algorithm, algorithm 2, can be extended to the case where sub-processes are spawned recursively and to include changing accept probabilities. If the implementation allows a new iteration to be started as soon as the accept/reject path leaves the tree, the time per iteration will depend on I.

The objective is then to maximize

$$\mathbb{E}\left[\frac{d_I}{t_I}\right] \tag{12}$$

d_i being the depth of node i, where the root has depth 1 and t_i is the time to finish the algorithm at node i.

As in Sect. 2, let P denote the stochastic accept/reject path, I the end node of this path and by $(\cdot)'$ the parent node or corresponding value in parent. Finally T_K is a subtree of the full binary tree on K nodes. Then

$$\begin{aligned}\mathbb{E}\left[\frac{d_I}{t_I}\right] &= \mathbb{E}\left[\sum_{i\in T_K} \mathbb{1}(i\in P)(\frac{d_i}{t_i}-\frac{d_i'}{t_i'})\right] \\ &= \sum_{i\in T_K} \mathbb{P}(i\in P)(\frac{d_i}{t_i}-\frac{d_i'}{t_i'})\end{aligned}$$

This suggests that in order to maximize (12) the K nodes in T_K should be chosen greedily using the selection function

$$\mathbb{P}(i\in P)(\frac{d_i}{t_i}-\frac{d_i'}{t_i'}) \tag{13}$$

That is, choose the K nodes with the highest value of (13). The property that parents will always be chosen before children, ensured the correctness of this algorithm in the simpler case, is not generally satisfied here. Proceed by assuming this is still satisfied, then algorithm 2 can still be used, just replacing the priorities with equation 13.

If the condition

$$\mathbb{P}(i\in P)(\frac{d_i}{t_i}-\frac{d_i'}{t_i'}) < \mathbb{P}(i'\in P)(\frac{d_i'}{t_i'}-\frac{d_i''}{t_i''}) \tag{14}$$

is satisfied, then algorithm 2 will give the tree on K processors that maximizes expected depth (12).

In the parallel computation model derived from discussed in Sect. 3, the computation times satisfy

$$t_i < t_e$$

for all $i \in T_K$. Also it is the case that

$$\begin{aligned}\frac{d_i}{t_i}-\frac{d_i'}{t_i'} &< \frac{d_i''+2}{t_i''}-\frac{d_i''+1}{t_i''} \\ \frac{d_i}{t_i}-\frac{d_i'}{t_i'} &< \frac{d_i''+2}{t_i''}-\frac{d_i''+1}{t_i''}\end{aligned} \tag{15}$$

Note the right-hand sides are equal in the limit of

$$\frac{t_c}{t_e}\to 0$$

As $\mathbb{P}(i\in P) < \mathbb{P}(i'\in P)$, the condition of equation 14 will roughly always be satisfied when the ratio between communication time and evaluation time is small.

B Numerical proof of ladder tree shape being optimal

Table 4

k	p	ladder	tree	k	p	ladder	tree	k	p	ladder	tree
2	0.1999	0.59	0.34	35	0.0443	3.22	2.98	68	0.0267	4.13	3.84
3	0.1758	0.81	0.58	36	0.0434	3.25	3.01	69	0.0264	4.15	3.86
4	0.1577	0.99	0.78	37	0.0426	3.29	3.05	70	0.0261	4.17	3.88
5	0.1434	1.15	0.95	38	0.0417	3.33	3.08	71	0.0258	4.19	3.90
6	0.1318	1.30	1.10	39	0.0409	3.36	3.11	72	0.0255	4.21	3.92
7	0.1221	1.43	1.23	40	0.0402	3.39	3.15	73	0.0253	4.23	3.93
8	0.1140	1.55	1.36	41	0.0394	3.43	3.18	74	0.0250	4.25	3.95
9	0.1069	1.66	1.47	42	0.0387	3.46	3.21	75	0.0247	4.27	3.97
10	0.1008	1.76	1.57	43	0.0381	3.49	3.24	76	0.0245	4.29	3.99
11	0.0955	1.86	1.66	44	0.0374	3.52	3.27	77	0.0242	4.31	4.00
12	0.0907	1.95	1.75	45	0.0368	3.55	3.30	78	0.0240	4.33	4.02
13	0.0864	2.03	1.83	46	0.0362	3.58	3.32	79	0.0237	4.35	4.04
14	0.0826	2.11	1.91	47	0.0356	3.61	3.35	80	0.0235	4.37	4.06
15	0.0791	2.19	1.99	48	0.0350	3.64	3.38	81	0.0232	4.38	4.07
16	0.0759	2.26	2.06	49	0.0345	3.67	3.41	82	0.0230	4.40	4.09
17	0.0730	2.33	2.12	50	0.0339	3.70	3.43	83	0.0228	4.42	4.11
18	0.0704	2.39	2.19	51	0.0334	3.72	3.46	84	0.0226	4.44	4.12
19	0.0679	2.46	2.25	52	0.0329	3.75	3.48	85	0.0224	4.45	4.14
20	0.0656	2.52	2.31	53	0.0325	3.78	3.51	86	0.0221	4.47	4.16
21	0.0635	2.58	2.36	54	0.0320	3.80	3.53	87	0.0219	4.49	4.17
22	0.0616	2.63	2.42	55	0.0315	3.83	3.56	88	0.0217	4.51	4.19
23	0.0597	2.69	2.47	56	0.0311	3.86	3.58	89	0.0215	4.52	4.20
24	0.0580	2.74	2.52	57	0.0307	3.88	3.60	90	0.0213	4.54	4.22
25	0.0564	2.79	2.57	58	0.0303	3.90	3.63	91	0.0212	4.55	4.23
26	0.0549	2.84	2.61	59	0.0299	3.93	3.65	92	0.0210	4.57	4.25
27	0.0534	2.88	2.66	60	0.0295	3.95	3.67	93	0.0208	4.59	4.26
28	0.0521	2.93	2.70	61	0.0291	3.98	3.69	94	0.0206	4.60	4.28
29	0.0508	2.97	2.75	62	0.0287	4.00	3.71	95	0.0204	4.62	4.29
30	0.0496	3.02	2.79	63	0.0284	4.02	3.74	96	0.0203	4.63	4.31
31	0.0484	3.06	2.83	64	0.0280	4.04	3.76	97	0.0201	4.65	4.32
32	0.0473	3.10	2.87	65	0.0277	4.07	3.78	98	0.0199	4.66	4.34
33	0.0463	3.14	2.90	66	0.0274	4.09	3.80	99	0.0198	4.68	4.35
34	0.0453	3.18	2.94	67	0.0270	4.11	3.82	100	0.0196	4.69	4.36

Table 4 For each number of cores $K \leq 100$ the optimal ladder probability, the optimal efficiency in the ladder (equation 9) and the optimal efficiency in a non-ladder (only under probabilities where the ladder doesn't guarantee greater expected depth) are given

C GIMH lowering proof