Running Regenerative Markov Chains in Parallel

Vikneswaran Gopal* George Casella†

IBM Research University of Florida

December 9, 2011

Abstract

Parallel computing is playing an increasingly large role in statistical computations. One reason for this is the computational complexity of modern techniques. In this paper, we present a parallel algorithm that provides a considerable speed-up in the running of a large class of Markov chain algorithms. We formally show that it remains theoretically sound, and derive a version of Amdahl's Law to demonstrate the benefits of parallelizing. Finally, we demonstrate the use of the methodology on Hierarchical Linear Models via our R-package `McParre`.

Keywords and phrases: Markov Central Limit Theorem, Minorization, Regeneration, Hierarchical models, Bayes models, Parallel Processing

*Research Staff Member, IBM Singapore Research Collaboratory. Email: `viknes@sg.ibm.com`. This research is part of the author's PhD thesis.

†Distinguished Professor, Department of Statistics, University of Florida, Gainesville, FL 32611. Supported by National Science Foundation Grants DMS-0631632 and SES-0631588. Email: `casella@stat.ufl.edu`.

1 Introduction

There is no doubt that the role of parallel computing in statistical computations is getting bigger by the day. Journal papers that exhort the use of parallel processors in statistical applications, such as Suchard *et al.* (2010) and Zhou *et al.* (2010), have become more frequent in recent times. For a comprehensive review of the many parallel computing techniques at our disposal today, the reader is referred to Kontoghiorghes (2006) and Matloff (2011).

What exactly is driving this need for parallel computing? All signs point to increasingly large datasets, and to the need to solve problems of increasing computational complexity. In the field of genetics, for example, it can be argued that datasets are increasing in size faster than the processing hardware is improving to handle them. On the other hand, methodologies such as bootstrapping require more computing resources than typical statistical techniques, simply by design. Similarly, Monte Carlo simulation techniques allow more sophisticated analyses of data, but again, they demand more processing power. These would naturally point to a need to boost the speed of our individual processors, but the trend is very clearly towards linking up processors of moderate speed rather than a concerted effort to increase the speed of individual processors. The reason for this is simple: The former option can be executed at a fraction of the cost of the latter. Doubling the speed of a processor is costly because it requires memory access speeds and cooling to be concomitantly improved. Adding another processor, on the other hand, would not give the same speed and would require a rewrite of programs in order to fully optimize the new parallel processing environment, but it is inordinately cheap, easy to do, and is a scalable solution. This explains why supercomputers are prohibitively expensive except to dedicated research institutes, whereas cluster computing possibilities are readily available and accessible. Most universities have their own cluster, while companies such as Amazon, IBM and now HP, have made cloud computing available to anyone at an extremely cheap cost. For this reason too, our desktops and notebooks already contain multiple processors.

In this section we shall touch on some of the topics covered in Suchard *et al.* (2010) and

Table 1: Taxonomy of parallel processing architectures, as per Flynn (1972)

SISD	Single Instruction Single Datastream
MISD	Multiple Instruction Single Datastream
SIMD	Single Instruction Multiple Datastream
MIMD	Multiple Instruction Multiple Datastream

Kontoghiorghes (2006), but our main aim is to introduce the parallel computing architecture and approach that we have chosen for our purpose. Years ago, a taxonomy of parallel processing architectures was put forward in Flynn (1972). Today, the dividing lines are not as clean as before since many contemporary parallel set-ups are hybrids of those outlined in that seminal paper. However the original breakdown is still a good starting point.

Referring to Table 1, models for parallel computation can be categorised into 4 basic kinds. Single-core desktops and notebooks from a few years ago fall under the SISD category. The MISD model is used only in specialised applications, such as fault-tolerant computing. The two most widely used architectures today are SIMD and MIMD. GPU processing is one good example of an SIMD machine, while a Beowulf[1] cluster is one example of an MIMD machine. The MIMD model can be further sub-divided into shared memory machines and distributed memory ones. Distributed memory MIMD machines typically co-ordinate their work by passing messages to one another.

Which architecture is chosen depends on the algorithm we intend to parallelize, and how we intend to parallelize it. In a medium-grain parallelization, it is possible to divide the domain of work into independent data "chunks" and work on them concurrently. In such problems, it is natural to utilize the SIMD architecture, in which each processor can access and work on the disparate chunks. In a more coarse-grained parallelization, where individual functions or procedures are to be run concurrently, the MIMD model is more appropriate.

Our intention is to tackle the problem of running a Markov chain Monte Carlo (MCMC) algorithm in parallel. It is not immediately apparent how such an algorithm can be adapted

[1] A cluster of computers connected by a fast network

to run in parallel, as such procedures are inherently sequential in the sense that they require the current state to be generated before proceeding. The goal here is to study the best method of adapting a general class of MCMC algorithms so as to fully capitalize on a parallel computing environment.

The rest of this paper is outlined as follows: Section 2 contains a short overview of current approaches to running Markov chains in parallel. Section 3 introduces our parallel algorithm and provides the theoretical justification for it. Section 4 provides pseudo-code for our algorithm and analyses the speed-up that it can potentially provide. Section 5 contains examples to demonstrate that the methodology works and provides considerable speed-up. Before finishing with a discussion on future directions, we introduce `McParre` - an R package for running our parallel algorithm on a cluster.

2 Parallel MCMC

One notable attempt at running a Metropolis-Hastings chain in parallel can be found in Brockwell (2006), where the algorithm calls on a cluster to pre-compute likelihoods of the candidate distribution and thus save time. It has been extended in Strid (2010) to the case where the current state is used to make "better" future candidate predictions. By "better", the authors mean that they can tune the candidate until an optimal acceptance rate is achieved. However, this destroys the Markovian nature of the chain, and would fall under the category of adaptive algorithms rather than a traditional MCMC algorithm.

The most direct method of transplanting an MCMC algorithm onto a cluster would be to run one chain of length N on each of n processors, and then use the mean of the means as a point estimate of the desired integral, and the standard error of the observed means as an estimate of the variability in order to get a confidence interval. Although appealing in its simplicity, this method provides no savings in terms of computing time and it will not always provide accurate results. For example, in Fishman (2001) (Chapter 6), it is shown that under mild assumptions on the rate at which the asymptotic bias reduces to 0, as the

number of chains n increases to infinity, the burnin in each chain, k, has to increase faster than $\log n$ in order for the coverage probability of the confidence interval to hold. Thus it almost defeats the purpose of running the chain on a cluster, where we hoped that having more processors would have gained us time and/or precision.

Regenerative simulation, on the other hand, provides a very clean way of parallelizing MCMC algorithms. At every regeneration point, the random variable generated is independent of its current state, and everything before it. Hence regeneration points provide us with independent segments or "tours" that can be run on separate processors, and then concatenated later!

Our intention, then, is to run individual tours on separate processors and then join them together. This is a coarse-grain parallelization of the MCMC algorithm, and hence is most suited to the MIMD distributed memory model architecture. There are instances where MCMC algorithms have been run on GPUs, that is, on an SIMD model. The best example we know of is in Suchard *et al.* (2010), where a fine grain parallelization is applied to a Gibbs sampler in order to parallelize and provide a speed up within the iterations. GPUs afford an incredible speed-up. However, they require very careful implementation of an algorithm, which translates to a long development time. What we aim for is to allow a researcher to run a regenerative MCMC algorithm, as long as they can provide the minorization (see Section 3 for the definition and explanation of required MCMC concepts). Any increase in processing power (by adding nodes) will then provide a speed-up by decreasing the execution time to run a fixed number of tours.

3 Regeneration in Parallel

In this section, we shall formally lay down our notation, thus clarifying, for example, what we have referred to as regenerations and tours. Let $\Phi = \{X_0, X_1, \ldots\}$ be a Markov chain taking values in the sample space $(E, \mathscr{E})$ with transition kernel $P(\cdot, \cdot)$, where E is typically $\mathbb{R}^k$ and $\mathscr{E}$ are the Borel sets on E. Also assume that Φ possesses the following properties:

1. Φ has a stationary distribution π. In other words, $\pi(A) = \int P(x, A)\pi(dx)$ for all $A \in \mathscr{E}$.

2. Φ is π-irreducible, aperiodic and Harris recurrent.

3.1 A Markov Chain Central Limit Theorem

Suppose we wish to use an MCMC algorithm to estimate $E_\pi g = \int g(x)\pi(dx)$. The assumptions above imply that we have an ergodic theorem, which yields

$$\bar{g} = \frac{1}{n}\sum_{i=0}^{n-1} g(X_i) \to E_\pi g \text{ a.s.} \quad \text{as} \quad n \to \infty.$$

If in addition we have that $E_\pi|g|^{2+\epsilon} < \infty$ for some $\epsilon > 0$, then, as shown in Chan & Geyer (1994), we can apply a Central Limit Theorem (CLT) to obtain a confidence interval for $E_\pi g$.

Theorem 3.1 (Chan & Geyer (1994))**.** *Suppose assumptions* **1.** *and* **2.** *hold for* Φ*. If* Φ *is geometrically ergodic and* $E_\pi|g|^{2+\epsilon} < \infty$ *for some* $\epsilon > 0$*, then*[2]

$$\sqrt{n}(\bar{g} - E_\pi g) \rightsquigarrow N(0, \gamma^2) \quad \textit{as} \quad n \to \infty,$$

where

$$\gamma^2 = Var_\pi[g(X_0)] + 2\sum_{i=1}^{\infty} Cov_\pi[g(X_0), g(X_i)].$$

Unfortunately, there does not exist a straightforward estimate of γ^2, even though there is a vast amount of research being done in this area. One of the more promising candidates to estimate γ^2 is the Batch Means estimator (see Jones *et al.* (2006), Flegal *et al.* (2008) and Flegal & Jones (2010)). However, it requires two levels of asymptotics in order to achieve consistency - both the batch size and the chain length have to tend to infinity, and the batch size is typically chosen by convention rather than by derivation. On the other hand,

[2] $\rightsquigarrow$ denotes convergence in distribution

introducing regeneration times into a Markov chain allows us to break it up into i.i.d blocks, or tours. Applying the classical CLT to these i.i.d blocks yields a CLT with a naturally arising consistent estimate of the variance. In addition, using the regeneration CLT removes *any* need for the chain to be in stationarity.

3.2 Minorization and Regereneration

In order to introduce regeneration times, we have to "split" the chain up first. For this we need to minorize it. The concept of minorization goes back to Nummelin (1978) and Athreya & Ney (1978); see also Nummelin (2004).

Definition 3.1 (Minorization Condition)**.** A Markov transition kernel $P(\cdot, \cdot)$ satisfies a *minorization condition* if there is a measurable function s and a finite non-negative measure ν such that for all $x \in E$ and $A \in \mathscr{E}$,

$$P(x, A) \geq s(x)\nu(A). \tag{1}$$

The function s is called a small function, and ν is referred to as a small measure. Since the latter is finite, we can always normalize it to ensure that it is a probability measure, and absorb the normalizing constant $\nu(E)$ into $s(x)$. Henceforth, we shall always assume that this is the case. Since $\nu(E) = 1$, it follows that $0 \leq s(x) \leq 1$.

Minorization implies that we can split P into a mixture of two transition kernels, because

$$P(x, dy) = s(x)\nu(dy) + (1 - s(x))P_r(x, dy), \tag{2}$$

where $P_r(x, dy) = (P(x, dy) - s(x)\nu(dy))/(1 - s(x))$. We call P_r the residual kernel. Given $X_n = x$, consider generating the chain in the following manner:

(*i*) Generate $\delta_n \sim$ Bernoulli$(s(x))$.

(*ii*) If $\delta_n = 1$, generate $X_{n+1} \sim \nu$.

(*iii*) Otherwise, generate $X_{n+1} \sim P_r(x, \cdot)$.

Several results can be shown immediately about this new sequence (X_n, δ_n) (Nummelin, 2004; Robert & Casella, 2004). The first result is that (X_n, δ_n) is a Markov chain. Second, the marginal sequence $\{X_n\}$ is a Markov chain that follows the law described by our original kernel P. Lastly, whenever $\delta_n = 1$, we have a *regeneration*, meaning that the subsequent portion of the chain $\{(X_{n+1}, \delta_{n+1}), (X_{n+2}, \delta_{n+2}), \ldots\}$ is independent of the past. This independence arises because X_{n+1} is generated from $\nu(\cdot)$ and not $P(X_{n+1}, \cdot)$.

In the mixture described above, it might be difficult to generate a random variable from P_r, but, following Mykland *et al.* (1995), this can be neatly sidestepped, since the Radon-Nikodym derivative of ν with respect to P is just

$$\Pr(\delta_n = 1 | X_n = x, X_{n+1} = y) = r(x, y) = \frac{s(x)\nu(dy)}{P(x, dy)}. \tag{3}$$

Thus, to generate the regenerative Markov chain, we only need generate Φ according to P, the original Markov chain. The "bell" variables δ_n are generated according to (3), and $\delta_n = 1$ signals that there is a regeneration. We never have to generate from the residual distribution P_r.

Definition 3.2 (Regeneration times and tours)**.** Let $0 = \tau_0 < \tau_1, \ldots$ be defined by $\tau_{i+1} = \min\{n > \tau_i : \delta_{n-1} = 1\}$ The τ_i's are referred to as the random *regeneration times. Tours* are defined by the sets of random variables $\{X_{\tau_i}, X_{\tau_i+1}, \ldots, X_{\tau_{i+1}-1}\}$.

Assume that Φ is run for a fixed number of tours, R. In other words, the chain was stopped on the Rth occasion that $\delta_i = 1$. Then if we let N_i be the length of the i-th tour, that is, $N_i = \tau_i - \tau_{i-1}$, and we define the sum of the observed values in the i-th tour to be

$$S_i = \sum_{j=\tau_{i-1}}^{\tau_i - 1} g(X_j),$$

then the random vectors (N_i, S_i) are independent and identically distributed for $i = 1, 2, \ldots, R$.

If we know that $E_\nu N_1^2$ and $E_\nu S_1^2$ are finite, then we can apply the classical CLT to obtain

$$\sqrt{R}(\bar{g}_R - E_\pi g) \rightsquigarrow N(0, \sigma^2) \quad \text{as} \quad R \to \infty, \tag{4}$$

where

$$\bar{g}_R = \frac{1}{\tau_R} \sum_{j=0}^{\tau_R - 1} g(X_j) = \frac{\sum_{i=1}^R S_i}{\sum_{i=1}^R N_i} \quad \text{and} \quad \sigma^2 = \frac{E_\nu[(S_1 - N_1 E_\pi g)^2]}{(E_\nu N_1)^2}. \tag{5}$$

A consistent estimator of the variance is given by

$$\hat{\sigma}^2 = \frac{\frac{1}{R}\sum_{i=1}^R (S_i - N_i \bar{g}_R)^2}{\bar{N}^2}. \tag{6}$$

We pointed out earlier, just prior to equation (4), that that the regenerative CLT requires S_i and N_i to have finite second moments. The achievement of Hobert *et al.* (2002) was proving that the same conditions as Theorem 3.1 are also sufficient for the regenerative CLT to hold.

Theorem 3.2 (Hobert *et al.* (2002))**.** *Suppose assumptions* **1.** *and* **2.** *hold for* Φ*, and that* Φ *can be minorized just as in Definition 3.1. If* Φ *is geometrically ergodic and* $E_\pi |g|^{2+\epsilon} < \infty$ *for some* $\epsilon > 0$*, then* $E_\nu N_1^2$ *and* $E_\nu S_1^2$ *are both finite. It follows that the CLT in equation (4) is applicable.*

3.3 Regeneration and Parallization

It should now be apparent that if we can introduce regeneration into a Markov chain Φ, then we can run several of them in parallel, and essentially concatenate the chains. Consider a case where we have d processors, each running a Markov chain dictated by the same transition kernel P. We shall index the d chains as follows:

$$\Phi_i = \{X_{i,0}, X_{i,1}, \ldots\} \text{ for } i = 1, \ldots, d.$$

Assume that $P(\cdot,\cdot)$ satisfies the minorization condition given in Definition 3.1, and that all chains are initialized with the small measure

$$X_{i,0} \sim \nu \text{ independent for } i = 1, \ldots, d.$$

We set R_i to be the (random) total number of tours in chain i, and $N_{i,j}$ to be the length of the j-th tour in chain i, where $i = 1, 2, \ldots, d$ and $j = 1, 2, \ldots, R_i$. Finally, we redefine R and $\bar{N}$ to be $R = \sum_{i=1}^{d} R_i$ and $\bar{N} = (1/R) \sum_i \sum_j N_{i,j}$. The key point of this procedure is that the chains are initialized and run independently, which leads to the pairs $(N_{i,j}, S_{i,j})$ still being independent and identically distributed. As a result, the following Proposition holds.

Proposition 3.1 (Regenerative CLT from parallel chains)**.** *Suppose Φ_i satisfies the conditions in Theorem 3.2 for all $i = i, \ldots, d$. Then if $E_\pi |g|^{2+\epsilon} < \infty$ for some $\epsilon > 0$, the following CLT holds:*

$$\sqrt{R}(\bar{g}_R - E_\pi g) \rightsquigarrow N(0, \sigma^2) \quad \text{as} \quad R \to \infty, \tag{7}$$

where

$$\bar{g}_R = \frac{\sum_{i=1}^{d} \sum_{i=1}^{R_i} S_{i,j}}{\sum_{i=1}^{d} \sum_{i=1}^{R_i} N_{i,j}}, \tag{8}$$

and a consistent estimate of σ^2 is given by

$$\hat{\sigma}_g^2 = \frac{\frac{1}{R} \sum_{j=1}^{d} \sum_{i=1}^{R_i} (S_{i,j} - N_{i,j} \bar{g}_R)^2}{\bar{N}^2}. \tag{9}$$

Current methods of regenerative MCMC algorithms typically involve first runnning a long sequential chain starting from an arbitrary distribution. Bell variables are then filled in at each iteration of the sequential chain by generating Bernoulli random variables with success probabilities given by equation (3). This process splits the chain into independent chunks, yielding the (N_i, S_i) pairs that we need in order to apply Theorem 3.2.

What we propose instead is generating the bell variables alongside the actual chain. Whenever the bell variable generated is a success, it indicates the completion of a tour. We

can then begin a new tour by *restarting with a draw from* ν. With this approach, we can run each tour on a separate processor, and form $\bar{g}_R$ and $\hat{\sigma}_g^2$ from the resulting independent tours. Although $\bar{g}_R$ is no longer the ergodic mean from a long non-regenerative chain, Proposition 3.1 proves that the individual Markov chains can, in fact, be treated as one long chain. The properties of the estimates of the mean and variance are identical to ones from one long chain.

4 Speed-up from Parallelization

In this section we present pseudo-code to outline the proposed parallel algorithm. We then analyze it by deriving bounds on the speed-up that it yields. Our intention in introducing the bounds is to point out that the speed up we attain from using this algorithm depends on the distribution of the tour lengths. Through an example, we demonstrate that the upper bound can give a very good indication of the possible speed-up.

In Amdahl (1967), a method to analyze the speed-up of an algorithm was introduced. Suppose that an algorithm consists of A float-point operations. Assume that a fraction f of those operations can be sped up to run at V_1 float-point operations per second (flops), and the remaining operations have to remain running at V_2 flops, with $V_2 \ll V_1$. Then the new execution time of the algorithm is

$$t_{fast} = f\frac{A}{V_1} + (1-f)\frac{A}{V_2} > \frac{(1-f)A}{V_2}.$$

Compared with $t_{slow} = A/V_2$, the speed-up afforded by the acceleration is

$$s = \frac{t_{slow}}{t_{fast}} < \frac{1}{1-f}. \tag{10}$$

The above inequality is known as *Amdahl's Law*, and can be used to derive a bound on the maximum speed up of an algorithm. It can be extended to parallelization of a serial algorithm in an analagous way. Suppose that a serial algorithm takes t_1 seconds to execute.

Assume that a fraction f of the algorithm can be executed in parallel on p processors ideally. In other words, this fraction f of the algorithm can be executed on p processors in exactly ft_1/p seconds. Though this is unrealistic, it greatly simplifies the analysis. The speed-up can again be bounded by the same quantity as above:

$$s = \frac{t_1}{ft_1/p + (1-f)t_1} = \frac{p}{f + (1-f)p} < \frac{1}{1-f}.$$

Now consider the serial algorithm that we wish to parallelize:

```
begin program
  read input; initialization;
  for (i=1,. . . ,R)
    generate 1 tour;
  end for;
  evaluate results;
end program
```

It is not clear what fraction f of our code can be parallelized. In situations like this, it would be better to consider an alternative view. In Gustafson (1988), the author considered the inverse question to Amdahl: Given a parallel algorithm, how much faster would it be compared to running it on a serial processor? Put in this way, our algorithm becomes easier to analyze, but it is still complicated by the fact the number of operations in each tour is a random quantity.

Consider the idealized version of our parallel algorithm, and run R tours. Suppose we have $R + 1$ identical processors at our disposal, each of which runs at a speed of V flops. The R processors actually run the tours are usually referred to as the slaves, while the processor controlling them is known as the master. Suppose also that there are no latencies in communicating with the slaves. The following code is run:

Code on master:

```
begin program
```

```
  read input; initialization;
  start slaves (i=1,. . . ,R);
  receive results(1,. . .,R);
  evaluate results;
end program
```

Code on slaves:

```
begin program
  receive parameters from master;
  generate 1 tour;
  send result to master;
end program
```

Let A_i be the random variable representing the number of float-point operations required to complete tour i on processor i. Recall that i runs from 1 to R and crucially, that the A_i are i.i.d. Then the following proposition summarizes the speed up we can expect to attain with our parallel algorithm. The proof is given in Appendix A.1.

Proposition 4.1 (Speed-up from regeneration in parallel)**.** *Suppose we have an i.i.d sample $A_1, A_2, \ldots, A_R$, with $\mathbb{E}[A_i^2] < \infty$ and $A_i \geq 1$ for all i. If we denote $A_{(R)}$ as the maximal order statistic, then the expected speed-up is bounded above and below by*

$$R \times \sup_{\alpha \in [0,1]} \alpha \Pr(A_1 \geq \alpha A_{(R)}) \leq \mathbb{E}[S] \leq R \times \min \left\{ \sqrt{\mathbb{E}[A_i^2]\mathbb{E}\left[\frac{1}{A_{(R)}^2}\right]}, 1 \right\}, \tag{11}$$

where the speed up S is given by

$$S = \frac{\sum_{i=1}^{R} A_i}{A_{(R)}}. \tag{12}$$

Recall that we want to illustrate how the extent of the speed-up depends on the distribution of the tour lengths. We do this with two examples.

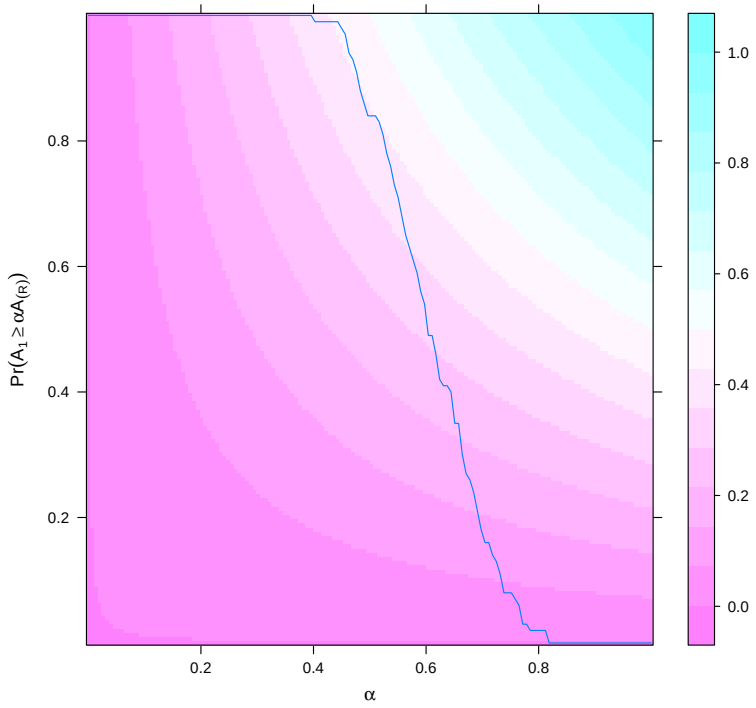

Figure 1: The level-plot is created using the equation $z = xy$, where $x, y \in [0, 1]$. The overlayed curve in blue is $\Pr(A_1 \geq \alpha A_{(R)})$ against α for a Poisson random variable with $\lambda = 50$ and taking $R = 50,000$ tours. 100 samples of 50,000 tours were taken in order to estimate the curve.

As a first example, consider the extreme case where the tours are degenerate random variables. Then $A_i = A_{(R)}$ almost surely and the upper and lower bounds would equal R. In that situation, we would obtain the maximum possible speed-up from the algorithm. On the other hand, suppose we know that the tour lengths are Poisson random variables with $\lambda = 50$, and we wish to run $R = 50,000$ tours. Figure 1 allows us to visualise the lower bound from Proposition 4.1. The closer the curve is to the top right hand corner, the greater the lower bound of the speed up will be. Here, we can estimate the lower bound to be slightly more than 40%.

In this small example, we can in fact empirically compute the true speed-up for each of the 100 realisations of the 50,000 tours. Thus we can estimate the true value of $\mathbb{E}[S]$

and compare it to the value provided by the lower bound. When we do so, we obtain $\mathbb{E}[S] = 60.9\% \times R$. With this simple simulation case, we can also estimate the upper bound from 11 to be $61.3\% \times R$. Thus the upper bound provided by Proposition 4.1 can be very tight. When we have knowledge of the tour distributions, the bounds can be used to decide if it is worth scaling up the number of processors to R.

5 Examples

In this section we look at two examples. The first subsection, which uses a oneway model, illustrates the accuracy of the calculations of the concatenated chains. The second example, which looks at a more general hierarchical model, show how our method works in a practical example.

5.1 Oneway Hierarchical Linear Model

In this section, we assess if running regenerative chains in the manner we purport - by generating tours completely independent of each other and then concatenating them, will provide confidence intervals with the correct coverage. The model we shall consider is the oneway hierarchical linear model, as specified in Hobert & Geyer (1998):

$$y_{ij} = \theta_i + \epsilon_{ij}, \quad i = 1, 2, \ldots, K; \quad j = 1, 2, \ldots m_i, \tag{13}$$

where

$$\begin{aligned} y_{ij}|\boldsymbol{\theta}, \lambda_e &\sim N(\theta_i, \lambda_e^{-1}), \\ \boldsymbol{\theta}|\mu, \lambda_\theta \sim N(\mathbf{1}\mu, \boldsymbol{I}\lambda_\theta^{-1}), &\qquad \lambda_e \sim \text{Gamma}(a_2, b_2), \\ \mu \sim N(\mu_0, \lambda_0^{-1}), &\qquad \lambda_\theta \sim \text{Gamma}(a_1, b_1). \end{aligned}$$

Suppose that $K = 2$, and $m_i = m$ for all i. In this simple case, we can analytically

integrate out μ and λ_e and then numerically obtain (any function of) the posterior means of θ_1, θ_2 and λ_θ. We used the `triplequad` function of MATLAB, which allows computation of a triple integral, but there are many other programs that will do the job too. Let the joint density be

$$h(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e, \boldsymbol{y}) = f(\boldsymbol{y}|\boldsymbol{\theta}, \lambda_e) f(\boldsymbol{\theta}|\mu, \lambda_\theta) f(\lambda_e) f(\mu) f(\lambda_\theta). \tag{14}$$

Then the following simple proposition reduces the expression for the posterior mean for $\boldsymbol{\theta}$ and λ_θ from a 5-dimensional integral to a 3-dimensional one. The proof is given in Appendix A.2

Proposition 5.1.

$$\mathbb{E}(\zeta|\boldsymbol{y}) = \frac{\int \zeta h(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e, \boldsymbol{y}) \mathrm{d}(\boldsymbol{\theta}, \lambda_\theta, \lambda_e, \mu)}{\int h(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e, \boldsymbol{y}) \mathrm{d}(\boldsymbol{\theta}, \lambda_\theta, \lambda_e, \mu)} = \frac{\int \zeta f_1(\lambda_\theta, \boldsymbol{\theta}) f_2(\boldsymbol{\theta}, \boldsymbol{y}) f(\lambda_\theta) \mathrm{d}(\theta_1, \theta_2, \lambda_\theta)}{\int f_1(\lambda_\theta, \boldsymbol{\theta}) f_2(\boldsymbol{\theta}, \boldsymbol{y}) f(\lambda_\theta) \mathrm{d}(\theta_1, \theta_2, \lambda_\theta)}, \tag{15}$$

where $\zeta = \theta_1, \theta_2$ *or* λ_θ*, and*

$$f_1(\boldsymbol{\theta}, \lambda_\theta) = \frac{\lambda_\theta}{(\lambda_0 + 2\lambda_\theta)^{1/2}} \exp\left\{ -\frac{\lambda_\theta}{2}\left(\theta_1^2 + \theta_2^2\right) + \frac{\left(\lambda_0\mu_0 + \lambda_\theta\theta_1 + \lambda_\theta\theta_2\right)^2}{2(\lambda_0 + 2\lambda_\theta)} \right\},$$

$$f_2(\boldsymbol{\theta}, \boldsymbol{y}) = \left(b_2 + \frac{1}{2}\sum_{j=1}^{K}\sum_{i=1}^{m}(y_{ij} - \theta_j)^2 \right)^{-m-a_2}.$$

5.1.1 Minorizing this chain

For this model, the usual way of obtaining samples from the posterior distribution is by running a Gibbs sampler. Suppressing the dependence on $\boldsymbol{y}$, the univariate conditional

densities needed for the Gibbs sampler are

$$\begin{aligned}
\lambda_\theta|\lambda_e,\mu,\boldsymbol{\theta} &\sim \text{Gamma}\left(\frac{K}{2}+a_1, b_1+\frac{1}{2}\sum_i(\theta_i-\mu)^2\right),\\
\lambda_e|\lambda_\theta,\mu,\boldsymbol{\theta} &\sim \text{Gamma}\left(\frac{N}{2}+a_2, b_2+\frac{1}{2}\sum_{i,j}(y_{ij}-\theta_i)^2\right),\\
\mu|\lambda_\theta,\lambda_e,\boldsymbol{\theta} &\sim N\left(\frac{\lambda_0\mu_0+K\lambda_\theta\bar{\boldsymbol{\theta}}}{\lambda_0+K\lambda_\theta}, \frac{1}{\lambda_0+K\lambda_\theta}\right),\\
\theta_i|\lambda_\theta,\lambda_e,\mu,\boldsymbol{\theta}_{-i} &\sim N\left(\frac{\lambda_\theta\mu+m_i\lambda_e\bar{y}_i}{\lambda_\theta+m_i\lambda_e}, \frac{1}{\lambda_\theta+m_i\lambda_e}\right) \quad i=i,\ldots,K.
\end{aligned} \tag{16}$$

It is not necessary to restrict the dimensions of the problem to $K=2$ in order to minorize this chain, so we shall work with the general case. Denote the state of the Gibbs sampler using a vector of length $K+3$. Thus $X_n=\boldsymbol{x}$ means that $\lambda_\theta^n=x_1$, $\lambda_e^n=x_2$, $\mu^n=x_3$ and $\boldsymbol{\theta}^n=(x_4^n,\ldots,x_{K+3}^n)$. Updating λ_0 first, then (λ_e,μ), and finally $\boldsymbol{\theta}$, yields a 3-stage Gibbs sampler. This is a cyclic permutation of the chain that was proven to be geometrically ergodic in Hobert *et al.* (2002). If we use $p_1,\ldots,p_{K+3}$ to represent each of the conditional densities above, we can write the entire Markov transition density as

$$\begin{aligned}
p(\boldsymbol{x},\boldsymbol{w}) =& p_1(w_1|x_2,x_3,\ldots,x_{K+3})\\
& p_2(w_2|w_1,x_3,x_4,\ldots,x_{K+3})\\
& p_3(w_3|w_1,w_2,x_4,\ldots,x_{K+3})\\
& p_4(w_4|w_1,w_2,w_3,x_5,\ldots,x_{K+3})\cdots\\
& p_{K+3}(w_{K+3}|w_1,w_2,w_3,w_4,\ldots,w_{K+2}).
\end{aligned}$$

Following Mykland *et al.* (1995), we shall minorize this Gibbs sampler in the following way. Choosing a distinguished point $\tilde{\boldsymbol{x}}$ and a compact set D, we have

$$p(\boldsymbol{x},\boldsymbol{w}) = \underbrace{\inf_{\boldsymbol{z}\in D}\frac{p(\boldsymbol{x},\boldsymbol{z})}{p(\tilde{\boldsymbol{x}},\boldsymbol{z})}}_{s(x)}\underbrace{p(\tilde{\boldsymbol{x}},\boldsymbol{w})I[\boldsymbol{w}\in D]}_{\nu(dw)}. \tag{17}$$

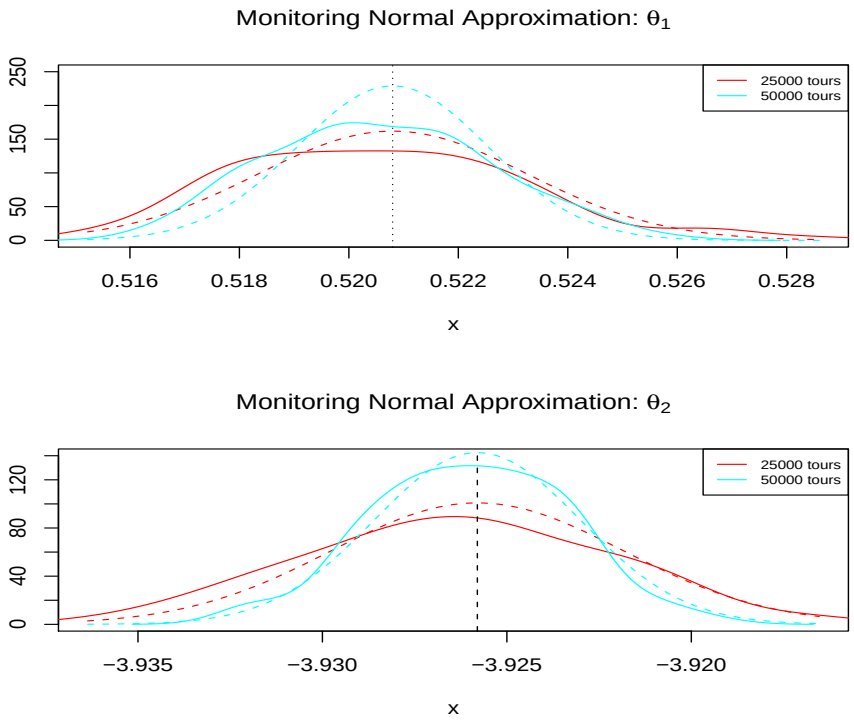

Figure 2: Density plots of the point estimator $\bar{g}_R$ of (8) from each of 100 runs to estimate the values in Table 2. The solid red line is based on the first $25,000$ tours, and the solid cyan line uses all $50,000$ tours. The corresponding dashed lines are the theoretical normal distributions that are being estimated.

The infimum over D can be computed analytically, so as to avoid using a software optimization routine. The details are given in Appendix A.3.

5.1.2 The experiment

In order to assess the performance of the regenerative chains, we shall fix the hyper-parameters as $\lambda_0 = 20.3$, $\mu_0 = 2.19$, $a_1 = 2.1$, $b_1 = 4.3$, $a_2 = 2.1$ and generate a dataset with $m_1 = m_2 = 7$ observations. Using the method outlined in Proposition 5.1, we can numerically compute the posterior means of θ_1, θ_2 and λ_θ. This is the gold standard that the regenerative procedure should pick up and form confidence intervals around.

Figure 2 contains density plots of the point estimator $\bar{g}_R$ from each of 100 runs to estimate

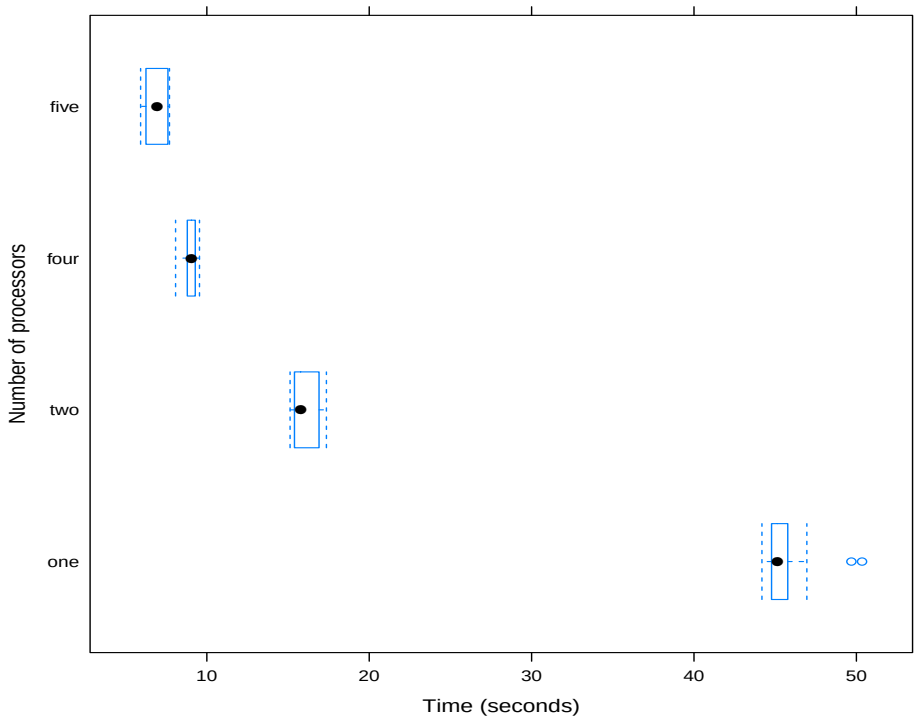

Figure 3: Boxplots of timings of 10,000 tours of the dataset in Figure 2. The 10,000 were run 20 times each on 1, 2, 4 and 5 slaves, shown on the y-axis of the graph. For each run in the 5 slave case, for example, each processor was tasked to run 2,000 tours.

the values in Table 2. All $100 \times 50000 = 5000000$ tours were used to compute an estimate of σ^2, the asymptotic variance in the regenerative CLT. Then taking into consideration only the first 25K tours, the density plot from the $\bar{X}$ of the 100 chains gives the solid red lines in the top 2 panels. The solid cyan line comes from using all 50000 tours in each of the 100 chains. The dashed lines depict the theoretical normal distributions that the $\bar{X}$ should be close to. The figure shows us that as we collect more and tours, the distribution of $\bar{g}_R$ approaches what we expect. Table 3 indicates that the coverage probabilities also approach the nominal value as we collect more tours.

It is also worth looking at the speed-up we gain from splitting this small example over multiple processors. This benefit is summarized in Figure 3. The maximum time taken (over

Table 2: Numerically computed posterior means for test dataset

$\mathbb{E}(\lambda_\theta \mid \boldsymbol{y})$	$\mathbb{E}(\theta_1 \mid \boldsymbol{y})$	$\mathbb{E}(\theta_1 \mid \boldsymbol{y})$
0.1361	0.5208	-3.9258

Table 3: Empirical coverage probabilities for test dataset

	$\mathbb{E}(\lambda_\theta \mid \boldsymbol{y})$	$\mathbb{E}(\theta_1 \mid \boldsymbol{y})$	$\mathbb{E}(\theta_1 \mid \boldsymbol{y})$
After 25,000 tours	0.89	0.93	0.94
After 50,000 tours	0.95	0.91	0.95

the 5 processors) is used to form the boxplot in the top row in the plot. The box-and-whisker plot suggests that the speed up gained as we utilise more processors is very significant.

5.2 Hierarchical Linear Mixed Model

In this section, we demontrate the use of our methodology on the Hierarchical Linear Mixed Model analyzed in Hobert *et al.* (2006), whose block Gibbs sampler has recently been proven to be geometrically ergodic. The minorization introduced there will also be presented and used in the numerical example that follows.

The model is the Bayesian version of the usual frequentist general linear mixed model:

$$\begin{aligned} Y|\beta, u, R, D &\sim N_n(X\beta + Zu, R^{-1}), \\ \beta|u, R, D &\sim N_p(\beta_0, B^{-1}), \\ u|D, R &\sim N_q(0, D^{-1}), \\ R &= \lambda_R I_n \text{ where } \lambda_R \sim Gamma(r_1, r_2), \\ D &= \lambda_D I_q \text{ where } \lambda_D \sim Gamma(d_1, d_2). \end{aligned} \tag{18}$$

The parameters $\beta_0, B, r_1, r_2, d_1$ and d_2 are all assumed to be known, and X is assumed to be of full column rank. If we let $\xi = (u, \beta)$, then the Gibbs chain samples the variables in the

following manner:

$$(\lambda_D, \lambda_R, \xi) \to (\lambda'_D, \lambda'_R, \xi) \to (\lambda'_D, \lambda'_R, \xi').$$

We essentially have a 2-stage Gibbs sampler, as we can see from the conditionals that λ_D and λ_R are independent of each other when given ξ. These are the full conditionals:

$$\begin{aligned}
(\lambda_D|\xi, Y) &\sim Gamma\left(\frac{q}{2} + d_1, d_2 + \frac{1}{2}u^T u\right), \\
(\lambda_R|\xi, Y) &\sim Gamma\left(\frac{n}{2} + r_1, r_2 + \frac{1}{2}(Y - X\beta - Zu)^T(Y - X\beta - Zu)\right), \\
(\xi|R, D, Y) &\sim N_{p+q}(\xi_0, \Sigma^{-1}),
\end{aligned}$$

where the mean and the covariance matrix for ξ are given by

$$\Sigma = \begin{pmatrix} Z^T RZ + D & Z^T RX \\ X^T RZ & X^T RX + B \end{pmatrix} \quad \text{and} \quad \xi_0 = \Sigma^{-1} \begin{pmatrix} Z^T RY \\ X^T RY + B\beta_0 \end{pmatrix}.$$

Choosing a distinguished set for the 2 variables λ_R and λ_D, the Mykland *et al.* (1995) method would lead us to the minorization

$$\begin{aligned}
p((\lambda_D, \lambda_R, \xi), (\lambda'_D, \lambda'_R, \xi')) \geq & \left[\inf_{\lambda_D \in D_1} \frac{\pi(\lambda_D|\xi)}{\pi(\lambda_D|\tilde{\xi})}\right] \left[\inf_{\lambda_R \in D_2} \frac{\pi(\lambda_R|\xi)}{\pi(\lambda_R|\tilde{\xi})}\right] \\
& \times p((\tilde{\lambda_D}, \tilde{\lambda_R}, \tilde{\xi}), (\lambda'_D, \lambda'_R, \xi'))I[\lambda'_D \in D_1]I[\lambda'_R \in D_2].
\end{aligned}$$

The small function can be analytically evaluated, avoiding any numerical optimization routines.

For a numerical example of this model of a realistic size, we shall use a dataset from genetics. In particular, we shall re-use the Carbon Isotope Dataset used in Gonzalez-Martinez *et al.* (2008). The phenotype was Carbon Isotope Discrimination (CID), and the aim of that paper was to use the Quantitative Transmission Disequilibrium Test to pick out SNPs that were associated with the CID, which is a long-used measure of water use efficiency. This dataset was re-analyzed using `BAMD` in Li *et al.* (2011). `BAMD` uses a hierarchical linear mixed

Table 4: Timings for 100 tours using the hierarchical linear mixed model on the CID dataset.

1 processor	9.60 mins
2 processors	4.66 mins
5 processors	1.34 mins

model to account for missing values in the SNPs. Briefly, it is a Gibbs sampler with the missing entries in the Z-matrix considered as parameters, and is available as an `R` package (Gopal *et al.* , 2011).

This dataset consisted of 894 observations from 61 families, and 46 SNPs. The X matrix consisted of columns defining the family from which an observation was obtained. Each column of the Z-matrix represented a SNP, and each entry was one of 3 possible values, reflecting the precise genotype that an individual carried for each particular SNP. Overall, there were about 10% of entries missing in the Z-matrix. For our purpose, we shall simply use the most frequently imputed genotype from the `BAMD` output to "complete" the Z-matrix and regenerate the Hierarchical Linear Mixed Model as our aim is to demonstrate the speed up we can obtain, as the previous sub-section has already highlighted the validity of our method. The average tour length for this dataset was found to be 4.38 and, as can be seen in Table 4, the use of 5 processors can result in a speed-up of 86% over using one processor.

5.3 Implementation in `R`

We have implemented the parallel methodology discussed above in `R`, in the package `McParre` (Mc-Pear-Ree). It is available on CRAN at `http://cran.r-project.org/package=McParre`. The minimal inputs to the algorithm are

1. The one-step regeneration function of the Markov chain. Given the current state, this function should return a draw from the Markov chain.

2. A function to compute the regeneration probability for the bell variable given in equation 3.

In its current incarnation, the package includes parallel regeneration code for the hierarchical oneway model with proper and improper priors (see Hobert & Geyer (1998) and Tan & Hobert (2009)), and the hierarchical linear mixed model presented in Hobert *et al.* (2006). Communication between slaves and master is carried out using the Message Passing Interface (MPI) standard, as specified in Snir *et al.* (1998) and Gropp *et al.* (1998). This is the de facto standard for MIMD architectures that our algorithm is suited for. `McParre` calls the MPI routines it needs through the `R`-package `Rmpi`, which provides an R interface to the MPI libraries.

6 Discussion

We have presented a method to run regenerative Markov chains in parallel. Via Proposition 3.1, we have shown that it is theoretically sound. Through the oneway HLM, we have empirically demonstrated that the parallel version of regenerative chains has the correct coverage probability, and that it can provide a considerable speed-up as processors are increased.

Running regenerative Markov chains in parallel introduces several new operational issues to consider. For example, if we have incomplete tours as a result of having limited time on the cluster, how best can the information in these tours be used? If the processors are of varying speeds, how can we learn the optimal allocation of tours to processors and share this information with minimal message passing? With the cheap proliferation of clusters, we strongly believe that these issues, which are peculiar to parallel and distributed algorithms, need to be considered and investigated.

A Proofs

A.1 Proof of Proposition 4.1

Proof. The first half of the upper bound is direct: Since $A_i \leq A_{(R)}$ for all i, it follows that

$$S = \frac{\sum_{i=1}^{R} A_i}{A_{(R)}} \leq \frac{\sum_{i=1}^{R} A_{(R)}}{A_{(R)}} = R$$

For all i, we know that $A_i \geq 1$. Then it follows that $0 \leq 1/A_{(R)}^2 \leq 1$. Thus we can apply the Cauchy-Schwarz inequality to obtain

$$\mathbb{E}[S] = \sum_{i=1}^{R} \mathbb{E}[A_i/A_{(R)}] \leq \sum_{i=1}^{R} \sqrt{\mathbb{E}[A_i^2]\mathbb{E}\left[\frac{1}{A_{(R)}^2}\right]} = R \times \sqrt{\mathbb{E}[A_1^2]\mathbb{E}\left[\frac{1}{A_{(R)}^2}\right]}$$

For the lower bound, first note that the bivariate random vectors $(A_i, A_{(R)})$ are identically distributed. This can be seen from their joint cdfs. For $b \geq a$ and for any $i \in \{1, 2, \ldots, R\}$

$$\begin{aligned}
\Pr(A_1 \leq a, A_{(R)} \leq b) &= \Pr(A_1 \leq a, \max\{A_2, \ldots, A_R\} \leq b) \\
&= \Pr(A_1 \leq a) \Pr(\max\{A_2, \ldots, A_R\} \leq b) \\
&= \Pr(A_i \leq a) \Pr(\max\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_R\} \leq b) \\
&= \Pr(A_i \leq a, \max\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_R\} \leq b) \\
&= \Pr(A_i \leq a, A_{(R)} \leq b)
\end{aligned}$$

For $b < a$, we have that

$$\begin{aligned}
\Pr(A_1 \leq a, A_{(R)} \leq b) &= \Pr(b < A_1 \leq a, A_{(R)} \leq b) + \Pr(A_1 \leq b, A_{(R)} \leq b) \\
&= 0 + \Pr(A_1 \leq b, A_{(R)} \leq b) \\
&= \Pr(A_i \leq b, A_{(R)} \leq b)
\end{aligned}$$

where the last equality follows from the $b \geq a$ case directly above. Going back to the lower bound, we have

$$\begin{aligned}
\mathbb{E}[S] &= \mathbb{E}\left[\frac{\sum_{i=1}^{R} A_i}{A_{(R)}}\right] \\
&= \sum_{i=1}^{R} \mathbb{E}\left[\frac{A_i}{A_{(R)}}\right] \\
&= R \times \mathbb{E}\left[\frac{A_1}{A_{(R)}}\right] \quad \text{(by above)} \\
&\geq R \times \alpha \Pr\left[\frac{A_1}{A_{(R)}} \geq \alpha\right] \quad \text{(by Markov's inequality)} \\
&\geq R \sup_{\alpha \in [0,1]} \Pr\left[\frac{A_1}{A_{(R)}} \geq \alpha\right]
\end{aligned}$$

□

A.2 Proof of Proposition 5.1

Proof. The first equality holds because the joint posterior density is given by

$$f(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e | \boldsymbol{y}) = \frac{h(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e, \boldsymbol{y})}{\int h(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e, \boldsymbol{y}) \mathrm{d}(\boldsymbol{\theta}, \lambda_\theta, \lambda_e, \mu)}$$

For the second equality, note that in simpifying the integrals, we can drop any terms that do not involve $\boldsymbol{\theta}, \lambda_\theta, \mu$ or λ_e since they will cancel out in the ratio of integrals. Whenever this is done in the sequel, we shall indicate it with a $\doteq$. To get rid of μ, observe that

$$
\begin{aligned}
\int f(\boldsymbol{\theta}|\mu,\lambda_\theta)f(\mu)\mathrm{d}\mu &= \left(\frac{\lambda_\theta}{2\pi}\right)\left(\frac{\lambda_0}{2\pi}\right)^{1/2}\int \exp\left\{-\frac{\lambda_\theta}{2}\left[(\theta_1-\mu)^2+(\theta_2-\mu)^2\right]^2-\frac{\lambda_0}{2}\left(\mu-\mu_0\right)^2\right\}\mathrm{d}\mu \\
&\doteq \lambda_\theta\int \exp\left\{-\frac{\lambda_0}{2}\left(\mu^2-2\mu\mu_0+\mu_0^2\right)-\frac{\lambda_\theta}{2}\left(\theta_1^2-2\mu\theta_1+\mu^2+\theta_2^2-2\mu\theta_2+\mu^2\right)\right\}\mathrm{d}\mu \\
&\doteq \lambda_\theta\int \exp\left\{-\frac{\lambda_0}{2}\left(\mu^2-2\mu\mu_0\right)-\frac{\lambda_\theta}{2}\left(2\mu^2-2\mu(\theta_1+\theta_2)+\theta_2^2+\theta_1^2\right)\right\}\mathrm{d}\mu \\
&= \lambda_\theta\exp\left\{-\frac{\lambda_\theta}{2}\left(\theta_1^2+\theta_2^2\right)\right\}\int \exp\left\{-\frac{1}{2}\left[(\lambda_0+2\lambda_\theta)\mu^2-2\mu(\lambda_0\mu_0+\lambda_\theta\theta_1+\lambda_\theta\theta_2)\right]\right\}\mathrm{d}\mu \\
&= \lambda_\theta\exp\left\{-\frac{\lambda_\theta}{2}\left(\theta_1^2+\theta_2^2\right)\right\}\left(\frac{2\pi}{\lambda_0+2\lambda_\theta}\right)^{1/2}\exp\left\{\frac{\left(\lambda_0\mu_0+\lambda_\theta\theta_1+\lambda_\theta\theta_2\right)^2}{2(\lambda_0+2\lambda_\theta)}\right\} \\
&\doteq \frac{\lambda_\theta}{(\lambda_0+2\lambda_\theta)^{1/2}}\exp\left\{-\frac{\lambda_\theta}{2}\left(\theta_1^2+\theta_2^2\right)+\frac{\left(\lambda_0\mu_0+\lambda_\theta\theta_1+\lambda_\theta\theta_2\right)^2}{2(\lambda_0+2\lambda_\theta)}\right\} \\
&\equiv f_1(\boldsymbol{\theta},\lambda_\theta)
\end{aligned}
$$

To get rid of λ_e, observe that

$$
\begin{aligned}
\int f(\boldsymbol{y}|\boldsymbol{\theta},\lambda_e)f(\lambda_e)\mathrm{d}\lambda_e &= \int\left(\frac{\lambda_e}{2\pi}\right)^m\exp\left\{\frac{\lambda_e}{2}\sum_{j=1}^K\sum_{i=1}^m(y_{ij}-\theta_j)^2\right\}\frac{b_2^{a_2}}{\Gamma(a_2)}\lambda_e^{a_2-1}\exp\left\{-\lambda_e b_2\right\}\mathrm{d}\lambda_e \\
&= \frac{b_2^{a_2}}{(2\pi)^m\Gamma(a_2)}\int\lambda_e^{m+a_2-1}\exp\left\{-\lambda_e\left(b_2+\frac{1}{2}\sum_{j=1}^K\sum_{i=1}^m(y_{ij}-\theta_j)^2\right)\right\}\mathrm{d}\lambda_e \\
&\doteq \frac{\Gamma(m+a_2)}{\left(b_2+\frac{1}{2}\sum_{j=1}^K\sum_{i=1}^m(y_{ij}-\theta_j)^2\right)^{m+a_2}} \\
&\doteq \left(b_2+\frac{1}{2}\sum_{j=1}^K\sum_{i=1}^m(y_{ij}-\theta_j)^2\right)^{-m-a_2} \\
&\equiv f_2(\boldsymbol{\theta},\boldsymbol{y})
\end{aligned}
$$

Substituting the above analytical expressions into $h(\cdot)$ in equations (14) and (15), we have that

$$
\int\zeta h(\boldsymbol{\theta},\mu,\lambda_\theta,\lambda_e,\boldsymbol{y})\mathrm{d}(\boldsymbol{\theta},\lambda_\theta,\lambda_e,\mu) = \int\zeta f_1(\lambda_\theta,\boldsymbol{\theta})f_2(\boldsymbol{\theta},\boldsymbol{y})f(\lambda_\theta)\mathrm{d}(\theta_1,\theta_2,\lambda_\theta) \tag{19}
$$

and

$$\int h(\boldsymbol{\theta}, \mu, \lambda_\theta, \lambda_e, \boldsymbol{y}) \mathrm{d}(\boldsymbol{\theta}, \lambda_\theta, \lambda_e, \mu) = \int f_1(\lambda_\theta, \boldsymbol{\theta}) f_2(\boldsymbol{\theta}, \boldsymbol{y}) f(\lambda_\theta) \mathrm{d}(\theta_1, \theta_2, \lambda_\theta) \tag{20}$$

□

A.3 Details of the Minorization in (17)

Consider $p(\boldsymbol{x}, \boldsymbol{z})/p(\tilde{\boldsymbol{x}}, \boldsymbol{z})$, and define $\bar{x} = (1/K)\sum_{i=1}^{K} x_{i+3}$ and $\bar{\tilde{x}} = (1/K)\sum_{i=1}^{K} \tilde{x}_{i+3}$. Then

$$\begin{aligned}
\frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\tilde{\boldsymbol{x}}, \boldsymbol{z})} =& \frac{\left[b_1 + \frac{1}{2}\sum_i (x_{i+3} - x_3)^2\right]^{K/2+a_1} z_1^{K/2+a_1-1} \exp\left(-z_1\left[b_1 + \frac{1}{2}\sum_i (x_{i+3} - x_3)^2\right]\right)}{\left[b_1 + \frac{1}{2}\sum_i (\tilde{x}_{i+3} - \tilde{x}_3)^2\right]^{K/2+a_1} z_1^{K/2+a_1-1} \exp\left(-z_1\left[b_1 + \frac{1}{2}\sum_i (\tilde{x}_{i+3} - \tilde{x}_3)^2\right]\right)} \\
&\times \frac{\left[b_2 + \frac{1}{2}\sum_{i,j} (y_{ij} - x_{i+3})^2\right]^{N/2+a_2} z_2^{N/2+a_2-1} \exp\left(-z_2\left[b_2 + \frac{1}{2}\sum_{i,j} (y_{ij} - x_{i+3})^2\right]\right)}{\left[b_2 + \frac{1}{2}\sum_{i,j} (y_{ij} - \tilde{x}_{i+3})^2\right]^{N/2+a_2} z_2^{N/2+a_2-1} \exp\left(-z_2\left[b_2 + \frac{1}{2}\sum_{i,j} (y_{ij} - \tilde{x}_{i+3})^2\right]\right)} \\
&\times \frac{\left(\frac{\lambda_0 + Kz_1}{2\pi}\right)^{1/2} \exp\left(-\frac{\lambda_0 + Kz_1}{2}\left[z_3 - \frac{\lambda_0\mu_0 + Kz_1\bar{x}}{\lambda_0 + Kz_1}\right]^2\right)}{\left(\frac{\lambda_0 + Kz_1}{2\pi}\right)^{1/2} \exp\left(-\frac{\lambda_0 + Kz_1}{2}\left[z_3 - \frac{\lambda_0\mu_0 + Kz_1\bar{\tilde{x}}}{\lambda_0 + Kz_1}\right]^2\right)}.
\end{aligned}$$

Minimizing this is equivalent to minimizing $\log(p(\boldsymbol{x}, \boldsymbol{z})/p(\tilde{\boldsymbol{x}}, \boldsymbol{z}))$, and

$$\begin{aligned}
\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\tilde{\boldsymbol{x}}, \boldsymbol{z})} = \text{Constant} \; & - z_1 \underbrace{\frac{1}{2}\sum_{i=1}^{K}\left[(x_{i+3} - x_3)^2 - (\tilde{x}_{i+3} - \tilde{x}_3)^2\right]}_{C_1} \\
& - z_2 \underbrace{\frac{1}{2}\sum_{i,j}\left[(y_{ij} - x_{i+3})^2 - (y_{ij} - \tilde{x}_{i+3})^2\right]}_{C_2} \\
& - \left(\frac{\lambda_0 + Kz_1}{2}\right)\left[\left(z_3 - \frac{\lambda_0\mu_0 + Kz_1\bar{x}}{\lambda_0 + Kz_1}\right)^2 - \left(z_3 - \frac{\lambda_0\mu_0 + Kz_1\bar{\tilde{x}}}{\lambda_0 + Kz_1}\right)^2\right].
\end{aligned}$$

The final expression in the above display can be simplified to yield

$$
\begin{aligned}
&-\left(\frac{\lambda_0 + Kz_1}{2}\right)\left(\frac{\lambda_0\mu_0 + Kz_1\bar{\tilde{x}} - \lambda_0\mu_0 - Kz_1\bar{x}}{\lambda_0 + Kz_1}\right)\left(2z_3 - \frac{2\lambda_0\mu_0 + Kz_1(\bar{\tilde{x}} + \bar{x})}{\lambda_0 + Kz_1}\right) \\
&= -\frac{Kz_1(\bar{\tilde{x}} - \bar{x})}{2}\left(2z_3 - \frac{2\lambda_0\mu_0 + Kz_1(\bar{\tilde{x}} + \bar{x})}{\lambda_0 + Kz_1}\right).
\end{aligned}
$$

Hence we can define the function

$$
g(z_1, z_2, z_3) = -z_1C_1 - z_2C_2 - \frac{Kz_1(\bar{\tilde{x}} - \bar{x})}{2}\left(2z_3 - \frac{2\lambda_0\mu_0 + Kz_1(\bar{\tilde{x}} + \bar{x})}{\lambda_0 + Kz_1}\right), \tag{21}
$$

and attempt to find (z_1^*, z_2^*, z_3^*) that minimizes g over

$$
D = [d_{11}, d_{12}] \times [d_{21}, d_{22}] \times [d_{31}, d_{32}].
$$

From (21), the partial derivatives are

$$
\begin{aligned}
\frac{\partial g}{\partial z_1} &= -C_1 - K(\bar{\tilde{x}} - \bar{x})z_3 + \frac{1}{2}K(\bar{\tilde{x}} - \bar{x})\left[\frac{2\lambda_0\mu_0 + Kz_1(\bar{\tilde{x}} - \bar{x})}{\lambda_0 + Kz_1}\right], \\
&\quad + \frac{1}{2}K(\bar{\tilde{x}} - \bar{x})z_1\left[\frac{K\lambda_0(\bar{\tilde{x}} + \bar{x}) - 2\lambda_0\mu_0K}{(\lambda_0 + Kz_1)^2}\right] \\
\frac{\partial g}{\partial z_2} &= -C_2, \\
\frac{\partial g}{\partial z_3} &= -Kz_1(\bar{\tilde{x}} - \bar{x}).
\end{aligned}
$$

Since $z_1 > 0$, we can set z_2^* and z_3^* once we know the signs of $\partial g/\partial z_2$ and $\partial g/\partial z_3$ above. For z_1^*, we can set $\partial g/\partial z_1 = 0$ and solve for the roots. Here are the coefficients of the quadratic equation in z_1:

$$
\begin{aligned}
\text{Coeff of } z_1^2 &= \frac{1}{2}K^2\left(-2C_1 - K(\bar{\tilde{x}} - \bar{x})(2z_3 - \bar{\tilde{x}} - \bar{x})\right), \\
\text{Coeff of } z_1 &= 2K\lambda_0(-C_1 - Kz_3(\bar{\tilde{x}} - \bar{x})) + K(\bar{\tilde{x}} - \bar{x})(K\lambda_0(\bar{\tilde{x}} + \bar{x})), \\
\text{Constant term} &= \lambda_0^2\left[-C_1 - K(\bar{\tilde{x}} - \bar{x})z_3 + K\mu_0(\bar{\tilde{x}} - \bar{x}))\right].
\end{aligned}
$$

References

Amdahl, G.M. 1967. Validity of the single processor approach to achieving large scale computing capabilities. *Pages 483–485 of: Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM.

Athreya, K, & Ney, P. 1978. A new approach to the limit theory of recurrent Markov chains. *Trans Amer. Math. Soc.*, **245**, 493–501.

Brockwell, AE. 2006. Parallel Markov chain Monte Carlo simulation by pre-fetching. *Journal of Computational and Graphical Statistics*, **15**(1), 246–261.

Chan, K.S., & Geyer, C.J. 1994. Comment on "Markov chains for exploring posterior distributions". *Annals of Statistics*, **22**(4), 1701–1728.

Fishman, G.S. 2001. *Discrete-event simulation: modeling, programming, and analysis*. Springer Verlag.

Flegal, J.M., & Jones, G.L. 2010. Batch means and spectral variance estimators in Markov chain Monte Carlo. *Annals of Statistics*, **38**(2), 1034–1070.

Flegal, J.M., Haran, M., & Jones, G.L. 2008. Markov chain Monte Carlo: Can we trust the third significant figure. *Statistical Science*, **23**(2), 250–260.

Flynn, M.J. 1972. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, **100**(9), 948–960.

Gonzalez-Martinez, SC, Huber, D., Ersoz, E., Davis, JM, & Neale, DB. 2008. Association genetics in Pinus taeda L. II: Carbon isotope discrimination. *Heredity*, **101**(1), 19–26.

Gopal, V., Li, Z., & Casella, G. 2011. *BAMD: Bayesian Association Model for Genomic Data with Missing Covariates*. R package version 3.5.

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., & Snir, M. 1998. *MPI: the complete reference (Vol. 2).*

Gustafson, J.L. 1988. Reevaluating Amdahl's law. *Communications of the ACM*, **31**(5), 532–533.

Hobert, J., Jones, G., Presnell, B., & Rosenthal, J. 2002. On the applicability of regenerative simulation in MCMC. *Biometrika*, **89**, 731–743.

Hobert, J.P., & Geyer, C.J. 1998. Geometric ergodicity of Gibbs and block Gibbs samplers for a hierarchical random effects model. *Journal of Multivariate Analysis*, **67**(2), 414–430.

Hobert, J.P., Jones, G.L., & Robert, C.P. 2006. Using a Markov chain to construct a tractable approximation of an intractable probability distribution. *Scandinavian Journal of Statistics*, **33**(1), 37–51.

Jones, G.L., Haran, M., Caffo, B.S., & Neath, R. 2006. Fixed-width output analysis for Markov chain Monte Carlo. *Journal of the American Statistical Association*, **101**(476), 1537–1547.

Kontoghiorghes, E.J. 2006. *Handbook of parallel computing and statistics.* CRC Press.

Li, Z., Gopal, V., Li, X., Davis, J.M., & Casella, G. 2011. Simultaneous SNP Identification in Association Studies with Missing Data. *Annals of Applied Statistics (to appear).*

Matloff, N. 2011. *Programming on parallel machines.* IEEE.

Mykland, P., Tierney, L., & Yu, B. 1995. Regeneration in Markov Chain Samplers. *Journal of American Statistical Association*, **90**, 233–241.

Nummelin, E. 1978. A Splitting Technique for Harris recurrent chains. *Zeit. Warsch. Verv. Gebiete*, **43**, 309–318.

Nummelin, E. 2004. *General irreducible Markov chains and non-negative operators.* Cambridge Univ Press.

Robert, C.P., & Casella, G. 2004. *Monte Carlo Statistical Methods*. Springer.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., & Dongarra, J. 1998. *MPI: The Complete Reference (Vol. 1)*.

Strid, I. 2010. Efficient parallelisation of Metropolis-Hastings algorithms using a prefetching approach. *Computational Statistics & Data Analysis*, **54**(11), 2814–2835.

Suchard, M.A., Wang, Q., Chan, C., Frelinger, J., Cron, A., & West, M. 2010. Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures. *Journal of Computational and Graphical Statistics*, **19**(2), 419–438.

Tan, A., & Hobert, J.P. 2009. Block Gibbs sampling for Bayesian random effects models with improper priors: Convergence and regeneration. *Journal of Computational and Graphical Statistics*, **18**(4), 861–878.

Zhou, H., Lange, K., & Suchard, M.A. 2010. Graphics Processing Units and High-Dimensional Optimization. *Statistical Science*, **25**(3), 311–324.