NEW ALGORITHMS FOR PARALLEL MCMC

GEOFF NICHOLLS

Abstract. The core paper for this project is the paper [3]. This parallelizes MCMC annealing. We will parallelize MCMC sampling using the same idea. We will look at an application in population genetics.

1. Introduction

Readers familiar with MCMC and core ideas of population genetics should simply read the paper [3] and consider how it might be applied in an MCMC sampling context. The idea is unknown in statistics.

This project will look at new schemes for parallelizing a single MCMC chain. Markov Chain Monte Carlo remains an important tool for fitting complex stochastic models to data. Despite its known weaknesses (convergence issues) MCMC is the generic method of last resort for statistical inference for complex models. As the problems we want to solve, and the computing environment within which we work, change, algorithms need to adapt. On the one hand there are many problems in which a single likelihood evaluation takes a significant fraction of a second. This is a long time for an MCMC algorithm which may need millions of likelihood evaluations to provide a representative sample. On the other hand we have now the ready availability of multi-core processors, with fast inter-processor communication at relatively high bandwidth.

The parallel algorithms we consider have the following distinctive properties. First they are very simple to implement. A simple parallel 'wrapper' of a few lines, on an existing piece of code is often enough to parallelize the code. Secondly, the simplest versions of these algorithms make relatively heavy use of inter-processor communications. This makes sense if communication time is small compared to on-processor computation time. Thirdly, They parallelize a single serial Markov Chain. It is trivial to parallelize MCMC by running multiple chains. However, it is often desirable to get 'serial depth' in a single MCMC chain, in order to avoid wasting CPU time on burn-in in many chains.

Consider the following problem of Bayesian inference. We have data $y \sim L(\theta; y)$ (where L is a likelihood, thought of as a distribution over y and a function over θ), a prior $\pi(\theta)$ for the unknown parameter $\theta \in \Omega$ and a posterior distribution $\pi(\theta|y) \propto \pi(\theta) f(\theta; y)$. We dont want a point estimate for θ (!) we want a credible set. One way to do that is using monte carlo samples distributed according to $\pi(\theta|y)$. MCMC is an algorithm for simulating such samples.

We assume a good knowledge of basic MCMC methods. For elementary MCMC see my Part A lecture notes on simulation. For a more advanced presentation see [1] and [8].

Key words and phrases. Parallel MCMC speculative computation.
Allen Rodrigo for finding related literature.

Here is Metropolis Hastings MCMC with proposal distribution q and acceptance probability α. Given an initialization $\Theta_0 = \theta_0$ this realizes a chain $\Theta_t = \theta_t, t = 0, 1, 2, 3, ..., T$ (the capitals are the random variables, the lower case are the realized values). Under certain conditions relating q and π, the chain is geometrically ergodic, so we have a CLT, and we can form an estimate of the posterior expectation $\mu_g = \mathbb{E}g(\Theta)|y$ of any function $g(\theta)$, or the posterior probability $\mathbb{E}\mathbb{I}_{\Theta\in S}|y$ of any set $S \subseteq \Omega$ of interest, via

$$T^{-1}\sum_{t=1}^{T} g(\Theta_t) \stackrel{D}{\rightarrow} N(\mu_g, \sigma_g^2/T),$$

for some constant σ_g^2.

Algorithm 1 (MCMC). At state $\Theta_t = \theta$, simulate Θ_{t+1} as follows:

(1) Simulate $\theta' \sim q(\theta, \cdot)$.
(2) With probability

$$\alpha(\theta, \theta') = \min\left\{1, \frac{\pi(\theta'|y)}{\pi(\theta|y)}\right\}$$

set $\Theta_{t+1} = \theta'$ and otherwise set $\Theta_{t+1} = \theta$.

Now the problem with this algorithm is that we have to evaluate $\pi(\theta'|y)/\pi(\theta|y)$. It is often the case that $\pi(\theta')/\pi(\theta)$ is simple to compute. It is common that $L(\theta'; y)/L(\theta; y)$ is time-consuming and in that case the following parallel algorithm may be helpful.

Suppose we K processors.

Algorithm 2 (parallel MCMC). If $\Theta_t = \theta$ is the last simulated state, the values of $\Theta_{t+1}, ..., \Theta_{t+k}$, with $k \in \{1, 2, ..., K\}$ random, are determined as follows:

(1) For $j = 1, 2, ..., K$, simulate iid $\theta'_j \sim q(\theta, \cdot)$.
(2) For $j = 1, 2, ..., K$, and with probability

$$\alpha_j = \min\left\{1, \frac{\pi(\theta'_j|y)}{\pi(\theta|y)}\right\}$$

set $B_j = 1$ and otherwise set $B_j = 0$.
(3) If $B_j = 0$ for all $j = 1, 2, ..., K$, then set $\Theta_{t+1} = \theta, ..., \Theta_{t+K} = \theta$ and $t \leftarrow t + K$.
If $B_1 = 1$ then set $\Theta_{t+1} = \theta'$ and $t \leftarrow t + 1$.
Otherwise, let $k = \min\{j; B_j = 1, j \in \{1, 2, ..., K\}\}$ and set $\Theta_{t+1} = \theta', ..., \Theta_{t+k-1} = \theta$ and $\Theta_{t+k} = \theta'$ and $t \leftarrow t + k$.

The computation of $\alpha_1 ... \alpha_k$ is done in parallel on the K processors. Computation on processors at $j > k$ is 'thrown away': we count the rejections out to the first acceptance, then take the state of the first acceptance, and no more.

Note that having a random number of updates ending on an acceptance might create a bias if we run the chain for a fixed number of calls to the K processors. Instead we should fix the run length N, and run to the smallest random time greater than N, then throw away any samples beyond the target run length.

One feature of this algorithm is that it may become more efficient if we deliberately lower the acceptance rate of the MCMC chain. We can do this by proposing more widely distributed candidates θ' at the first step (taking $q(\theta, \theta')$ with heavier

tails at large $|\theta' - \theta|$). This is because the mean number of steps the chain will take at each call to the K processors will be increased (it is related to the inverse of the mean acceptance probability) but the mixing rate of the chain (its statistical efficiency) is usually only reduced very slightly, or may even increase, as the acceptance rate is decreased. [7] have shown how to calculate approximately optimal acceptance rates for the standard MCMC chain, and it would be good to extend that result to parallel chains of the kind above.

The approach can be thought of as implementing the MCMC as a random walk down a binary tree. Think of this as a tree drawn on the page in front of you, with the root at the top and branches left and right at each node. The current state θ is fed into the root node. Each node j receives an input state θ_j, and proposes a candidate state θ'_j. The input state is emitted as the input state to the left child and the candidate state is emitted as the input state to the right child. In this way each node on the tree gets an input state and a candidate state. We compute at each node the acceptance probability α_j in parallel at each tree node. We can now run down the tree, branching left at node j (reject candidate) with probability $1 - \alpha_j$ and right with probability α_j. Each time we take a step down the tree we add the emitted state on the chosen child branch to the end of the Markov Chain we are simulating. In the algorithm we have described, this tree is skewed towards the left: the K nodes computed in parallel are all the rejection nodes. If there is an acceptance, the walk branches to the right, hits a leaf, and stops. Other authors have proposed algorithms in which the tree is balanced. This will usually be less efficient, as the K processors give a balanced tree of depth $\log_2(K)$, typically much smaller than the expected number of updates in our skewed tree.

If you look at the paper from the mid nineties on parallel speculative annealing you will see that they have considered different tree shapes, depending on the acceptance probability. They are trying to find the optimal balance of the tree, skewed towards acceptances or rejections, as a function of the mean acceptance probability. The problem for sampling (as opposed to annealing, and optimization) is similar. However, it is complicated by the possibility that we may increase the efficiency of the chain by lowering its acceptance rate.

Some R-code is available illustrating the parallel algorithm. There are some advantages to using R to explore this algorithm. The parallelization is straightforward. If students have experience parallelizing other languages (especially Matlab) they may wish to work in their favorite environment. However, we have R parallelization 'up and running' and so it is a good fall-back.

Example Application

The example application is a simple form of the problem treated in [2]. There are some relevant comments in chapter 8 of [6]. Parallel methods are needed to help scale to problem sizes of current interest. To some extent we could choose any fairly hard, time consuming MCMC sampling application to illustrate and test the parallel methods, and if students have personal preferences they are welcome to follow them.

For a given single sex population of fixed size let L sample individuals be sequenced yielding L aligned sample DNA sequences $y_1, y_2, ..., y_L$ each of length N. The bases $y_{i,j} \in \{A, C, G, T\}$ at site $j \in \{1, 2, ..., N\}$ are homologous for $i \in \{1, 2, ..., L\}$.

Let N_e be the effective population size in a Wright-Fisher model for ancestry, and μ be the substitution rate (base substitutions per generation per site) in the DNA sequence data. Let time be measured in units of substitutions. The lineages ancestral to two individuals coalesce independently at instantaneous rate $\theta = \mu N_e$. This determines a probability distribution $\pi(g|\theta)$ over rooted trees with branch lengths in units of substitutions. See chapter 26 of [4] for details.

If Q is a rate matrix (units are inverse substitutions) for base substitution and $p = (p_A, p_C, p_G, p_T)$ is the equilibrium base frequency (so that $pQ = 0$) then the likelihood $f(y|g) = \prod_{j=1}^{L} f(y_{:,j}|g)$ for a tree g is given in terms of the EBF p and a 4×4 transition probability matrix function $P_{a,b}(t) = [\exp(Qt)]_{a,b}, a, b \in \{A, C, G, T\}$. Given that the base $x_j(0)$ at site j at time 0 is $x_j(0) = a$, the probability to find that it has changed to $x_j(t) = b$ after a time t is $[\exp(Qt)]_{a,b}$, where $\exp(Qt)$ is a matrix exponential. This likelihood may be computed efficiently using an algorithm known as pruning. See chapter 13 of [4].

As an example application consider the problem of inferring genealogy g and scaled population size θ in a Bayesian framework with θ-prior $\pi(\theta)$. The posterior for g, θ is

$$\pi(g, \theta|y) \propto f(y|g)\pi(g|\theta)\pi(\theta).$$

We can sample this distribution using MCMC. There are two approaches.

If we are interested mainly in the posterior for θ, we can target

$$\pi(\theta|y) \propto f(y|\theta)\pi(\theta).$$

Here $f(y|\theta) = \int f(y|g)\pi(g|\theta)dg$ is a marginal likelihood. Essentially, we average $f(y|g)$ over trees g in the coalescent distribution $\pi(g|\theta)$. This is a hard estimation problem in its own right. In this approach we update θ in the MCMC. At each update $\theta \to \theta'$ we must compute $f(y|\theta')$ and $f(y|\theta)$ in order to compute the Hastings ratio. If we make good progress we may be able to return to this approach to θ-estimation.

Our first approach will simple MCMC on the joint tree and θ distribution. At each step of the chain we either vary θ (use for example a simple random walk), or vary the tree. To vary the tree we can (for example) take an edge (and hence a sub-tree) at random, disconnect it, and reconnect at a new location chosen randomly over the tree. We may also wish to scale the overall tree depth jointly or independently with μ. In order to do this we need to have a data structure for a tree, with branch lengths. We need to be able to perform the above operations on the tree, and to compute the likelihood $f(y|g)$ and coalescent probability density $\pi(g|\theta)$ rapidly.

Road map

I assume programming is in R and that just two-core parallel laptops are available. It may be that larger board sizes are available. However, this gives a baseline.

Before coming: (1) References on basic MCMC, Metropolis Hastings. Effective sample size and output analysis for MCMC. Simplest form of parallel algorithm [5]. Find out about simple models for parallel computation (especially effects due to bandwidth and communication-time). (2) The coalescent. Finite sites sequence models. Posterior distribution $\pi(g, \theta|y)$ for genealogy g and effective population size Θ given sequence data y.

Week 1: Implement MH MCMC for $\pi(g, \theta|y)$ using pruning to evaluate tree likelihood.
(1) Read a nexus file of sequence data.
(2) Simulate the posterior distribution for g and θ.
(3) Plot consensus tree for g and make a histogram for θ.

Week 2: Add a parallel wrapper to the code from Week 1. Look at how the runtime scales with the number of processors, and how this scaling depends on problem parameters (with L and N).

Week 3-4: Consider how the effective sample size per CPU second depends on mean acceptance probability. Look at the papers of [7] and [3] and consider how over-dispersed proposals may lower acceptance rates but improve parallel MCMC efficiency. Consider how parallel MCMC tree shapes for speculative execution depend on acceptance probabilities.

Weeks 5-6: Consider the other approach to inference for θ in which the marginal likelihood is estimated. This should give a maroe dramatic parallel speedup, as each MCMC step is potentially more time consuming.

References

[1] Steve Brooks, Andrew Gelman, Galin L. Jones, and Xiao-Li Meng. *Handbook of MCMC*. Chapman and Hall/CRC, 2011.
[2] R Durbin, S. Eddy, A Krogh, and G Mitchison. *Biological sequence analysis*. Cambridge, 1998/2004.
[3] J Felsenstein. *Inferring Phylogenies*. Sinauer, 2004.
[4] DRUMMOND A. J., NICHOLLS G. K., A. G. RODRIGO, and W. SOLOMON. Estimating mutation parameters, population history and genealogy simultaneously from temporally spaced sequence data. *Genetics*, 161:1307–1320, 2002.
[5] Geoff Nicholls. Code for parallelization. `http://www.stats.ox.ac.uk/~nicholls/linkfiles/papers/basicexample.R`.
[6] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics, 2004.
[7] Gareth O. Roberts and Jeffrey S. Rosenthal. Optimal scaling of discrete approximations to langevin diffusions. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 1998.
[8] Ellen E. Witte, Roger D. Chamberlain, and Mark A. Franklin. Parallel simulated annealing using speculative computation. *IEEE TRANSACTlONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 2:483, 1991.

Geoff Nicholls, Department of Statistics, 1 South Parks Road, Oxford, OX1 3TG, UK,
E-mail address: `nicholls@stats.ox.ac.uk`