Accelerating MCMC with Parallel Predictive Prefetching

Accelerating MCMC via Parallel Predictive Prefetching

Elaine Angelino, Eddie Kohler, Amos Waterland, Margo Seltzer and Ryan P. Adams
School of Engineering and Applied Sciences
Harvard University
{elaine,margo}@eecs.harvard.edu {kohler,apw,rpa}@seas.harvard.edu

Abstract

Parallel predictive prefetching is a new framework for accelerating a large class of widely-used Markov chain Monte Carlo (MCMC) algorithms. It speculatively evaluates many potential steps of an MCMC chain in parallel while exploiting fast, iterative approximations to the target density. This can accelerate sampling from target distributions in Bayesian inference problems. Our approach takes advantage of whatever parallel resources are available, but produces results *exactly equivalent* to standard serial execution. In the initial burn-in phase of chain evaluation, we achieve speedup close to linear in the number of available cores.

1 INTRODUCTION

Probabilistic modeling is one of the mainstays of modern machine learning, and Bayesian methods are particularly appealing due to their ability to represent uncertainty in parameter estimates and latent variables. Unfortunately, Bayesian inference can be difficult in the real world. Many problems are not amenable to exact inference, and so require approximate inference in the form of Monte Carlo estimates or variational approximations. These procedures require many evaluations of a target posterior density, and each evaluation can be expensive, especially on large data sets. Our work accelerates Markov chain Monte Carlo (MCMC) but, in contrast to other recent proposals, we arrive at a method in which the stationary distribution is *exactly* the target posterior. This method exploits approximations to the target density to speculatively evaluate many potential future steps of the chain in parallel.

The increasing availability of multi-core machines, and many-core cluster deployments, led to our focus on parallelism. Unfortunately, the execution of MCMC algorithms such as Metropolis-Hastings (MH) is inherently serial. One can run many independent chains at once, but this does not change the mixing time for any single chain. Since mixing time can be prohibitively large, especially when the target function is high-dimensional and multi-modal, this embarrassingly parallel approach tends not to reduce the time to achieve a useful estimator. Sometimes the target function evaluation can be parallelized, or multiple chains in an ensemble method can be run in parallel, but these strategies are not available in general.

We instead use speculative execution to parallelize a large class of MCMC methods. This approach, sometimes called *prefetching*, has received some attention in the past decade, but does not seem to be widely recognized. Speculative execution is the general technique of optimistically performing computational work that might be eventually useful. To understand speculative execution in the context of MCMC, consider the MH algorithm in Algorithm 1, in which each iteration consists of a proposal that is stochastically accepted or rejected (Metropolis et al., 1953). MH uses randomness in two ways: to generate uniform random variables and to generate proposals. Given a random stream and an initial state, all possible future states of the chain can be thought of as the nodes of a binary tree (Figure 1). Serial execution of MH yields a sequence of states that maps to a single path of nodes through the tree. Starting at the root, each transition stochastically chooses between the current state (left child) and the proposal (right child). This requires evaluating the target density at the root and each subsequent proposal. The main goal of prefetching is to perform these evaluations in parallel. However, only the immediate transition that compares the root of the tree to the first proposal is known *a priori* to be on the true computational path. Prefetching schemes use parallel cores to evaluate these two nodes and also speculatively evaluate additional nodes further down the tree.

An effective prefetching implementation must overcome several challenges. Some involve correctness; for example, care is required in the treatment of pseudo-randomness lest bias be introduced (*i.e.*, each node's source of randomness must produce exactly the same results as it would in a serial

Algorithm 1 Metropolis-Hastings

```
Input: initial state $\theta_0$, number of iterations $T$, target $\pi(\theta)$, proposal $q(\theta' \mid \theta)$
Output: samples $\theta_1, \dots, \theta_T$
for $t = 0, \dots, T-1$ do
    $\theta' \sim q(\theta' \mid \theta_t)$
    $u \sim \mathrm{Unif}(0,1)$
    if $\frac{\pi(\theta')q(\theta_t \mid \theta')}{\pi(\theta_t)q(\theta' \mid \theta_t)} > u$ then
        $\theta_{t+1} = \theta'$
    else
        $\theta_{t+1} = \theta_t$
    end if
end for
```

execution). But the key challenge is performance. A naïve scheduling scheme always requires $\approx 2^s$ parallel cores to achieve a speedup of s. Less naïve schemes improve on this speedup using the average proposal acceptance rate: if most proposals are rejected, a prefetching implementation should prefetch more heavily among the right children of the left-most branch, *i.e.*, the path representing a sequence of rejected proposals. Although in practice the optimal acceptance rate is less than 0.5 (Gelman et al., 1996), tiny acceptance rates, which lead to good speedup, cause less effective mixing. If the acceptance rate is set near the 0.234 value of Gelman et al., speedup is still at most logarithmic.

We evaluate a new scheduling approach that uses local information to improve speedup relative to other prefetching schemes. We adaptively adjust speculation based not only on the local average proposal acceptance rate (which changes as evaluation progresses), but also on the actual random deviate used at each state. Even better, we make use of any available fast approximations to the transition operator. Though these approximations are not required, when they are available or learnable, we leverage them to make better scheduling decisions.

We present results using a series of increasingly expensive but more accurate approximations. These decisions are further improved by modeling the error of these approximations, and thus the uncertainty of the scheduling decisions. Performance depends critically on how we model the approximations, and a key insight is in our error model for this setting; much smaller error, and therefore more precise prediction, is obtained by modeling the error of the *difference* between two proposal evaluations, rather than evaluating the errors of the proposals separately. Our current implementation uses approximations that correspond to incremental evaluation of the target distribution, but our framework does not require this. We could use other examples of target density approximation, including exploiting closed form approximations such as Taylor series (Christen and Fox, 2005) and fitting linear or Gaussian Process regressions (Conrad et al., 2014).

Motivated by large-scale Bayesian inference, we present results using incremental approximations that arise from evaluating a subset of factors in a larger product. As we show on inference problems using both real and synthetic data, our system takes advantage of parallelism to speed up the wall-clock time of serial Markov chain evaluation. Unlike prior systems, we achieve near-linear speedup during burn-in on up to 64 cores spread across two or more machines. As evaluation progresses, speedup eventually decreases to logarithmic in the number of cores; we show why this is hard to avoid.

2 RELATED WORK

In this section, we summarize existing parallel strategies for accelerating MCMC, motivated by the computational cost of MCMC. This cost is most often determined by evaluation of the target density relative to mixing. In Metropolis–Hastings, it is incurred when the target is evaluated to determine the acceptance ratio of a proposed move; in slice sampling (Neal, 2003) an expensive target slows both bracket expansion and contraction. We focus on the increasingly common case where the target is expensive and the dominant computational cost. This evaluation can sometimes be parallelized directly, *e.g.*, when the target function is a product of many individually expensive terms. This can arise in Bayesian inference if the target easily decomposes into one likelihood term for each data item. Practically achievable speedup in this setting is limited by the communication and computational costs associated with aggregating the partial evaluations. In general, the target function cannot be parallelized; we divide methods that accelerate MCMC via other sources of parallelism into two classes: ensemble sampling and prefetching.

Other work speeds up MCMC evaluation using approximation. Stochastic variational inference techniques achieve scalable approximate inference via randomized approximations of gradients (Hoffman et al., 2013), while recent developments in MCMC have implemented efficient transition operators that lead to approximate stationary distributions (Welling and Teh, 2011; Korattikara et al., 2014; Bardenet et al., 2014). Recent other work uses a lower bound on the local likelihood factor to simulate from the exact posterior distribution while evaluating only a subset of the data at each iteration (Maclaurin and Adams, 2014). Unlike such prior work, we speed up *exact* evaluation of many existing MCMC algorithms.

2.1 ENSEMBLE SAMPLERS

Ensemble (or *population*) methods for sampling run multiple chains and accelerate mixing by sharing information between the chains. The individual chains can be simu-

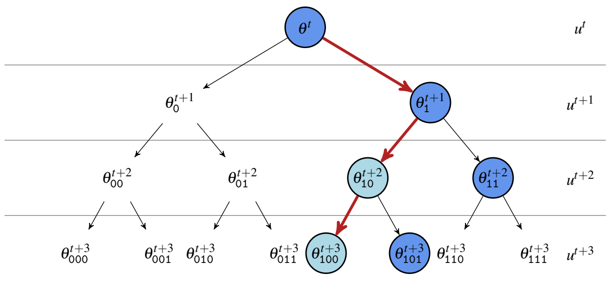


Figure 1: Schematic of a MH simulation superimposed on the binary tree of all possible chains. Each level of the tree represents an iteration, where branching to the right/left indicates accepting/rejecting a proposal. Random variates (on right) are shared across each layer. Thick red arrows highlight one simulated chain starting at the root θ^t; the first proposal is accepted and the next two are rejected, yielding as output: $\theta_1^{t+1}, \theta_{10}^{t+2}, \theta_{100}^{t+3}$. Dark filled circles indicate states where the target density is evaluated during simulation. Those not on the chain's path correspond to rejected proposals. Their siblings are pale filled circles on this path; since each is a copy of its parent, the target density does not need to be reevaluated to compute the next transition.

lated in parallel; any information sharing between chains requires communication. Examples include parallel tempering (Swendsen and Wang, 1986), the emcee implementation (Foreman-Mackey et al., 2012) of affine-invariant ensemble sampling (Goodman and Weare, 2010), and a parallel implementation of generalized elliptical slice sampling (Nishihara et al., 2014).

2.2 PREFETCHING

The second class of parallel MCMC algorithms uses parallelism through speculative execution to accelerate individual chains. This idea is called *prefetching* in some of the literature. To the best of our knowledge, prefetching has only been studied in the context of MH-style algorithms where, at each iteration, a single new proposal is drawn from a proposal distribution and stochastically accepted or rejected. The typical body of an MH implementation is a loop containing a single conditional statement and two associated branches. One can then view the possible execution paths as a binary tree, as illustrated in Figure 1. The vanilla version of prefetching speculatively evaluates all paths in this binary tree (Brockwell, 2006). The correct path will be exactly one of these, so with J cores, this approach achieves a speedup of $\log_2 J$ with respect to single core execution, ignoring communication and bookkeeping overheads.

Naïve prefetching can be improved by observing that the two branches are not taken with equal probability. On average, the left-most branch, corresponding to a sequence of rejected proposals, tends to be more probable; the classic result for the optimal MH acceptance rate is 0.234 (Roberts et al., 1997), so most prefetching scheduling policies have been built around the expectation of rejection. Let $\alpha \leq 0.5$ be the expected probability of accepting a proposal. Byrd et al. (2008) introduced a procedure, called "speculative moves," that speculatively evaluates only along the "reject" branch of the binary tree; in Figure 1, this corresponds to the left-most branch. In each round of their algorithm, only the first k out of $J-1$ extra cores perform useful work, where k is the number of rejected proposals before the first accepted proposal, starting from the root of the tree. The expected speedup is then:

$$1+\mathbb{E}(k) < 1+\sum_{k=0}^{\infty} k(1-\alpha)^k\alpha < 1+\frac{1-\alpha}{\alpha} = \frac{1}{\alpha}.$$

Note that the first term on the left is due to the core at the root of the tree, which always performs useful computation in the prefetching scheme. When $\alpha = 0.23$, this scheme yields a maximum expected speedup of about 4.3; it achieves an expected speedup of about 4 with 16 cores. If only a few cores are available, this may be a reasonable policy, but if many cores are available, their work is essentially wasted. In contrast, the naïve prefetching policy achieves speedup that grows as the log of the number of cores. Byrd et al. (2010) later considered the special case where the evaluation of the likelihood function occurs on two timescales, slow and fast. They call this method "speculative chains"; it modifies "speculative moves" so that whenever the evaluation of the likelihood function is slow, any available cores are used to speculatively evaluate the subsequent chain, assuming the slow step resulted in an accept.

In work closely related to ours, Strid (2010) extend the naïve prefetching scheme to allocate cores according to the optimal "tree shape" with respect to various assumptions about the probability of rejecting a proposal, *i.e.*, by greedily allocating cores to nodes that maximize the depth of speculative computation expected to be correct (Strid, 2010). Their static prefetching scheme assumes a fixed acceptance rate; versions of this were proposed earlier in the context of simulated annealing (Witte et al., 1991). Their dynamic scheme estimates acceptance probabilities, *e.g.*, at each level of the tree by drawing empirical MH samples (100,000 in the evaluation), or at each branch in the tree by computing $\min\{\beta, r\}$ where β is a constant ($\beta = 1$ in the evaluation) and r is an estimate of the MH ratio based on a fast approximation to the target function. Alternatively, Strid proposes using the approximate target function to identify the single most likely path on which to perform speculative computation. Strid also combines prefetching with other sources of parallelism to obtain a multiplicative effect. To the best of our knowledge, these methods have been developed for MH algorithms and evaluated on up to 64 cores, although usually many fewer.

3 PREDICTIVE PREFETCHING

We propose *predictive prefetching*, an improved scheduling approach that accelerates exact MCMC. Like Strid's dynamic prefetching procedure, we also exploit inexpensive but approximate target evaluations. However, there are several fundamental differences between our approach and existing prefetching methods. We combine approximate target evaluation with the fact that the random stream used by a MCMC algorithm can be generated in advance and thus incorporated into the estimates of the acceptance probabilities at each branch in the binary tree. Critically, we also model the error of the target density approximation, and thus the uncertainty of whether a proposal will be accepted. In addition, we identify a broad class of MCMC algorithms that could benefit from prefetching, not just MH, and we show how prefetching can exploit a series of approximations, not just a single one.

3.1 MATHEMATICAL SETUP

Consider a transition operator $T(\theta \to \theta')$ which has $\pi(\theta)$ as its stationary distribution on state space Θ. Simulation of such an operator typically proceeds using an "external" source of pseudo-random numbers that can, without loss of generality, be assumed to be drawn uniformly on the unit hypercube $\mathscr{U}$. The transition operator is then a deterministic function $T : \Theta \times \mathscr{U} \to \Theta$. Most practical transition operators – Metropolis–Hastings, slice sampling, *etc.* – are actually compositions of two such functions, however. The first function produces a countable set of candidate points in Θ, here denoted $Q : \Theta \times \mathscr{U}_Q \to \mathscr{P}(\Theta)$, where $\mathscr{P}(\Theta)$ is the power set of Θ. The second function $R : \mathscr{P}(\Theta) \times \mathscr{U}_R \to \Theta$ then chooses one of the candidates for the next state in the Markov chain. Here we have used $\mathscr{U}_Q$ and $\mathscr{U}_R$ to indicate the orthogonal parts of $\mathscr{U}$ relevant to each part of the operator. In this setup, the basic Metropolis–Hastings algorithm uses $Q(\cdot)$ to produce a tuple of the current point and a proposed point, while multiple-try MH (Liu et al., 2000) and delayed-rejection MH (Tierney and Mira, 1999; Green and Mira, 2001) create a larger set that includes the current point. In the exponential-shrinkage variant of slice sampling (Neal, 2003), $Q(\cdot)$ produces an infinite sequence of candidates that converges to, but does not include, the current point.

This setup is a somewhat more elaborate treatment than usual, but this is intended to serve two purposes: 1) make it clear that there is a separation between generating a set of possible candidates via $Q(\cdot)$ and selecting among them with $R(\cdot)$, and 2) highlight that both of these functions are deterministic functions, given the pseudo-random variates. Others have pointed out this "deterministic given the randomness" view, and used it to construct alternative approaches to MCMC (Propp and Wilson, 1996; Neal, 2012).

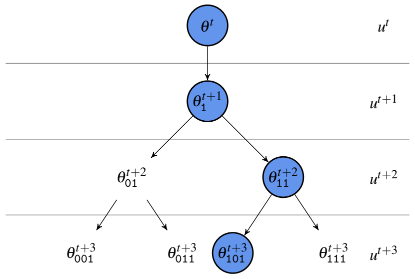


Figure 2: Schematic of the same MH simulation as in Figure 1, this time superimposed on the jobtree. This tree includes only those nodes in the original MH tree where a new state is introduced and thus the target density must be evaluated when comparing such a state to another. The filled circles, corresponding to states where the target density is evaluated in a serial MH execution, are now directly connected by a single path.

We separately consider $Q(\cdot)$ and $R(\cdot)$ because it is generally the case that $Q(\cdot)$ is inexpensive to evaluate and does not require computation of the target density $\pi(\theta)$, while $R(\cdot)$ must compare the target density at the candidate locations and so represents the bulk of the computational burden. Parallel predictive prefetching observes that, since $Q(\cdot)$ is cheap and the pseudo-random variates can be produced in any order, the tree of possible future states of the Markov chain can be constructed before any of the $R(\cdot)$ functions are evaluated, as in Figure 1. The sequence of $R(\cdot)$ evaluations simply chooses a path down this tree. Parallelism can be achieved by speculatively choosing to evaluate $R(\{\theta_i\}, u)$ for some part of the tree that has not yet been reached. If this node in the tree is eventually reached, then we achieve a speedup.

For clarity, we henceforth focus on the straightforward random-walk Metropolis–Hastings operator. In this special case, $Q(\cdot)$ produces a tuple of the current point and a proposal, and the function $R : \Theta \times \Theta \times (0,1) \to \Theta$ takes these two points, along with a uniform random variate u in $(0,1)$, and selects one of the two inputs via:

$$R(\theta, \theta', u) = \begin{cases} \theta' & \text{if } u \frac{q(\theta' \mid \theta)}{q(\theta \mid \theta')} < \frac{\pi(\theta')}{\pi(\theta)} \\ \theta & \text{otherwise} \end{cases}, \tag{1}$$

where $q(\cdot \mid \cdot)$ is the proposal density corresponding to $Q(\cdot)$. We write the acceptance ratio in this somewhat unusual fashion to highlight the fact that the left-hand side of the inequality does not require evaluation of the target density and is easy to precompute.

3.2 THE JOBTREE

While the MH state tree in Figure 1 effectively represents a simulated chain as a path, it yields an awkward representation of the computation necessary to produce a chain. Specifically, transitions to right children (when a proposal is accepted) align with this path but transitions to left children (when a proposal is rejected) branch off it. For our prefetching framework, we wanted a better representation of this computation. To this end, we introduced the Metropolis–Hastings *jobtree*, depicted in Figure 2. It contains the same information as the MH state tree but represents only those states where new computation occurs, *i.e.*, where the target density must be evaluated in order to compare such a state to another. Like the original tree, the jobtree is generally binary, except that the root has only one child. It includes the root node and all right children of the MH state tree, corresponding to the current state and all possible subsequent proposals – together, these specify the possible distinct states and at what iteration each would first appear. Paths on the jobtree represent computation in the sense that they map to sequences of states where the target density is evaluated during serial MH simulation; we call any such path a *computation path*.

3.3 EXPLOITING PREDICTIONS

Consider a prefetching framework with J cores that uses one core to compute the immediate transition and the others to precompute transitions for possible future iterations. If each precomputation falls along the actual Markov chain, the framework will achieve the ideal linear speedup proportional to J. If some of them do not fall along the chain, the framework will fail to scale perfectly with the available resources. For instance, recall that the naïve framework that evaluates transitions based on breadth-first search of the prefetching state tree (Figure 1) will achieve speedup proportional to $\log_2 J$. Good speedup thus depends on making good predictions of what path will be taken on the tree, which is in turn determined by our prediction of whether the threshold will be exceeded in Eq. 1.

Let ρ denote a node on the tree, θ_ρ indicate the current state at ρ, and θ'_ρ indicate the proposal. We define

$$r_\rho = u_\rho \frac{q(\theta'_\rho \,|\, \theta_\rho)}{q(\theta_\rho \,|\, \theta'_\rho)} \tag{2}$$

where u_ρ is the MH threshold variate for node ρ. The Markov chain's steps are determined by iterations of computing the indicator function $\iota_\rho = \mathbb{I}(r_\rho < \pi(\theta'_\rho)/\pi(\theta_\rho))$, where a proposal is accepted iff $\iota_\rho = 1$. The quantities θ_ρ, θ'_ρ, and r_ρ can be inexpensively computed at any time from the stream of pseudo-random numbers, without examining the expensive target $\pi(\cdot)$. The random variate u_ρ depends only on the depth (iteration) of ρ.

The precomputation schedule should maximize expected speedup, which corresponds to the expected number of precomputations along the true computation path in the jobtree. To maximize this quantity, the framework needs to anticipate which branches of the jobtree are likely to be taken. The root node and its only child are always evaluated. We associate with each remaining node ρ in the jobtree a predictor ψ_ρ that models the probability that the proposal is accepted, given approximate or partial information. For example, suppose that $\tilde{\pi}(\cdot)$ is an approximation to the target density $\pi(\cdot)$, and assume that in log space, the error of this approximation relative to the target density is normally distributed with some variance σ^2. Then, we could write the predictor as:

$$\psi_\rho = \Pr\left(\log r_\rho < \log \pi(\theta'_\rho) - \log \pi(\theta_\rho) \;\middle|\; \tilde{\pi}(\cdot), \sigma^2\right) \tag{3}$$

$$= \int_{\log r_\rho}^{\infty} \mathcal{N}\left(z \;\middle|\; \log \tilde{\pi}(\theta'_\rho) - \log \tilde{\pi}(\theta_\rho), \sigma^2\right) dz. \tag{4}$$

As more information becomes available in the form of better approximators, the predictor ψ_ρ will change. When $\tilde{\pi}(\cdot) = \pi(\cdot)$, the predictor equals the indicator ι_ρ. We label the edges in the jobtree with *branch probabilities*: the edge from a node ρ to its right child has branch probability equal to the predictor ψ_ρ and the edge to its left child has branch probability $1 - \psi_\rho$. Assuming that the predictions at each node are independent, the probability that a node's computation is used is the product of the branch probabilities along the path connecting the root to ρ; we call this quantity the node's *expected utility*. Those nodes with maximum expected utility should be scheduled for precomputation. (The immediate transition will always be chosen: it has no ancestors and utility 1.)

A predictor is always available – for instance, one can use the recent acceptance probability – but many problems can improve predictions using computation. To model this, we define a sequence of predictors

$$\psi_\rho^{(m)} \approx \psi_\rho, \quad m = 0, 1, 2, \ldots, N, \tag{5}$$

where increasing m implies increasing accuracy, and $\psi_\rho^{(N)} = \iota_\rho$. Workers move through this sequence until they perform the exact computation. The predictor sequence affects scheduling decisions: once it becomes sufficiently certain that a worker's branch will not be taken, that worker and its descendants should be reallocated to more promising branches. Ultimately, every true step of the Markov chain is computed to completion. The approach simulates from the true stationary distribution, not an approximation thereof. The estimators are used only in prefetching.

There are several schemes for producing this estimator sequence, and predictive prefetching applies to any Markov chain Monte Carlo problem for which approximations are available. We focus on the important case where improved estimators are obtained by including more and more of the data in the posterior target distribution.

3.4 LARGE-SCALE BAYESIAN INFERENCE

In Bayesian inference with MCMC, the target density is a (possibly unnormalized) posterior distribution. In most modeling problems, such as those using graphical models, the target density can be decomposed into a product of terms. If the data are conditionally independent given the model parameters, there is a factor for each of the N data:

$$\pi(\theta \,|\, \mathbf{x}) \propto \pi_0(\theta)\, \pi(\mathbf{x} \,|\, \theta) = \pi_0(\theta) \prod_{n=1}^{N} \pi(x_n \,|\, \theta)\,. \tag{6}$$

Here $\pi_0(\theta)$ is a prior distribution and $\pi(x_n \,|\, \theta)$ is the likelihood term associated with the nth datum. The logarithm of the target distribution is a sum of terms

$$\mathscr{L}(\theta) = \log \pi(\theta \,|\, \mathbf{x}) = \log \pi_0(\theta) + \sum_{n=1}^{N} \log \pi(x_n \,|\, \theta) + c\,,$$

where c is an unknown constant that does not depend on θ and can be ignored. Our predictive prefetching algorithm uses this to form predictors ψ_ρ as in Eq. 3; we again reframe ψ_ρ using log probabilities as

$$\psi_\rho \approx \Pr\left(\log r_\rho < \mathscr{L}(\theta') - \mathscr{L}(\theta)\right), \tag{7}$$

where r_ρ is the precomputed random MH threshold of Eq. 2. One approach to forming this predictor is to use a normal model for each $\mathscr{L}(\theta)$, as in Korattikara et al. (2014). However, we can achieve a better estimator with lower variance by modeling $\mathscr{L}(\theta)$ and $\mathscr{L}(\theta')$ together, rather than separately. Expanding each log likelihood gives:

$$\mathscr{L}(\theta') - \mathscr{L}(\theta) = \log \pi_0(\theta') - \log \pi_0(\theta) + \sum_{n=1}^{N} \Delta_n \tag{8}$$

$$\Delta_n = \log \pi(x_n \,|\, \theta') - \log \pi(x_n \,|\, \theta)\,. \tag{9}$$

In Bayesian posterior sampling, the proposal θ' is usually a perturbation of θ and so we expect $\log \pi(x_n \,|\, \theta')$ to be correlated with $\log \pi(x_n \,|\, \theta)$. In this case, the differences Δ_n occur on a smaller scale and have a smaller variance compared to the variance due to $\log \pi(x_n \,|\, \theta)$ across data terms.

A concrete sequence of estimators is obtained by subsampling the data. Let $\{\Delta_n\}_{n=1}^{m}$ be a subsample of size $m < N$, without replacement, from $\{\Delta_n\}_{n=1}^{N}$. This subsample can be used to construct an unbiased estimate of $\mathscr{L}(\theta') - \mathscr{L}(\theta)$. We model the terms of this subsample as i.i.d. from a normal distribution with bounded variance σ^2, leading to:

$$\mathscr{L}(\theta') - \mathscr{L}(\theta) \sim \mathcal{N}(\hat{\mu}_m, \hat{\sigma}_m^2)\,. \tag{10}$$

The mean estimate $\hat{\mu}_m$ is empirically computable:

$$\hat{\mu}_m = \log \pi_0(\theta') - \log \pi_0(\theta) + \frac{N}{m} \sum_{n=1}^{m} \Delta_n\,. \tag{11}$$

The error estimate $\hat{\sigma}_m$ may be derived from $s_m/\sqrt{m}$, where s_m is the empirical standard deviation of the m subsampled Δ_n terms. To obtain a confidence interval for the sum of N terms, we multiply this estimate by N and the finite population correction $\sqrt{(N-m)/N}$, giving:

$$\hat{\sigma}_m = s_m \sqrt{\frac{N(N-m)}{m}}\,. \tag{12}$$

We can now form the predictor $\psi_\rho^{(m)}$ by considering the tail probability for $\log r_\rho$:

$$\psi_\rho^{(m)} = \int_{\log r_\rho}^{\infty} \mathcal{N}(z \,|\, \hat{\mu}_m, \hat{\sigma}_m^2)\, \mathrm{d}z \tag{13}$$

$$= \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{\log \hat{\mu}_m - \log r_\rho}{\sqrt{2}\hat{\sigma}_m}\right)\right]. \tag{14}$$

3.5 SYSTEM

Our system is fully parallel and runs on network clusters of computers, each of which may comprise multiple cores. We do not perform any affinity scheduling, so all cores are treated identically whether they co-reside on the same machine or not. Our system does not use shared memory; rather, cores communicate via message passing. Note that we assign one thread to each core. To date, the largest installation on which we have run is a shared cluster of 5 machines with a total of 160 cores, on which we have used in parallel at least 64 cores spanning a minimum of 2 machines.

Our system executes predictive prefetching as follows. A master node manages the jobtree and distributes a different node in the jobtree to each worker. When a worker receives a message to compute on node ρ, it first computes the corresponding proposal θ_ρ (which may consume values from the random sequence). It asynchronously transmits the proposal and the new point in the random sequence back to the master. It then starts evaluating the target density, producing progressively improved approximations to the target that it periodically reports back to the master. Meanwhile, the master uses estimates of $\mathscr{L}(\theta'_\rho) - \mathscr{L}(\theta_\rho)$ values, the appropriate r_ρ constants, and an adaptive estimate of the current acceptance probability to calculate the predictor $\psi_\rho^{(m)}$ for each node in the evaluation tree. To assign a worker to a node, the master stochastically traverses down the jobtree from the root, following branches according to their branch probabilities, until it finds a node that is inactive, *i.e.*, no other worker is currently working on it. In this way, the master stochastically assigns workers to those nodes with highest expected utility. During computation, expected utilities change. When the master notices that the expected utility of a worker's node falls below that of other inactive nodes, the master tells the worker to abandon its work. If the abandoned proposal becomes likely again, a worker will pick it up where the earlier worker left off.

	Burn-in					
J	$i_1 = 9575$		$i_2 = 24000$		$i_3 = 50000$	
1	16674	—	41978	—	87500	—
16	2730	6.1×	8678	4.3×	20318	4.3×
32	1731	9.6×	7539	5.6×	19046	4.6×
64	989	16.8×	5894	7.1×	15146	5.8×

Table 1: Cumulative time (in seconds) and speedup for evaluating the Gaussian mixture model with different numbers of workers J.

In our implementation, the target posteriors $\log \pi(\theta \mid \mathbf{x})$ and $\log \pi(\theta' \mid \mathbf{x})$ are evaluated by separate workers. Our normal model for the MH ratio based on a subsample of size m depends on the empirical mean and standard deviation of the differences Δ_n, but we use an approximation to avoid the extra communication required to keep track of all these differences. The worker for θ calculates

$$G_m(\theta) = \log \pi_0(\theta) + \frac{N}{m} \sum_{n=1}^{m} \log \pi(x_n \mid \theta) \qquad (15)$$

rather than the difference mean $\hat{\mu}_m$ from Eq. 11. The master can then compute $\hat{\mu}_m = G_m(\theta') - G_m(\theta)$, but the empirical standard deviation of differences, s_m in Eq. 12, must be estimated. We set

$$s_m = \sqrt{S_m(\theta)^2 + S_m(\theta')^2 - 2\tilde{c}S_m(\theta)S_m(\theta')}\,, \qquad (16)$$

where $S_m(\theta)$ denotes the empirical standard deviation of the m $\log \pi(x_n \mid \theta)$ terms, and $\tilde{c}$ approximates the correlation between $\log \pi(x_n \mid \theta)$ and $\log \pi(x_n \mid \theta')$. We empirically observe this correlation to be very high; in all experiments we set $\tilde{c} = 0.9999$. Note that this approximation only affects the quality of our speculative predictions; it does not affect the actual decision to accept or reject the proposal θ'.

Our implementation requires at least two cores, one master and one worker. Note that when there is only one worker, it is always performing useful computation for the immediate transition at the root, leaving the master with essentially nothing to do besides some bookkeeping.

4 EXPERIMENTS

Our evaluation focuses on MH for large-scale Bayesian inference using the approximations described above (though our framework can use any approximation scheme for the target distribution). Our implementation is written in C++ and Python, and uses MPI for communication between the master and worker cores.[1] We evaluate our implementation on up to 64 cores in a multicore cluster environment in which machines are connected by 10GB ethernet and each machine has 32 cores (four 8-core Intel Xeon E7-8837 processors). We report speedups relative to serial computation with one worker.

[1]`https://github.com/elaine84/fetching`

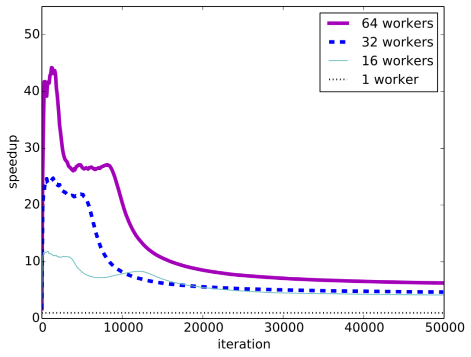


Figure 3: Cumulative speedup relative to our baseline, as a function of the number of MH iterations, for the mixture of Gaussians problem. The different curves correspond to different numbers of workers.

We evaluate our system on both synthetic and real Bayesian inference problems. First, we consider the posterior density of the eight-component mixture of eight-dimensional Gaussians used by Nishihara et al. (2014), where the likelihood involves 10^6 samples drawn from this model. Next, we consider the posterior density of a Bayesian Lasso regression (Park and Casella, 2008) that models molecular photovoltaic activity. The likelihood involves a dataset of 1.8×10^6 molecules described by 56-dimensional real-valued cheminformatic features (Olivares-Amaya et al., 2011; Amador-Bedolla et al., 2013); each response is real-valued and corresponds to a lengthy density functional theory calculation (Hachmann et al., 2011, 2014).

In our experiments, we use a spherical Gaussian for the proposal distribution. A simple adaptive scheme sets the scale of this distribution, improving convergence relative to standard MH. Our approach falls under the provably convergent adaptive algorithms studied by Andrieu and Moulines (2006); we easily incorporated them into our framework.

We expect predictive prefetching to perform best when the densities at a proposal and corresponding current point are significantly different, which is common in the initial burn-in phase of chain evaluation. In this phase, early estimates based on small subsamples effectively predict whether the proposal is accepted or rejected. When the density at the proposal is very close to that at the current point – for example, as the proposal distribution approaches the target distribution – the outcome is inherently difficult to predict; early estimates will be uncertain or even wrong. Incorrect estimates could destroy speedup (no precomputations would be useful). We hope to do better than this worst case, and to at least achieve logarithmic speedup.

	mean	standard deviation	min	max
n_{eff}	3405	7253	50	26000
$\hat{R}$	1.005	0.006	1.000	1.020

Table 2: Convergence statistics after burn-in (over iterations i_2–i_3) for the Gaussian mixture model, computed over the 64 dimensions of the model.

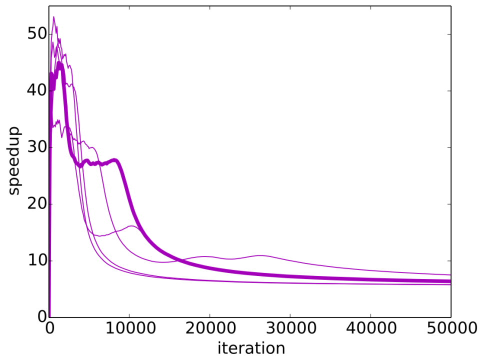
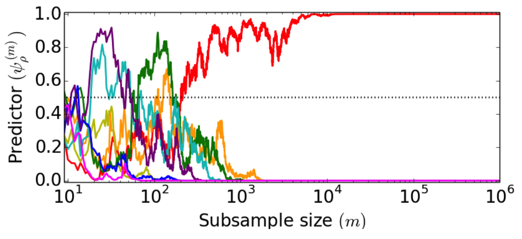


Figure 4: Cumulative speedup relative to our baseline, as a function of the number of MH iterations, for the mixture of Gaussians problem. The different curves correspond to different initial conditions; all curves are for 64 workers.

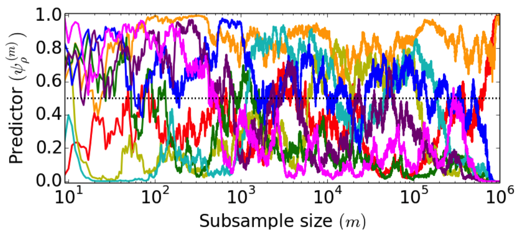
In our experiments, we divide the evaluation of the target function into 100 batches. Thus, for the mixture problem, each subsample contains 10^4 data items.

Table 1 shows the results for the Gaussian mixture model. We run the model with the same initial conditions and pseudorandom sequences with varying numbers of worker threads. All experiments produce identical chains. We evaluate the cumulative time and speedup obtained at three different iteration counts. The first, $i_1 = 9575$ iterations, are burn-in. After i_1 iterations, all dimensions of samples achieve the Gelman-Rubin statistic $\hat{R} < 1.05$, computed using two independent chains, where the first $i_1/2$ samples have been discarded (Gelman and Rubin, 1992). We then run the model further to i_3 iterations. Iterations $i_2 = 24000$ through $i_3 = 50000$ are used to compute an effective number of samples n_{eff}. (Table 2 shows convergence statistics after i_3 iterations.) The results are as we hoped. The initial burn-in phase obtains better-than-logarithmic speedup (though not perfect linear speedup). With 64 workers, the chain achieves burn-in 16.8× faster than with one worker. After burn-in, efficiency drops as expected, but we still achieve logarithmic speedup (rather than sub-logarithmic). At 50000 iterations, speedup for each number of workers J rounds to $\log_2 J$.

Figure 3 explains these results by graphing cumulative speedup over the whole range of iterations. The initial speedup is close to linear – we briefly achieve more than 40× speedup at $J = 64$ workers. As burn-in proceeds, cumulative speedup falls off to logarithmic in J. Figure 4 shows cumulative speedup for the Gaussian mixture model with several different initial conditions. We see a range of variation due to differences in the adaptive scheme during burn-in. The overall pattern is stable, however: good speedup during burn-in followed by logarithmic speedup later. Also note that speedup does not necessarily decrease steadily, or even monotonically. At some initial conditions, the chain enters an easier-to-predict region before truly burning in; while in such a region, speedup is maintained. Our system takes advantage of these regions effectively.



(a) Burn-in



(b) Convergence

Figure 5: Example predictor trajectories for the mixture of Gaussians. We show the predictor $\psi_\rho^{(m)}$ as a function of subsample size m. Different colors indicate different proposals. Burn-in is much easier to predict than convergence.

Figure 5 shows how our predictors behave both during and after burn-in. During burn-in, estimates are effective, and the predictor converges quite quickly to the correct indicator. After burn-in, the new proposal's target density is close to the old proposal's, and the estimates are similarly hard to distinguish. Sometimes the random variate r_ρ is small enough for the predictor to converge quickly to 1; more often, the predictor varies widely over time, and does not converge to 0 or 1 until almost all data are evaluated. This behavior makes logarithmic speedup a best case. Luckily, the predictor is more typically uncertain (with an intermediate value) than wrong (with an extreme value that eventually flips to the opposite value): incorrect predictors could lead to sublogarithmic speedup.

Figure 6 shows that good speedups are achievable for real problems. The speedup distribution for the Bayesian Lasso problem for molecular photovoltaic activity appears similar

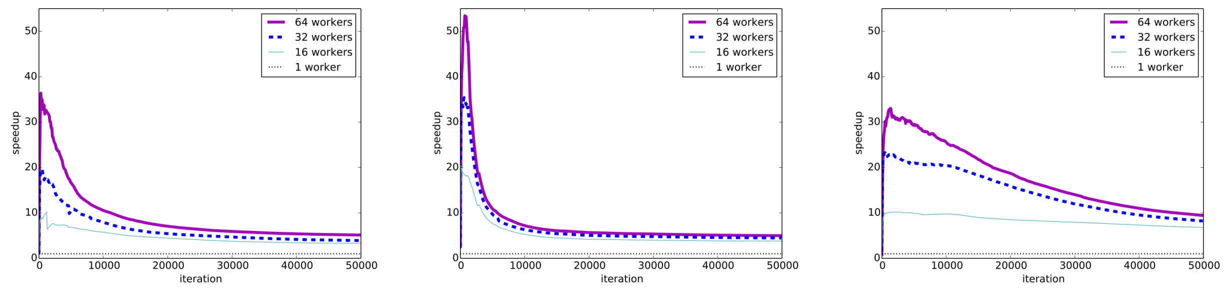


Figure 6: Cumulative speedup relative to our baseline, as a function of the number of MH iterations, for the Bayesian Lasso problem. Different curves indicate different numbers of workers. Each figure corresponds to a different initial condition.

to that of the mixture of Gaussians. There are differences, however: Lasso evaluation did not converge by 50000 iterations according to standard convergence statistics. On several initial conditions, the chain started taking small steps, and therefore dropped to logarithmic speedup, before achieving convergence. Overall performance might be improved by detecting this case and switching some speculative resources over to other initial conditions, an idea we leave for future work.

5 CONCLUSIONS

We presented parallel predictive prefetching, a general framework for accelerating many widely used MCMC algorithms that are inherently serial and often slow to converge. Our approach applies to MCMC algorithms whose transition operator can be decomposed into two functions, one that produces a countable set of candidate proposal states and a second that selects the best candidate. Predictive prefetching uses speculative computation to exploit the common setting in which (1) generating candidates is computationally fast compared to the evaluation required to select the best candidate, and (2) this evaluation can be approximated quickly. Our first focus has been on the MH algorithm, in which predictive prefetching exploits a sequence of increasingly accurate predictors for the decision to accept or reject a proposed state. Our second focus has been on large-scale Bayesian inference, for which we identified an effective predictive model that estimates the likelihood from a subset of data. The key insight is that we model the uncertainty of these predictions with respect to the difference between the likelihood of each datum evaluated at the proposal and current state. As these evaluations are highly correlated, the variance of the differences is much smaller than the variance of the states evaluated separately, leading to significantly higher confidence in our predictions. This allows us to justify more aggressive use of parallel resources, leading to greater speedup with respect to serial execution or more naïve prefetching schemes.

The best speedup that is realistically achievable for this problem is sublinear in the number of cores but better than logarithmic, and our results achieve this. Our approach generalizes both to schemes that learn an approximation to the target density and to other MCMC algorithms with more complex structure, such as slice sampling and more sophisticated adaptive techniques.

Acknowledgments

We thank M.P. Brenner, E.D. Cubuk, V. Kanade, Z. Liu, D. Maclaurin, A.C. Miller and R. Nishihara for helpful discussions, Aspuru-Guzik, J. Hachmann and R. Olivares-Amaya for the use of the Clean Energy Project dataset and introduction to the cheminformatic feature set, and M. Tingley for the derived features used here. This work was partially funded by DARPA Young Faculty Award N66001-12-1-4219, the National Institutes of Health under Award Number 1R01LM010213-01, a Microsoft Research New Faculty Fellowship award, and Google.

REFERENCES

C. Amador-Bedolla, R. Olivares-Amaya, J. Hachmann, and A. Aspuru-Guzik. Towards materials informatics for organic photovoltaics. In K. Rajan, editor, *Informatics for Materials Science and Engineering*. Elsevier, Amsterdam, 2013.

C. Andrieu and E. Moulines. On the ergodicity properties of some adaptive MCMC algorithms. *The Annals of Applied Probability*, 16(3):1462–1505, 2006.

R. Bardenet, A. Doucet, and C. Holmes. Towards scaling up Markov chain Monte Carlo: An adaptive subsampling approach. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

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

J. M. R. Byrd, S. A. Jarvis, and A. H. Bhalerao. Reducing the run-time of MCMC programs by multithreading on SMP architectures. In *International Symposium on Parallel and Distributed Processing*, pages 1–8, 2008.

J. M. R. Byrd, S. A. Jarvis, and A. H. Bhalerao. On the parallelisation of MCMC by speculative chain execution. In *IPDPS Workshops*, pages 1–8, 2010.

J. A. Christen and C. Fox. Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical Statistics*, 14(4):795–810, 2005.

P. R. Conrad, Y. M. Marzouk, N. S. Pillai, and A. Smith. Asymptotically exact MCMC algorithms via local approximations of computationally intensive models. *ArXiv e-prints*, Feb. 2014.

D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. emcee: The MCMC Hammer. *Publications of the Astronomical Society of the Pacific*, 125(306), 2012.

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, pages 457–472, 1992.

A. Gelman, G. O. Roberts, and W. R. Gilks. Efficient Metropolis jumping rules in Bayesian statistics. *Bayesian Statistics 5*, pages 599–607, 1996.

J. Goodman and J. Weare. Ensemble samplers with affine invariance. *Communications in Applied Mathematics and Computational Science*, 5(1):65–80, 2010.

P. J. Green and A. Mira. Delayed rejection in reversible jump Metropolis-Hastings. *Biometrika*, 88(4):pp. 1035–1053, 2001.

J. Hachmann, R. Olivares-Amaya, S. Atahan-Evrenk, C. Amador-Bedolla, R. S. Sánchez-Carrera, A. Gold-Parker, L. Vogt, A. M. Brockway, and A. Aspuru-Guzik. The Harvard Clean Energy Project: Large-scale computational screening and design of organic photovoltaics on the world community grid. *The Journal of Physical Chemistry Letters*, 2(17):2241–2251, 2011.

J. Hachmann, R. Olivares-Amaya, A. Jinich, A. L. Appleton, M. A. Blood-Forsythe, L. R. Seress, C. Román-Salgado, K. Trepte, S. Atahan-Evrenk, S. Er, S. Shrestha, R. Mondal, A. Sokolov, Z. Bao, and A. Aspuru-Guzik. Lead candidates for high-performance organic photovoltaics from high-throughput quantum chemistry - the Harvard Clean Energy Project. *Energy Environ. Sci.*, 7: 698–704, 2014.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

A. Korattikara, Y. Chen, and M. Welling. Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

J. S. Liu, F. Liang, and W. H. Wong. The multiple-try method and local optimization in Metropolis sampling. *Journal of the American Statistical Association*, 95(449): pp. 121–134, 2000.

D. Maclaurin and R. P. Adams. Firefly Monte Carlo: Exact MCMC with subsets of data. In *30th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

R. M. Neal. Slice sampling. *The Annals of Statistics*, 31 (3):705–767, 06 2003.

R. M. Neal. How to view an MCMC simulation as a permutation, with applications to parallel simulation and improved importance sampling. Technical Report 1201, Dept. of Statistics, University of Toronto, 2012.

R. Nishihara, I. Murray, and R. P. Adams. Parallel MCMC with generalized elliptical slice sampling. *Journal of Machine Learning Research*, Oct. 2014.

R. Olivares-Amaya, C. Amador-Bedolla, J. Hachmann, S. Atahan-Evrenk, R. S. Sánchez-Carrera, L. Vogt, and A. Aspuru-Guzik. Accelerated computational discovery of high-performance materials for organic photovoltaics by means of cheminformatics. *Energy Environ. Sci.*, 4: 4849–4861, 2011.

T. Park and G. Casella. The Bayesian Lasso. *Journal of the American Statistical Association*, 103(482):681–686, 2008.

J. G. Propp and D. B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1&2): 223–252, 1996.

G. O. Roberts, A. Gelman, and W. R. Gilks. Weak convergence and optimal scaling of random walk Metropolis algorithms. *Annals of Applied Probability*, 7:110–120, 1997.

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

R. H. Swendsen and J. S. Wang. Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21): 2607–2609, Nov. 1986.

L. Tierney and A. Mira. Some adaptive Monte Carlo methods for Bayesian inference. *Statistics in Medicine*, 18: 2507–2515, 1999.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

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