SIAM J. SCI. COMPUT.
Vol. 38, No. 5, pp. S539–S565

ACCELERATED DIMENSION-INDEPENDENT ADAPTIVE METROPOLIS*

YUXIN CHEN†, DAVID KEYES‡, KODY J. H. LAW§, AND HATEM LTAIEF¶

Abstract. This work describes improvements by algorithmic and architectural means to black-box Bayesian inference over high-dimensional parameter spaces. The well-known adaptive Metropolis (AM) algorithm [H. Haario, E. Saksman, and J. Tamminen, *Bernoulli*, (2001), pp. 223–242] is extended herein to scale asymptotically uniformly with respect to the underlying parameter dimension for Gaussian targets, by respecting the variance of the target. The resulting algorithm, referred to as the dimension-independent adaptive Metropolis (DIAM) algorithm, also shows improved performance with respect to adaptive Metropolis on non-Gaussian targets. This algorithm is further improved, and the possibility of probing high-dimensional (with dimension $d \geq 1000$) targets is enabled, via GPU-accelerated numerical libraries and periodically synchronized concurrent chains (justified a posteriori). Asymptotically in dimension, this GPU implementation exhibits a factor of four improvement versus a competitive CPU-based Intel MKL (math kernel library) parallel version alone. Strong scaling to concurrent chains is exhibited, through a combination of longer time per sample batch (weak scaling) with fewer necessary samples to convergence. The algorithm performance is illustrated on several Gaussian and non-Gaussian target examples, in which the dimension may be in excess of one thousand.

Key words. Markov chain Monte Carlo, big data, Bayesian inference, adaptive Metropolis, Metropolis-Hastings, BLAS, GPU acceleration, high performance computing

AMS subject classifications. 65C05, 65C40

DOI. 10.1137/15M1026432

1. Introduction.

Recent years have seen increasing activity in the areas of uncertainty quantification and big data, largely enabled by the progress of computational science, which itself is enabled by ever more powerful computers and the symbiosis of this architectural brute force with innovative algorithmic advances. In particular, the solution of a forward problem, given by an ordinary differential equation (ODE) or partial differential equation (PDE), may be viewed as a distributed quantity induced by the uncertainty of input parameters [46], rather than as a deterministic quantity. When the input parameters themselves are spatially (and/or temporally) extended, one is faced with much higher-dimensional problems and indeed distributions over function spaces in principle [9, 50, 79]. In the context of Bayesian inference, this leads to the notion of a Bayesian analogue of the classical inverse problem [73, 75, 40, 8]. Such problems are enormously challenging both algorithmically and computationally and largely motivate the present work. At the same time, a very similar problem of

*Received by the editors June 16, 2015; accepted for publication (in revised form) June 7, 2016; published electronically October 27, 2016.

http://www.siam.org/journals/sisc/38-5/M102643.html

Funding: This work was supported by the King Abdullah University of Science and Technology (KAUST). The work of the third author was partially supported by Oak Ridge National Laboratory Directed Research and Development Strategic Hire grant 32112590 LDRD.

†Computer Science Program and Extreme Computing Research Center, KAUST, Thuwal, Saudi Arabia (yuxin.chen@kaust.edu.sa).

‡Applied Mathematics and Computational Sciences and Extreme Computing Research Center, KAUST, Thuwal, Saudi Arabia (david.keyes@kaust.edu.sa).

§Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831 (lawkj@ornl.gov).

¶Extreme Computing Research Center, KAUST, Thuwal, Saudi Arabia (hatem.ltaief@kaust.edu.sa).

big data is recently attracting a lot of attention. In the inverse problem case, even in the hypothetical case of full-field measurements, when the amount of data is infinite, the *effective* dimension of the data, or the space where posterior measure concentrates with respect to the prior, is often quite small with respect to that of the underlying parameter of interest, due to smoothing of the forward problem [73, 16, 45, 69]. The big data problem directly confronts the case of genuinely high-dimensional posterior distributions; i.e., the posterior differs significantly from the prior in the whole space [70, 42, 55, 28, 48]. Both of these problems involve sampling from high-dimensional distributions, meaning the dimension of the parameter space $d \geq 1000$ and in principle may be in the millions or arbitrarily high. For concrete examples of Bayesian inverse problems, consider inverting pointwise surface pressure field measurements for the three-dimensional subsurface permeability field, or inverting various observations of the atmosphere for the full velocity vector-field, pressure, and temperature. For a concrete big data example, consider m observed values which can be related to d covariates by Bayesian regression, where m and/or d may be a million or more.

The primary contribution of this paper is the detailed investigation and acceleration of a new adaptive Metropolis (AM) Hastings algorithm [33], which was introduced in [45] but has yet to be implemented. The benefit of this algorithm over standard AM is its scaling with respect to dimension. Indeed it is illustrated that, asymptotically as the number of samples tends to infinity, its convergence rate will remain stable as the underlying parameter dimension d tends to infinity. It is hence named dimension-independent adaptive Metropolis (DIAM). The cost scales as $\mathcal{O}(\max\{d^2, C_{\text{forward}}\})$, where C_{forward} is the cost of a single forward simulation, which is not too bad considering that one cannot expect to do better than C_{forward}. It is shown that GPU acceleration is able to reduce the constant and improves performance substantially for high dimension. Furthermore, a concurrent chain parallelization strategy is shown to provide strong scaling for up to 50% of the available cores on a multicore machine. The merging of chains is a posteriori justified by a potential scale reduction factor (PSRF) convergence diagnostic [26, 10]. It is shown also how to tune the parameters in the algorithm and how to modify it with an inflation term for improved performance in the case of non-Gaussian posteriors.

The rest of this paper is organized as follows. In section 2 some background is provided on both Markov chain Monte Carlo (MCMC) methodologies for Bayesian inverse problems and the computational issues arising in the methodology utilized herein. In section 3 the problem of Bayesian inference in high dimensions is introduced precisely, detailed definitions of the baseline and benchmark algorithms are given, and finally the concurrent formulation is presented as well as the convergence diagnostic for its a posteriori justification. In section 4 the algorithms are illustrated by some numerical experiments. In section 5 advanced GPU acceleration techniques are introduced, as well as the logistical framework for extending to multiple chains. Performance results are highlighted in section 6, and we conclude in section 7.

2. Literature review. Some review is now provided in order to set the stage and orient the nonexpert reader.

2.1. Algorithm review. Probability distributions over low-dimensional spaces are straightforward to represent via the associated probability density. It is impossible, however, to represent densities in higher than a few dimensions. But one can do something that is usually sufficient in scientific utility: one can sample the probability distribution with Monte Carlo. Probability distributions arising from a Bayesian framework introduce another layer of complexity in Monte Carlo, as typically one

can only *evaluate* the posterior distribution, up to a normalizing constant, while direct sampling methods typically do not exist. One must resort to methods such as importance or rejection sampling or MCMC [29, 30]. The posterior distribution will therefore henceforth be referred to as the *target distribution*, or simply the target, since the aim of the sampling algorithm is to sample from this target.

A primary workhorse of Bayesian computation is Markov chain Monte Carlo (MCMC). A popular and versatile MCMC algorithm is the Metropolis-Hastings (MH) algorithm, introduced in [53] and later revised to its current form in [36]. It requires as input only the target distribution and a proposal Markov chain. New samples are drawn according to the proposal Markov chain and then accepted or rejected in such a way that the target is kept invariant. The AM algorithm [33], and derivatives thereof (DRAM [32], ASWAM [4], SCAM [34], RAM [77], etc.), construct proposals based on the empirical covariance arising from the current trajectory, i.e., the past samples. These proposals are perhaps the most versatile, effective, and useful among the MH-type algorithms for low-dimensional and reasonably well-behaved targets, for example targets unimodal up to a dimension of 100. As the proposal depends on the chain history, it is no longer Markov, although there is theoretical work guaranteeing convergence under fairly general conditions [3, 65, 68, 23, 24]. For targets in which the Hessian of the logarithm has a strong local dependence, gradient-based proposals such as the Metropolis-adjusted Langevin algorithm (MALA) [67, 66], the Hamiltonian Monte Carlo (HMC) algorithm [21, 56], or their manifold extensions [31] can improve the convergence time, at the cost of providing the gradients, which may be nontrivial to obtain or may not even exist.

It can be shown that such proposals, as well as the random walk (RW) proposal upon which the AM algorithms are based, can be derived from the explicit discretization of a certain stochastic differential equation (SDE). Based on such diffusion limits, it has been shown that for underlying dimension d, the variance, or squared step-size, taken by the random walk (RW) Metropolis algorithm, the MALA, and the HMC algorithms must scale as $\mathcal{O}(1/d)$ [62, 6, 51], $\mathcal{O}(d^{-1/3})$ [63, 60], and $\mathcal{O}(d^{-1/4})$ [5], respectively. This naturally translates to decorrelation time of the inverse order; i.e., the number of steps required to obtain an almost independent sample is $\mathcal{O}(d)$, $\mathcal{O}(d^{1/3})$, and $\mathcal{O}(d^{1/4})$ [64]. For high-dimensional targets, this is naturally impractical, and this has been a limiting factor for the application of these algorithms to targets over higher-dimensional spaces, although the gradient-based methods can still be effective in high dimensions if Hessian information is incorporated efficiently [49, 11].

If a target arising from a Bayesian inverse problem is well-defined in the function-space limit, as it should be, then proposals can be designed to respect that limit [76]. When the problem is discretized, such proposals exhibit a decorrelation time that is independent of the refinement of the mesh towards that limit; in other words, independent of the underlying dimension [7, 13], or $\mathcal{O}(1)$. Recently the work [45] introduced an algorithm that incorporates general operator-weighting, and in particular Hessian information, into function-space proposals which may be derived from time-inhomogeneous discretization of the Ornstein–Uhlenbeck SDE. The work [16] goes one step further, using prior-preconditioned Hessian information to adaptively identify the space of posterior concentration, and then using empirical covariance information within that low-dimensional space to adaptively precondition a time-inhomogeneous discretization of the Langevin SDE.

In general, the amount of elaborate forward simulation code in the world, whether it be high-dimensional ODE or PDE, far outweighs the associated gradient and adjoint codes, so often such information may not be available. The possibility of avoid-

ing the person-hours required to construct such code is therefore highly valuable and provides good motivation for constructing *nonintrusive*, *black-box*, or gradient-free algorithms. This work presents an alternative approach to those described above, in an attempt to combine the best of each technique without resorting to gradient information. Indeed, the preconditioned Crank–Nicolson (pCN) proposal of [13] arises from a Crank–Nicolson discretization of an Ornstein–Uhlenbeck SDE which preserves a certain Gaussian measure. In contrast, the RW Metropolis algorithm proposal arises from an Euler–Maruyama discretization of a diffusion which spreads mass to infinity and has no invariant measure. It is this property that provides the $\mathcal{O}(1)$ decorrelation time of the former versus the $\mathcal{O}(d)$ time of the later. From this viewpoint, the advantage of the former is clear even in the absence of a function-space limit. Herein we construct a proposal inspired by the pCN approach, which preserves a distribution proportional to the empirical Gaussian obtained from past samples, yielding an asymptotically dimension-independent AM algorithm, which will be abbreviated DIAM. That is, the decorrelation time is expected to scale as $\mathcal{O}(1)$ for reasonably well-behaved distributions, and this can be proven for the Gaussian case. Nonetheless, this will result in a gain of only $\mathcal{O}(d^{1/2})$ in convergence time for root mean-squared error (RMSE) quantities. Therefore, the value is still limited as long as one is limited to $d \leq 100$. On the other hand, when the dimension of the target becomes much larger, the cost of adaptation itself may become a limiting factor due to the required linear algebra. The computational contribution consists of mitigating this effect.

2.2. Computational aspects. From the computational perspective, the fundamental limiting operations that comprise the AM algorithm, and the DIAM extension proposed here, are levels 2 and 3 basic linear algebra subprogram (BLAS) operations, scaling traditionally as $\mathcal{O}(d^2)$ and $\mathcal{O}(d^3)$, particularly the dense matrix-vector, matrix-matrix multiplication, and Cholesky-based matrix inversion. These operations prevent the use of the algorithms in high dimensions, even given the algorithmic advances outlined in the previous section. However, it is shown here that one may impose a lag-time of $\mathcal{O}(d)$ between Cholesky-based matrix inversion, and hence block updates of the covariance, without increasing the required number of samples to convergence. The algorithm is thereby immediately reduced to $\mathcal{O}(d^2)$ rather than $\mathcal{O}(d^3)$, in the sense that the cost to obtain N samples is $\mathcal{O}(Nd^2)$ (assuming the cost of evaluating the logarithm of the unnormalized density is at most $\mathcal{O}(d^2)$). It is also feasible to reduce the cost of the algorithm to $\mathcal{O}(d^2)$ by using low-rank Cholesky updates [20, 77].

We here propose using state-of-the-art GPU acceleration of dense linear algebra operations within the fundamental operations of the AM and DIAM algorithms. Compute-bound operations, i.e., level 3 BLAS kernels, usually benefit the most from these hardware accelerators because they are able to stress the floating-point units with significant data reuse at high levels of the memory hierarchy, and they attain a decent percentage of the theoretical peak performance of the underlying hardware. Memory-bound operations, i.e., level 2 BLAS kernels, are, however, limited by the bus bandwidth and how fast the requested data can be fetched to the floating-point units, due to negligible data reuse. Accelerators provide much higher bandwidth compared to standard x86 architecture, and therefore memory-bound kernels can still be accelerated on such hardware. All these statements assume that the data resides already on the GPU memory, which is not always the case for current architecture models. Data has to be offloaded from the host (CPU) memory to the device (GPU) memory through a thin pipe called the peripheral component interconnect express (PCIe),

which has an order of magnitude lower bandwidth than the GPU. By distributing the level 2 BLAS operations across the GPU, the transition to quadratic scaling is reduced by a factor of almost 4, leading to a factor-4 gain in efficiency asymptotically in d. This net gain arises from a combination of slow data transfer through PCIe, mitigated by asynchronous processing, and the speed-up of the resultant level 2 BLAS operations owing to the increased memory bandwidth on the GPU.

The clock frequency of a single processor of CMOS (complementary metal-oxide-semiconductor) logic has nearly reached its physical limit due to power dissipation constraints. The multicore era has permitted the introduction of multiple low-frequency cores on a single chip. This trend has been reinforced, moving forward, by the international exascale roadmap [19], which specifies that streaming multiprocessor architectures (NVIDIA GPUs, Intel Xeon Phi, etc.) composed of lightweight cores will be the norm for future exascale systems. The value of brute force concurrent parallelization has therefore increased. While traditional Monte Carlo methods enjoy this property, MCMC methods do not, as they are inherently serial in nature. Nonetheless, one can a posteriori justify the merging of concurrent parallel chains within the framework of [26, 10], using the so-called potential scale reduction factor (PSRF) as a diagnostic to measure convergence. This is the approach to parallelization of AM taken in the recent works [15, 71], although neither paper confronts a high-dimensional parameter. In [15] the objective is to sufficiently explore the state-space in order to identify a partition for regional adaptation. In [71] this approach is used to mitigate the cost of very expensive forward solves. Herein, the approach is proposed as a general parallelization strategy for the algorithm, indeed with almost perfect scaling efficiency in terms of time. The convergence time of the empirical covariance is decreased by concatenating samples from the concurrent chains through periodic synchronization. This gain makes up for the slight slow-down in the collection of a given batch of samples, resulting in effectively strong scaling with respect to convergence time. It is shown that this allows black-box sampling of targets over very high dimensions. As the focus of this work is the new DIAM algorithm, the principle is illustrated for that algorithm, but the same principle is expected to apply to AM.

It should be noted that many more elaborate approaches to parallelization of Bayesian computation have recently emerged, including [74, 80, 72, 14, 47, 39, 12]. For example, the authors in [74] and [47] developed a CUDA (compute unified device architecture) kernel to tackle the most time-consuming phase of their MCMC simulation using SIMD (single instruction, multiple data) parallelizations to run on the massive number of CUDA cores available on the GPU card. Our numerical algorithm relies on BLAS operations, for which most vendors provide highly optimized implementations on their hardware (e.g., cuBLAS for NVIDIA). Moreover, our implementation is portable across a range of vendor hardware, thanks to the legacy of the BLAS library.

It should also be noted that more advanced Monte Carlo methods exist for Bayesian computation, such as population-based MCMC [27, 37], equi-energy samplers [43], and sequential Monte Carlo samplers [18]. Such methods are indeed necessary for sampling from very complex multimodal distributions, but it should be noted that MH algorithms appear *within* these algorithms as a fundamental component, similarly to the way the BLAS operations appear in the MH algorithms as a fundamental component. The proposed DIAM algorithm is therefore expected to have a great impact as a fundamental black-box MH algorithm.

3. Bayesian inference in high dimensions.

3.1. General problem formulation. Given a quantity of interest $\varphi : \mathbb{R}^d \to \mathbb{R}$, estimate its expectation with respect to a probability measure π:

$$\pi(\varphi) := \mathbb{E}_\pi(\varphi) = \int_{\mathbb{R}^d} \varphi(x)\pi(x)dx \approx \frac{1}{N}\sum_{n=1}^{N} \varphi(x_i), \quad x_i \sim \pi. \tag{3.1}$$

The notation "$x \sim \pi$" indicates that the random variable x follows the distribution of π. The convergence of the approximation given above is a consequence of the law of large numbers for independent identically distributed (i.i.d.) random variables x_i [61] and an extension thereof under an assumption of sufficient decay of correlation [54].

Let $\eta : \mathbb{R}^d \to \mathbb{R}_+$, where $\mathbb{R}_+ = \{t \in \mathbb{R}; t \geq 0\}$, and assume $Z := \int_{\mathbb{R}^d} \eta(x)dx < \infty$. Then $\pi = \eta/Z$ is a probability density, in the sense that $\pi : \mathbb{R}^d \to \mathbb{R}_+$ *and* $\int_{\mathbb{R}^d} \pi(x)dx = 1$. Assume that, given $x \in \mathbb{R}^d$, $\eta(x)$ can be readily evaluated, but that there is no direct method for sampling from π. Probability measures in the present work will always have densities with respect to Lebesgue measure, and the same notation will be used both for the measure $\pi : \sigma(\mathbb{R}^d) \to [0, 1]$, where $\sigma(\mathbb{R}^d)$ refers to the *sigma algebra* of measurable sets in $\mathbb{R}^d$, and its density $\pi : \sigma(\mathbb{R}^d) \to \mathbb{R}^+$ with $\int_{\mathbb{R}^d} \pi(x)dx = 1$. This should not cause confusion.

Such a problem often arises in a Bayesian context, in which case one has some observation y such that $y|x \sim L(x, \cdot)$, where $L(x, \cdot)$ is the *likelihood* which gives the distribution of the data y conditional on x, and one knows how to evaluate the density $L(x, y)$ pointwise. The density of the *posterior* distribution of $x|y$ is given by

$$\pi(x) = \frac{1}{Z}L(x; y)\pi_0(x), \qquad Z = \int_{\mathbb{R}^d} L(x; y)\pi_0(x)dx, \tag{3.2}$$

where π_0 is the *prior* distribution of x before any observation is made, $L(x; y)$ is the density associated to the law of $y|x$, and the ";" notation is used to emphasize that the observation $y \in \mathbb{R}^{d_y}$ is fixed to a given observed value, while x is allowed to vary [61].

Particular attention will be paid to the case in which d is large. For example, in the context of Bayesian inverse problems, $d \to \infty$ in principle, and it is appropriate to formulate the problem as the discretization of a limiting measure on a function-space X. In this case the target is a measure $\mu : X \to \mathbb{R}_+$, $\mu(X) = 1$, and (3.2) takes the form

$$\frac{d\mu}{d\mu_0}(x) = \frac{1}{Z}L(x; y), \qquad Z = \int_X L(x; y)\mu_0(dx), \tag{3.3}$$

where $d\mu/d\mu_0$ denotes the *Radon–Nikodym derivative* of μ with respect to μ_0, i.e., the ratio $\mu(du)/\mu_0(du)$ of infinitesimal volume elements at the point u. A sufficient requirement for the above to be well-defined is that $c^{-1} < \mu_0(L(\cdot; y)) < c$ for some $c \in (0, \infty)$ [73]. This context will not be considered further; however, this is the problem to keep in mind when we refer to the $d \to \infty$ limit for Bayesian inverse problems.

The case of big data may also come increasingly to fit into this scenario. While this term has come to refer in the statistics community to the case of large d_y [42, 55], which need not imply large d, it would be natural to try to explain high-dimensional data in terms of a high-dimensional parameter. This may again lead to a posterior

distribution over a high-dimensional space. For example, in the context of regression, access to an increasing number of observations and potential covariates may inspire one to consider an increasing number of covariates as well as an increasing number of observations. In the Bayesian inverse problem context, the data may often be given as a noisy observation of the solution of a PDE with the parameter as input, and the intrinsic smoothing property which provides well-posedness of PDE may hence reduce the effective dimension of the data even in the case of full-field measurements when $d_y \to \infty$. In the big-data context, on the other hand, the data may be genuinely informative over increasingly high-dimensional parameter spaces, which can lead to higher *effective dimension* of the posterior with respect to the prior in comparison with the Bayesian inverse problem, albeit with a generally much simpler forward model connecting the parameter to the observations. The general black-box methods developed here are expected to be effective in both cases and more.

3.2. Markov chain Monte Carlo. Introduce a Markov chain with transition kernel $\mathcal{K} : \mathbb{R}^d \times \sigma(\mathbb{R}^d) \to \mathbb{R}^+$. Let $\mathcal{P}(\mathbb{R}^d)$ denote the set of probability densities over $\mathbb{R}^d$, i.e., functions $p :\to \mathbb{R}^d \to \mathbb{R}_+$ such that $\int_{\mathbb{R}^d} p(x)dx = 1$. By the definition of Markov kernel, for $q \in \mathcal{P}$ one has that $p(y) = \int_{\mathbb{R}^d} q(x)\mathcal{K}(x,y)dx \in \mathcal{P}$. The shorthand notation $p = q\mathcal{K}$ is therefore commonly used, while the equation $p(\varphi) = \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} q(x)\mathcal{K}(x,y)dx\varphi(y)dy$ inspires the analogous notation $f = \mathcal{K}\varphi = \int_{\mathbb{R}^d} \mathcal{K}(x,y)\varphi(y)dy$, so that $p(\varphi) = (q\mathcal{K})(\varphi) = q(\mathcal{K}\varphi) = q(f)$. The unfamiliar reader can think of the discrete state-space analogy of row vectors as representing probability distributions, column vectors as representing quantities of interest, and the transition kernel as given by a row stochastic matrix. A density π such that $\pi = \mathcal{K}\pi$ is referred to as (the density of) an invariant measure, and a sufficient condition is reversibility,

$$\int_{\mathbb{R}^d\times\mathbb{R}^d} \pi(dx)\mathcal{K}(x,dx') = \int_{\mathbb{R}^d\times\mathbb{R}^d} \pi(dx')\mathcal{K}(x',dx). \tag{3.4}$$

Under additional assumptions of irreducibility and aperiodicity, one has ergodicity of the chain, i.e., $\lim_{N\to\infty} |\mathcal{K}^N(x_0,\cdot) - \pi|_{TV} = 0$ for any $x_0 \in \mathbb{R}^d$, and rates can be derived depending essentially on the rate of decorrelation of the chain. A consequence of this is that if one sets $x_n \sim \mathcal{K}^n(x_0,\cdot) = \mathcal{K}(x_{n-1},\cdot)$, then x_n is distributed approximately according to the target π; hence such $\{x_{n-M}\}_{n=M+1}^{N+M}$ can be used in the approximation (3.1).

Indeed, if $x_M \sim \pi$ and the *autocorrelation function* (ACF) $\rho_n := \mathbb{E}[x_{m+n} - \mathbb{E}(x)][x_m - \mathbb{E}(x)]/(\mathbb{E}[x - \mathbb{E}(x)]^2) = \rho^n$ for some $\rho \in (0,1)$, then a simple calculation shows that

$$\mathbb{E}_{\prod_{n=1}^N(\pi\mathcal{K}^{n-1})}\left|\frac{1}{N}\sum_{n=M+1}^{N+M}\varphi(x_n) - \pi(\varphi)\right|^2 \le \frac{1}{N}\mathbb{E}_\pi[x - \mathbb{E}_\pi(x)]^2\left(1 + \frac{2}{1-\rho}\right), \tag{3.5}$$

where the geometric series identity $\Theta = \sum_{n=1}^\infty \rho^n = 1/(1-\rho)$ was used to simplify the *integrated autocorrelation time* (IACT) $1 + 2\Theta$. Notice that, by comparison to the celebrated central limit theorem [61] for i.i.d. draws, the *effective* sample size of the correlated ensemble, with respect to the i.i.d. case, may be defined as $N_{\text{eff}} = N/(1+2\Theta)$.

The MH algorithm, introduced in [53] and refined to its present version in [35], is perhaps the most popular and versatile amongst the MCMC methods. It states that an essentially arbitrarily chosen transition kernel $\mathcal{Q}$ [76] can be composed with an accept/reject step as follows in order to satisfy reversibility (3.4) with respect to π. Given x_n, the next sample $x_{n+1} \sim \mathcal{K}(x_n,\cdot)$, where the kernel $\mathcal{K}$ is defined as follows:

- Let $x' \sim \mathcal{Q}(x_n, \cdot)$.
- Let

$$x_{n+1} = \begin{cases} x' & \text{w.p. } \min\{1, \alpha(x_n, x')\}, \\ x_n & \text{else}, \end{cases} \tag{3.6}$$

where "w.p." stands for "with probability" and the *acceptance probability* α is defined as

$$\alpha(x_n, x') = \frac{\pi(x')\mathcal{Q}(x', x_n)}{\pi(x_n)\mathcal{Q}(x_n, x')}. \tag{3.7}$$

There are clearly infinitely many possible choices of $\mathcal{Q}$, which leads to a wide range of behaviors of the associated kernels $\mathcal{K}$. Essentially one aims to minimize the correlation between the subsequent samples, which in turn results in a smaller ρ in (3.5) above and hence smaller Θ and larger effective sample size N_{eff}. The MH algorithm is ubiquitous, not only as a method in its own right, but also as a fundamental component for many other Bayesian computation algorithms, as mentioned at the end of section 2.

3.3. Advanced MH proposals. This subsection will focus on the MH algorithm introduced in the previous subsection. The most basic MH proposal will be introduced first (indeed, the Metropolis algorithm), followed by the more advanced black-box, or gradient-free, algorithms which were mentioned in section 2. Finally, the algorithm introduced in the present work will be defined.

3.3.1. Random walk. The presentation begins with the SDE

$$dx = AdW, \tag{3.8}$$

where $A \in \mathbb{R}^{d\times d}$ is positive definite and dW is an independent increment of Brownian motion $dW \sim N(0, dt \times I)$ [59]. An Euler–Maruyama discretization of this equation with step-size β (time-step β^2) gives (see [41])

$$x_{n+1} = x_n + \beta A W_n, \tag{3.9}$$

where $W_n \sim N(0, I)$ and $W_n \perp W_m$ for all n, m. The standard random walk (RW) is defined by the above equation so that $\mathcal{Q}(x_n, x_{n+1}) = \mathcal{Q}(x_{n+1}, x_n) \propto \exp\{-\frac{1}{2\beta^2}|A^{-1}(x_n - x_{n+1})|^2\}$. The fact that the proposal density is symmetric means that $\alpha(x, x') = \pi(x')/\pi(x)$. Often $A = I$ is chosen as the identity matrix, although it is possible to make other educated choices, for example, the prior covariance in a Bayesian context, the Hessian close to the maximizer, or some other approximation of the covariance of the target.

3.3.2. Preconditioned Crank–Nicolson. In turn, the Ornstein–Uhlenbeck process is defined by the following SDE:

$$dx = -Bxdt + \sqrt{2B}AdW, \tag{3.10}$$

where A is as above, B is symmetric and positive definite, $\sqrt{B}$ denotes the symmetric matrix square root, and it is assumed that $\sqrt{B}A = A\sqrt{B}$. It is easy to verify, by substituting $\tilde{x} = e^{-Bt}x$ and using Itô isometry [59] to evaluate the stochastic integral, that the above equation has unique invariant distribution $N(0, AA^\top)$, making it a reasonable equation to aim to approximate if $AA^\top$ is a good approximation of

the covariance of the target. It was proposed in [7, 13] to use the above SDE as a starting point with $A = \sqrt{C}$ and $B = I$ for posterior measures with Gaussian prior $N(0, C)$, and furthermore to use a Crank–Nicolson discretization scheme, leading to the following update, for time-step δ (upon multiplication by 2):

$$(2+\delta)x_{n+1} = (2-\delta)x_n + 2\sqrt{2\delta}AW_n. \tag{3.11}$$

Setting step-size $\beta = 2\sqrt{2\delta}/(2+\delta)$, one has the pCN proposal (see [13])

$$x_{n+1} = \sqrt{1-\beta^2}x_n + \beta AW_n, \tag{3.12}$$

with W_n as above. Notice that this equation *preserves* the measure $N(0, AA^\top)$, just like its continuum counterpart (3.10). This means that if p is the density of $N(0, AA^\top)$, then $p = p\mathcal{Q}$, which in turn implies $p(x)\mathcal{Q}(x, x') = p(x')\mathcal{Q}(x', x)$. So, if $\pi(x) = q(x)p(x)$ for some q, then the MH algorithm with this proposal has the following acceptance probability: $\alpha(x, x') = q(x')/q(x)$. This is useful in case the prior is Gaussian, as only the likelihood appears in the acceptance. There is nothing intrinsically finite-dimensional about (3.10), or its temporal discretization (3.12), so one can see how this allows the definition of a *function-space* algorithm, i.e., one which is defined in the limit $d \to \infty$ for targets of the form (3.3) in which μ_0 is Gaussian. Indeed, as long as one can construct a proposal which is reversible with respect to the prior, then the same theory extends to non-Gaussian prior [78]. By observing that the form of (3.12) may be extended with *operators* B replacing the scalar β, the work of [45] introduced general operator-weighted proposals which are reversible with respect to priors of the form $N(m, AA^\top)$:

$$x_{n+1} = m + A(I - BB^\top)^{1/2}A^{-1}(x_n - m) + ABW_n. \tag{3.13}$$

For the above proposals, Hessian information may be incorporated if it is available, and this was the strategy of [45]. This was extended to more general proposals, including also gradient information, and given the general name of dimension-independent likelihood-informed (DILI) proposals in [16]. The name derives from judicious incorporation of the linear subspace where the posterior concentrates with respect to the prior, the *likelihood-informed space* (LIS) [17].

It has been shown in [62] that for proposals of the form (3.9) one must have $\beta^2 = \mathcal{O}(1/d)$, thereby leading to a decorrelation-time of $\mathcal{O}(d)$. In turn, by virtue of being defined in the function-space limit, the proposals described above allow $\beta = \mathcal{O}(1)$ with respect to parameter dimension. Of course, the effective data dimension, i.e., the dimension of the LIS, will indeed still play a role for the above proposals, although it can be mitigated for DILI proposals, in particular those of the type (3.13), by scaling the data-informed directions appropriately.

3.3.3. Adaptive Metropolis. When gradients are unavailable, as assumed in the present work, one way to improve upon the proposals (3.9) and (3.12) above is with empirical covariance information, and this leads to the AM algorithm [33]. Let

$$C_n = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^\top - m_n m_n^\top, \tag{3.14}$$

$$m_n = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{3.15}$$

and choose A_n such that $A_n A_n^\top = C_n$. Plugging this into (3.9) yields the classical AM proposal. The work [62] identifies an optimal acceptance rate of 0.234, and the later work [4] proposes scaling the step-size adaptively within the AM algorithm to target such an acceptance ratio. This will be the version of the AM considered here.

3.3.4. Dimension-independent adaptive Metropolis. The new algorithm introduced here was already alluded to in [45]. It follows naturally from the above presentation by substituting an A_n such that $A_n A_n^\top = C_n$ into (3.12). In fact, a reference point should possibly also be taken into account, in which case the proposal Markov chain takes the form

$$x_{n+1} = x_{\text{ref}} + \sqrt{1-\beta^2}(x_n - x_{\text{ref}}) + \beta A_n W_n. \tag{3.16}$$

The reference point x_{ref} may be chosen as the maximum a posteriori (MAP) estimator; i.e., $x_{\text{MAP}} = \text{argmax}_x \pi(x)$ if this is available. Or else it may be adapted to the empirical mean.[1] It is worth dwelling on several points that make this proposal, and the resultant MH algorithm, attractive:

- This proposal asymptotically targets $\mu = N(x_{\text{ref}}, C_\infty)$, which is the best Gaussian approximation of the target in case $x_{\text{ref}} = m_n \to m_\infty$, for example as measured by the Kullback–Liebler (KL) distance [44].
- For $\beta = 1$, this proposal asymptotically gives independent samples from μ (from the previous bullet). Hence, for a Gaussian target, $\pi = \mu$, and it is easy to verify that the acceptance probability will be 1. One will then obtain perfect independent samples from the posterior.
- For any target, the variance of draws from the proposal will asymptotically coincide with the variance of the target for any $\beta \in (0, 1]$. In turn, the variance of draws from the proposal of the AM algorithm will be $(1 + \beta^2)C_n$. So in order for the trace of draws from the proposal, i.e., the expected ℓ^2 norm, to be on par with the target, one will necessarily need to choose $\beta^2 = \mathcal{O}(1/d)$ for the AM algorithm, while it will suffice to choose $\beta^2 = \mathcal{O}(1)$ for the new proposal. The decorrelation-time of an MH algorithm is limited by that of the proposal, which in this case can be computed directly and is controlled by $\mathbb{E}x_n x_0 = \mathcal{O}(\beta^{-2})$. This leads to a decorrelation-time of $\mathcal{O}(d)$ for the AM algorithm and $\mathcal{O}(1)$ for the new algorithm. The new algorithm will hence be called dimension-independent adaptive Metropolis (DIAM).

For nonlinear/non-Gaussian targets the spread of the Gaussian approximation may actually miss some of the mass of the target, in the sense that the Gaussian may have negligible mass where there is actually significant mass of the target. For example, a Gaussian approximation of a two-dimensional boomerang-shaped target will clearly miss the tail ends of the boomerang. When the Gaussian is broadened, however, the tail ends of the boomerang may be captured. Following this rationale, it will be necessary to modify the above with some additive inflation factor $\rho > 1$ as follows:

$$x_{n+1} = x_{\text{ref}} + \sqrt{1-\beta^2}(x_n - x_{\text{ref}}) + \beta\rho A_n W_n. \tag{3.17}$$

Notice that as long as $\rho \in [1, \sqrt{2}]$, the covariance is still smaller than in the AM case, and one therefore expects improved performance, although some dependence on dimension will then exist. This point requires further investigation.

[1] In this case, it should be set to zero until some sufficiently large n to avoid too many abrupt jumps of the pivot.

3.4. Concurrent chains. In addition to the inflation introduced at the end of the previous section, this section introduces further new concepts in the implementation of the DIAM algorithm, not previously considered. It is relevant to discuss the potential of "embarrassingly parallel" MCMC. This is a controversial topic, since MCMC is an intrinsically serial algorithm and convergence proofs typically rely on this fact. Nonetheless, [26, 10] describe a convergence diagnostic based on running multiple chains and comparing the between-chain and within-chain covariances. Once this diagnostic indicates convergence, one is then justified a posteriori in merging the samples from the different chains.

3.4.1. Logistics. Denote P chains by $\{x^p\}_{p=1}^P$, and let k denote the number of batches which have been done. Each of these is run for M intervals of length $n_{\rm lag}$, and the local first two moments are collected periodically. We get

$$S^p_{k,m} = \frac{1}{mn_{\rm lag}} \sum_{i=1}^{mn_{\rm lag}} x_i^p (x_i^p)^\top = \frac{(m-1)n_{\rm lag}}{mn_{\rm lag}} S^p_{k,m-1} + \frac{n_{\rm lag}}{mn_{\rm lag}} \sum_{i=(m-1)n_{\rm lag}+1}^{mn_{\rm lag}} x_i^p (x_i^p)^\top, \tag{3.18}$$

$$m^p_{k,m} = \frac{1}{mn_{\rm lag}} \sum_{i=(m-1)n_{\rm lag}+1}^{mn_{\rm lag}} x_i^p = \frac{(m-1)n_{\rm lag}}{mn_{\rm lag}} m^p_{k,m-1} + \frac{n_{\rm lag}}{mn_{\rm lag}} \sum_{i=1}^{mn_{\rm lag}} x_i^p. \tag{3.19}$$

After each $n_{\rm lag}$ update, local updates of the global moments are made,

$$S^{p,{\rm glob}}_{k,m} = \frac{kMPn_{\rm lag}}{(kMP+m)n_{\rm lag}} S^{\rm glob}_k + \frac{mn_{\rm lag}}{(kMP+m)n_{\rm lag}} S^p_{k,m}, \tag{3.20}$$

$$m^{p,{\rm glob}}_{k,m} = \frac{kMPn_{\rm lag}}{(kMP+m)n_{\rm lag}} m^{\rm glob}_k + \frac{mn_{\rm lag}}{(kMP+m)n_{\rm lag}} m^p_{k,m}, \tag{3.21}$$

followed by a local update of the global covariance,

$$C^{p,{\rm glob}}_{k,m} = S^{p,{\rm glob}}_{k,m} - m^{p,{\rm glob}}_{k,m} (m^{p,{\rm glob}}_{k,m})^\top. \tag{3.22}$$

This is used within the individual steps of the algorithm (3.16). Then, each time $m = M$, the local samples from the P chains are merged into global moments so they can be shared:

$$S^{\rm glob}_k = \frac{(k-1)}{k} S^{\rm glob}_{k-1} + \frac{1}{kP} \sum_{p=1}^P S^p_{k-1,M}, \tag{3.23}$$

$$m^{\rm glob}_k = \frac{(k-1)}{k} m^{\rm glob}_{k-1} + \frac{1}{kP} \sum_{p=1}^P m^p_{k-1,M}. \tag{3.24}$$

At this point, one can compute the global covariance once, or just return the moments to the individual chains to continue in parallel. This procedure can be optimized, but that is outside the scope of the present work.

3.4.2. Potential scale reduction factor. As mentioned above, the potential scale reduction factor (PSRF) convergence diagnostic [26, 10] will be used for a posteriori justification of chain merging. It is defined as follows. Start P chains, with initial conditions which are overdispersed with respect to the target. Define the following within-chain quantities for each p:

$$S^p_{MK} = \frac{1}{K}\sum_{k=1}^{K} S^p_{k,M},$$

$$m^p_{MK} = \frac{1}{K}\sum_{k=1}^{K} m^p_{k,M}. \tag{3.25}$$

Now define the global quantities for $i = 1, \ldots, d$:

$$B_i = \frac{MKn_{\text{lag}}}{P-1}\sum_{p=1}^{P}(m^p_{MK} - m^{\text{glob}}_K)^2_i, \tag{3.26}$$

$$W_i = \frac{MKn_{\text{lag}}}{(MKn_{\text{lag}}-1)P}\sum_{p=1}^{P}(C^p_{MK})_{ii}, \tag{3.27}$$

where $C^p_{MK} = S^p_{MK} - (m^p_{MK})(m^p_{MK})^\top$. The first quantity is referred to as the *between-chain* variance, representing (a factor MKn_{lag} times) the variance between the means computed in the individual chains. The second is the average within-chain variance across the chains, and is referred to as the *within-chain* variance. These quantities both approximate the variance. Now define

$$R_i = \frac{MKn_{\text{lag}}-1}{MKn_{\text{lag}}} + \left(\frac{P+1}{PMKn_{\text{lag}}}\right)\frac{B_i}{W_i}. \tag{3.28}$$

The PSRF in this ith direction is given by $\sqrt{R_i}$. One expects that $\sqrt{R_i} > 1$, and clearly one has that $\sqrt{R_i} \to 1$ as $K \to \infty$. The indicator for convergence is $\sqrt{R_i} - 1 \leq \text{TOL}$, where TOL is taken to be some number smaller than 0.2. See [26, 10] for further details.

4. Numerical experiments. This section consists of a systematic collection of numerical experiments that present the algorithms defined in this paper.

4.1. Description of the test cases. To begin with, several random posterior densities are introduced. First a standard normal random matrix $A \in \mathbb{R}^{d\times r}$ is generated and used to construct a random symmetric matrix $B = AA^\top$. Such a matrix has a spectrum with maximum eigenvalue $\mathcal{O}(d)$ and minimum eigenvalue close to zero ($r = d$) or at zero ($r < d$). To mimic the case of a posterior distribution, with standard normal prior and log-likelihood $-\frac{1}{2}x^\top Bx$, the target is fixed as $N(0, C)$, where the covariance is set to the form $C = (B + I)^{-1}$. This covariance has smallest eigenvalue $\mathcal{O}(1/d)$ and largest close to 1, which will emphasize the effect of anisotropy. Furthermore, evaluating the target in these cases requires a dense matrix vector multiplication which has a complexity $\mathcal{O}(d^2)$ and is thus greater than or equal to to the cost of a typical black-box PDE forward solver that one may encounter in a more realistic example. The following "twisting" function is introduced:

$$\phi(x) = (x_1, x_2 + b_1x_1^2, x_3, x_4 + b_3x_3^2, \ldots, x_{d/10} + b_{d/10-1}x^2_{d/10-1}, x_{d/10+1}, \ldots, x_d),$$

which allows the construction of simple "boomerang"-shaped targets with exactly computable moments. The following four Gaussian cases and two non-Gaussian cases are considered:

- $\pi_1 = N(0, C = (B+I)^{-1})$, $r = d$ (full-rank, $\mathrm{cond}(C) = \mathcal{O}(d)$);
- $\pi_2 = N(0, C = (B/d+I)^{-1})$, $r = d$ (full-rank, $\mathrm{cond}(C) = \mathcal{O}(1)$);
- $\pi_3 = N(0, C = (B+I)^{-1})$, $r = d/10$ (low-rank, $\mathrm{cond}(C) = \mathcal{O}(d)$);
- $\pi_4 = N(0, C = V\mathrm{diag}[(\sigma^{-2}n^{-4}+1)^{-1}]_{n=1,\dots,d}V^\top)$ (full-rank, $\mathrm{cond}(C) = \mathcal{O}(\sigma^{-2})$);
- $\pi_5 = \pi_1 \circ V \circ \phi \circ V^\top$, $b_i = b\sigma_i^{-2}/\sqrt{d}$, $b = 0.3$ (non-Gaussian, *mildly* twisted);
- $\pi_6 = \pi_1 \circ V \circ \phi \circ V^\top$, $b_i = b\sigma_i^{-2}/\sqrt{d}$, $b = 2$ (non-Gaussian, *strongly* twisted),

where $V\Sigma V^\top = C$ is the *ordered* eigendecomposition of C from π_1 such that the first eigenpair corresponds to the smallest eigenvalue of C. Notice that the Jacobian determinant of ϕ is 1, so a change of variables is trivial. Also, one can compute the maximizer of π_j for all j, and it is 0. Furthermore, the mean and variance of π_5 and π_6 for $i = 2, 4, \dots, d/10$ are given by

$$\mathbb{E}[(V^\top x)_i] = -b_{i-1}\sigma_{i-1}^2, \qquad \mathbb{E}[(V^\top x)_i - \mathbb{E}[(V^\top x)_i]]^2 = \sigma_i^2 + 2b_{i-1}^2\sigma_{i-1}^4,$$

where σ_i^2 are the variances of the ith component under $\pi_1 \circ V^\top$, i.e., the ith diagonal element in Σ. For the others, the mean is 0 and the covariance is C, of course. For the last two non-Gaussian distributions, $\xi > 1$ must be tuned in (3.17) to allow sufficient spread in the proposal. This is purely heuristic.

The above targets are all randomly generated but chosen to mimic certain problems that arise in practice. We fix a modestly high dimension $d = 100$. Although the problems we aim to solve will have much higher dimension, this section is designed merely to illustrate the key properties of the algorithms—i.e., the fidelity of samples with respect to d and the condition number of the covariance. In particular it will be shown that DIAM performs favorably with respect to both. The target π_1 has the structure one might encounter in a big data problem, where we reduce the dimension of the data to $d_y = d$. This target is highly anisotropic because the covariance has a big condition number, which may or may not be the case for a big-data problem but which makes the problem more challenging. The target π_2 is generated by deliberately reducing the condition number from $\mathcal{O}(d)$ to $\mathcal{O}(1)$, thus making a clear comparison with π_1 to show how condition number impacts the algorithm efficiency. The target π_3 simulates the context of a Bayesian inverse problem, in which the posterior is low-rank with respect to the prior. The targets π_5 and π_6 are non-Gaussian distributions: π_5 is a mildly twisted Gaussian, and π_6 is a strongly twisted Gaussian.

The target π_4 has the structure of a Bayesian inverse problem with "smoothing" forward map, for example from a PDE forward solve, given by the decaying spectrum of the likelihood. To lend a concrete context to this problem, consider the following elliptic ODE to be solved for $p(\cdot; x) : [0, \pi] \to \mathbb{R}$, given a right-hand side x:

$$\frac{d^2}{ds^2}p = x, \tag{4.1}$$

$$p(0) = p(\pi) = 0. \tag{4.2}$$

Assume we are given plentiful full-field data $y \equiv 0$ (for simplicity), where

$$y = p(\cdot; x) + \zeta$$

and $x \perp \zeta \sim \sigma N(0, I)$, and assume the prior is given as $x \sim \pi_0 \equiv N(0, I)$ as well.

Upon a finite-dimensional truncation, the likelihood is given by

$$\mathcal{L}(x; y = 0) = \exp\left\{-\frac{1}{2}|Lx|^2\right\},$$

where $L : H^2([0,\pi]) \cap H_0^1([0,\pi]) \to L^2([0,\pi])$. Setting $x = \sum_{k=1}^{\infty} x_k \sin(ks)$, $p = \sum_{k=1}^{\infty} p_k \sin(ks)$, and $y = \sum_{k=1}^{\infty} y_k \sin(ks)$, the problem can be solved exactly, and the resulting posterior Gaussian measure on $x|y$ is then given in terms of the coefficients of its mean $m = \sum_{k=1}^{\infty} m_k \sin(ks)$ and covariance kernel operator $C = \sum_{k,\ell=1}^{\infty} C_{k,\ell} \sin(ks)\sin(\ell t)$ by

$$\begin{aligned} m_k &= (k^{-4} + \sigma^2)^{-1} k^{-2} y_k = 0, \\ C_{k,j} &= (\sigma^{-2} k^{-4} + 1)^{-1} \delta_{jk}. \end{aligned} \tag{4.3}$$

Truncating at d modes, it can be seen that this is π_4 above, up to rotation. Furthermore, many image reconstruction and deconvolution problems share this identical fundamental structure. Notice that the parameter σ^2 in this case corresponds to the variance on the data. Smaller variance implies bigger condition number, which makes this distribution more anisotropic and thus harder to sample from.

4.2. Autocorrelation assessment. In Figure 1 the numerical performance of the DIAM, AM, pCN, and RW algorithms is compared by looking at their ACFs with underlying distributions π_1 through π_6 for $d = 100$. The step-size β is adapted by targeting the optimal acceptance ratio range, which is 0.1 to 0.3 for AM and RW and is 0.3 to 0.5 for DIAM and pCN. It is chosen initially as $2.4/\sqrt{d}$, which is suggested in [33, 25]. The top four panels of Figure 1 show the ACF of $\varphi(x) = \log \pi_i(x)$, for $i = 1, 2, 4$, and 5, as a single global measure of DIAM, AM, pCN, and RW. The middle and bottom four panels of Figure 1 show the ACFs of $\varphi(x) = v_d^T x$ and $\varphi(x) = v_1^T x$, the projections onto the eigenvector associated to the largest eigenvalue and the smallest eigenvalue, respectively. One expects that DIAM will perform the best and RW will perform the worst. The performance of the pCN and AM algorithms is subtle since, on the one hand, pCN is a dimension-independent but isotropic algorithm and may become competitive in high-dimensional and well-conditioned cases. On the other hand, the AM algorithm performs equally in all directions, although it suffers from an $\mathcal{O}(d)$ dependence on the dimension, and therefore it performs better than pCN for targets of modest dimension whose covariance has a large condition number. The condition number is deliberately increased in target π_4 from $\mathcal{O}(d)$ (for $\sigma^2 = 1/d$) to $\mathcal{O}(d^2)$, so that one can have a more clear idea on how AM and pCN react as the condition number of C increases. This is shown in the bottom left panels, where there are two curves for each of pCN and AM. The AM algorithm performs the same. The pCN algorithm, on the other hand, performs the same for the eigendirection corresponding to the smallest eigenvalue (bottom panels), but performs significantly worse on the two other functionals. Numerical experiments confirm the behavior described above.

As mentioned, it is expected that the cost of a forward solve, say $C(d)$, will be bounded by $\mathcal{O}(d^2)$, so these experiments should give a good measure of the general usability of the algorithm. For example, if the forward solve involves a dense matrix-vector multiplication, it is $\mathcal{O}(d^2)$; if it involves an iterative solution of a sparse system, it is $\mathcal{O}(d)$; and if it involves fast Fourier transform (FFT), it is $\mathcal{O}(d\log(d))$. The argument found in [45, section 3.1] indicates that the scaling of pCN is roughly $\mathcal{O}(N\sigma_{\min}^{-2} C(d))$, where $\sigma_{\min}^2$ is the smallest eigenvalue of the posterior covariance,

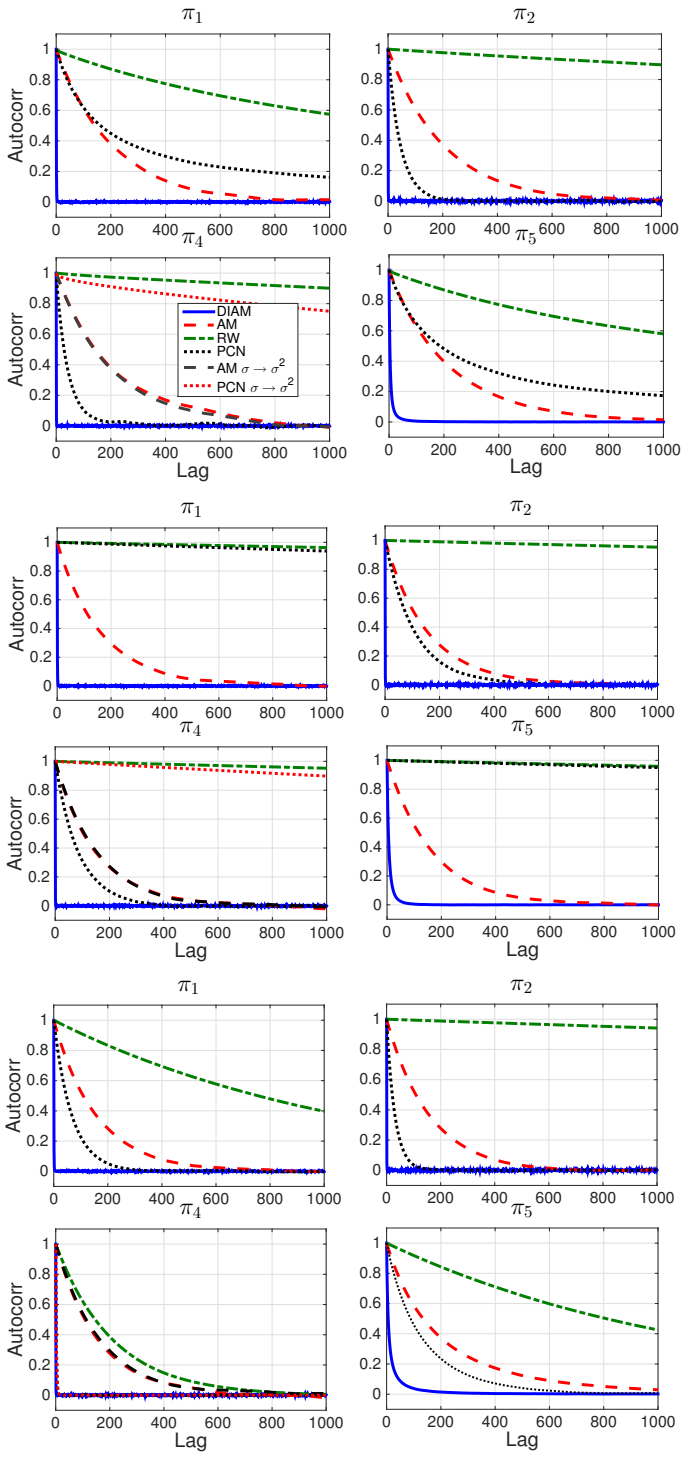
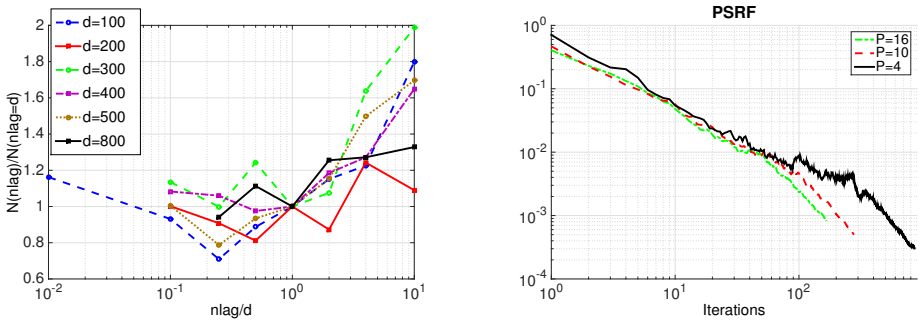


FIGURE 1. *Comparison of ACF of the log posterior (top four panels), and the projection onto the eigenvector associated with the largest eigenvalue (middle four) and the smallest eigenvalue (bottom four) of DIAM, AM, pCN, and RW on targets* π_i *for* $i = 1, 2, 4, 5$.

with whitened prior (approximately the inverse of the largest eigenvalue of the prior preconditioned Hessian [22, 16]), at least in the case of Gaussian targets. In turn, AM is $\mathcal{O}(N \max\{d^2, C(d)\}d)$, and DIAM is $\mathcal{O}(N \max\{d^2, C(d)\})$. One can therefore conclude that DIAM will outperform AM, and DIAM will outperform pCN if $C(d) \gtrsim d^2\sigma_{\min}^2$. AM will outperform pCN only if $C(d) \gtrsim d^3\sigma_{\min}^2$.

4.3. Impact of n_{lag} choice. The outcome of any MCMC simulation depends on, in addition to the natural variations due to random sampling, the specific way the run is performed. First, the chain length must be sufficient, and the burn-in has to be dealt with properly. In addition, any algorithm contains a number of tuning parameters that may decisively affect the results, and the frequency with which we update our proposal, denoted by n_{lag}, is one of the parameters that needs to be tuned.



(a) Total required number of samples as a function of n_{lag}/d (normalized by the number for $n_{\text{lag}} = d$) needed to satisfy a given convergence criterion. The missing points for small n_{lag} and larger d correspond to a "max-time" criterion of 12 hours.

(b) PSRF convergence criterion [10] for a range of number of chains $P = 4, 10, 16$, for $d = 1000$, with the number of outer batch iterations k given on the x-axis. In this case, the chains are stopped when our convergence criterion is satisfied.

FIGURE 2. *Tuning and convergence diagnostic.*

Test cases with target π_1 were run separately at various values of d. For each $d = 100, 200, \ldots, 500,$ and 800, n_{lag} varied over $\{d/100, d/10, d/4, d/2, d, 2d, 4d, 10d\}$, and the program was run until a certain stopping criterion was reached. The number of samples necessary to reach convergence, normalized by the number for $n_{\text{lag}} = d$, is shown in the left panel of Figure 2 as a function of n_{lag}/d. It is interesting that in fact the number of necessary samples *increases* for small enough n_{lag}. The corresponding time to convergence (not shown) is large for either small or large n_{lag}, due to the increased number of $\mathcal{O}(d^3)$ operations in the former case and the increased number of required samples in the latter. The curves are not convex, although this is presumably due to random effects, and it is expected that they would smooth out if averages were taken over sufficiently many simulations. While it would be interesting to identify the optimal value of n_{lag} and see whether it converges over multiple values of d, and even targets, to a universal value, for the present purposes this is not necessary. It suffices to observe that the minimum occurs for some $n_{\text{lag}} = \mathcal{O}(d)$. The value of n_{lag} is chosen as $d/2$ in the experiments to follow. This means that the total cost of the algorithm is $\mathcal{O}(d^2N)$, where N is the total number of samples. A similar effect could be obtained by performing low-rank Cholesky updates, although Figure 2 indicates that this may actually lead to a *larger* number of samples necessary to

achieve convergence for $n_{\text{lag}} = \mathcal{O}(1)$, and hence a larger cost. Furthermore, profiling with this choice shows that level 3 BLAS operations take less than 10% of the total simulation time, a consequence of the fact that, for sufficiently large d, the time to complete d level 2 BLAS operations of cost d^2 is significantly greater than one d^3 operation, due to memory constraints. This is discussed more in the next section.

For the examples illustrated here, convergence is diagnosed based on the exactly computable moments. In general, however, such ad hoc techniques as the PSRF (described in section 3.4.2) are required. The PSRF for π_1 with $d = 1000$ is shown in the right-hand panel of Figure 2 over various P, illustrating its convergence. The convergence criterion that is used to stop the chains is when the relative error of the sample covariance with respect to the truth in the Frobenius norm falls below some tolerance, denoted TOL. The same convergence criterion, with $\text{TOL} = 0.001$, is used for all the runs except for the tuning of n_{lag}. For the latter, we use the weaker convergence criterion of the absolute error of the sample mean with respect to the truth in the Euclidean norm, with $\text{TOL} = 0.01$.

5. High performance implementation. In this section, we describe the high performance implementation of the DIAM algorithm using standard x86 and GPU-accelerated numerical libraries.

5.1. Typical CPU-GPU architecture ecosystem. Today's hardware landscape is composed of lightweight x86 multicores associated with accelerators through a weak link called the peripheral component interconnect express (PCIe). The architectural discrepancies between the host (CPU) and the device (GPU) are manifest. GPU accelerators have thousands of CUDA cores, which provide unprecedented parallel performance and computing capabilities, i.e., more than an order of magnitude higher in terms of theoretical peak performance than the standard x86 CPU. Moreover, the speed at which data can be fetched from GPU main memory is higher than the bandwidth of a standard x86 CPU, by a factor of two or more, depending on the CPU system specifications. (In our testbed, it is almost a factor of five.) However, the PCIe bus cannot transfer the data from the CPU memory to the GPU memory as fast as the latter can compute. And this is precisely where the challenge resides, in keeping the CUDA cores always busy and not starving for computational work. This problem is further exacerbated by the limited size of the GPU memory, which can be smaller by one or two orders of magnitude than the CPU memory. All in all, application performance can usually be leveraged using GPU technology (i.e., massive thread parallelism, high computing power, and high memory bandwidth), as long as the overhead of moving data across the PCIe bus can be mitigated by using communication-reducing algorithms, and/or mandatory communications can be overlapped by useful computations.

5.2. High performance CPU-GPU numerical software stack. Fortunately, the high performance numerical software stack targeting the complexity of the CPU-GPU hardware is rich in kernel implementations and available from optimized open-source and vendor distributions. In particular, dense linear algebra (DLA) operations are well supported on multicore and hardware accelerators, thanks to their regularity in terms of memory accesses. The fundamental DLA kernels are categorized into three levels: levels 1, 2, and 3, which form the basic linear algebra subroutines (BLAS) library. Level 1 BLAS involves vector-vector operations (e.g., dot product), level 2 BLAS corresponds to matrix-vector operations (e.g., matrix-vector multiplication), and level 3 BLAS includes matrix-matrix operations (e.g., matrix-matrix multiplica-

tion). While level 1 and 2 BLAS operations are mostly memory-bound (limited by the bus bandwidth), level 3 BLAS kernels are compute-bound, thanks to a higher data reuse rate. BLAS kernels are often at the bottom of the software chain and, therefore, are critical for parallel performance. Vendors provide support for the BLAS kernels on their respective architectures. For instance, Intel provides its own high performance BLAS library on CPUs, distributed in the Math Kernel Library (MKL) [38]. On GPUs, NVIDIA provides the cuBLAS library [58], which implements BLAS kernels using the CUDA programming model [57]. The open-source KAUST BLAS (KBLAS) library [1] also provides a subset of level 2 BLAS operations on GPUs that perform better than the corresponding kernel from NVIDIA cuBLAS. Last but not least, LAPACK [2] provides CPU implementations of high-level DLA operations, such as solvers of linear equations and covariance (symmetric) matrix inversion.

5.3. The DIAM software framework. Algorithm 1 shows the work flow of DIAM[2] for sampling the target $\pi : \sigma(\mathbb{R}^N) \to [0, 1]$. There is an evaluation of the log target at each iteration, which is a level 2 BLAS operation for all of our random targets and another level 2 BLAS evaluation for the multiplication by A_n^{-1} in the evaluation of the weighted quadratic. Every n_{lag} iterations there is a level 2 BLAS operation for evaluation of the mean and level 3 BLAS operations for evaluation of the second-moment, Cholesky-based matrix inversion and evaluation of the next n_{lag} random search directions. Nonetheless, the bottleneck with increasing dimension turns out to be the n_{lag} level 2 BLAS operations in between updates, given that $n_{\text{lag}} = \mathcal{O}(d)$ and the level 2 BLAS operations are memory-bound. Notice that $A_{n_{\text{lag}}+m} = A_{n_{\text{lag}}}$ for $m < n_{\text{lag}}$. Therefore, from this work flow, the DIAM framework is basically composed of the following level 2 and 3 BLAS operations:

- LARNV: random matrix generation function (auxiliary LAPACK function).
- TRMV: performs triangular matrix-vector operations (level 2 BLAS).
- SYMV: performs symmetric matrix-vector operation (level 2 BLAS).
- GEMV: performs general matrix-vector operations (level 2 BLAS).
- SYR: performs the symmetric rank 1 operation (level 2 BLAS).
- GEMM: performs general matrix-matrix operations (level 3 BLAS).
- POTRF: performs Cholesky factorization (LAPACK function, mostly composed of level 3 BLAS).
- POTRI: computes the inverse of a real symmetric positive definite matrix A using the Cholesky factorization (POTRF) $A = U^T U$ or $A = LL^T$ (LAPACK function, mostly composed of level 3 BLAS).

All these functions are available from the high performance numerical CPU and GPU libraries, introduced in section 5.2.

5.4. Single chain parallelization implementation challenges. The challenge now resides in composing with all libraries and in determining which kernels need to run on which platform. Level 2 and 3 BLAS operations usually perform best on GPUs, i.e., the Cholesky-based symmetric matrix inversion of the sample covariance computation from (3.14) and the dense matrix-vector multiplication, as highlighted in (3.16). On the one hand, the Cholesky-based matrix inversion is compute-intensive, and its complexity may impede performance scalability of the overall parallel DIAM approach, if frequently requested for solving high-dimension problems. On the other

[2]It should be noted that there are other empirical details which are omitted. For example, a transient number of initial iterates n_0 are collected, with burn-in discarded, before the covariance is updated for the first time.

Algorithm 1 DIAM algorithm.

Initialize $x_0 \sim N(0, \mathrm{Id})$, $A_0 = \mathrm{Id}$, $\beta = \min\{2.4/\sqrt{d}, 0.5\}$, $n = 0, n_{\text{accepted}} = 0$;

for convergence criterion $\geq$ TOL **do**
  Propose: $x^* = x_{\text{ref}} + \sqrt{1-\beta^2}(x_n - x_{\text{ref}}) + \xi_n$;
  $u \sim \text{Uniform}(u; 0, 1)$
  $\log \alpha = \log \pi(x^*) + \frac{1}{2\rho^2}(A_n^{-1}x^*)^\top (A_n^{-1}x^*) - \left(\log \pi(x_n) + \frac{1}{2\rho^2}(A_n^{-1}x_n)^\top (A_n^{-1}x_n)\right)$
  if $\log u < \log \alpha$, **then**
    Accept the proposal draw: $x_{n+1} = x^*$; $n_{\text{accepted}} = n_{\text{accepted}} + 1$;
  else
    Reject the proposal draw: $x_{n+1} = x_n$.
  end if
  if $n = k n_{\text{lag}}$, $k \in \mathbb{Z}$, **then**
    Calculate acceptance ratio $\hat{\alpha} = n_{\text{accepted}}/n_{\text{lag}}$ and update β:
    if $\hat{\alpha} > 0.5$, **then**
      $\beta = \min\{1.1\beta, 1\}$;
    else
      if $\hat{\alpha} < 0.3$, **then**
        $\beta = \max\{0.9\beta, 0.1/\sqrt{N}\}$;
      end if
    end if
    $n_{\text{accepted}} = 0$;
    Calculate empirical mean and covariance m_n, C_n as (3.15), (3.14);
    Update $A_n = \text{Cholesky}(C_n)$; Compute A_n^{-1}
    Batch update $[\xi_{n+1}, \ldots, \xi_{n+n_{\text{lag}}}] = \beta\rho A_n [W_{n+1}, \ldots, W_{n+n_{\text{lag}}}]$,
    where $W_m \sim N(0, \mathrm{Id})$ i.i.d.;
  end if
  $n = n + 1$;
end for

hand, the dense matrix-vector multiplication is memory-bound and, therefore, exhibits a lower arithmetic complexity and slows the parallel DIAM implementation down if it becomes predominant. The lag-time is then paramount in balancing these two operations and in further reducing the time to solution, and it warrants further investigation. We rely on existing high performance implementations of both operations: we use the KBLAS [1] and the NVIDIA cuBLAS [58] libraries for the level 2 BLAS operations on GPU occurring each iteration, and the Intel MKL library [38] to perform the Cholesky-based matrix inversion and other level 3 BLAS operations occurring once every n_{lag} iterations. This hybrid CPU-GPU implementation requires data movement between CPU and GPU memory through the slow PCIe link. Ideally, one should try to operate on persistent data once on GPU memory to increase data reuse within the simulation. When this is not feasible, data motion has to be hidden using asynchronous data communication to mitigate the overhead of the slow PCIe bridge. The cuBLAS and KBLAS libraries provide API functionalities to ensure that communication can be overlapped with computation, through the CUDA programming model using the function CUDA_MEMCPY_ASYNC.

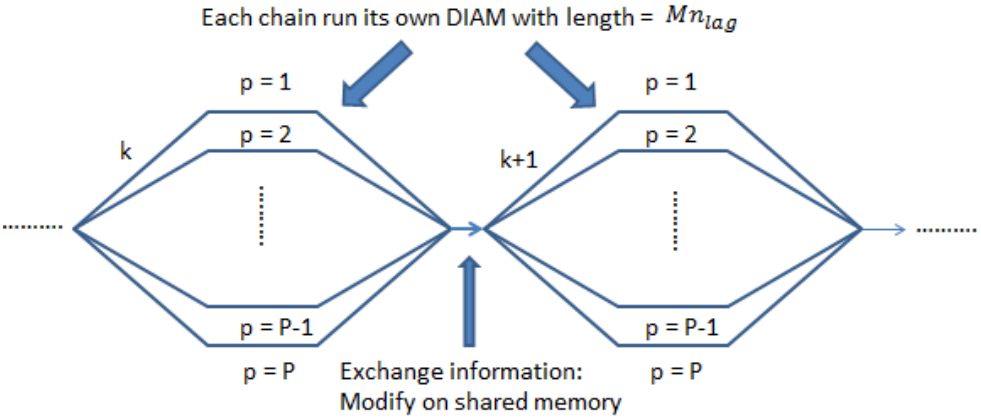


FIGURE 3. *DIAM using the fork-and-join parallel programming model.*

5.5. Concurrent chain parallelization using multithreading. The degree of parallelism of DIAM can be further leveraged by running concurrent chains (see section 3.4). Thanks to the POSIX threads programming model (Pthreads), threads are instantiated and work in an embarrassingly parallel fashion. We rely on the usual fork-and-join parallel programming model to take advantage of the parallelism exposed by the concurrent chains. Once P threads are created, each thread p will have its own private memory containing all information necessary to run on its own, as depicted in Figure 3. In Figure 3, k denotes the number of batches which have been completed. At the end of each batch processing, the threads are joined using a shared memory lock to facilitate and ensure safe synchronization. This may engender load imbalance if the workload per thread is not similar. However, this can be overcome using a more sophisticated dynamic scheduler to reduce the idle time [81].

This second level of parallelism introduces another complexity on the CPU because it mixes threads created by the Intel MKL library as well as the concurrent chains (Pthreads). Indeed, MKL implements multithreading in BLAS functions, and the default number of threads that MKL uses corresponds to the number of physical cores available on the system, except if the environment variable MKL_NUM_THREADS is defined by the user. Thus, the total number of threads running in the system is $P \times P_{\text{mkl}}$, where P is the number of chains launched and P_{mkl} is the number of threads MKL functions fork. When $P \times P_{\text{mkl}}$ is higher than the actual number of cores (P_{cores}) the system has, the overall performance may drop down because of thread oversubscription. Therefore, it is critical to keep $P \times P_{\text{mkl}} \leq P_{\text{cores}}$.

6. Performance results. This section presents the performance results of various DIAM implementations.

6.1. Environment settings. Table 1 defines the CPU specifications of the computing system used in these experiments. Sustained bandwidth is determined by the Stream benchmark. The system has two Intel Ivy Bridge CPU sockets with ten cores each. The total number of cores is therefore 20.

The system has three NVIDIA Tesla K40 GPU Accelerators with 1.4 Tflops sustained performance, 12 GB memory, and ultrafast memory bandwidth of 288 GB/s each. The GPUs are connected to the motherboard through a Gen3 PCIe bus. The

TABLE 1
Specifications for Intel Xeon Ivy Bridge E5-2680 v2.

Specifications	
CPU	2
Cores/CPU	10
Clock frequency (GHz)	2.8
Cache size (MB)	25
Memory bandwidth (GB/s)	59.7
Main memory (GB)	256
PCI express	3.0

machine runs Ubuntu 14.04.1 LTS and provides Intel Compilers Suite v13.0 together with the MKL library. The DIAM code is written in C and relies on multithreaded MKL and Pthreads for the multiple-chains implementation as well as CUDA through cuBLAS and KBLAS, for the CPU and GPU interfaces, respectively.

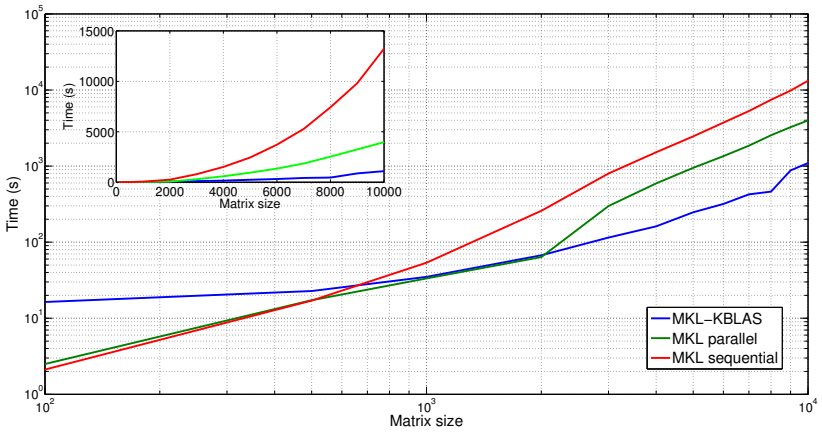
6.2. Empirical tuning. One can notice that level 3 BLAS functions in DIAM are called only every $d/2$ iterations, reducing the algorithm complexity to $\mathcal{O}(d^2)$. The strategy we use here is that, when dealing with small problems, e.g., problem sizes smaller than 1000, the optimized Intel MKL [38] is preferred (only CPU), while when dealing with larger problems, e.g., problem sizes larger than 1000, high performance libraries such as cuBLAS [58] and KBLAS [1] are preferred (GPU). This tuning choice helps mitigate the overhead of copying data between the host (CPU) and the device (GPU).

6.3. CPU-GPU performance profiling. Performance profiling of the MKL-based DIAM CPU implementation indicates that, as the dimension increases, SYMV becomes the bottleneck and impedes scaling to higher dimensions. SYMV is a level 2 BLAS function and thus is limited by the bus bandwidth. As described in section 5.1, accelerators provide several times higher bandwidth compared to standard x86 architecture, and therefore memory-bound kernels can still be accelerated on such hardware.

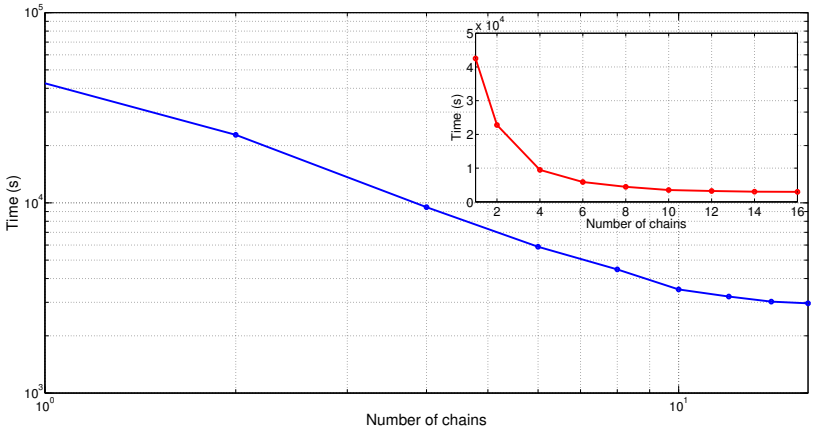
6.4. Performance scalability of DIAM. One of the approaches to statistical inference in high dimensions, beside algorithm improvement, is to reorganize the code into a faster implementation. In Figure 4(a), we show performance scalability in seconds to collect 10^5 samples from $d = 100$ to $d = 10000$ using MKL sequential (by setting MKL_NUM_THREADS = 1), MKL parallel (by setting MKL_NUM_THREADS = 20), and MKL-KBLAS (hybrid) high performance libraries combined. The target distribution used here is π_1. The MKL-KBLAS curve represents the implementation using both MKL and GPU libraries. The MKL curve represents the implementation using only MKL and run with 20 threads, and MKL sequential represents the implementation written on C and run with only one thread with no parallel techniques involved. The time required to collect 10^5 samples of MKL-KBLAS code outperforms that of MKL parallel code for $d \geq 3000$. Fitting these three curves to quadratic functions results in the following:

- MKL-KBLAS $T = 56.39 - 0.036d + 1.34 \times 10^{-5}d^2$,
- MKL parallel $T = 7.49 - 0.033d + 4.32 \times 10^{-5}d^2$,
- MKL sequential $T = 253.63 - 0.3983d + 1.65 \times 10^{-4}d^2$.

These functions make it easy to read the d_{quad} such that quadratic scaling begins, as well as the asymptotic gain factor of between 3 and 4 in MKL-KBLAS over MKL alone, and a similar difference between MKL parallel and serial.



(a) Performance scalability to collect 10^5 samples.



(b) Scalability of concurrent chains.

FIGURE 4. *These panels illustrate the performance scalability of DIAM: the time required to obtain a batch of* 10^5 *samples as a function of problem size* d (a) *and the time required to obtain a suitable convergence criterion as a function of number of cores* P *for concurrent DIAM (strong scaling)* (b).

6.5. Performance scalability of concurrent chain DIAM. In Figure 4(b), the scaling to concurrent chains is illustrated for target π_1 with $d = 1000$, and $M = 40$ fixed. The scaling is essentially $T \propto P^{-1}$ at first, but for $P > 10$ it slows down on a machine with 20 cores (see discussion at the end of section 5.5). This algorithm is memory-bound and needs synchronization after each chain generates a certain number of samples; thus, once the memory bandwidth is saturated, adding more threads will have limited benefit because more time is spent in each batch (the interval between each two synchronizations; see Table 2). We refer to the results in Figure 4(b) as "subtle" strong scaling because, in contrast to traditional strong scaling, the problem size actually is shrinking; namely, the total number of samples required to get convergence is decreasing as we add more chains. This can therefore still be considered

TABLE 2
Scaling to concurrent chains in terms of convergence time for $d = 1000$. Here the total number of samples is $N = PMKn_{\text{lag}}$.

P	**Total time [s]**	**Time per batch [s]**	$PMKn_{\text{lag}}$
1	42517.73	5.98	142260000
2	22779.92	8.46	107760000
4	9486.76	8.78	86480000
6	5878.11	9.09	77640000
8	4466.00	9.67	73920000
10	3506.19	9.9	70800000
12	3215.66	11.01	70080000
14	3024.47	12.1	70000000
16	2962.85	13.47	70400000

a form of strong scaling because the same convergence criterion is used, and in this sense the problem is the same. However, it is clear that the reduction in number of samples is converging. The reduction in required number of samples is likely due to the fact that more chains translates to more total samples used for a given update of the proposal covariance, and hence the proposal adapts faster.

These experiments performed on shared-memory systems suggest new opportunities in further scaling DIAM to multiple distributed-memory nodes. As shown in this section, single-node performance starts to decay after running beyond one socket (i.e., ten cores in our testbed) due to the saturation of the bus bandwidth, which is typical for memory-bound applications. We can then weak-scale the simulation by adding more nodes, each equipped with GPUs, and solve higher-dimensional problems on a distributed-memory environment using the message passing interface (MPI) [52]. The synchronization scheme described in Figure 3 will have to be adjusted, and explicit function calls will have to be made in order to handle communications across the computational nodes. In particular, collective communication operations will be required to synchronize between the distributed nodes. This may generate overhead due to the higher latency and lower bandwidth of the network interconnect when moving data off-chip. However, the latest MPI 3.0 standard allows for nonblocking collective communication operations, which may mitigate the overhead when running on large distributed-memory systems.

7. Summary. A black-box MCMC algorithm is introduced for Bayesian inference of highly anisotropic targets in high dimensions, herein named DIAM. In particular, it is shown that for Gaussian target distributions the integrated autocorrelation time, and hence efficiency of the algorithm, is independent of the underlying dimension, asymptotically as the number of samples tends to infinity. The algorithm is shown to perform as expected on Gaussian targets and also performs favorably with respect to standard AM on non-Gaussian targets. These algorithms are also compared to some other standard Metropolis variants. GPU-accelerated level 2 operations enable the efficient exploration of high-dimensional targets with $d \geq 1000$. The speedup versus standard serial C code is a factor of twelve as dimension tends to infinity. This improvement in conjunction with the combination of concurrent chains (justified a posteriori) may in principle allow exploration of very high-dimensional targets with $d \gg 1000$, since the number of cores per node will further increase in the near future. This will enable a considerable on-node level of concurrency, which our implementation can really leverage. A form of strong scaling with respect to convergence

time is illustrated on up to 16 cores. The parallelization strategy used for the DIAM algorithm will work also for the standard AM algorithm.

We mentioned in the introduction that the cost of the algorithm is $\mathcal{O}(\max\{d^2, C_{\text{forward}}\})$. It is also fundamentally limited by a $\mathcal{O}(d^2)$ memory requirement. The work [16] introduced in its conclusion a similar dimension-independent likelihood-informed (DILI) algorithm, which is the same as DIAM on a certain linear subspace with dimension d_{LIS}, the likelihood-informed space (LIS), and is combined with pCN [13] on the complement space. This algorithm has a cost of $\mathcal{O}(\max\{d_{\text{LIS}}^2, C_{\text{forward}}\})$ and so is genuinely dimension-independent. The memory requirement is $\mathcal{O}(dd_{\text{LIS}})$. However, the standard version of DILI requires gradients, unlike the algorithm herein which is *black-box*, in the sense that it requires only a forward model solver as input. It is the subject of our active research to identify the LIS using samples alone, hence providing a complexity proportional to C_{forward} in fully black-box format. Note that this approach will provide a gain in efficiency only when (a) the target posterior is a low-rank modification of the prior and (b) $C_{\text{forward}} = o(d^2)$, both of which often hold for Bayesian inverse problems.

Acknowledgments. YC, DK, and HL are members of the Extreme Computing Research Center at KAUST. KJHL was a member of the SRI Center for Uncertainty Quantification (SRI-UQ) at KAUST while this paper was written. We thank Heikki Haario and Youssef Marzouk for useful conversations about related topics, and the anonymous referees for many useful suggestions which have improved the manuscript.

REFERENCES

[1] A. Abdelfattah, H. Ltaief, and D. Keyes, *KBLAS: An optimized library for dense matrix-vector multiplication on GPU accelerators*, ACM Trans. Math. Softw., 42 (2016), 18, doi:10.1145/2818311.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, 3rd ed., Software Environ. Tools 9, SIAM, Philadelphia, 1999.

[3] C. Andrieu and É. Moulines, *On the ergodicity properties of some adaptive MCMC algorithms*, Ann. Appl. Probab., 16 (2006), pp. 1462–1505.

[4] C. Andrieu and J. Thoms, *A tutorial on adaptive MCMC*, Statist. Comput., 18 (2008), pp. 343–373.

[5] A. Beskos, N. Pillai, G. Roberts, J.-M. Sanz-Serna, and A. Stuart, *Optimal tuning of the hybrid Monte Carlo algorithm*, Bernoulli, 19 (2013), pp. 1501–1534.

[6] A. Beskos, G. O. Roberts, and A. M. Stuart, *Optimal scalings of Metropolis-Hastings algorithms for non-product targets in high dimensions*, Ann. Appl. Probab., 19 (2009), pp. 863–898.

[7] A. Beskos, G. O. Roberts, A. M. Stuart, and J. Voss, *MCMC methods for diffusion bridges*, Stochast. Dyn., 8 (2008), pp. 319–350.

[8] L. Biegler, G. Biros, O. Ghattas, M. Heinkenschloss, D. Keyes, B. Mallick, L. Tenorio, B. van Bloemen Waanders, K. Willcox, and Y. Marzouk, *Large-Scale Inverse Problems and Quantification of Uncertainty*, Wiley Ser. Comput. Statist. 712, John Wiley & Sons, New York, 2011.

[9] V. I. Bogachev, *Gaussian Measures*, Math. Surveys Monogr. 62, American Mathematical Society, Providence, RI, 1998.

[10] S. P. Brooks and A. Gelman, *General methods for monitoring convergence of iterative simulations*, J. Comput. Graph. Statist., 7 (1998), pp. 434–455.

[11] T. Bui-Thanh and M. Girolami, *Solving large-scale PDE-constrained Bayesian inverse problems with Riemann manifold Hamiltonian Monte Carlo*, Inverse Problems, 30 (2014), 114014.

[12] B. Calderhead, *A general construction for parallelizing Metropolis-Hastings algorithms*, Proc. Natl. Acad. Sci. USA, 111 (2014), pp. 17408–17413.

[13] S. L. Cotter, G. O. Roberts, A. M. Stuart, and D. White, *MCMC methods for functions: Modifying old algorithms to make them faster*, Statist. Sci., 28 (2013), pp. 424–446.

[14] R. V. Craiu and X.-L. Meng, *Multiprocess parallel antithetic coupling for backward and forward Markov chain Monte Carlo*, Ann. Statist., 33 (2005), pp. 661–697.
[15] R. V. Craiu, J. Rosenthal, and C. Yang, *Learn from thy neighbor: Parallel-chain and regional adaptive MCMC*, J. Amer. Statist. Assoc., 104 (2009), pp. 1454–1466.
[16] T. Cui, K. J. H. Law, and Y. M. Marzouk, *Dimension-independent likelihood-informed MCMC*, J. Comput. Phys., 304 (2016), pp. 109–137, doi:10.1016/j.jcp.2015.10.008.
[17] T. Cui, J. Martin, Y. M. Marzouk, A. Solonen, and A. Spantini, *Likelihood-informed dimension reduction for nonlinear inverse problems*, Inverse Problems, 30 (2014), 114015.
[18] P. Del Moral, A. Doucet, and A. Jasra, *Sequential Monte Carlo samplers*, J. Roy. Statist. Soc. Ser. B, 68 (2006), pp. 411–436.
[19] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, S. Moore, T. A. R., Trefethen, and M. Valero, *The international exascale software project: A call to cooperative action by the global high performance community*, Int. J. High Perform. Comput. Appl., 23 (2009), pp. 309–322.
[20] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, 1979.
[21] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, *Hybrid Monte Carlo*, Phys. Lett. B, 195 (1987), pp. 216–222.
[22] H. P. Flath, L. C. Wilcox, V. Akçelik, J. Hill, B. van Bloemen Waanders, and O. Ghattas, *Fast algorithms for Bayesian uncertainty quantification in large-scale linear inverse problems based on low-rank partial Hessian approximations*, SIAM J. Sci. Comput., 33 (2011), pp. 407–432.
[23] G. Fort, E. Moulines, and P. Priouret, *Convergence of adaptive and interacting Markov chain Monte Carlo algorithms*, Ann. Statist., 39 (2011), pp. 3262–3289.
[24] G. Fort, E. Moulines, P. Priouret, and P. Vandekerkhove, *A central limit theorem for adaptive and interacting Markov chains*, Bernoulli, 20 (2014), pp. 457–485.
[25] A. Gelman, G. Roberts, and W. Gilks, *Efficient metropolis jumping rules*, Bayesian Statist., 5 (1996), pp. 599–608.
[26] A. Gelman and D. B. Rubin, *Inference from iterative simulation using multiple sequences*, Statist. Sci., 7 (1992), pp. 457–472.
[27] C. J. Geyer, *Markov chain Monte Carlo maximum likelihood*, in Computer Science and Statistics: Proceedings of the 23rd Symposium on the Interface, Interface Foundation, Fairfax Station, VA, 1991, pp. 156–163.
[28] S. Ghosal, J. K. Ghosh, and A. W. Van Der Vaart, *Convergence rates of posterior distributions*, Ann. Statist., 28 (2000), pp. 500–531.
[29] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice*, Chapman Hall/CRC, New York, 1996.
[30] W. R. Gilks, *Markov Chain Monte Carlo*, Wiley Online Library, 2005, doi:10.1002/0470011815.b2a14021.
[31] M. Girolami and B. Calderhead, *Riemann manifold Langevin and Hamiltonian Monte Carlo methods*, J. Roy. Statist. Soc. Ser. B, 73 (2011), pp. 123–214.
[32] H. Haario, M. Laine, A. Mira, and E. Saksman, *DRAM: Efficient adaptive MCMC*, Statist. and Comput., 16 (2006), pp. 339–354.
[33] H. Haario, E. Saksman, and J. Tamminen, *An adaptive Metropolis algorithm*, Bernoulli, 7 (2001), pp. 223–242.
[34] H. Haario, E. Saksman, and J. Tamminen, *Componentwise adaptation for high dimensional MCMC*, Computat. Statist., 20 (2005), pp. 265–273.
[35] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, 57 (1970), pp. 97–109.
[36] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, 57 (1970), pp. 97–109.
[37] K. Hukushima and K. Nemoto, *Exchange Monte Carlo method and application to spin glass simulations*, Journal of the Physical Society of Japan, 65 (1996), pp. 1604–1608.
[38] Intel, *Math Kernel Library*; http://software.intel.com/en-us/articles/intel-mkl/.
[39] P. Jacob, C. P. Robert, and M. H. Smith, *Using parallel computation to improve independent Metropolis–Hastings based estimation*, J. Comput. Graph. Statist., 20 (2011), pp. 616–635.
[40] J. P. Kaipio and E. Somersalo, *Statistical inversion and Monte Carlo sampling methods in electrical impedance tomography*, Inverse Problems, 16 (2000), pp. 1487–1522.
[41] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Appl. Math. 23, Springer-Verlag, Berlin, 1992.
[42] 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, J. Machine Learning Res., 32 (2014).

[43] S. C. Kou, Q. Zhou, and W. H. Wong, *Discussion paper equi-energy sampler with applications in statistical inference and statistical mechanics*, Ann. Statist., (2006), pp. 1581–1619.
[44] S. Kullback and R. A. Leibler, *On information and sufficiency*, Ann. Math. Stat., 22 (1951), pp. 79–86.
[45] K. J. H. Law, *Proposals which speed up function-space MCMC*, J. Comput. Appl. Math., 262 (2014), pp. 127–138.
[46] O. P. Le Maître and O. M. Knio, *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics*, Springer Science & Business Media, New York, 2010.
[47] A. Lee, C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes, *On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods*, J. Comput. Graph. Statist., 19 (2010), pp. 769–789.
[48] D. Maclaurin and R. P. Adams, *Firefly Monte Carlo: Exact MCMC with subsets of data*, in Proceedings of the International Joint Conference on Artificial Intelligence, 2015, http://www.aaai.org/ocs/index.php/IJCAI/IJCAI15/paper/view/11279.
[49] J. Martin, L. C. Wilcox, C. Burstedde, and O. Ghattas, *A stochastic Newton MCMC method for large-scale statistical inverse problems with application to seismic inversion*, SIAM J. Sci. Comput., 34 (2012), pp. A1460–A1487.
[50] B. Matérn et al., *Spatial Variation. Stochastic Models and Their Application to Some Problems in Forest Surveys and Other Sampling Investigations*, Meddelanden fran statens Skogsforskningsinstitut, 49 (1960), 5.
[51] J. C. Mattingly, N. S. Pillai, and A. M. Stuart, *SPDE limits of the random walk Metropolis algorithm in high dimensions*, Ann. Appl. Probab., 22 (2012), pp. 881–930.
[52] Message Passing Interface Forum, *MPI: A message passing interface*, in Proceedings of Supercomputing '93, IEEE Computer Society Press, Piscataway, NJ, 1993, pp. 878–883.
[53] N. Metropolis, R. W. Rosenbluth, M. N. Teller, and E. Teller, *Equations of state calculations by fast computing machines*, J. Chem. Phys., 21 (1953), pp. 1087–1092.
[54] S. P. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, Communications and Control Engineering Series, Springer-Verlag, London, 1993.
[55] S. Minsker, S. Srivastava, L. Lin, and D. B. Dunson, *Robust and Scalable Bayes via a Median of Subset Posterior Measures*, preprint, https://arxiv.org/abs/1403.2660, 2014.
[56] R. M. Neal, *Bayesian Learning for Neural Networks*, Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, ON, 1994.
[57] NVIDIA, *CUDA: Compute Unified Device Architecture – Parallel Computing Platform and Programming Model*, http://www.nvidia.com/object.
[58] NVIDIA, *The CUDA Basic Linear Algebra Subroutines (cuBLAS)*, http://www.nvidia.com/object/cuda_home_new.html.
[59] B. Øksendal, *Stochastic Differential Equations*, 5th ed., Universitext, Springer-Verlag, Berlin, 1998.
[60] N. S. Pillai, A. M. Stuart, and A. H. Thiery, *Optimal scaling and diffusion limits for the Langevin algorithm in high dimensions*, Ann. Appl. Probab., 22 (2012), pp. 2320–2356.
[61] C. P. Robert and G. C. Casella, *Monte Carlo Statistical Methods*, Springer Texts Statist., Springer-Verlag, Berlin, 1999.
[62] G. O. Roberts, A. Gelman, and W. R. Gilks, *Weak convergence and optimal scaling of random walk Metropolis algorithms*, Ann. Appl. Probab., 7 (1997), pp. 110–120.
[63] G. O. Roberts and J. Rosenthal, *Optimal scaling of discrete approximations to Langevin diffusions*, J. Roy. Statist. Soc. Ser. B, 60 (1998), pp. 255–268.
[64] G. O. Roberts and J. Rosenthal, *Optimal scaling for various Metropolis-Hastings algorithms*, Statist. Sci., 16 (2001), pp. 351–367.
[65] G. O. Roberts and J. S. Rosenthal, *Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms*, J. Appl. Probab., 44 (2007), pp. 458–475.
[66] G. O. Roberts and R. L. Tweedie, *Exponential convergence of Langevin distributions and their discrete approximations*, Bernoulli, 2 (1996), pp. 341–363.
[67] P. J. Rossky, J. D. Doll, and H. L. Friedman, *Brownian dynamics as smart Monte Carlo simulation*, J. Chem. Phys., 69 (1978), pp. 4628–4633.
[68] E. Saksman and M. Vihola, *On the ergodicity of the adaptive Metropolis algorithm on unbounded domains*, Ann. Appl. Probab., 20 (2010), pp. 2178–2203.
[69] C. Schillings and Ch. Schwab, *Scaling Limits in Computational Bayesian Inversion*, Technical Report 2014-26, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2014.
[70] S. L. Scott, A. W. Blocker, F. V. Bonassi, H. Chipman, E. George, and R. McCulloch, *Bayes and big data: The consensus Monte Carlo algorithm*, Int. J. Management Sci. and Engrg. Management, 11 (2016), pp. 78–88.

[71] A. Solonen, P. Ollinaho, M. Laine, H. Haario, J. Tamminen, and H. Järvinen, *Efficient MCMC for climate model parameter estimation: Parallel adaptive chains and early rejection*, Bayesian Anal., 7 (2012), pp. 715–736.
[72] I. Strid, *Efficient parallelisation of Metropolis–Hastings algorithms using a prefetching approach*, Comput. Statist. Data Anal., 54 (2010), pp. 2814–2835.
[73] A. M. Stuart, *Inverse problems: A Bayesian approach*, Acta Numer., 19 (2010), pp. 451–559.
[74] M. A. Suchard, Q. Wang, C. Chan, J. Frelinger, A. Cron, and M. West, *Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures*, J. Comput. Graph. Statist., 19 (2010), pp. 419–438.
[75] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*, SIAM, Philadelphia, 2005.
[76] L. Tierney, *A note on Metropolis-Hastings kernels for general state spaces*, Ann. Appl. Probab., 8 (1998), pp. 1–9.
[77] M. Vihola, *Robust adaptive Metropolis algorithm with coerced acceptance rate*, Statist. Comput., 22 (2012), pp. 997–1008.
[78] S. J. Vollmer, *Dimension-independent MCMC sampling for inverse problems with non-Gaussian priors*, SIAM/ASA J. Uncertainty Quant., 3 (2015), pp. 535–561.
[79] P. Whittle, *Prediction and Regulation by Linear Least-Square Methods*, English University Press, London, 1963.
[80] D. J. Wilkinson, *Parallel Bayesian computation*, Statist. Textbooks Monogr., 184 (2006), p. 477.
[81] A. YarKhan, J. Kurzak, and J. Dongarra, *QUARK Users' Guide: QUeueing And Runtime for Kernels*, University of Tennessee Innovative Computing Laboratory Technical Report ICL-UT-11-02, Knoxville, TN, 2011.