Massively Parallel Dimension Independent Adaptive Metropolis

Thesis by

Yuxin Chen

In Partial Fulfillment of the Requirements

For the Degree of

Masters of Science

King Abdullah University of Science and Technology, Thuwal,
Kingdom of Saudi Arabia

May, 2015
The thesis of Yuxin Chen is approved by the examination committee

Committee Chairperson: David Keyes
Committee Member: Mikhail Moshkov
Committee Member: Xin Gao
Committee Member: Hatem Ltaief
ABSTRACT

Massively Parallel Dimension Independent Adaptive Metropolis

Yuxin Chen

This work considers black-box Bayesian inference over high-dimensional parameter spaces. The well-known and widely respected adaptive Metropolis (AM) algorithm is extended herein to asymptotically scale uniformly with respect to the underlying parameter dimension, by respecting the variance, for Gaussian targets. 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 targets is enabled, via GPU-accelerated numerical libraries and periodically synchronized concurrent chains (justified a posteriori). Asymptotically in dimension, this massively parallel dimension-independent adaptive Metropolis (MPDIAM) GPU implementation exhibits a factor of four improvement versus the CPU-based Intel MKL version alone, which is itself already a factor of three improvement versus the serial version. The scaling to multiple CPUs and GPUs exhibits a form of strong scaling in terms of the time necessary to reach a certain convergence criterion, through a combination of longer time per sample batch (weak scaling) and yet fewer necessary samples to convergence. This is illustrated by efficiently sampling from several Gaussian and non-Gaussian targets for dimension $d \geq 1000$. 
ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor, Professor David Keyes, whose expertise, understanding, and patience, added considerably to my graduate experience, and led me into this interesting field.

My special and genuine thanks have to go out to Dr. Kody Law and Dr. Hatem Ltaief. I appreciate their various expertise and skills in many areas (e.g., vision, ethics, interaction with participants), and their assistance in writing reports (i.e., academic proposals, research paper, and this thesis). I own Dr. Kody a big thanks for his contribution in this thesis. Without Dr. Kody’s patient guidance, I would not be able to build my basic knowledge in the field of uncertainty quantification, then start to implement improved Metropolis algorithms using high performance computing techniques, which resulted in this thesis. I would like to thank Dr. Hatem, who gives me an important guidance on high performance computing. With his help, I became familiar with many skills such as the CUDA programming model, shared-memory Pthread programming model, and the right way to optimize a code.

I am also thankful to Professor Mikhail Moshkov and Professor Xin Gao to being part of my committee members and for the assistance they provided at all levels of the research project.

I would also like to thank my family and my friends, without whose motivation and encouragement, I would not have considered a graduate career in computer science.
# TABLE OF CONTENTS

Examination Committee Approval 2
Copyright 3
Abstract 4
Acknowledgements 5
List of Abbreviations 8
List of Figures 9
List of Tables 10

1 Introduction 11

2 Related Work 15

3 Bayesian Inference in High Dimensions 19
   3.1 General Problem Formulation 19
   3.2 Markov Chain Monte Carlo 21
   3.3 Advanced Metropolis-Hastings Proposals 23
       3.3.1 Random Walk 23
       3.3.2 Preconditioned Crank-Nicolson 24
       3.3.3 Adaptive Metropolis 26
       3.3.4 Dimension-Independent Adaptive Metropolis 26
       3.3.5 Concurrent Chains 28

4 Numerical Experiments 31
   4.1 Description of the Test Cases 31
   4.2 Autocorrelation Assessment 33
5 High Performance Implementation

5.1 Typical CPU-GPU Architecture Ecosystem .......................... 41
5.2 High Performance CPU-GPU Numerical Software Stack .......... 43
5.3 The DIAM Software Framework ...................................... 44
5.4 Massively Parallel DIAM Implementation Challenges .............. 45
5.5 Concurrent Chains using Multithreading .......................... 47

6 Performance Results

6.1 Environment Settings ............................................. 50
6.2 Impact of the $n_{\text{lag}}$ Choice .................................. 51
6.3 Empirical Tuning ................................................ 52
6.4 CPU-GPU Performance Profiling .................................. 53
6.5 Performance Scalability of DIAM ................................. 53
6.6 Performance Scalability of MPDIAM .............................. 58

7 Conclusion

7.1 Summary ...................................................... 60
7.2 Future Research Work ......................................... 61

References .......................... 63
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>adaptive Metropolis algorithm</td>
</tr>
<tr>
<td>BLAS</td>
<td>Basic Linear Algebra Subprograms</td>
</tr>
<tr>
<td>DIAM</td>
<td>dimension-independent adaptive Metropolis algorithm</td>
</tr>
<tr>
<td>DILI</td>
<td>dimension-independent likelihood-informed</td>
</tr>
<tr>
<td>HMC</td>
<td>Hamiltonian Monte Carlo</td>
</tr>
<tr>
<td>MALA</td>
<td>Metropolis-Adjusted Langevin algorithm</td>
</tr>
<tr>
<td>MCMC</td>
<td>Markov chain Monte Carlo</td>
</tr>
<tr>
<td>MH</td>
<td>Metropolis-Hastings algorithm</td>
</tr>
<tr>
<td>MPDIAM</td>
<td>massively parallel dimension-independent adaptive Metropolis algorithm</td>
</tr>
<tr>
<td>ODE</td>
<td>ordinary differential equation</td>
</tr>
<tr>
<td>pCN</td>
<td>pre-conditioned Crank-Nicolson</td>
</tr>
<tr>
<td>PDE</td>
<td>partial differential equation</td>
</tr>
<tr>
<td>RMSE</td>
<td>root mean squared error</td>
</tr>
<tr>
<td>RW</td>
<td>random walk Metropolis algorithm</td>
</tr>
<tr>
<td>SDE</td>
<td>stochastic differential equation</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

4.1 Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_1$. .................................................................................................................. 34
4.2 Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_2$. .................................................................................................................. 35
4.3 Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_3$. .................................................................................................................. 36
4.4 Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_4$. .................................................................................................................. 37
4.5 Comparison of autocorrelation function of AM and pCN on target $\pi_4$ with condition number $O(d^2)$. ................................................................. 38
4.6 Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_5$. .................................................................................................................. 39
4.7 Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_6$. .................................................................................................................. 40
5.1 CPU-GPU hardware architecture [1]. ............................................................. 42
5.2 MPDIAM using the fork and join parallel programming model. .......... 48
6.1 Comparison of number of samples necessary to reach convergence criterion, as a function of $n_{\text{lag}}/d$. ................................................................. 52
6.2 Performance profiling of MKL-based CPU implementation. ............... 54
6.3 Performance profiling of GPU-based implementation. ....................... 55
6.4 Performance scalability to collect $1e5$ samples. .................................. 56
6.5 Loglog plot of performance scalability to collect $1e5$ samples. ......... 57
6.6 Scalability of concurrent chains. ............................................................... 58
LIST OF TABLES

6.1 Hardware description. ................................. 50
6.2 Scaling to concurrent chains. .......................... 59
Chapter 1

Introduction

Recent years have seen an increasing amount of 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 [2], rather than as 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 [3, 4, 5]. In the context of Bayesian inference, this leads to the notion of a Bayesian analogue of the classical inverse problem [6, 7, 8, 9]. Such problems are enormously challenging both algorithmically and computationally, and largely motivate the present work. At the same time, a very similar problem of big data is attracting a lot of attention. In the former 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 [6, 10, 11, 12]. The big data problem directly confronts the case of genuinely high-dimensional posterior distributions, i.e.,
the posterior concentrates with respect to the prior in the whole space [13, 14].

When a probability distribution has been defined over a space of low dimension (from one to four dimensions), we can directly present the associated probability density. This is trivial in one or two dimensions. It is routine in three dimensions, and some tricks may allow us to represent a four-dimensional probability distribution. However, in the problems we mention above, when the model space has a large number of dimensions, representing a probability density is impossible, but we can do something that is largely equivalent in scientific utility: we can sample the probability density. In fact, there are two problems in a Monte Carlo sampling of a probability distribution in a large-dimensional space: (i) locating the regions of significant probability, and (ii) sampling the whole of the regions densely enough. Many sampling methods are available: inversion method, rejection method, sequential realization, Gibbs sampler, Metropolis algorithm, and some more advanced algorithms allowing a high efficiency sampling in non-Gaussian targets (DRAM [15], ASWAM [16], etc). Herein, we present an alternative approach to those above, combining the best of the worlds above without resorting to gradient information. This is especially important when such information is not available.

From the computer science perspective, the fundamental limiting operations that comprise the adaptive Metropolis algorithm (AM) algorithm, and the dimension-independent adaptive Metropolis algorithm (DIAM) extension proposed here, are Level 2 and 3 Basic Linear Algebra Subprograms (BLAS) linear algebra operations, scaling traditionally as $O(d^2)$ and $O(d^3)$, in particular, dense matrix-vector, matrix-matrix multiplication, and Cholesky factorization. These are the operations prevent its use in high dimensions. Indeed, it is shown here that one may impose a lag-time of $O(d)$ between Cholesky factorizations, and hence block updates of the covariance, without extending the converging time. The algorithm is thereby immediately reduced to $O(d^2)$ rather than $O(d^3)$, by reducing the frequency of the latter by a factor
It is also feasible to reduce the cost of the algorithm to $\mathcal{O}(d^2)$ by using low-rank Cholesky updates \cite{17, 18}. In addition to the simple “embarrassingly parallel” strategy above, it is proposed here to use 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 the high level 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 the high rate of compulsory cache misses (nearly no data reuse). Accelerators provide a factor of two to three higher bandwidth compared to standard x86 architecture and, therefore, memory-bound kernels can still be accelerated on such hardware. All these assume that the data resides already on the GPU memory, which is not always the case for scientific applications. 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 magnitude lower bandwidth than the GPU. Data transfer may then appear to be very expensive, especially in the context of multiple GPU system. It is illustrated that by distributing the Level 2 and 3 BLAS operations across the GPU, the scaling is hidden and the algorithm scales like $\mathcal{O}(d)$ for $d \leq 5000$. For $d > 5000$ the quadratic scaling appears to resume. It should be noted that only prototype targets are considered here, so the cost of the forward model is $\mathcal{O}(1)$. In general, this cost will be seen by the algorithm once it surpasses $\mathcal{O}(d)$, and will be the limiting cost, thereby providing an optimal algorithm.

The rest of the thesis is organized as follows. Section 2 presents some related work. In Section 3 the problem of Bayesian inference in high dimensions is introduced pre-
cisely, 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 the advanced GPU acceleration techniques implemented here 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.
Chapter 2

Related Work

Either in solving an inverse problem or a big data problem, for probability measures that cannot be sampled from exactly, the workhorse of Bayesian computation is Markov chain Monte Carlo (MCMC) [19, 20]. A popular and versatile MCMC algorithm is the Metropolis-Hastings algorithm (MH), introduced in [21] and later revised to its current form by [22]. The adaptive Metropolis algorithm AM [23], and derivatives thereof (DRAM [15], ASWAM [16], SCAM [24], RAM [18], 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 algorithms for relatively low-dimensional and reasonably well-behaved targets, for example unimodal up to a dimension of 100. For targets with strong local dependence, gradient-based proposals such as Metropolis-Adjusted Langevin algorithm (MALA) [25, 26], Hamiltonian Monte Carlo (HMC) [27, 28] or their manifold extensions [29] can improve the convergence time, at the cost of providing the gradients, which may be nontrivial to obtain. It can be shown that such proposals, as well as the random walk Metropolis algorithm (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 RW, MALA, and HMC algorithms must scale as $O(1/d)$ [30, 31, 32], $O(d^{-1/3})$ [33, 34], and $O(d^{-1/4})$
This naturally translates to decorrelation time of the inverse order, i.e., the number of steps required to obtain an almost independent sample is $O(d)$, $O(d^{1/3})$, and $O(d^{1/4})$ [36]. For higher-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 [37, 38]. 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 [39]. 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 [40, 41], or $O(1)$. Recently the work [11] introduces an algorithm that incorporates Hessian information into function-space proposals. The work [10] goes one step further, using prior-preconditioned Hessian information to adaptively identify the space of posterior concentration, and then adapting empirical covariance information within that low-dimensional space.

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. Indeed the possibility of avoiding the man-hours required to construct such code is therefore highly valuable, and provides good motivation for constructing non-intrusive, black-box, or gradient-free algorithms. This work presents an alternative approach to those described above, combining the best of the worlds above without resorting to gradient information. Indeed, the pre-conditioned Crank-Nicolson (pCN) proposal of [41] arises from a Crank-Nicolson discretization of an Ornstein-Uhlenbeck SDE preserves a certain Gaussian measure. In contrast, the RW proposal arises from an Euler-Maruyama discretization of a diffusion which spreads mass to infinity and has no invariant measure. It is this property which provides the $O(1)$ convergence time of the former versus the $O(d)$ 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 that preserves a distribution proportional to the empirical Gaussian obtained from past samples, yielding an asymptotically dimension-independent adaptive metropolis 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 converging time for root mean squared error (RMSE) quantities. Therefore, the value is still limited as long as one is limited to $d = 100$.

Another important limiting factor impeding its use comes from the expensive computational cost. The clock frequency of a single processor of CMOS 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 in favor of increasing concurrency. And this trend has been reinforced moving forward with the international exascale roadmap [42], where 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 (embarrassing) parallelization is therefore seeing an increase in value. While traditional Monte Carlo methods enjoy this property, Markov chain Monte Carlo 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 [43, 44], 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 [45, 46], although neither work confronts high dimension. In [45] the objective is to sufficiently explore the state-space in order to identify a partition for regional adaptation. In [46] this approach is used to 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. It decreases the convergence time of the empirical covariance by concatenating samples from the concurrent chains through periodic synchronization. This gain makes up for the price of a slight slow-down in the collection of a given batch of samples. It is shown that this allows black-box sampling of targets over very high dimensions. Many more elaborate approaches to parallelization of Bayesian computation have recently emerged, including [47, 48, 49, 50, 51, 52, 53].

The authors in [47] and [51] developed a CUDA kernel to tackle the most time-consuming phase of their MCMC simulation using SIMD 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 be noted that more advanced Monte Carlo methods exist, such as population-based MCMC [54, 55], equi-energy sampler [56], and sequential Monte Carlo samplers [57]. Such methods are indeed necessary for sampling from very complex multi-modal distributions, but it should be noted that Metropolis-Hastings 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.
Chapter 3

Bayesian Inference in High Dimensions

3.1 General Problem Formulation

The problem considered here is the following. Given a measure $\eta$ defined in a finite-dimensional space $\mathbb{R}^d$, if $\eta$ is normalized to unity: $\int_{\mathbb{R}^d} \eta(x) dx = 1$, $\eta$ is termed a probability over space $\mathbb{R}^d$. However, in general cases, $\int_{\mathbb{R}^d} \eta(x) dx \neq 1$, so we define: $\pi(x) = \eta(x)/Z$ with $Z = \int_{\mathbb{R}^d} \eta(x) dx$, and $\pi$ is a probability density: $\pi : \mathbb{R}^d \to \mathbb{R}_+ \forall x \in \mathbb{R}^d, \pi(x) \geq 0, \int_{\mathbb{R}^d} \pi(x) dx = 1$. The notation “$x \sim \pi$” denotes that the random variable $x$ follows the distribution $\pi$. Given the quantities of interest $\varphi : \mathbb{R}^d \to \mathbb{R}$, we can obtain estimates of expectations of quantities of interest

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

The convergence of the approximation is a consequence of the Law of large numbers for independent identically distributed (i.i.d.) $x_i$ [58], and an extension thereof under an assumption of sufficient decay of correlation.

Such problem often arises in a Bayesian context, in which case one has some observation $y|x \sim L(x;y)$, where $L(x;y)$ is the likelihood coming from the forward
function connecting data \( y \) and \( x \)

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

where \( \pi_0 \) 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.

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

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

where \( d\mu/d\mu_0 \) denotes the Radon-Nikodym derivative of \( \mu \) with respect to \( \mu_0 \), 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 \( c \in (0, \infty) \) \cite{6}. This context will not be considered further, however this is the problem to have 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 it has come to refer in the statistics community to the case of large \( d_y \) \cite{13, 14}, 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 \( \mathcal{K} : \mathbb{R}^d \times \sigma(\mathbb{R}^d) \to \mathbb{R}^+ \), where \( \sigma(\mathbb{R}^d) \)
refers to the sigma algebra of measurable sets in \( \mathbb{R}^d \). Let \( \mathcal{F}(\mathbb{R}^d) \) denotes the set of
probability densities over \( \mathbb{R}^d \), i.e., functions \( p \) of the form \( p : \mathbb{R}^d \to \mathbb{R}^+ \) and such that
\( \int_{\mathbb{R}^d} p(x)dx = 1 \). By the definition of Markov kernel, for \( q \in \mathcal{F} \), one has that \( p(y) = \int_{\mathbb{R}^d} q(x) \mathcal{K}(x, y)dx \in \mathcal{F} \). The following short-hand notation is therefore commonly used
for the former \( p = q \mathcal{K} \), 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 representing probability distributions, column vectors representing
quantities of interest, and the transition kernel given by a row stochastic matrix. A
density \( \pi \) such that \( \pi = \mathcal{K}\pi \) is referred to as (the density of) an invariant measure,
and a sufficient condition is reversibility

\[
\pi(x)\mathcal{K}(x, x') = \pi(x')\mathcal{K}(x', x). \tag{3.4}
\]
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 \( \pi \), hence such \( \{x_{n-M}\}_{n=M+1}^N \) 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

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

(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 [58] for i.i.d., 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 Metropolis-Hastings (MH) algorithm, introduced in [21] and refined to its present version in [59], is perhaps the most popular and versatile amongst the MCMC methods. It states that an essentially arbitrarily chosen transition kernel \( \mathcal{Q} \) [39] can be composed with an accept/reject step as follows in order to satisfy reversibility (3.4) with respect to \( \pi \). 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

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

(3.6)
where the acceptance probability $\alpha$ is defined as

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

(3.7)

There are clearly infinitely many possible choices of $Q$, which leads to a wide range of behaviors of the associated kernels $K$. Essentially one aims to minimize the correlation between the subsequent samples, which in terms results in a smaller $\rho$ in (3.5) above and hence smaller $\Theta$ and larger effective sample size $N_{\text{eff}}$. The Metropolis-Hastings 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 1.

### 3.3 Advanced Metropolis-Hastings Proposals

This subsection will focus on the MH algorithm introduced in the previous subsection. The most basic Metropolis-Hastings proposal will be introduced (indeed the Metropolis algorithm), followed by the more advanced black-box, or gradient-free, algorithms which were mentioned in Sec. 1. Finally, the algorithm introduced in the present work will be defined.

#### 3.3.1 Random Walk

The presentation begins with the SDE

$$dx = AdW$$  

(3.8)

where $A \in \mathbb{R}^{d \times d}$ is positive definite and $dW$ is an independent increment of Brownian motion $dW \sim N(0, dt)$ [60]. An Euler-Maruyama discretization of this equation with
step-size $\beta$ (time-step $\beta^2$) gives [61]

$$x_{n+1} = x_n + \beta AW_n,$$

(3.9)

where $W_n \sim N(0,1)$ and $W_n \perp W_m$ for all $n,m$. The standard random walk (RW) is defined by the above equation so that $Q(x_n, x_{n+1}) = Q(x_{n+1}, x_n) \propto \exp\{-\frac{1}{2\beta^2} |A^{-1}(x_n - x_{n+1})|^2\}$. The fact that the proposal density is symmetric means that $\pi(x, x') = \pi(x')/\pi(x)$. 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 or some other approximations of the covariance of the target.

### 3.3.2 Preconditioned Crank-Nicolson

In turn, the Ornstein-Uhlenbeck process is defined by the following SDE

$$dx = Bx dt + \sqrt{2}BA dW,$$

(3.10)

where $A$ is as above, $B$ is symmetric and positive definite, $\sqrt{B}$ denotes the symmetric matrix square root, and assume that $BA = AB$. It can be shown that the above equation has 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 [40, 41] 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 $\delta$ (upon multiplication by 2):

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

(3.11)
Setting step-size $\beta = 2\sqrt{2\delta}/(2 + \delta)$, one has the pre-conditioned Crank-Nicolson (pCN) proposal $[41]$
\[ x_{n+1} = \sqrt{1 - \beta^2}x_n + \beta AW_n, \]  
(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 if $p$ is the density of $N(0, AA^\top)$ then $p = pQ$, which in turn implies $p(x)Q(x, x') = p(x')Q(x', x)$. 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. As there is nothing intrinsically finitedimensional about (3.10), or its temporal discretization (3.12), 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 $\mu_0$ 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 $[62]$. By observing that the form of (3.12) may be extended with operators $B$ replacing the scalar $\beta$, the work of $[11]$ introduced general operator-weighted proposals which are reversible with respect to priors of the form $N(m, AA^\top)$:
\[ x_{n+1} = m + A(I - BB^\top)^{1/2}A^{-1}(x_n - m) + ABW_n. \]  
(3.13)

For the above proposals, Hessian information may be incorporated if it is available, and this was the strategy of [11]. This was extended to more general proposals including also gradient information, and given the general name of dimension-independent likelihood-informed (DILI) proposals in $[10]$.

It has been shown in $[30]$ that for proposals of the form (3.9) one must have $\beta^2 = O(1/d)$, thereby leading to a decorrelation-time of $O(d)$. In turn, by virtue of being defined in the function-space limit, the proposals described above have $\beta = \ldots$
\(\mathcal{O}(1)\) with respect to parameter dimension. Of course, the effective data dimension, i.e., the effective dimension of the likelihood, or the dimension of the subspace of posterior concentration with respect to the prior, will indeed still play a role for the above proposals, although it can be mitigated for dimension-independent likelihood-informed (DILI) proposals, i.e., 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 adaptive Metropolis (AM) algorithm [23]. Let

\[
C_n = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^\top - m_n m_n^\top 
\]

and choose \(A_n\) such that \(A_n A_n^\top = C_n\). Plugging this into (3.9) yields the classical adaptive Metropolis proposal. The work [30] identifies an optimal acceptance rate of 0.234, and the later work [16] proposes to scale adaptively the step-size within the AM algorithm to target such acceptance ratio. This will be the version of AM considered here.

### 3.3.4 Dimension-Independent Adaptive Metropolis

The new algorithm introduced here is already alluded to in [11]. This 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 takes the form

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

The reference point \( x_{\text{ref}} \) may be chosen as the maximum \emph{a posteriori} (MAP) estimator, i.e., \( x_{\text{MAP}} = \arg \max_x \pi(x) \), if this is available. Or else, it may be adapted to the empirical mean \footnote{In this case, it should be set to zero until some sufficiently large \( n \) to avoid abrupt jumps of the pivot.}. It is worth dwelling on several points which make this proposal, and the resultant MH algorithm attractive:

- This proposal asymptotically targets \( N(u_{\text{ref}}, C_\infty) \), which is the best Gaussian approximation of the target in case \( u_{\text{ref}} = m_n \rightarrow m_\infty \), for example as measured by Kullback-Liebler (KL) distance \cite{63}.

- As \( \beta \rightarrow 1 \), the algorithm converges to the independence sampler. Hence, for a Gaussian target, it is easy to see that the acceptance probability approaches 1, following from the previous point.

- In the non-Gaussian case, the variance of the proposals will asymptotically coincide with the variance of the target, for \( \beta = \mathcal{O}(1) \). In turn, the variance of the proposals from the AM algorithm will be \( (1 + \beta^2)C_n \). So in order for its trace, i.e., the expected \( \ell^2 \) norm of the AM proposals, to be on par with the target, one will necessarily need to choose \( \beta^2 = \mathcal{O}(1/d) \).

- Based on the above, for a Gaussian target, the asymptotic decorrelation-time of the new algorithm is \( \mathcal{O}(1) \) as opposed \( \mathcal{O}(d) \) for the AM algorithm. The new algorithm will hence be called dimension-independent adaptive Metropolis (DIAM).
For nonlinear/non-Gaussian targets, it will be necessary to modify the above with some additive inflation factor $\alpha > 1$ as follows

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

Notice that as long as $\alpha \in [1, 2]$, one still has a smaller proposal covariance than in the AM case, and therefore expects improved performance.

### 3.3.5 Concurrent Chains

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, the works [43, 44] 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 \textit{a posteriori} to merge the samples from the different chains.

#### Logistics

Denote $P$ chains by $\{x^p\}_{p=1}^P$. $k$ denotes the number of batches that have been done. Each of these is run for $M$ intervals of length $n_{\text{lag}}$ and the local first two moments are collected periodically.

$$S^p_{k,m} = \frac{1}{mn_{\text{lag}}} \sum_{i=1}^{mn_{\text{lag}}} x^p_i (x^p_i)^\top$$ (3.18)

$$= \frac{(m-1)n_{\text{lag}}}{mn_{\text{lag}}} S^p_{k,m-1} + \frac{n_{\text{lag}}}{mn_{\text{lag}}} \sum_{i=(m-1)n_{\text{lag}}+1}^{mn_{\text{lag}}} x^p_i (x^p_i)^\top$$

$$m^p_{k,m} = \frac{1}{mn_{\text{lag}}} \sum_{i=(m-1)n_{\text{lag}}+1}^{mn_{\text{lag}}} x^p_i$$ (3.19)

$$= \frac{(m-1)n_{\text{lag}}}{mn_{\text{lag}}} m^p_{k,m-1} + \frac{n_{\text{lag}}}{mn_{\text{lag}}} \sum_{i=1}^{mn_{\text{lag}}} x^p_i.$$
After each $n_{\text{lag}}$ update, local updates of the global moments are made

\[
S^{p,\text{glob}}_{k,m} = \frac{kMPn_{\text{lag}}}{(kMP + m)n_{\text{lag}}} S^{\text{glob}}_k + \frac{mn_{\text{lag}}}{(kMP + m)n_{\text{lag}}} S^p_{k,m},
\]

\[
m^{p,\text{glob}}_{k,m} = \frac{kMPn_{\text{lag}}}{(kMP + m)n_{\text{lag}}} m^{\text{glob}}_k + \frac{mn_{\text{lag}}}{(kMP + m)n_{\text{lag}}} m^p_{k,m},
\]

followed by a local update of the global covariance

\[
C^{p,\text{glob}}_{k,m} = S^{p,\text{glob}}_{k,m} - m^{p,\text{glob}}_{k,m} (m^{p,\text{glob}}_{k,m})^\top.
\]

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

\[
S^{\text{glob}}_k = \frac{(k-1)}{k} S^{\text{glob}}_{k-1} + \frac{1}{kP} \sum_{p=1}^P S^p_{k-1,M},
\]

\[
m^{\text{glob}}_k = \frac{(k-1)}{k} m^{\text{glob}}_{k-1} + \frac{1}{kP} \sum_{p=1}^P m^p_{k-1,M}.
\]

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 it is outside the scope of the present work.

**Potential Scale Reduction Factor**

As mentioned above, the potential scale reduction factor (PSRF) convergence diagnostic will be used for a posteriori justification of chain merging. It is defined as follows. Start with $P$ chains that are over-dispersed with respect to the target. Define
the following within-chain quantities for each $p$ as follows

$$
S_{MK}^p = \frac{1}{K} \sum_{k=1}^{K} S_{k,M}^p,
$$

$$
m_{MK}^p = \frac{1}{K} \sum_{k=1}^{K} m_{k,M}^p,
$$

(3.25)

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

$$
B_i = \frac{MKn_{\text{lag}}}{P - 1} \sum_{p=1}^{P} (m_{MK}^p - m_{MK}^\text{glob})_i^2,
$$

(3.26)

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

(3.27)

where $C_{MK}^p = S_{MK}^p - (m_{MK}^p)(m_{MK}^p)^\top$. The first quantity is referred to as the *between-chain* variance, representing (a factor $MKn_{\text{lag}}$ times) the variance of 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

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

(3.28)

The PSRF in this $i^{th}$ 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 [43, 44] for further details.
Chapter 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 matrix has a spectrum with maximum eigenvalue $\mathcal{O}(d)$ and minimum eigenvalue close to zero ($r = d$) or zero ($r < d$). To mimic the case of a posterior distribution, with standard normal prior, the covariance is set to be of the form $C = (B + I)^{-1}$, which has smallest eigenvalue $\mathcal{O}(1/d)$ and largest close to 1. The following “twisting” function is introduced

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

which allows the construction of simple “banana” shaped functions with exactly computable moments. The following four Gaussian cases and two non-Gaussian cases are considered:

- $\pi_1 = N(0, (B + I)^{-1})$, $r = d$ (full-rank, cond($C$) = $\mathcal{O}(d)$),
\[ \pi_2 = N(0, (B/d + I)^{-1}), \ r = d, \ \text{(full-rank, cond}(C) = \mathcal{O}(1)), \]

\[ \pi_3 = N(0, (B + I)^{-1}), \ r = d/10, \ \text{(low-rank, cond}(C) = \mathcal{O}(d)), \]

\[ \pi_4 = N(0, V \text{diag}[(\sigma^2 n^{-4} + 1)^{-1}]_{n=1}^d V^\top), \ \text{(full-rank, cond}(C) = \mathcal{O}(\sigma^2)) \]

\[ \pi_5 = \pi_1 \circ V \circ \phi \circ V^\top, \ b_i = 0.3\sigma_i^{-2}/\sqrt{d}, \ \text{(non-Gaussian, mildly twisted)}, \]

\[ \pi_6 = \pi_1 \circ V \circ \phi \circ V^\top, \ b_i = 2\sigma_i^{-2}/\sqrt{d}, \ \text{(non-Gaussian, strongly twisted)}, \]

where \( C = V \Sigma V^\top \) is the ordered eigendecomposition of \( C \), such that the first eigenpair corresponds to the smallest eigenvalue of \( C \). Notice that the Jacobian determinant of \( \phi \) is 1, so a change of variables is trivial. Also, one can compute the maximizer of \( \pi_j \) for all \( j \) and it is 0. Furthermore, the mean and variance of \( \pi_5 \) and \( \pi_6 \) for \( i = 2, 4, \ldots, d/10 \) are given by

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

where \( \sigma_i^2 \) are the variances of the \( i^{\text{th}} \) component under \( \pi_1 \circ V^\top \), i.e., the \( i^{\text{th}} \) diagonal element in \( \Sigma \). For the last two non-Gaussian distributions it will be necessary to define \( \alpha = 1.5 \) in (3.17).

The above targets are all randomly generated, chosen to mimic certain problems that arise in practice. We fix a modestly high dimension \( d=100 \). \( \pi_1 \) may be an example of big data problem, where we reduce the dimension to \( d \), but could be complicated and highly anisotropic because of big a condition number. \( \pi_2 \) is generated by deliberately reducing the condition number from \( \mathcal{O}(d) \) to \( \mathcal{O}(1) \), thus making a clear comparison with \( \pi_1 \) to show how condition number could affect the algorithm efficiency. \( \pi_3 \) could be an example from Bayesian inverse problem context, where the posterior is low-rank with respect to the prior. \( \pi_4 \) could be an example of a smoothing inverse problem with decaying spectrum, and the data comes from PDE forward solve. \( \sigma \) corresponds to 1/variance on the data and smaller variance implies
bigger condition number, which makes this distribution harder to sample from. \(\pi_5\) and \(\pi_6\) are non-Gaussian distributions. \(\pi_5\) is a mildly twisted Gaussian and \(\pi_6\) is a strongly twisted Gaussian.

4.2 Autocorrelation Assessment

From Figure 4.1 to Figure 4.7 we compare the numerical performance of the DIAM with AM, pCN, RW Metropolis algorithms by looking into their autocorrelation functions with underlying distributions: \(\pi_1\) through \(\pi_6\) for \(d = 100\). We try to be fair in choosing the proposal distributions for the random RW, pCN and AM. In the cases \(\pi_1\) through \(\pi_4\), we choose the corresponding 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, and use Gaussian proposal distribution with known covariances together with the optimal scaling: \(2.4/\sqrt{d}\), which is detailedly illustrated and proved in [43]. In Figure 4.1 - Figure 4.7, i.e., \(\pi_1 - \pi_6\), the autocorrelation function is evaluated in the eigenvector directions associated with the 10 largest (red lines), 10 middle (green lines), and 10 smallest (blue lines) eigenvalues of the target’s covariance matrix, as well as the logarithm of the target as a single global measure (black lines). We could expect DIAM to do the best and RW to do the worst. The performance of pCN and AM could be subtle since, on the one hand, pCN is dimension-independent but isotropic algorithm and may become competitive in high-dimensional and well-conditioned cases, on another hand, the AM algorithm performs equally in all directions, although suffers from a \(\mathcal{O}(d)\) dependence on the dimension, and, therefore, it thrives with pCN for targets whose covariance has a large condition number. Numerical experiments confirm this behavior.

Thus we could expect pCN to win in the case that the dimension is rather large, but the covariance is rather isotropic This is the case in \(\pi_2\), and the intuition is
confirmed in Figure 4.2.

Figure 4.1: Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_1$.

Figure 4.1 is the comparison of the performance of the DIAM, AM, pCN and RW with a fairly optimal acceptance rate. The target distribution is $\pi_1$. It is clear that DIAM performs the best and RW performs the worst. pCN only has better decorrelation performance than AM in the directions corresponding to the 10 smallest eigenvalues of the target covariance, as expected.
In Figure 4.2, the autocorrelation function of DIAM, AM, pCN, RW are drawn for target $\pi_2$. It is not surprise that DIAM performs the best and RW performs the worst. pCN wins AM in this case, since pCN is the isotropic algorithm and become competitive in high dimensions problems.
In Figure 4.3, the autocorrelation function of DIAM, AM, pCN, and RW are drawn for target $\pi_3$. DIAM performs the best and RW performs the worst. AM has better overall performance than pCN, but pCN still has superior de-correlation speed in the 10 smallest directions.
In Figure 4.4, the autocorrelation function of DIAM, AM, pCN, RW are drawn for target $\pi_4$. DIAM performs the best and RW performs the worst. pCN has better overall performance than AM here, presumably because the dimension is playing a larger role than the isotropy.
In Figure 4.5, we increase the condition number of $\pi_4$ from $O(d)$ to $O(d^2)$ deliberately to illustrate that AM thrives in comparison to pCN when the condition number of the covariance is large enough.
Figure 4.6: Comparison of autocorrelation function of DIAM, AM, pCN, and RW on target $\pi_5$.

Figure 4.6 compares the performance of the DIAM algorithm, AM algorithm, pCN algorithm and RW algorithm with a fairly optimal acceptance rate. The target distribution is $\pi_5$, which is a mildly twisted Gaussian distribution. It is clear that DIAM performs the best and RW performs the worst. pCN only has better decorrelation performance than AM in the directions corresponding to the 10 smallest eigenvalues. The behavior is similar to the corresponding Gaussian $\pi_1$ presented in Figure 4.1.
In Figure 4.7, the autocorrelation of the various algorithms is presented above for a strongly twisted Gaussian target $\pi_6$. For this highly nonlinear target the DIAM algorithm performs comparably to the AM algorithm. It may be possible to separate the nonlinear part of the space, and implement ordinary DIAM on the linear part to retain the dimension-independent performance (see comments in Section 7.2).
Chapter 5

High Performance Implementation

In this section, we describe the high performance implementation of the massively parallel dimension-independent adaptive Metropolis algorithm (MPDIAM) 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 multicore processors associated with accelerators through a weak link called the Peripheral Component Interconnect Express (PCIe), as depicted in Figure 5.1. 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 compared to the standard x86 CPU. Moreover, the speed to fetch data from GPU main memory is roughly more than two times higher than the standard x86 CPU’s bandwidth. 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 maintaining 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 order of magnitude,
compared to 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 have to 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 in three levels: Level 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 multiplication). 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) [64]. On GPUs, NVIDIA provides the cuBLAS library [65], which implements BLAS kernels using the CUDA programming model [66]. The open-source KAUST BLAS (KBLAS) library [67] provides also a subset of Level 2 BLAS operations on GPUs, which performs better than the corresponding kernel from NVIDIA cuBLAS. Last but not least, LAPACK [68] 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

Below is the pseudo-code of DIAM with the target to sample \( \pi_1 = N(0, (B + I)^{-1}) \), and the dimension of the target is \( N \).

\[
D=I, \ j=1; \ ST=\text{zeros}(N, \ nlag); \ nacc=0; \\
\text{while} \ (\text{criterion is not satisfied}) \\
\quad \text{if} \ (j \% nlag) == 1 \\
\quad \quad \text{randmatrix} = b*\text{randn}(N, \ nlag); \\
\quad \end{end} \\
\quad x = zm + \sqrt{1-b^2}\cdot \text{ad}\cdot (xo-zm) + \text{randmatrix}(:,(j-1)\%nlag); \\
\quad a = \text{rand}; \\
\quad \text{if} \ \log(a) < 1/2\times((xo'\cdot(B+I)\cdot xo) + x'\cdot\text{inv}(D)\cdot x \\
\quad \quad - (x'\cdot(B+I)\cdot x) - xo'\cdot\text{inv}(D)\cdot xo) \\
\]
xo = x;
nacc = nacc + 1;
end
ST(:, mod(j-1, nlag)+1) = xo;

if (j\%nlag == 0)
    m = sum(ST, 2) / (j+1) + (j+1-nlag) / (j+1)*m;
    S = (ST*(ST))' / (j+1) + (j+1-nlag) / (j+1)*S;
    C = S - m*m';
    D = chol(C)';
    acc = nacc/nlag;
    nacc = 0;
    if acc > .5
        b = min(1, 1*b, 1);
    else if acc < .3
        b = max(0.9*b, 1/10/sqrt(N));
    end
end
j = j+1;
end

From the pseudo-code, the DIAM framework is basically composed by the following
Level 2 and 3 BLAS functions:

- 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) \( A = U^TU \) 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 Massively Parallel DIAM 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 Equation (3.14) and the dense matrix-vector multiplication, as highlighted in Equation (3.16). On the one hand, the Cholesky-based matrix inversion is compute-intensive and at the same time its complexity may impede performance scalability of the overall MPDIAM approach if frequently requested for solving high-dimension problems. On the other hand, the dense matrix-vector multiplication is memory-bound and, therefore, exhibits a lower arithmetic complexity and slows our MPDIAM implementation down if it becomes predominant. The lag-time is then paramount to balance these two operations and to further reduce the time to solution. We rely on existing high-performance implementations of both operations: using the MKL library [64] for the Cholesky-based matrix inversion and the KBLAS library [67] for the dense matrix-vector multiplication. The remaining kernels are executed on the CPU using Intel MKL [64]. This hybrid CPU-GPU implementation requires the 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 communication can be overlapped with computation, through
the CUDA programming model using the function CUDA_MEMCPY_ASYNC. Below is the MPDIAM pseudo-code highlighting the asynchronous execution using GPU:

```plaintext
D=I, j=1; ST=zeros(N, nlag); nacc=0; xo=zeros(N,1); m=zeros(N,1);
// Asynchronous transfer B to d_B, I to d_I, D to d_inv(C), m to d_m, and xo to d_xo
while (criterion is not satisfied)
    if (j%nlag == 1)
        randmatrix = b=1+randn(N, nlag);
    end
    x = zm + sqrt((1-b^2)^a*(xo-zm) + randmatrix(:,(j-1)%nlag));
    // Asynchronous transfer x to d_x
    a = rand;
    // Check if d_m transfer has finished, if not, wait until finished
    lo = ((d_xo-zed_m)'*d_B+d_I)*((d_xo-zed_m));
    // Check if d_inv(C) transfer has finished, if not, wait until finished
    po = (d_xo-zed_m)*d_inv(C)*d_xo-zed_m);
    // Check if d_x transfer has finished, if not, wait until finished
    l = (d_xz)'*d_B+d_I)*d_x;
    p = ((d_xz-zed_m)*d_inv(C)*d_xz-zed_m);
    if log(a) < 1/2*(lo+1+po-po)
        xo = x;
        nacc = nacc + 1;
    end
    ST(:,mod(j-1,nlag)+1) = xo;
end
```

5.5 Concurrent Chains using Multithreading

The degree of parallelism of MPDIAM can be further leveraged by running concurrent chains (see Section 3.3.5). 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 dimension independent adaptive Metropolis algorithm. Once a number of $P$ threads created, each thread $p$ will have its own private memory containing all needed information to independently process, as depicted in Figure 5.2. In Figure 5.2, $k$ denotes the number of batches have been done. At the end of each batch processing, threads join is forced by synchronizing on a shared-memory lock to busy wait on all running threads. Comparing to MPDIAM, exchanging information after each batch is not required in concurrent chain pCN and concurrent chain RW, and they terminate as long as one of chains reaches the appropriate criterion. To make that happen, we need a indicator and let all chains spin on that until one of the threads reaches criterion, then takes the lock, and terminates all other threads.

This second level of parallelism introduces another complexity on the CPU because it mixes threads created by the Intel MKL library (OpenMP) as well as the concurrent chains (Pthreads). Indeed, MKL implements multithreading in BLAS functions and the default number of threads MKL uses corresponds to the number of physical cores available on the system, except if the environment variable MKL_NUM_THREADS

![Figure 5.2: MPDIAM using the fork and join parallel programming model.](image-url)
is defined by the user. Thus, the total number of threads running in the system is 
\(N_{chains} \times N_{mkl}\), where \(N_{chains}\) is the number of chains launched and \(N_{mkl}\) is the number of threads MKL functions fork. When \(N_{chains} \times N_{mkl}\) is higher than the actually number of cores (\(N_{cores}\)) the system has, the overall performance may drop down because of thread oversubscription. Therefore, it is critical to keep \(N_{chains} \times N_{mkl} \leq N_{cores}\).
Chapter 6

Performance Results

This section presents the performance results of various MPDIAM implementations.

6.1 Environment Settings

Table 6.1 defines the CPU specifications of the computing system used in these experiments. Sustained bandwidth is determined by the Stream benchmark. The total number of cores is 20.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Intel Xeon Ivy Bridge E5-2680 v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2</td>
</tr>
<tr>
<td>Cores/CPU</td>
<td>10</td>
</tr>
<tr>
<td>Clock frequency (GHz)</td>
<td>2.8</td>
</tr>
<tr>
<td>Cache size (MB)</td>
<td>25</td>
</tr>
<tr>
<td>Memory Bandwidth (GB/s)</td>
<td>59.7</td>
</tr>
<tr>
<td>Main Memory (GB)</td>
<td>256</td>
</tr>
<tr>
<td>PCI Express</td>
<td>3.0</td>
</tr>
</tbody>
</table>

The system has three NVIDIA Tesla K40 GPU Accelerators with 1.4 TFLOPS sustained performance, 12 GB memory, and ultra-fast memory bandwidth 288 GB/s each. The machine runs Ubuntu 14.04.1 LTS and provides Intel Compilers Suite v13.0 together with the MKL library. The MPDIAM code is written in C and relies on OpenMP for 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 Impact of the $n_{\text{lag}}$ Choice

The outcome of any MCMC simulation depends, aside the natural variations due to random sampling, on the specific way the run is performed. First of all, 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 we update our proposal, denoted by $n_{\text{lag}}$, is one of the parameters that needs to be tuned.

Since the total chain length required for reliable results increases with the dimension $d$, the test cases, with a known Gaussian objective function, were run separately at various values of $d$. For each $d = 100, 200, 300, 400$, $n_{\text{lag}}$ was systematically paired, while we kept running the program until a certain stopping criteria has been reached. The value of $n_{\text{lag}}$ is chosen as the largest one before the number of generated samples necessary for convergence begins to increase. Therefore, the efficiency is the greatest, since the Level 3 BLAS operations occur every $n_{\text{lag}}$ iterations. The results are plotted in Figure 6.1. All in all, the $n_{\text{lag}}$ parameter allows to trade algorithmic complexity with parallel performance.
Empirical Tuning

One can notice that Level 3 BLAS functions in MPDIAM are only called every \(d/2\) iterations, reducing the algorithm complexity to \(O(d^2)\). The strategy we use here is that when dealing with small problems, e.g., problem sizes smaller than 2000, the optimized Intel MKL [64], is preferred (only CPU), while when dealing with larger problems, e.g., problem sizes larger than 3000, high performance libraries such as cuBLAS [65] and KBLAS [67] are preferred (GPU). This tuning choice helps mitigate the overhead of copying data between the host (CPU) and the device (GPU).
6.4 CPU-GPU Performance Profiling

Figure 6.2 illustrates the performance profiling of the MKL-based DIAM CPU implementation. As the dimension increases, SYMV becomes the bottleneck and impedes DIAM to scale to higher dimensions. However, SYMV is a Level 2 BLAS function and, thus, is limited by the bus bandwidth. As described in Section 5.1, accelerators provide a factor of two to three higher bandwidth compared to standard x86 architecture and, therefore, memory-bound kernels can still be accelerated on such hardware.

![Figure 6.2: Performance profiling of MKL-based CPU implementation.](image)

Figure 6.3 shows the performance profiling of GPU-based DIAM implementation. The dot product function (DOT) takes the majority of time, which has $O(d)$ complexity. The practical implication of Figure 6.3 is that the time complexity is compressed to $O(d)$ by taking advantage of GPU computing capabilities as well as keeping data transfer overhead low, thanks to the asynchronous memory data offloading.
6.5 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 6.4 and Figure 6.5, 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 $\pi_1$. 

Figure 6.3: Performance profiling of GPU-based implementation.
In Figure 6.4, MKL-KBLAS represents program implemented using both MKL and GPU-libraries. MKL is the program implemented only using MKL and MKL sequential is the program written on C with no parallel techniques involved. The time required to collect 1e5 samples of MKL-KBLAS code outperforms that of MKL parallel code after d=2000 and rises more slowly than that of both MKL parallel code and MKL sequential code. Doing regression on all three curves results in:

- 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 $T = 253.63 - 0.3983d + 2 \times 10^{-4}d^2$
In Figure 6.5, it is more easy to see that MKL sequential, MKL parallel, and MKL-KBLAS speed-up in scaling from quadratic, to faster quadratic, to linear then faster quadratic.

Figure 6.5: Loglog plot of performance scalability to collect 1e5 samples.
6.6 Performance Scalability of MPDIAM

In Figure 6.6, as we add more threads, the time used to converge drops monotonically. But after adding 12 threads, the benefit we get from adding more threads is modest.

Based on the illustration of chapter 3.3.5, the more chains run simultaneously, the faster the algorithm to converge. However, this algorithm is memory-bound and needs synchronization after each chain generates a certain number of samples, thus, as long as the memory bandwidth is saturated, adding more threads will have limited benefit because we spend more time in each batch (the interval between each two synchronization see Table 6.2). The reason we call Figure 6.6 “subtle” strong scaling on target $\pi_1$ is that comparing to the 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 (per thread per chain). Despite this, we still call it strong scaling because we use the same convergence criteria, and in terms of the convergence criteria, the whole problem is the same.

The possible reason that fewer samples are required to obtain convergence is that
Table 6.2: Scaling to concurrent chains.

<table>
<thead>
<tr>
<th># threads</th>
<th>Total time (s)</th>
<th>Time per batch (s)</th>
<th>total # samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42517.73</td>
<td>5.9</td>
<td>142260000</td>
</tr>
<tr>
<td>2</td>
<td>22779.92</td>
<td>8.4</td>
<td>107760000</td>
</tr>
<tr>
<td>4</td>
<td>9486.76</td>
<td>8.6</td>
<td>86480000</td>
</tr>
<tr>
<td>6</td>
<td>5878.11</td>
<td>8.9</td>
<td>77640000</td>
</tr>
<tr>
<td>8</td>
<td>4466.00</td>
<td>9.3</td>
<td>73920000</td>
</tr>
<tr>
<td>10</td>
<td>3506.19</td>
<td>9.6</td>
<td>70800000</td>
</tr>
<tr>
<td>12</td>
<td>3215.66</td>
<td>10</td>
<td>70080000</td>
</tr>
<tr>
<td>14</td>
<td>3024.47</td>
<td>10.6</td>
<td>70000000</td>
</tr>
<tr>
<td>16</td>
<td>2962.85</td>
<td>14</td>
<td>70004000</td>
</tr>
</tbody>
</table>

the more chains used from the beginning, the earlier we adapt a better proposal within the same number of batches and thus fewer samples we need in total to get convergence.
Chapter 7

Conclusion

7.1 Summary

A black-box MCMC algorithm is introduced for Bayesian inference in high dimensions, herein named DIAM. In particular, it is illustrated 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 illustrated 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 BLAS3 operations enable the efficient exploration of very high-dimensional targets $d \geq 1000$. 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 up to concurrent chains (justified \textit{a posteriori}) may in principle allow exploration of very high-dimensional targets in the presence of very many GPUs. A form of strong scaling with respect to time is illustrated on up to 16 cores.
7.2 Future Research Work

The work presented in this thesis can be extended in the following directions.

- **Real world examples.** The examples here have been chosen to emulate real-world Bayesian inference problems in high dimensions, however they are toy examples which have been constructed so that the target distribution is known. This allows direct convergence analysis, leaving questions of burn-in and other convergence diagnostics out of the picture so that the algorithms can be evaluated directly without ambiguity. Nonetheless, the objective is to tackle high-dimensional inference problems, for example where the data is related to the parameter of interest by a partial differential equation (PDE). Such problems may arise, for example, in subsurface exploration. In this case, it may cost many person-hours to compute derivatives, in order to improve behavior of sampling algorithms with respect to isotropy of the posterior. This algorithm and its descendants may be very useful for this purpose.

- **Separation of space.** It is observed that as the target becomes strongly twisted, and hence strongly nonlinear, in the example $\pi_6$, and the covariance needs to be inflated, the DIAM algorithm performance begins to degenerate towards that of the AM algorithm. It would be of interest to determine how to separate out the nonlinear part of the space, in such cases as this, where it is only a fraction of the whole space. In this way, the original DIAM algorithm may be used on the complement of this space and the dimension-independence can be retained, which yields the enormous speed-up in comparison to AM, for Gaussian targets. This idea is similar to that of the DILI algorithm [10].

- **Fast low-rank covariance approximation.** This applies to the Bayesian inverse problem context, but not the big data problem. In the case that the posterior is low-rank with respect to the prior, one may attempt to identify
this low-rank space with even fewer than $d$ samples, and then use a splitting together with pCN, similar to [10]. The cost of the algorithm may therefore be reduced to $\mathcal{O}(d)$, the cost of a forward solve, for a certain class of problems.

- **Other parallelization strategies.** Several parallelization strategies have recently emerged for Metropolis-Hastings algorithms, and it would be interesting to explore some of these as well, in a computing world where we are becoming ever more parallel.
REFERENCES


A  Paper Under Preparation