An Unbiased MCMC FPGA-based Accelerator in the Land of Custom Precision Arithmetic

Shuanglong Liu, Grigorios Mingas, and Christos-Savvas Bouganis, *Member, IEEE*

Abstract—Markov Chain Monte Carlo (MCMC) based methods have been the main tool used for Bayesian Inference by practitioners and researchers due to their flexibility and theoretical properties that guarantee unbiased sampling-based estimates. Nevertheless, with the availability of large data sets and the constant need to develop more complex models that better capture the targeted problem, significant computational challenges have been presented. Current approaches, based on multi-core CPUs, GPUs, and FPGAs, aim to accelerate the execution time of the MCMC methods using subsampling techniques or custom precision arithmetic, resulting to biased estimates. In this work, a novel FPGA-based construction is proposed that utilises the custom precision support of FPGA devices in order to accelerate the computations, guaranteeing at the same time asymptotically unbiased estimates. Key to this approach is the extension of the parameter space by an extra parameter that indicates the required precision in the computation of the likelihood of a data point. The work proposes an FPGA architecture for the above algorithm, as well as discuss its tuning for maximising the performance of the system. The performance of the FPGA-mapped sampler is evaluated using two Bayesian logistic regression case studies of varying complexity, which show significant speedups compared to existing FPGA- and CPU-based works that utilise double floating point arithmetic, without any bias on the sampling-based estimates.

Index Terms—Field Programmable Gate Array, Markov Chain Monte Carlo, Custom Arithmetic Precision, Logistic Regression, MNIST Database

1 Introduction

Bayesian methods play a central role in modern Machine Learning mainly due to their ability to capture uncertainty in parameter estimation [1]. A key step in Bayesian inference is the sampling from an arbitrary probability distribution [2]–[5]. Markov chain Monte Carlo (MCMC; Chapter 6 of [3]) method is one of the most popular and successful tools to draw samples effectively from arbitrary probability distributions in Bayesian inference problems. For this reason, it has been widely used in a range of statistical applications, including computational physics, population genetics and statistical classifications [2], [4], [6].

The MCMC algorithms allow sampling from a large class of distributions and scales well with the dimensionality of the sample space. They are often used to tackle the problem of sampling from a probability distribution known up to a normalizing constant, with the purpose of using the generated samples to estimate otherwise intractable integrals (this task is known as Monte Carlo integration). For the estimation of the above integrals, the MCMC algorithms need to estimate how well the data are explained by the sampled parameters (i.e. likelihood function estimation), which becomes the dominant computational bottleneck when large datasets are targeted. Thus, speeding up the likelihood computation has attracted the focus in academia and industry in order to allow the application of MCMC to models with large-scale dataset. Currently, the lack of sufficiently fast MCMC methods limits their applicability in many modern applications like genetics and machine learning, and this situation is bound to get worse given the increasing adoption of big data in many fields of industry and research.

This challenge has motivated approximate MCMC approaches [7]–[10] that are based on approximations of the target distribution. A summary and review of current MCMC methods used for large datasets can be found in [11]. Most of them tend to use subsampling based approaches to provide a faster estimation of the likelihood for only a subset of the whole data [12]–[15]. Other recent works focus on the computation engine, and investigate the calculation of the likelihood function based on custom precision arithmetic in order to achieve low latency and allow for more parallelism for a given set of hardware resources. However, both approaches lead to biased estimates that exhibit large variance due to the approximations of the target distribution. Even though a controlled biased estimate can be accepted in certain applications [16], there is a large number of applications where unbiased estimates [1] of the given parameters of the sampling distribution are required, and MCMC algorithms are expected to perform exact inference in these problems [17].

In this work we are focusing on problems that are compute-bounded, having the evaluation of the likelihood function as the limiting performance factor. Towards addressing this problem, a novel MCMC construction is proposed, the custom-precision firefly MCMC (CF-MCMC), that samples from the exact posterior distribution even though it operates under a custom precision regime, and its implementation in an FPGA device. The key idea behind this work, that enables custom precision arithmetic in the

• *The authors are with the Department of Electrical and Electronic Engineering, Imperial College, London, SW7 2AZ, UK.*
E-mail: s.liu13@imperial.ac.uk

1. Please note that the estimates are at best asymptotically unbiased for MCMC [3]. In the rest of the paper, for brevity and clarity, the term "unbiased" is used instead of "asymptotically unbiased".

computation of the likelihood function, is the introduction of an extra parameter in the problem parameter space that models the mode of the computations, i.e. the arithmetic precision, in the calculation of the likelihood function. The paper shows that by properly sampling the new augmented space, unbiased estimates of the parameter of the distribution are computed even though part of the computations are performed under custom precision arithmetic.

This article extends our prior work [18], by (1) investigating and comparing alternative custom precision likelihood construction approximates targeting improved performance (i.e. effective samples per second) and (2) proposing a method to maximize the performance of the algorithm by selecting the optimal arithmetic precision based on performing short MCMC pre-runs on a set of candidate precisions. A summary of the main contributions of this work are as follows:

- A custom-precision firefly MCMC (CF-MCMC) algorithm which guarantees unbiased sampling and custom precision arithmetic, leading to significant performance gains;
- An optimised FPGA-based implementation of the CF-MCMC algorithm, that capitalises on the nature of FPGA devices to support custom arithmetic precision;
- A novel methodology for the construction of tight lower bound functions of the target probability distribution function based on the selection of the rounding mode of the FPGA arithmetic operators in combination with verification tools for modelling numerical behaviour (i.e. Gappa++) in order to maximise the performance of the proposed algorithm;
- A methodology for selecting the custom arithmetic precision of the system that would maximise its performance based on the system's performance model and the estimates of the parameters from pre-runs.

2 Background

2.1 Markov Chain Monte Carlo

In scientific computing, we often need to compute some integrals in a very high dimensional space such as:

$$I(f) = \int f(\theta)p(\theta)d\theta \tag{1}$$

The probability distribution $p(\theta)$ i.e. the target density, can be a distribution from statistical physics or a conditional distribution arising in data modelling - for example, the posterior probability of a model's parameters given some observed data, where $f(\theta)$ is the function of interest.

In the field of statistics, these integrals are vital for calculating the expectation or expected values of distributions. However many functions and distributions cannot be integrated analytically especially for higher-dimensional integrals. For most probabilistic models of practical interest, these expectations cannot be evaluated by exact methods. In these cases, a general and powerful framework, i.e. the Markov chain Monte Carlo (MCMC) method, is employed, which can be used to generate samples from any given probability distribution. Using the generated samples, the integral $I(f)$ can be approximated by tractable sums that converge (as the number of samples N_s tends to infinity) to $I(f)$. The following central limit theorem holds for suitable test functions f under weak assumptions [19]:

$$\tilde{I}(f) = \frac{1}{N_s}\sum_{n=1}^{N_s} f(\theta_n) \longrightarrow \text{Normal}(I(f), \sigma^2_{lim}(f)) \tag{2}$$

i.e. the sum is an asymptotically unbiased estimator of the integral $I(f)$ [3].

MCMC generates samples from the probability distribution $p(\theta)$ by sequentially constructing a Markov chain that satisfies (2). In practice it is often advisable to discard some initial states of the chain (throwing away a number of iterations at the beginning of an MCMC run is often called "burn-in"), in order to reduce the initialisation bias. In this work, the parameters of interest are denoted by θ of D-dimensions, and it is assumed that N data points $\{x_n\}_{n=1}^N$ (with each component x_n as a vector) have been observed. An MCMC sampler makes transitions from a given θ to a new θ' such that the posterior distribution $p(\theta \mid \{x_n\}_{n=1}^N)$ remains invariant. Consider the most commonly used MCMC algorithm (Metropolis MCMC; Chapter 7.3 of [3]) in Algorithm 1. In each iteration, a proposed move of the chain is considered, by using a proposal such as a Gaussian random walk (line 2) to generate the new θ' that is accepted or rejected with the probability based on the ratio of the posterior probabilities (i.e. how well the new value explains the data) (line 4-9). The main computation load lies in the evaluation of the full posterior probability at every iteration in line 3. Using the Bayesian theorem and assuming that the data $\{x_n\}_{n=1}^N$ are *i.i.d.* (it is often assumed in real applications) and θ has the prior $p(\theta)$, the posterior distribution breaks down into a product of the likelihood of each data point i.e. $p(x_n \mid \theta)$ as:

$$p(\theta \mid \{x_n\}_{n=1}^N) \propto p(\theta)\prod_{n=1}^{N} p(x_n \mid \theta) \tag{3}$$

For notational convenience, we write the nth likelihood term as

$$L_n(\theta) = p(x_n \mid \theta) \tag{4}$$

Although MCMC generates statistically consistent samples from the target distribution, the samples are correlated due to the use of a Markov chain. This dependence leads to an increase in asymptotic variance σ^2_{lim} of the MCMC estimate in (2), compared to the case where independent samples of the target distribution are used. This loss in efficiency can be quantified by the Effective Sample Size (ESS) [20] in (5):

$$ESS = N_s/(1 + 2\sum_{j=1}^{k}\rho(j)) \tag{5}$$

where N_s is the number of post burn-in MCMC samples and $\sum_{j=1}^{k}\rho(j)$ is the sum of the first k monotone sample autocorrelations. The ESS estimates the "effective" number of samples, which is always lower than N_s. Thus the adopted performance metric for MCMC samplers is ESS/sec, which combines raw sampling speed (runtime) and ESS [20].

ALGORITHM 1: Metropolis MCMC

Input: initial setting θ_0, number of samples N_s;
Output: parameter samples $\theta_i, i = 1, ..., N_s$;

```
1:  for $i = 1$ to $N_s$ do
2:      Propose $\theta' \sim \theta_{i-1}$+Normal(0, $s^2 I_D$); // a random walk proposal with step size $s$.
3:      Compute $a = \frac{p(\theta' \mid \{x_n\}_{n=1}^N)}{p(\theta_{i-1} \mid \{x_n\}_{n=1}^N)}$;
4:      $u \sim$ Uniform(0,1);
5:      if $u \leq a$ then
6:          $\theta_i = \theta'$;
7:      else
8:          $\theta_i = \theta_{i-1}$;
9:      end if
10: end for
```

2.2 Likelihood Computation Acceleration

For problems with large data-set, the evaluation of all N likelihoods in (3) dominates the computational cost. The increased computational time lies in the complete scan of the data at each iteration through likelihood evaluations. Significant effort has been recently spent on proposing approximate methods focusing on how to minimize the cost of evaluating the full likelihood.

2.2.1 Data Subsampling

Many recent works are based on subsampling methods. These approaches use subsets of data to provide a faster estimation of the likelihood in which only a fraction of the whole data set is employed to estimate the full likelihood. [12] introduce an approximate Metropolis-Hasting rule with controlled bias that allows accepting or rejecting samples with high confidence using only a fraction of the data. [13] propose an adaptive subsampling technique which is an alternative approximate implementation to [12] that only requires evaluating the likelihood of a random subset of the data. This algorithm is a more robust approach compared to [12] and can provide estimates under a user-controlled error. However, both algorithms in [12] and [13] are approximate, and they rely on a bound for the difference between the log-likelihood contributions at the proposed and current sample, and that of the control variates [21]. [15] propose a subsampling of the data based on the contribution on each likelihood term, which by a bias-correction can be turned into an unbiased estimator of the likelihood function. This algorithm needs to build a surrogate of the true likelihood, using either a Gaussian process or a spline approximation. As such, it requires computing the surrogate likelihood for all data before running the subsampling step, thus introducing another costly requirement. [14] present an auxiliary variable MCMC algorithm that also queries the likelihoods of a small subset of the data but achieves exact posterior distribution. The fundamental assumption of this approach is that each product term in the likelihood can be approximated from below by a function easier to compute. The drawback of this method lies in the construction of these functions as the quality of the bound depends on the target distribution. Furthermore, the authors have demonstrated that an acceleration is only achieved when the approximation is tight enough.

2.2.2 Specialised Hardware Approaches

When running MCMC methods in computational devices such as GPUs and FPGAs, relaxing the requirement for high precision allows the MCMC algorithms to execute faster and with less energy. By utilising low precision (custom floating point) datapaths, it consumes fewer resources and leads to a higher degree of parallelism compared to full precision (double floating point) datapaths for a fixed area resource. As such, recent works that target GPUs and FPGAs have been investigating the utilisation of custom arithmetic precision for the estimation of the likelihood. However, departing from double precision arithmetic for likelihood evaluation leads to biased estimates with respect to systems that employ double precision arithmetic throughout the computations, because of the approximations in each likelihood term. When exact estimates are required, the utilisation of high precision data-paths with lower performance is unavoidable.

Previous works on FPGAs using custom precision can be found in [16], [17], [22], [23]. [23] propose the use of custom precision arithmetic for population-based MCMC methods where multiple parallel chains are used to improve the mixing properties of the chain. In [22], high- and low- precision estimations are compared during run-time using the Kolmogorov-Smirnoff metric and the precision is adapted such that the distribution of the custom precision estimate does not deviate more than a user specified thershold from the high-precision distribution estimate. However, placing such a threshold does not limit the bias in the estimate. [17] employ an auxiliary mixed precision run to correct the bias in the output estimates. However, their method requires knowledge of the function of interest during the design of the system, and as such the generated samples cannot be used for other estimates. [16] propose a method under which they can estimate using short pre-runs of the system the bias of the estimate under various custom precision schemes. The final run is performed utilising the lowest possible precision that does not violate the user's acceptable bias at the estimate. In summary, the above methods do not provide any guarantees of an unbiased estimate, and thus are not applicable in applications where such guarantees are required.

This work is focused on the acceleration of the likelihood function evaluation part of the MCMC algorithm, but in relation to the existing works, the proposed method guarantees unbiased estimates even when custom precision arithmetic is employed for the generation of the samples. The work is based on the underlying idea initially proposed in [14], and we adapt it to the custom hardware world for allowing the use of custom precision approximates in the probability distribution evaluation without leading to biased estimates. To the best of our knowledge, this is the first work to produce and guarantee an unbiased MCMC estimation using mixed precision designs.

3 Mixed Precision MCMC Methodology

3.1 Custom-Precision Firefly MCMC (CF-MCMC)

On each iteration of MCMC, the likelihood term for each data point must be evaluated to obtain the target density, which is the most computation expensive part of the algorithm. [14] proposed Firefly Monte Carlo (FlyMC), which introduces an auxiliary variable for each observation which determines whether it should be included in the exact evaluation of the posterior distribution or not. A lower bound function for each likelihood term caters for the observations that are not included in the evaluation of the posterior, and an extra sampling step is included in the algorithm in order to sample the above indication parameter. As such, FlyMC generates samples from the exact target posterior rather than from an approximation distribution. Nevertheless, the drawback of FlyMC is that useful lower bounds can be difficult to obtain for many problems. Moreover, [14] have shown that the algorithm's performance depends on the tightness of the bound; it only achieves significant gains when computational light and tight bounds are applicable. The idea of using lower bounds to reduce the cost of MCMC has been exploited previously in [24]; [11] (Section 4.3) propose construction of the lower bound that avoids specifying a resampling fraction, but it requires the integrals of the exponents of the lower bound functions to be tractable.

This work is based on the same principle as the FlyMC, but the introduced auxiliary parameter is utilised to indicate whether or not the likelihood computation for each data point is performed under double precision or custom precision regime. Thus, instead of requiring the derivation and use of approximate functions for the likelihood terms, the work utilises custom precision approximations and utilize precision-related tools to guarantee that these approximations are indeed a lower bound to the true likelihood term (which is a requirement for [14] to generate samples from the posterior distribution), removing the need to manually design the approximation function as in FlyMC. Thus, the proposed framework produces lower bounds automatically regardless of the class of problem.

In the rest of the paper, $LD_n(\theta)$ and $LC_n(\theta)$ denote the double precision likelihood term and the custom precision lower bound of the likelihood of the nth data point, respectively. For each data point x_n, a binary auxiliary variable $z_n \in \{0,1\}$ is introduced, indicating the type of the likelihood term computation i.e. double or custom precision. Assuming that $LC_n(\theta)$ has been constructed such as it is always less than the double precision likelihood $LD_n(\theta)$, i.e. $LC_n(\theta) \leq LD_n(\theta)$ (such construction is shown later on in the paper), then each z_n is modelled to have the following Bernoulli distribution conditioned on the relative difference between these two precision values:

$$z_n \sim Bernoulli(1 - LC_n(\theta)/LD_n(\theta)) \tag{6}$$

The augmented posterior distribution is shown below:

$$p(\theta, \{z_n\}_{n=1}^N \mid \{x_n\}_{n=1}^N) \propto p(\theta) \prod_{n=1}^N p(x_n \mid \theta) p(z_n \mid x_n, \theta) \tag{7}$$

As in other auxiliary variable methods, this augmentation does not damage the target distribution in (3):

$$\begin{aligned}
&\sum_{z_1} \dots \sum_{z_N} p(\theta) \prod_{n=1}^N p(x_n \mid \theta) p(z_n \mid x_n, \theta) \\
&= p(\theta) \prod_{n=1}^N p(x_n \mid \theta) \sum_{z_n} p(z_n \mid x_n, \theta) \\
&= p(\theta) \prod_{n=1}^N p(x_n \mid \theta)
\end{aligned} \tag{8}$$

Therefore, the marginal distribution over θ in (7) is still the correct posterior distribution given in Equation (3).

Consider each product of the joint distribution:

$$\begin{aligned}
&p(x_n \mid \theta) p(z_n \mid x_n, \theta) \\
&= LD_n(\theta) [\frac{LD_n(\theta) - LC_n(\theta)}{LD_n(\theta)}]^{z_n} [\frac{LC_n(\theta)}{LD_n(\theta)}]^{1-z_n} \\
&= \begin{cases} LD_n(\theta) - LC_n(\theta) & \text{if } z_n = 1 \\ LC_n(\theta) & \text{if } z_n = 0 \end{cases}.
\end{aligned} \tag{9}$$

Please note that the double precision likelihood $LD_n(\theta)$ now only appears in those data (let's call them "bright data" to follow FlyMC terminology) for which $z_n = 1$. At any given iteration, the data points for which $z_n = 0$ (let's call them "dark data"), we compute their likelihoods in reduced precision. Therefore, the full likelihood is now given by:

$$L(\theta) = \prod_i^{z_i=1} (LD_i(\theta) - LC_i(\theta)) * \prod_j^{z_j=0} LC_j(\theta) \tag{10}$$

This algorithm can be seen as shifting the computational burden from evaluating $LD_n(\theta)$ to evaluating $LC_n(\theta)$ plus a step to sample this new parameter. The computational gains are coming from evaluating some likelihoods in custom precision instead of utilising double precision in all likelihood evaluations.

For the rest of the paper, the above proposed algorithm is called custom-precision firefly MCMC (CF-MCMC) algorithm and its steps are shown in Algorithm 2. The overhead of introducing a sampling stage of the auxiliary variable z_n, has a small penalty in the performance of the algorithm as this resampling is performed only for a random fixed-size subset of the data [14]. This results from the fact that at every iteration most of the binary variables are kept unchanged. The sampling step for z_n is shown in lines 8-11 and 14-19 of Algorithm 2, which is performed immediately after the computation of the likelihood. Since the likelihoods of the bright data points have already been evaluated in the MCMC step of line 7, the implementation of the algorithm can reuse these values and resample all the instances that correspond to "bright data" points without any extra computational cost. As only few z_n variables that $z_n = 0$ change in each iteration (assuming a tight lower bound), the resampling of the dark points' variables is performed at a fixed rate (1/ResampleFraction as shown in line 15), to avoid computing the full precision likelihoods for all the dark data in each iteration[2]. The above partial resampling leads to a chain with slower mixing rate. However, as

2. Setting the value of ResampleFraction is discussed in Section 4.

ALGORITHM 2: CF-MCMC Algorithm

```
Input: initial setting $\theta_0$ and $\{z_n\}_{n=1}^{N}$, $N_s$;
Output: parameter samples $\theta_i, i = 1, ..., N_s$;
 1: for $i = 1$ to $N_s$ do
 2:   Propose $\theta' \sim \theta_{i-1}$+Normal(0, $s^2 I_D$);
 3:   $L(\theta') = 1$; // likelihood initialization
 4:   for $n = 1$ to $N$ do
 5:     $u_1 \sim$ Uniform(0,1);
 6:     if $z_n = 1$ then
 7:       // likelihood computation for bright data
          $L(\theta') = L(\theta') * (LD_n(\theta') - LC_n(\theta'))$;
 8:       // $z_n$ sampling
 9:       if $1 - LC_n(\theta')/LD_n(\theta') \leq u_1$ then
10:         $z_n = 0$;
11:       end if
12:     else
13:       // likelihood computation for dark data
          $L(\theta') = L(\theta') * LC_n(\theta')$;
14:       // partial sampling of $z_n$ for dark data
15:       if $n\%ResampleFraction =$
          $RandInteger(1, ResampleFraction)$ then
16:         if $1 - LC_n(\theta')/LD_n(\theta') > u_1$ then
17:           $z_n = 1$;
18:         end if
19:       end if
20:     end if
21:   end for
22:   Compute $a = \frac{L(\theta')}{L(\theta_{i-1})}$;
23:   $u_2 \sim$ Uniform(0,1);
24:   if $u_2 \leq a$ then
25:     $\theta_i = \theta'$;
26:   else
27:     $\theta_i = \theta_{i-1}$;
28:   end if
29: end for
```

indicated in [14], the approach works well in practice as the bottleneck for mixing is usually in the space of θ. For a given budget in likelihood evaluations, allowing more steps in the θ space is intuitively likely to reduce initialization bias faster than resampling all variables at each iteration.

3.2 Lower Bound Function Construction

In order for the samples to come from the original posterior distribution when the augmented posterior distribution is utilised, the custom precision likelihood $LC_n(\theta)$ is required to be a lower bound on the full precision likelihood $LD_n(\theta)$, i.e. $LC_n(\theta) \leq LD_n(\theta)$. In order to achieve this in a custom precision setting, in our prior work [18] we proposed to use the tool Gappa++ [25] which determines and verifies numerical behaviour, and particularly rounding error in computations with floating point operations. The tool manipulates logical formulas stating the enclosures of expressions in some intervals. In particular, Gappa++ allows bounding computational errors due to floating point arithmetic. It works effectively and fast across a range of function constructions and especially for the linear functions. For most problems it takes less than a minute to obtain the precision-related error bound [25].

Let's denote the maximum absolute error bound between two floating point precision constructions of $p(x_n \mid \theta)$, one under double precision arithmetic (i.e. $LD_n(\theta)$) and one under a custom precision $p(x_n \mid \theta)_c$, that is provided by the Gappa++ tool as ε, where $\varepsilon \geq 0$ (i.e. $|LD_n(\theta) - p(x_n \mid \theta)_c| < \varepsilon$). Then, $LC_n(\theta)$, is defined as:

$$LC_n(\theta) = p(x_n \mid \theta)_c - \varepsilon \tag{11}$$

which ensures that $LC_n(\theta) \leq LD_n(\theta)$, i.e. that $LC_n(\theta)$ is a lower bound of $LD_n(\theta)$.

The tightness of the lower bound construction is important to the performance of the CF-MCMC algorithm because it impacts the number of bright data points at each iteration, which essentially determines the execution time of the CF-MCMC algorithm. In our prior work, Gappa++ was used solely in order to obtain the lower bounds for the likelihood function. However, the tightness of the bounds provided by Gappa++ (i.e. ε) depend on the actual operations involved in the function under investigation. [25].

In this work, we propose an alternative lower bound function construction in order to provide a tighter custom precision bounds, further boosting the performance of the algorithm.

The proposed approach capitalises on the fact that in FPGA designs the user can tune the rounding modes of the floating point operators. As such, by appropriately tuning the rounding mode of the operators under a custom precision implementation, the user can guarantee a lower bound by construction. To take advantage of this, we separate the parts of the likelihoods for which lower bound guarantees are obtained by construction (for example in the case where there is an addition of two positive quantities) and to parts for which this is not possible and their error bounds are estimated through Gappa++. The proposed lower bound function design allows utilization of the Gappa++ tool in combination with the rounding mode configuration of the arithmetic operators on FPGAs, producing tighter lower bounds for a given custom precision, with respect to the existing methodology.

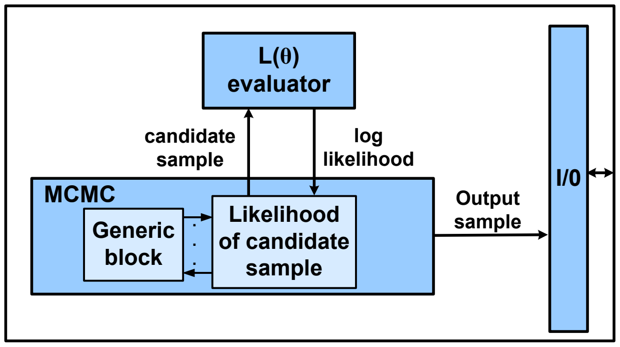
Given the logistic regression likelihood function in Equation (12) as an example, the previous proposal in [18] of the lower bound function (13) is based on the error bound ε_1 of the whole function which is provided by Gappa++.

$$L_n(\theta) = \frac{1}{1 + exp\{\theta^T x_n\}} \tag{12}$$

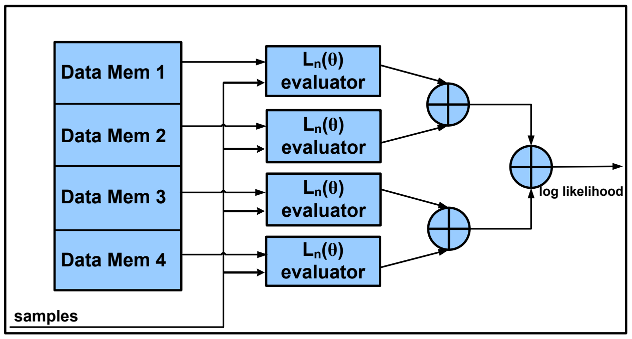
$$LC_n(\theta) = \frac{1}{1 + exp\{\theta^T x_n\}} - \varepsilon_1 \tag{13}$$

Here we show how the lower bound function is constructed in this work. Firstly, we use Gappa++ to obtain the rounding error ε_2 of the dot product operation inside the exponent operation. Then we add ε_2 to the custom precision dot product values. Secondly, we set specific rounding modes (round up or round down) for the other operators that are monotonic, in order to guarantee the final result is a lower bound. The proposed lower bound function can be shown as the following equation:

$$\begin{aligned} LC_n(\theta) =& div(1, add(1, exp(\theta^T x_n + \varepsilon_2, \mathbf{RoundUp}), \\ & \mathbf{RoundUp}), \mathbf{RoundDown})) \end{aligned} \tag{14}$$

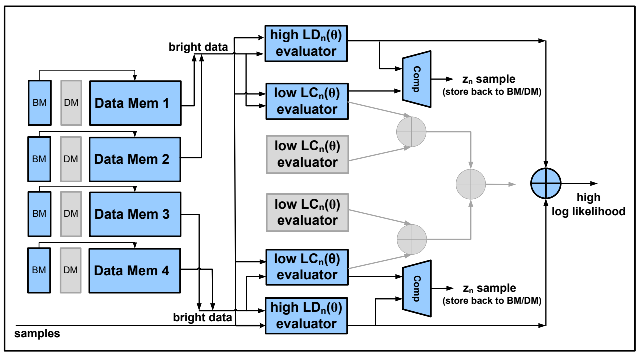


(a) The overall architecture

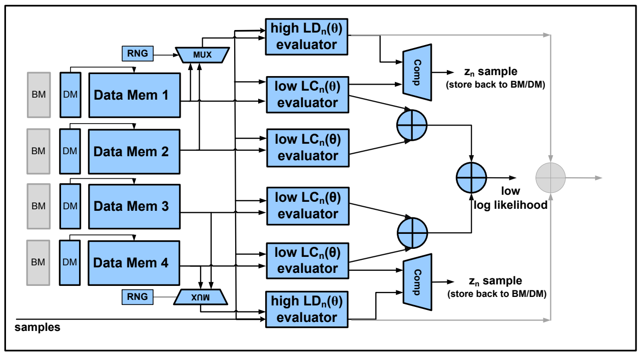


(b) The parallel likelihood architecture

Fig. 1. (a) The overall architecture of the FPGA-mapped MCMC sampler which mainly contains the generic and likelihood $L(\theta)$ evaluator block. (b) The architecture of double-precision floating point likelihood $L(\theta)$ evaluator design with the conventional parallel implementation at $P = 4$.



Step 1 – Bright data likelihood computation and z_n sampling



Step 2 – Dark data likelihood computation and partial z_n sampling

Fig. 2. The architecture of CF-MCMC algorithm using mixed precision design at $P_H = 2$ and $P_L = 4$. The full likelihood is computed and the binary variables are updated by two steps: 1) Bright data likelihood computation in parallel using 2 blocks and z_n sampling; 2) Dark data likelihood computation in parallel using 4 blocks and partial z_n sampling. The light gray blocks indicate they remain idle at the corresponding step.

4 FPGA Implementation

4.1 Proposed Hardware Architecture

FPGA devices have been considered by researchers and practitioners for MCMC acceleration because of their ability to implement many processing elements for the likelihood calculation, as well as due to their flexibility to implement any custom arithmetic precision regime. Assuming that the data points can fit in the on-chip memory blocks (an assumption that will be lifted later on), an FPGA system that implements an MCMC sampler is given in Figure 1a, where high memory bandwidth that matches the computational capabilities of the processing elements (for likelihood evaluation) is provided through the on-chip memories.

The FPGA-mapped MCMC sampler (not considering the off-chip memory access) generally contains two blocks as shown in Figure 1a: a hardware block for the generic MCMC operations (i.e. propose new sample, accept/reject ratio calculation) and a block to compute the full likelihood $L(\theta)$ in the logarithmic domain in order to avoid numerical instability in the evaluation of the likelihood [26]. Because likelihood evaluations dominate the computational cost, the performance of the MCMC sampler can be improved by implementing many parallel likelihood evaluation blocks. When the likelihood evaluation can be decomposed into sub-components due to i.i.d assumption of the data (which is also assumed in this article), FPGA implementations typically use a likelihood evaluation block which consists of parallel likelihood modules [27]. Let's denote the number of modules by P, and an example of the conventional double-precision floating point design at $P = 4$ is given in Figure 1b. Accordingly the data memory is partitioned into P blocks, thus each evaluation block processes one block of data. Finally the total sum (i.e. the full log likelihood) is computed by combining the outputs of the P blocks. For compute-bounded problems (as the one considered in this work), the goal is to maximize the number of parallel block within the available resources in the FPGA device, minimizing as such the execution time of a single MCMC iteration. This motivates the idea in this article to implement low precision data paths in order to save computational resources and increase the sampling throughput.

In our prior work, we showed the FPGA architecture of the CF-MCMC Algorithm where only a single high-precision datapath was utilised in the design [18]. Building on the previous architecture, a more generic design which utilizes multiple high-precision datapaths is presented here and it is depicted in Figure 2. We denote each parallel degree of the high and low precision datapaths as P_H and P_L respectively ($P_H < P_L$). Accordingly, the data are stored in P_L memories and each data memory is attached with a set of BM and DM memories to store the indexes of the bright and dark data points respectively. Rather than storing the binary value of each z_n, we store the indexes of the bright and dark binary variables separately in the two independent memories (BM and DM). Also, for each memory, the system keeps track of the total number of bright and dark points.

For each iteration, the system needs to perform the likelihood computation and z_n sampling for the bright and dark points. Thus two steps (as shown in Figure 2) are performed in sequence to accept/reject the proposed sample. First, in Step 1, the system accesses P_H BM memories in parallel to use the bright data index to access the bright data in the corresponding data memories, which are then passed to the high-precision data paths to evaluate the sum of the log likelihood of the data points with $z_n = 1$. At the same time, these bright data are also passed to the first P_H low-precision data paths to compare the difference between these two values $LD_n(\theta)$ and $LC_n(\theta)$ in order to update z_n required for the next iteration. Therefore, the other $(P_L - P_H)$ low-precision datapaths remain idle (appear light gray in the figure) during this step. The above process continues till all the bright data points are processed. Then, in Step 2, the P_L DM memories are read in parallel to access the dark data points in parallel, which are passed through the P_L low-precision data paths. Accordingly, at every cycle, P_H out of the P_L dark data will be chosen randomly to go through the P_H high-precision data paths for updating the corresponding z_n. Therefore only P_H/P_L of the dark variables are resampled, and all the data paths are fully utilised in this second step. Since the dark data can be resampled at a fixed fraction rate as mentioned previously, the system samples the dark data at the fraction rate P_H/P_L based on the degree of parallelism that has been achieved for the high and low precision paths, in order to maintain all data paths busy and maximize utilization.

As many applications target sets of data that do not fit in the on-chip memory of FPGAs, external memories are utilised to store the data. In such situation, the system segments the data set into smaller subsets, and operates on them in a sequential order until all the subsets are processed. Standard techniques can be applied such as double buffering, in order to match the computational and memory bandwidth capabilities of the system. The overall processing on the FPGA device remains the same, but for every iteration the system needs to transfer data from the external memory. As a result, compared to the on-chip memory which can be directly addressed and accessed by the datapaths, the communication between the FPGA and off-chip memory can limit the FPGA performance if the memory bandwidth is not enough to constantly feed the processing elements.

4.2 Intelligent Data Distribution

In order to maximise the performance of the system, the bright data and dark data points need to be equally distributed in the memories, otherwise there may be a deviation in the utilization of the datapaths when executed in parallel. This work proposes a methodology based on proportional allocation for redistributing the data to the memories in an intelligent way in order to maximize the performance of the system. The goal is to rebalance the dark data points across the available memories, in order to reduce the overall latency of each iteration of the MCMC algorithm.

Assuming the case where all data can fit in the on-chip memories, the system can introduce a rebalancing step after each iteration in order to minimize the latency of each iteration by maximizing the utilisation of the processing elements.

At every point in time, the system keeps track of the number of total bright and dark points stored in each memory block. After each iteration, the system checks the percentage of the bright or dark points stored in each memory block against to the average number of bright and dart points respectively in the system. Then, a reshuffle of the data points is initiated in order to result in memory blocks that have equal proportion of dark points.

To briefly demonstrate the benefit of rebalancing the dark data points among memories, assume that we have two memories and each memory contains M_1 and M_2 dark data ($M_1 > M_2$) at one iteration. Let C_{PE} denote the input throughput of the evaluation block in cycles. The execution time (in clock cycles) to process these data points without data distribution is $T_1 = max(M_1 C_{PE}, M_2 C_{PE})$, where if a distribution step is introduced to the system, the execution time T_2 is given by

$$T_2 = \frac{M_1 - M_2}{2} C_M + \frac{M_1 + M_2}{2} C_{PE}$$

where C_M models the clock cycles needed to transfer a data point from one memory to another, and $\frac{M_1 - M_2}{2} C_M$ models the time taken for data distribution. Assuming that one data point can be read/stored in the memory in every cycle (i.e. $C_M = 1$), it is easy to show that $T_1 \geq T_2$ always holds if $C_{PE} \geq 1$. The above implies that it always pays off to rebalance the dark data points, when it takes more than one clock cycle to consume a data point and the data points can be transferred from one memory block to another in one clock cycle. The above model is used to decide whether a redistribution of the data would improve the performance of the system.

In the case where off-chip memories are used to store the data, the above redistribution of the data takes place when the data are transferred from the external memory to the on-chip memories on FPGA, removing any time penalty imposed by the distribution of the data as a separate process as in the case where the data are stored in on-chip memories.

4.3 Performance Model

In this section, an analytical performance model of the system is derived in order to reason on how the selected custom precision impacts the execution time of the system. The total processing time (in cycles) of the MCMC method for generating N_s samples consists of the time spent for performing the MCMC sampling T_{MCMC}, assuming the data are already on chip, and the time required to transfer the data on-chip/off-chip, $T_{transfer}$, which is shown in Equation (15).

$$T_{total} = T_{MCMC} + T_{transfer} \tag{15}$$

In the case of a double-precision MCMC design (i.e. baseline), a sample is generated every N/P_{DP} clock cycles. Thus, the total time spent for generating N_s samples is:

$$T_{DP-MCMC} = N_s * N/P_{DP} \tag{16}$$

where N is the number of the data points and P_{DP} is the parallelism of double-precision MCMC design. In comparison, the time needed for the CF-MCMC architecture in total is given by:

$$T_{CF-MCMC} = N_s * (N\alpha/P_H + N(1-\alpha)/P_L) \tag{17}$$

where α is the proportion of bright data, and P_H, P_L are the parallelism of the high- and low-precision paths respectively.

In the case where no off-chip memory is used, $T_{transfer}$ can be omitted otherwise $T_{transfer}$ can be modelled as:

$$T_{transfer} = \frac{N_s * (N * D * sizeof(data) + N * \lceil logN \rceil)}{bandwidth} \tag{18}$$

where D is the dimension of the each data point and $\lceil logN \rceil$ is the bit-width of the index for the $z(n)$ variables.

Please note that the above execution time T_{total} only refers to the raw execution time (i.e. time needed to generate N_s samples) of the corresponding MCMC sampler. As mentioned in Section 2, the sampling efficiency metric which is used to compare the performance of different MCMC algorithms and their implementations also needs to include the effect of sample dependency using the ESS metric given in (5). The rate of effective samples per clock cycle can be derived by dividing ESS by the execution time i.e. ESS/T_{total}, which can also be seen as the effective throughput of the MCMC system.

5 Custom Precision Tuning

Even though the above construction guarantees unbiased estimates for any adopted custom precision, the selected custom precision has impact on the performance of the system and needs to be tuned. In order to achieve the maximum performance in terms of effective throughput, the designer needs to consider the impact of custom precision selection to the parallelisation factor achieved, as well as the percentage of bright data points during the execution of the algorithm. As the precision is reduced, more parallelism can be obtained for a set of resources, and thus reducing the total execution time of the system; however the percentage of the bright data points increases accordingly, as the gap between the lower bound function and the target likelihood function increases, which in turn introduces additional runtime as more high-precision computations need to be performed. Here, the work focuses on exploiting the optimal precision selection, in order to maximise the performance of the system.

In this work, a static analysis selection method is proposed by modelling the processing time and resources versus the utilised custom precision. Please note that the ESS cannot be modelled during static analysis, so the work targets to maximise samples per cycle that are generated by the system (i.e. raw speed). The proposed methodology consists of two steps:

1) Resources v.s. Precision

The first step is to compute how the resource utilisation varies with the custom precision of the system. This step only requires the pre-synthesis of the floating point IPs under different precisions on FPGA in order to estimate the total resource utilisation of the likelihood function evaluation block under different custom precision regimes. Then, for a give target FPGA device the maximum achievable parallel degree, i.e. P, can be obtained for each custom precision candidate.

2) Bright Data v.s. Precision

The second step needs to consider the percentage of the bright data points during the execution of the algorithm in order to estimate the optimal configuration of the system. An estimate of the number of bright data points M, is given by:

$$M = \sum_{n=1}^{N} \int p(\theta \mid \{x_n\}_{n=1}^{N}) \frac{LD_n(\theta) - LC_n(\theta)}{LD_n(\theta)} d\theta \tag{19}$$

However, in order to estimate the above quantity, we need to draw samples from the actual target distribution $p(\theta \mid \{x_n\}_{n=1}^{N})$, which is the reason for the design of such system. Following [16], M can be estimated by short MCMC pre-runs. As the $\frac{LD_n(\theta) - LC_n(\theta)}{LD_n(\theta)}$ factor in equation (19) takes small values, the variance of the estimation of M will drop down fast as the number of samples increases. Here we propose that M is estimated using short FPGA-mapped pre-runs, taking advantage at the same time of the parallelism offered by FPGA devices across the different runs, as have been demonstrated in [16].

Utilising information from these two steps, i.e. information on the resource requirements for likelihood function evaluation under different custom precisions and an estimate of the number of bright points M, the performance model introduced before is utilised in order to provide estimates of the theoretical raw speedup leading to the selection of the optimal custom precision of the system. Please note that the ESS effect is only known at runtime and cannot be captured by the above static analysis based model. However, as the obtained results indicate, the above model can provide an informative prediction of the performance of the system.

6 Performance Evaluation

6.1 Case Studies

Logistic regression is used in many fields, including medical and social sciences. Here we consider two Bayesian problems with different dimensionality and data size that utilize a logistic regression model. Both case studies are representative of the distributions normally targeted by MCMC, both in terms of the types of arithmetic operators used, as well as the problem size they incorporate.

6.1.1 Synthetic Problem

Initially, the performance of the proposed system is evaluated by performing logistic regression on a synthetic data set, a two-class classification problem in two dimensions (and one bias dimension). As such, the ground truth of the parameters is known and it is used for the evaluation of the obtained estimates. Here the linear model in (20) is used: a set of 500 independent data $\mathbf{x} = x_{1:500}$ is generated randomly; the data set $y_{1:500} \in \{-1, 1\}$ is simulated using the parameters $\beta = (-10, 5, 10)$. The logistic regression likelihood of each data point is given by (21), where $\theta = (\beta_0, \beta_1, \beta_2)$ are the parameters, and the bias parameter is absorbed into θ by including 1 as an entry in x_n.

$$y = sign(\beta_0 + \beta_1 x_1 + \beta_2 x_2) \tag{20}$$

$$L_n(\theta) = \frac{1}{1 + exp\{-y_n \theta^T x_n\}} \tag{21}$$

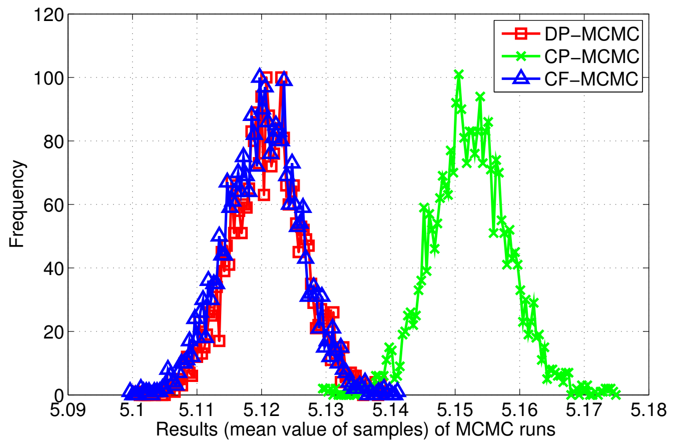


Fig. 3. Distribution of the mean value of the first parameter samples in the synthetic problem with 3,000 runs of DP-MCMC (at precision s52e11), CP-MCMC and CF-MCMC (both at precision s8e8).

6.1.2 *MNIST Classification*

The second case study focuses on a real problem, which is the logistic regression task described in [5]. The task is to classify handwritten digits (7s and 9s) in the large MNIST database, which has been widely used for training and testing in the field of machine learning [28]. The first 12 principal components (and one bias) are used as features. A set of 2000 data points are chosen from the total 12,2214 data so the MCMC algorithm queries 2000 likelihood terms per iteration in this experiment. Each likelihood takes the form shown in (21) where x_n is the set of features for the nth data point. As opposed to the first case study, the parameter dimension increases to 12 plus a bias, where the total number of likelihood terms increases from 500 to 2000.

6.2 Hardware Implementation Details

The architecture in Figure 2 is implemented on a Xilinx Virtex-6 LX240T FPGA. The arithmetic operators of the generic MCMC block are implemented in double-precision floating point. The high and low likelihood evaluation blocks which are fully pipelined are implemented under double precision floating point arithmetic and reduced custom precision arithmetic respectively, using floating point operators generated by FloPoCo [29]. All designs run on a single 150 MHz clock and fully utilized the available FPGA's resources. All results are post place and route.

6.3 Quality of MCMC Samples

We first assess the quality of the generated samples of three algorithms: the proposed CF-MCMC, a double-precision implementation MCMC (DP-MCMC), which is used as a reference design, and a custom-precision MCMC (CP-MCMC), which uses the reduced precision for the computation of all likelihood terms i.e. existing approach in digital hardware design community. For convenience, the notation sAeB is used in this article to denote a floating point representation, where A is the number of significant bits and B is the number of exponent bits.

Figure 3 shows the distributions of the predictive mean of parameter β_1 for the synthetic problem using the above three Monte Carlo simulations. In each MCMC simulation, $N = 30,000$ sample points are generated, and each of three algorithms are repeated for $3,000$ times with different random seeds. Both CP-MCMC and CF-MCMC algorithms utilised a custom precision of s8e8 (i.e. 8 significant bits and 8 bits for the exponent). As the results indicate, for CP-MCMC simulations where the reduced precision datapaths are used for all the likelihood terms and models the existing hardware design approaches, the mean value of the parameter has a significant bias and also a larger variance compared to the DP-MCMC results, supporting the results obtained in [16], [17]. However, the CF-MCMC sample distributions have the same variance and mean as DP-MCMC samples, which demonstrates that the proposed algorithm removes any bias introduced in the results due to low-precision computations. Please note that even though the actual value of the estimated parameter is 5, the data that have been generated by the model support a parameter value of 5.12 (the mean estimate of DP-MCMC and CF-MCMC algorithms).

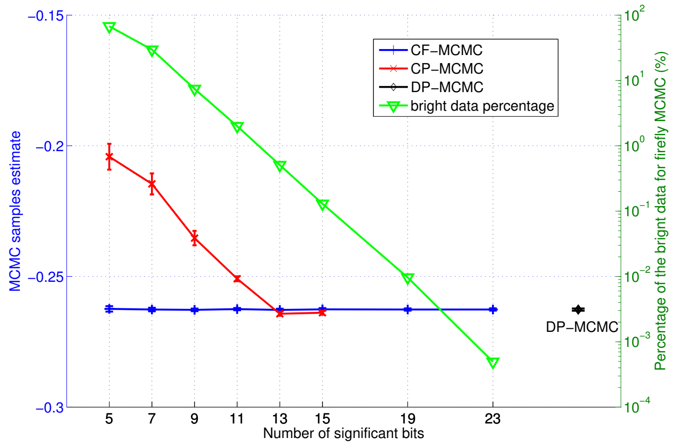


Fig. 4. The bias and its variance estimates of the parameter for MNIST problem, where DP-MCMC runs in double precision (s52e11), CP-MCMC and CF-MCMC run in the precision with the number of significant bits from 5 to 23. The green line indicates the average percentage of the bright data by 10,000 iterations in each of the 100 CF-MCMC runs.

Figure 4 depicts the bias and the variance in estimating the predictive mean on MNIST dataset, for a range of reduced custom precisions (with the number of significant bits varying from 5 to 23 and a fixed exponent bits number of 8) for CP-MCMC and CF-MCMC algorithms with comparison to DP-MCMC simulations (where double precision with the significant bits 52 and the exponent bits 11 used). For each algorithm, 10,000 sample points are generated, and each MCMC simulation is repeated for 100 times. As shown in the figure, for every design point, both bias and standard deviation increase as the utilised custom precision uses fewer bits in the case of CP-MCMC. However, for the proposed algorithm, CF-MCMC, the obtained estimates have the same mean value and deviation as in the case of DP-MCMC algorithm regardless of how much the precision is reduced (apart when the number of significant bit is reduced to 5, but still there is no bias in the estimate). Figure 4 also shows the proportion of bright data point of the data set for CF-MCMC under different reduced precisions. The proportion of bright data is only 0.0005% at single-precision s23e8, but increase to 67% at a very small precision s5e8, indicating the the lower bound function becomes less tight as the precision decreases.

TABLE 1
Resources of the generic block and one log-likelihood evaluation block using double floating point arithmetic operators, and memory size for CF-MCMC on Xilinx Virtex-6 LX240T.

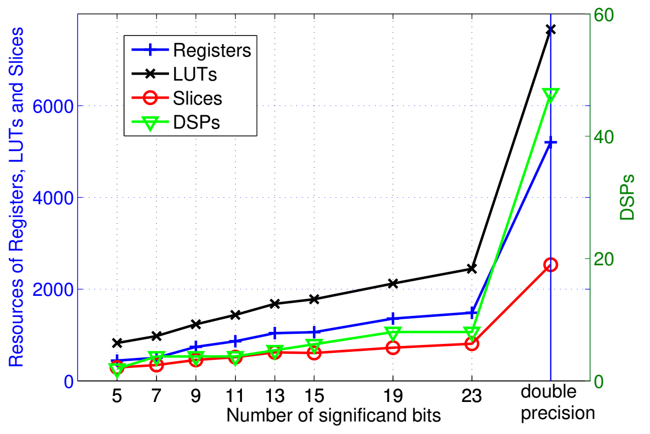
Resources (double-precision)		Slices Registers	LUTs	Slices	DSP48E1s	Memory Size
Generic block		2853	4837	1722	11	-
Log Likelihood Evaluation block	Synthetic Problem	5203	7666	2533	47	9.2 KB
	MNIST Problem	13168	18919	6242	143	0.2 MB

In summary, the two case studies indicate that the proposed system can produce unbiased estimates even under custom precision regimes without any noticeable difference in the variance of the estimate compared to an implementation that utilises double-precision floating point arithmetic throughout the system, except in the case where the precision is reduced significantly (i.e. 5 bits for the significant). Furthermore, it is observed that the current techniques that utilise custom precision in the likelihood evaluation lead to biased estimate, as it is expected.

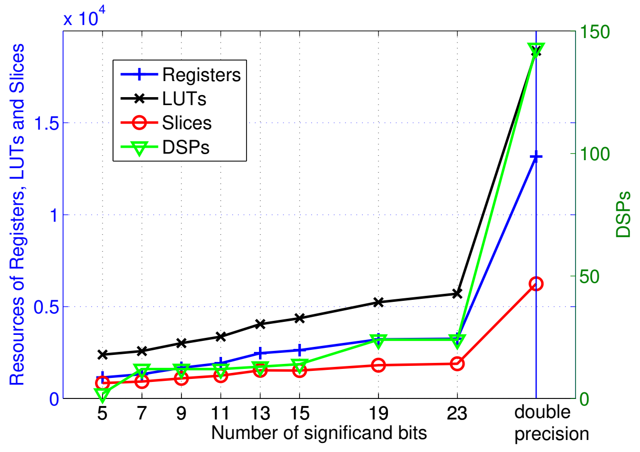
6.4 Resource Utilization

The proposed architecture CF-MCMC shown in Figure 2, and the double-precision architecture (DP-MCMC) shown in Figure 1b have been implemented in the target device. Table 1 gives the resource utilization (Registers, LUTs, Slices etc.) of the generic MCMC block which uses double precision floating point arithmetic operators, and also the resources required by one log-likelihood evaluation block at double-precision for both case studies. Here we also show the total memory size needed by both architectures to store the data and the auxiliary variables.

Figures 5a and 5b show the resource utilization of a single likelihood evaluation block of the synthetic example and MNIST problem under different reduced precisions respectively. The double-precision block's resources in Table 1 are also plotted in this figure. The figures imply that a factor of parallelism between 4-5 can be extracted when part of the computations can be mapped to reduced precision likelihood evaluation blocks utilising a precision between 8-18 significant bits. The above figure provides an expectation of the maximum gain in the performance that can be delivered by the proposed system compared to a double precision floating point implementation.



(a) Synthetic Problem



(b) MNIST Problem

Fig. 5. The resource utilization of a single likelihood evaluation block of (a) the two-dimension (plus a bias) problem and (b) the MNIST problem with custom precision where the significant bits range from 5 to 23 and a fixed exponent bits at 8, and also double precision.

6.5 Effective Sampling Throughput

In this section, the Effective Sampling Throughout speed-up is investigated for the proposed architecture. As the aim is to achieve the maximum throughput, full utilization of the FPGA logic resources is assumed for both architectures (CF-MCMC and DP-MCMC). Therefore, a maximum number of parallel log-likelihood evaluation blocks is obtained for each sampler, by selecting the optimal configuration of high and custom precision likelihood calculation blocks, P_H and P_L, using the proposed performance model. The effective sampling throughput is measured by ESS/T_{total} as described in Section 4.3. We compare the speedups in the throughput for our proposed CF-MCMC accelerator over DP-MCMC design, and the results for both problems are shown in Figure 6 for a range of values of the number of significant bits utilised in the CF-MCMC system. For the precision shown in the figure, the optimal configurations (P_H and P_L) that fully utilise the targeted FPGA device are: $(P_H, P_L) = \{(8,22),(8,20),(4,36),(1,46),(1,38),(1,36),(1,34),(1,30)\}$ for the synthetic problem, and $(P_H, P_L) = \{(2,22),(2,20),(2,16),(1,14),(1,12),(1,12),(1,10),(1,10)\}$ for the MNIST problem.

As the figure shows, the speedups obtained for the two problems with the evaluated precisions are in the order of 0.72x-3.67x and 0.51x-4.07x, for the synthetic and the MNIST problem respectively. The results show that by reducing the utilised precision in CF-MCMC, a higher degree of paral-

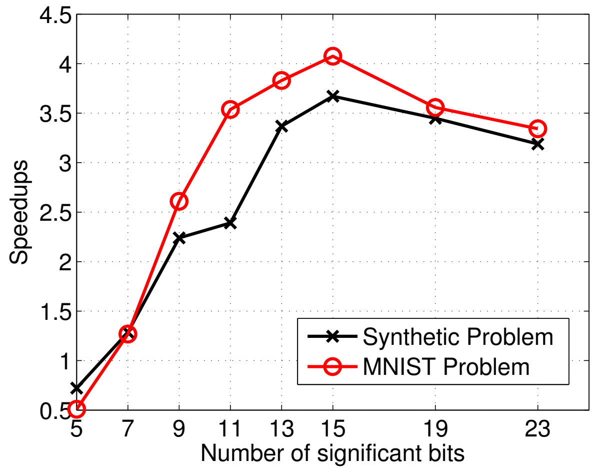


Fig. 6. The speedups in terms of the effective sampling throughput of CF-MCMC architecture over DP-MCMC on the target device for the two case studies.

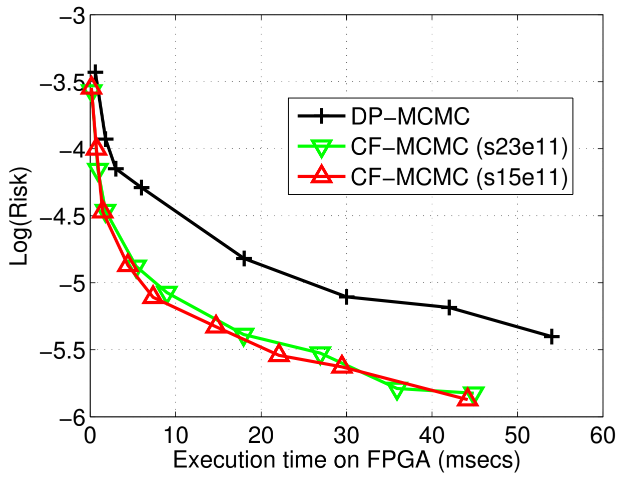


Fig. 7. The Risk in the estimate of the mean of the parameter.

lelism is possible. However, at the same time, the proportion of the bright data increases, which in turn introduces extra latency as the computation for bright data likelihood is executed in a lower parallel degree. Furthermore, there is a reduction in the effective sample size (ESS) as the precision is reduced. This is evident by the obtained results, where a peak in effective sampling throughput speedup is observed at a specific precision configuration. Furthermore, it is observed that the proposed system can be outperformed by the double precision implementation when few significant bits are utilised (less than 6), indicating the need to have in place a performance model that predicts the throughput of the system. The optimal precisions for both problems are (s, e) = (15, 8), with the corresponding speedup of 3.67x and 4.07x.

Another metric to compare the performance of MCMC algorithms is the mean squared error (or risk) in the estimate of (2), i.e. $R = (I - \tilde{I})^2$, where the expectation is taken over multiple simulations of the Markov chain [12]. The risk can be decomposed as the sum of squared bias and variance, and the objective of MCMC in practice is to obtain estimates with lower risk. Figure 7 shows how the logarithm of the risk in estimating the mean of the parameter for MNIST, decreases as a function of the execution time. The experiment configuration is the same as in [12]: we first estimate the true mean using a long run of regular MCMC; then we compute multiple estimates of the mean from the algorithms under investigation and obtain the risk in these estimates. In our test, the average risk is based on 100 runs for each algorithm. The figure demonstrates that the proposed CF-MCMC algorithm largely reduces the risk compared to that of DP-MCMC, by reducing the variance faster within the same time period.

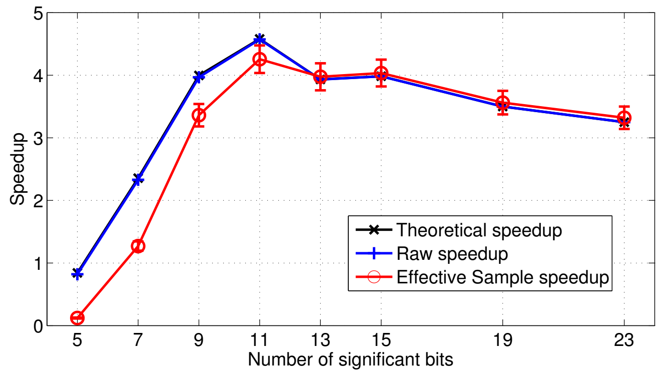
6.6 Theoretical Performance Model Evaluation

In this subsection, the accuracy of the theoretical performance model proposed in Section 5 is evaluated, and it is shown how it can be utilised in order to maximise the performance of the proposed system. For this investigation, the MNIST case study is utilised.

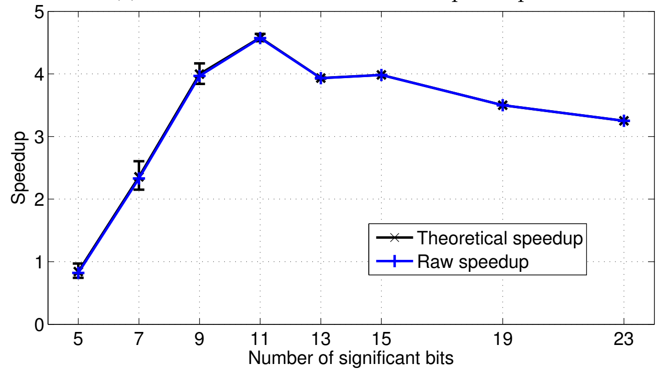
The evaluation of the framework is performed under three metrics: 1) the Theoretical speedup, which is computed by the theoretical performance model $T_{CF-MCMC} = M/P_H + (N-M)/P_L$ using the parallelism P_H and P_L we achieved in the target hardware and the bright data point M in Equation (19); 2) the actual Raw speedup, which is computed using the actual runtime by executing the configuration in the FPGA device at each precision; 3) the Effective Sample speedup, which is computed through ESS/T_{total} i.e. the effective sampling throughput as described in the above subsection, and it includes the actual runtime when executed in the FPGA device and also the ESS effect in the generated samples (this speedup is used in the rest of the article unless explicitly stated).

Figure 8a shows the Theoretical, Raw, and Effective speedups for a range of precisions between the proposed architecture CF-MCMC and DP-MCMC. In total 100 runs were conducted in order to capture the possible variations in ESS. The results confirm that the derived performance model captures well the performance of the system under all the precisions, where the ESS effect, even though it is not captured by the performance model, does not have significant impact on the model's performance prediction accuracy. The confidence interval bars denote the variation in the Effective Sample speedup performance along different runs of the system for 95% confidence interval.

As has been described before, the performance model utilises the estimation of the number of bright points based on short pre-runs (e.g. 1000 samples). Given that this estimate is not exactly the same as the converged parameter values and thus it only provides an approximation of the number of bright points in the system, it is necessary to investigate the sensitivity of the predicted speedups obtained by the performance model with respect to the variation of the actual number of bright points from the predicted one. In this investigation, the number of bright points M is set to take values in an interval $[M^*(1-x) \quad M^*(1+x)]$, where M^* denotes the estimate of (19) based on short pre-runs, and then compute the theoretical speedups according to the number of bright data in this interval. The results for $x = 15\%$ are shown in Figure 8b.



(a) Theoretical, Raw, and Effective speed-ups.



(b) Theoretical and Raw speed-ups.

Fig. 8. The theoretical performance model is evaluated by (a) comparisons of the Theoretical speed (estimated time), actual Raw speedup (execution time in FPGA), and the Effective Sample speedup (execution time in FPGA and including the ESS effects); and (b) the range (max and min values as denoted by the bars) of the theoretical speedup assuming a maximum deviation between the actual and predicted number of bright points of up to 15%.

The results indicate that at high precisions, the theoretical speedups have little variation with respect to the number of bright points. However, the above variations do not have an impact on the choice for the optimal custom precision that should be employed by the system, and thus the provided theoretical performance model and optimal precision selection method are still valid even in the case where the estimates of (19) have up to 15% deviation from the real values.

6.7 Lower Bound Construction Comparison

In this subsection, the construction of the lower bound functions proposed in Section 3.2 is investigated in order to assess how the lower bound constructions impact the speedups of the CF-MCMC algorithm. The MNIST case study is used for this investigation. Three different constructions are compared. The first construction is the method proposed in our prior work that utilises only Gappa++ in order to estimate the error bound (Gappa++). The second method is the method proposed in this work which estimates the error bound for part of the function through Gappa++ and utilises specific rounding modes in order to ensure a lower bound construction (proposed). The third method utilises the round mode for part of the function (similar as before), but now the final error bound for the whole function is estimated through simulations using the MPFR library [30] (MPFR). The third method does not provide any guarantees,

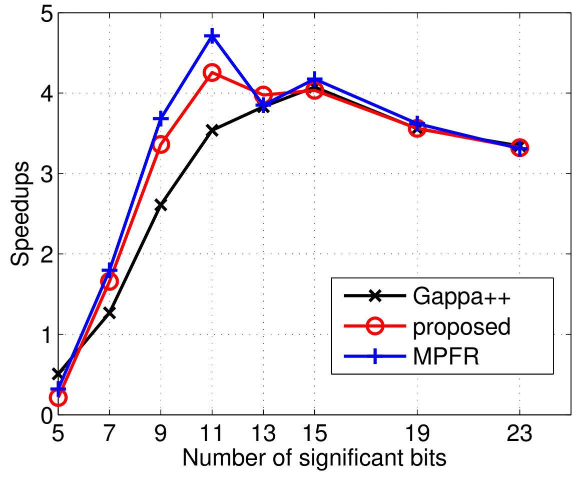


Fig. 9. The speed-ups (Effective Sample speed-up) of CF-MCMC architecture over DP-MCMC implementations on the target device for the MNIST problem, under the three lower bound function constructions.

but it can be seen as a reference of the maximum speedups of the CF-MCMC algorithm that could be achieved. The results in terms of Effective Sample Speedup performance for the MNIST problem are shown in Figure 9, while all constructions have no obvious influence on the bias and variance of the generated samples.

As shown in the figure, the proposed construction, which is based on the rounding mode, results in systems that outperform systems that are based on the construction of the lower bound function based on our previous work (Gappa++). Furthermore, the proposed approach which guarantees a lower bound construction gives similar performance results to the method that is based on simulations (MPFR) and no guarantees in the construction can be provided. Nevertheless, all constructions lead to similar speedups when high precisions are utilised.

The above observation can be further generalised beyond the specific case study. Let us rewrite the performance model provided in Section 4.3. Assuming $P_H = 1$, the execution time to generate one sample: $T_{sample} = N\alpha + N(1-\alpha)/P_L$ can be rewritten as: $T_{sample} = N * (1/P_L + (1-1/P_L)\alpha)$. For a given utilised precision, the number of parallel low precision units P_L is fixed on a target device. If the proportion of the bright data points (i.e. α, which depends on the lower bound proposal) satisfies $(1-1/P_L)\alpha << 1/P_L$ i.e. $\alpha << 1/(P_L-1)$, then the execution time T_{sample} will be almost equal to N/P_L which doesn't depend on the quality of the lower bound construction. As such, all constructions would provide similar speedup when high custom precision evaluation blocks are utilised.

For example, for the MNIST problem when the precision bits are at 5, 7 and 9, $P_L = [29\ 26\ 22]$ (for $P_H = 1$) and $1/(P_L-1) = [0.0357\ \ 0.04\ \ 0.048]$ while $\alpha = [0.10\ 0.025\ 0.0066]$. When more than 11 significant bits are used, $\alpha << 1/(P_L-1)$ and thus all the constructions provide similar speedups.

Looking further into the obtained results, the last two methods (i.e. the proposed method and the MPFR method), provide the best performance when $P_H = 1$ instead of $P_H = 2$, which is the case for the first method (Gappa++) for

TABLE 2
Speed-ups of DP-MCMC and CF-MCMC FPGA samplers against an 8-core CPU sampler.

FPGA designs	DP-MCMC	CF-MCMC at various precisions (number of significant bits)							
		5	7	9	11	13	15	19	23
Speedup vs. 8-core CPU	44.3x	22.5x	56.1x	115.6x	156.7x	169.6x	180.5x	157.5x	148.0x

precisions 7 and 9. This is due to the fact that the proposed method and MPFR provide tighter bounds than the first method (Gappa++), and as a result the proportion of bright data decreases at these two precisions points, leading to configurations that utilise fewer high precision evaluation units.

6.8 Comparison to a Multi-core CPU Implementation

The proposed system was also evaluated against an optimised version of the standard (i.e. double precision) MCMC algorithm that was running on a multicore system using the MNIST case study. The selected system was running CentOS 7 64-bit and utilised an Intel i7-3770 processor with 8 cores and 8 GBs of RAM, where the compiler is the gcc version 4.8.3. The CPU-based system was developed using OpenMP in order to utilise all the available cores in the system (i.e. 8), as well as using -O3 optimisations. The program takes full advantage of the available cores (100% utilisation), as the likelihood calculations are distributed evenly across the available cores (assumption of i.i.d data).

The obtained speedup results (i.e. Effective Speed-up) are shown in Table 2. The DP-MCMC implementation achieves a speedup around 44.3x against the CPU, where the proposed CF-MCMC achieves speedups in the range between 22.5x to 180.5x. depending on the utilised custom-precision. The above results demonstrate the speedup gains of the proposed system.

6.9 Comparison to an FPGA Implemented FlyMC Algorithm

Finally, a comparison against the FlyMC algorithm proposed in [14] is performed in this section, as this is the closest work to ours. FlyMC algorithm can provide considerable speed ups when it is compared against a regular MCMC algorithm implementation in software, as is acknowledged by the authors [14]. For this to be the case, a lower bound function needs to be constructed as well as tuning for each data point needs to be performed through MAP in order to ensure tight bounds at the data points. The authors in [14] call this version of the algorithm MAP-tuned MCMC. However, such lower bound functions can be difficult to be obtained for many problems [14], and MAP tuning for each data point is required prior to the execution of the system imposing overheads to the overall execution time of the algorithm. The authors also investigate an alternative implementation of FlyMCMC that skips the MAP tuning for each data point and call this algorithm Untuned FlyMC. However, their obtained results show that the actual speed ups compared to regular MCMC are less impressive and sometimes it can lead to longer execution times compared to regular MCMC.

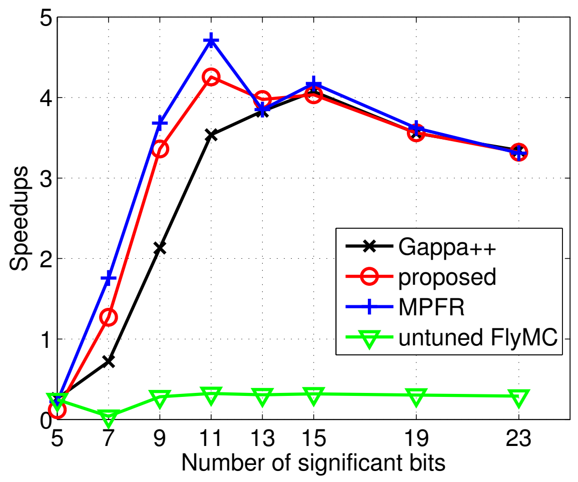


Fig. 10. Achieved speed-ups in terms of the resulting sampling efficiency of the original FlyMC algorithm in [14] and our proposed algorithms over the double-precision MCMC implementation (DP-MCMC) on the target device for the MNIST problem.

The important point of departure between this work and [14], is that the lower bound functions are custom precision versions of the target distribution and are automatically calculated by our system. As such, our proposed algorithm can be applied to any problem but as a trade-off it does not give the impressive speed ups claimed in [14] (i.e. for the MNIST problem, the authors claim 22x speedup for the MAP-tuned FlyMC compared to their regular MCMC implementation in a CPU).

In order to investigate the performance of the FlyMC algorithm in an FPGA, and how it compares against an implementation of a regular MCMC and the proposed CF-MCMC algorithm, the untuned FlyMC was mapped in an FPGA. To further explore the custom precision supported by the FPGA device, the lower bound function in the FlyMC architecture was implemented under different precision regimes (i.e. 23 significant bits correspond to a single precision floating point implementation of the lower bound function suggested by [14] for the MNIST problem). The obtained results are shown in Figure 10. As the results indicate, the FlyMC's performance (in terms of effective samples per second) is inferior to the regular MCMC FPGA implementation following the patters that was observed in the corresponding CPU implementations [14]. Furthermore, the use of custom precision in the implementation of the lower bound function leads to similar performance systems. In both cases, the underlying reason for leading to these performance points is the loose lower bound function used in the system. In any case, the proposed algorithm outperforms the FPGA implementation of the FlyMC.

7 Conclusions

In this work, the CF-MCMC algorithm and its mapping to an FPGA device was presented. The proposed algorithm exploits the custom precision support of FPGAs in order to accelerate computational bounded MCMC problems, by utilising low precision calculations of the likelihood function in an intelligent way. The key contribution of this work is that by introducing a set of auxiliary variables, the proposed CF-MCMC accelerator guarantees the generation of unbiased estimates which is important for applications that cannot tolerate any bias in the estimates. Experimental results show that notable speedups over double-precision designs can be achieved with our proposed architecture in both SW and HW implementations. Future work will focus on extending this methodology to support multiple lower bound functions and their corresponding lower precision computational blocks, as well as investigating the impact to the performance of the system of MAP tuned lower bound functions per data point as in MAP-tuned FlyMC.

References

[1] E. Angelino, E. Kohler, A. Waterland, M. Seltzer, and R. P. Adams, "Accelerating MCMC via parallel predictive prefetching," *arXiv preprint arXiv:1403.7265*, 2014.

[2] J. S. Liu, *Monte Carlo strategies in scientific computing*. Springer, 2001.

[3] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[4] C. Geyer, "Introduction to Markov Chain Monte Carlo," *Handbook of Markov Chain Monte Carlo*, pp. 3–48, 2011.

[5] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 681–688.

[6] G. Mingas and C.-S. Bouganis, "Population-Based MCMC on Multi-Core CPUs, GPUs and FPGAs," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1283–1296, April 2016.

[7] M. Hoffman, F. R. Bach, and D. M. Blei, "Online learning for latent dirichlet allocation," in *advances in neural information processing systems*, 2010, pp. 856–864.

[8] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[9] A. Gothandaraman, G. D. Peterson, G. L. Warren, R. J. Hinde, and R. J. Harrison, "FPGA acceleration of a quantum Monte Carlo application," *Parallel Computing*, vol. 34, no. 4, pp. 278–291, 2008.

[10] X. Tian and K. Benkrid, "Design and implementation of a high performance financial Monte-Carlo simulation engine on an FPGA supercomputer," in *Proc. FPT*, Dec 2008, pp. 81–88.

[11] R. Bardenet, A. Doucet, and C. Holmes, "On Markov chain Monte Carlo methods for tall data," *arXiv preprint arXiv:1505.02827*, 2015.

[12] 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 (ICML-14)*, 2014, pp. 181–189.

[13] R. Bardenet, A. Doucet, and C. Holmes, "Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 405–413.

[14] D. Maclaurin and R. P. Adams, "Firefly Monte Carlo: Exact MCMC with Subsets of Data," in *Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 07/2014 2014.

[15] M. Quiroz, M. Villani, and R. Kohn, "Speeding up MCMC by efficient data subsampling," *arXiv preprint arXiv:1404.4178*, 2014.

[16] G. Mingas, F. Rahman, and C.-S. Bouganis, "On Optimizing the Arithmetic Precision of MCMC Algorithms," in *Proc. FCCM*, April 2013, pp. 181–188.

[17] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. Leong, and D. B. Thomas, "A mixed precision Monte Carlo methodology for reconfigurable accelerator systems," in *Proc. FPGA*, 2012, pp. 57–66.

[18] S. Liu, G. Mingas, and C.-S. Bouganis, "An exact MCMC accelerator under custom precision regimes," in *Proc. FPT*, Dec 2015, pp. 120–127.

[19] R. Douc, E. Moulines, and D. Stoffer, *Nonlinear time series: theory, methods and applications with R examples*. CRC Press, 2014.

[20] C. Holmes. (2008) Markov Chain Monte Carlo and Applied Bayesian Statistics: a short course.

[21] M. Quiroz, M. Villani, and R. Kohn, "Exact Subsampling MCMC," *arXiv preprint arXiv:1603.08232*, 2016.

[22] X. Tian and C.-S. Bouganis, "A Run-Time Adaptive FPGA Architecture for Monte Carlo Simulations," in *Proc. FPL*, 2011, pp. 116 –122.

[23] G. Mingas and C.-S. Bouganis, "A Custom Precision Based Architecture for Accelerating Parallel Tempering MCMC on FPGAs without Introducing Sampling Error," in *Proc. FCCM*, 2012, pp. 153 –156.

[24] C. Mak, "Stochastic potential switching algorithm for monte carlo simulations of complex systems," *The Journal of chemical physics*, vol. 122, no. 21, p. 214110, 2005.

[25] M. D. Linderman, M. Ho, D. L. Dill, T. H. Meng, and G. P. Nolan, "Towards Program Optimization Through Automated Analysis of Numerical Precision," in *Proc. CGO*, 2010, pp. 230–237.

[26] D. Sorensen and D. Gianola, *Likelihood, Bayesian, and MCMC methods in quantitative genetics*. Springer Science & Business Media, 2002.

[27] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel resampling for particle filters on FPGAs," in *Proc. FPT*, Dec 2014, pp. 191–198.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design and Test of Computers*, vol. 28, pp. 18–27, 2011.

[30] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding," *ACM Trans. Math. Softw.*, vol. 33, no. 2, Jun. 2007.

PLACE PHOTO HERE

Shuanglong Liu is a PhD candidate in the Department of Electrical and Electronic Engineering of Imperial College London, UK. He received the B.S and M.S degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2010 and 2013 respectively. His main research interests are hardware accelerations for Statistical Inference problems, including MCMC and SMC using FPGAs and GPUs.

PLACE PHOTO HERE

Grigorios Mingas is a PhD candidate and Research Assistant in the Department of Electrical and Electronic Engineering of Imperial College London, UK. He received the M.Eng. degree in Electrical and Computer Engineering in 2010 from the Aristotle University of Thessaloniki, Greece. His main research interests are reconfigurable and high performance computing for MCMC/SMC, linear algebra and SLAM acceleration.

PLACE PHOTO HERE

Christos-Savvas Bouganis is a Senior Lecturer in the Department of Electrical and Electronic Engineering of Imperial College London, UK. He is an editorial board member of IET Computers and Digital Techniques and Journal of Systems Architecture. His research interests include the theory and practice of reconfigurable computing and design automation, mainly targeting digital signal processing algorithms.