An Exact MCMC Accelerator Under Custom Precision Regimes

Shuanglong Liu, Grigorios Mingas, Christos-Savvas Bouganis
Department of Electrical and Electronic Engineering
Imperial College London
London, UK
Email: {s.liu13, g.mingas10, christos-savvas.bouganis}@imperial.ac.uk

***Abstract*—Markov chain Monte Carlo (MCMC) is one of the most popular and important tools to generate random samples from probability distributions over many variables which occur frequently in Bayesian inference. However, MCMC cannot be practically applied to models with large data sets because of the prohibitive costly likelihood evaluations for each data point. Previous solutions propose to compute the likelihood approximately, e.g. by sub-sampling data or by using custom precision implementations. These methods involve a trade-off between bias in the output and sampling speed; therefore they cannot guarantee unbiased sampling, which is critical in many applications. This paper introduces a novel mixed precision MCMC accelerator for FPGAs, which simulates from the exact probability distribution in contrast to existing approximate MCMC samplers. An auxiliary binary variable is appended to each data point to indicate the corresponding likelihood term evaluation in full or reduced precision. The proposed method guarantees unbiased samples, while the large majority of likelihood computations are performed in reduced precision. Moreover, a tailored FPGA architecture for the algorithm is introduced, and its performance is evaluated using two Bayesian logistic regression case studies of varying complexity: a 2-dimension synthetic problem and MNIST classification with 12-dimension parameters. The achieved speedups over double-precision FPGA designs are 4.21x and 4.76x respectively.**

I. Introduction

MCMC is a popular and successful general-purpose algorithm to draw samples effectively from arbitrary probability distributions in Bayesian inference problems such as parameter estimation and logistic regression [1]–[4]. It has thus been widely used in a range of statistical applications, including machine learning, computational physics, population genetics and statistical classifications [1]–[3].

The MCMC algorithm tackles the problem of sampling from a known probability distribution with the purpose of using the generated samples to estimate otherwise intractable integrals (this task is known as Monte Carlo integration). These integrals are commonly encountered in various scientific fields, notably Bayesian inference; most tasks in Bayesian inference, e.g. prediction and model comparison, boil down to estimating such complex integrals. When dealing with large data sets, the computational bottleneck of MCMC is the evaluation of the Bayesian likelihood function in every step of the algorithm. Assuming independent and identically distributed (*i.i.d.*) data, the likelihood function complexity is $O(N)$, where N is the size of the data set.

Therefore, accelerating the likelihood computation is the most crucial task in order to allow the application of MCMC to complex models with large-scale data sets. 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.

Recent technological progress tends to approach such problems using approximate methods [5]–[8]. The most popular one is data subsampling [9]–[12], which only computes the likelihood for a subset of the whole data. Other approaches are based on implementing likelihood-related arithmetic operators in custom precision arithmetic in reconfigurable hardware such as FPGAs, in order to achieve low latency and allow for more parallelism. However, both methods lead to biased samples and also big variance in the output estimate. There is a large number of applications where unbiased estimates are required, and MCMC algorithms are expected to generate exact samples in these problems [13].

In this paper we propose a novel mixed precision MCMC algorithm for exact MCMC simulations, along with a tailored architecture that maps the algorithm to an FPGA. The obtained samples are always bias-free, independent of the application, which is not the case in the state-of-the-art systems. The key idea behind this work is the introduction of auxiliary variables that assign the likelihood computation to be performed in high or low precision. By properly integrating these variables within the MCMC construction, unbiased estimates are computed, and at the same time a significant part of the likelihood computation is performed in low precision with a significant impact on the overall performance of the system. The main contributions of this work are:

1) A novel mixed precision MCMC algorithm which leads to unbiased estimates, taking into account the unique custom precision capabilities of FPGAs;

2) A novel architecture which maps the algorithm to an FPGA. The architecture includes the necessary data structures and sampling mechanisms to accommodate the use of the auxiliary binary variables. With the proposed data structure for storing the auxiliary variables, both high and low precision data paths can be fully pipelined, and thus further improving the throughput;

The proposed methodology is evaluated using two Bayesian logistic regression case studies of varying complexity, which confirms the unbiased MCMC accelerator. The optimal precisions for the two applications are also given, in which the achieved speedups over double-precision FPGA designs are 4.21x and 4.76x respectively.

II. Background

A. Markov Chain Monte Carlo

MCMC methods can be used to sample from any arbitrary distributions $p(\theta)$ that are complex. The samples generated by MCMC are typically used to estimate integrals such as:

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

where $p(\theta)$ is the probability distribution of θ, $f(\theta)$ is the function of interest (e.g. mean, moment). Using the MCMC samples we can approximate the integral $I(f)$ (or very large sums) with tractable sums that converge (as the number of samples N_s grows) to $I(f)$:

$$\tilde{I}(f) = \frac{1}{N_s}\sum_{n=1}^{N_s} f(\theta_n) \longrightarrow I(f), as\, N_s \to \infty \tag{2}$$

which is an unbiased estimator of the integral $I(f)$.

Let's denote the parameters of interest as θ of D dimension, and assume that N data points $\{x_n\}_{n=1}^N$ 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 common MCMC algorithm (Metropolis-Hastings) in Code 1. In each iteration it consists of a proposal (a Gaussian random walk proposal is presented in line 2) that is stochastically accepted or rejected (in line 4-9). The major computation load lies in the evaluation of the full posterior likelihood 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 the case) 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

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

When the data set is large, the evaluation of all N likelihoods is the computational bottleneck. For this reason, this work focuses on how to minimize the cost of evaluating these $L_n(\theta)$, but without introducing error in the output estimate in (2).

MCMC samples are correlated due to the use of a Markov chain to generate samples. This dependence leads to an increase in the variance of the estimate (2), compared to the case where independent samples are used. This loss in efficiency can be quantified by the Effective Sample Size (ESS) 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 reduction in the true number of samples, compared to *i.i.d.* samples. Therefore, ESS is necessary to include the effect of sample dependence in performance. In other words, the proper performance metric for all MCMC samplers is ESS/sec, which combines raw sampling speed (runtime) and ESS.

Code 1 Metropolis-Hastings MCMC Algorithm

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 $u \le a$ then
6:       $\theta_i = \theta'$;
7:    else
8:       $\theta_i = \theta_{i-1}$;
9:    end if
10: end for
```

B. Related Work

Several recent works have proposed to use subsets of data to perform MCMC in which only a fraction of the whole data set is computed to give the full likelihood. As these proposals only provide approximations, in many cases we have to bound the error of the resulting samples. The ability to support customizable data-paths of different precisions in reconfigurable hardware motivates FPGA researchers to use reduced-precision data paths to implement the likelihood computations. As the low precision (custom floating point) data-paths usually consume fewer resources and lead to a higher degree of parallelism compared to full precision (double floating point) data-paths for a fixed area resources, this methodology can lead to higher performance. However, it also affects the quality of the samples because of the approximations in each likelihood evaluation. When exact estimates are required, the utilisation of high precision data-paths with lower performance are unavoidable. This makes FPGAs less attractive in MCMC applications with high accuracy requirements [13].

The cost of likelihood computations in MCMC approaches has been particularly acute as it necessarily needs to access all of the data at each iteration in order to arrive at a correct answer (accept or reject) to the proposal [11]. This makes MCMC prohibitively slow to converge, especially when the distribution is high-dimensional and multi-modal. A method using subsets of data to compute the likelihood has been firstly proposed by Korattikara et al. in 2014 [9]. It introduces an approximate Metropolis-Hasting rule with controlled bias that allows accepting or rejecting samples with high confidence using only a fraction of the data. [10] proposes an adaptive subsampling technique which is an alternative approximate implementation that only requires evaluating the likelihood of a random subset of the data. This algorithm is a more robust approach compared to [9] and can provide estimates under a user-controlled error. However, both algorithms in [9] and [10] still lead to a biased result, making them not applicable for unbiased estimates. [12] uses an efficient 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. It requires computing the surrogate likelihood for all data before running the subsampling step, thus introducing another costly requirement. [11] presents an auxiliary variable MCMC algorithm that also queries the likelihoods of a small subset of the data but achieves exact posterior distribution. However, the fundamental assumption of this approach is that each product term in the likelihood can be bounded by a function easier to compute. There is a gain only if the lower bound is tight enough.

Most FPGA-based Monte Carlo designs exploit the reduced precision data-paths to produce MCMC outputs within the required error tolerance. Previous works on FPGAs using mixed precision can be found in [13]–[16]. [15] can only be applied to the MCMC algorithms which use multiple parallel chains. In [14], high- and low- precision simulations are compared using the Kolmogorov-Smirnoff metric and the precision is adapted so that a threshold is not violated. However, placing such a threshold is empirical and does not constrain the bias. [13] uses 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] uses the optimal precision under a predefined set of precision configurations to satisfy the user's bias tolerance requirements. Besides the introduced error and bias correction in the outputs of the system, the biggest problem for all previous works using custom or mixed precision on FPGAs is that it's hard to guarantee which precision to be utilised that would lead to sufficiently accurate result defined by the user.

Here we also focus on accelerating the likelihood evaluation part of MCMC, but we arrive at a method in which the samples are generated from true posterior distribution. Here we apply the underlying idea in [11] to the custom precision support on FPGAs, in order to reduce the computation time as well as achieve an unbiased estimator. To the best of our knowledge, this is the first work to produce and guarantee an unbiased MCMC estimation using mixed precision design on FPGAs. Also it is the first work to present the combination of the ideas on custom precision design and data subsampling in MCMC for big data applications, which results in an unbiased MCMC accelerator.

III. Mixed Precision MCMC Methodology

Our novel mixed precision methodology is motivated by the underlying idea in [11], but we adapt it so that it can be used in combination with custom precision arithmetic operators. Instead of using approximate functions for the likelihood terms as in [11], we use custom precision approximations and we show how we can make sure that these approximations are a lower bound to the true likelihood term (which is a requirement for [11] to generate accurate samples from the posterior distribution).

The aim is to use reduced precision for as many terms in the likelihood as possible. We denote the value of likelihood term $L_n(\theta)$ when computed in double precision with $LD_n(\theta)$ and when computed in reduced precision with $LC_n(\theta)$. For each data point x_n, we introduce a binary auxiliary variable $z_n \in \{0, 1\}$ which indicates the type of the likelihood term computation i.e. full or low precision. Each z_n has the following Bernoulli distribution conditioned on the relative difference between $LD_n(\theta)$ and $LC_n(\theta)$:

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

We augment the posterior distribution with these N variables:

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

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

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

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

According to [11], in order for the posterior distribution to be correct the main requirement is that $LC_n(\theta)$ is a lower bound on the full precision likelihood $LD_n(\theta)$, i.e. $LC_n(\theta) \leq LD_n(\theta)$. [11] uses specific expressions (distribution classes) for the lower bound. In order to achieve this in a custom precision setting, we make use of the tool Gappa++ [17] which determines and verifies numerical behaviour, and particularly rounding error in computations with floating point operations. Gappa++ provides tight and guaranteed error bounds between the two floating precision evaluations $LD_n(\theta)$ and $LC_n(\theta)$ i.e. $|LD_n(\theta) - LC_n(\theta)| < \varepsilon$. The likelihood $LC_n(\theta)$ is thus obtained by computing (4) in custom precision and then subtracting the bounded error ε, i.e.

$$LC_n(\theta) = p(x_n \mid \theta) - \varepsilon \tag{9}$$

in order to make sure it is a lower bound on $LD_n(\theta)$ and ensure that $LC_n(\theta) > 0$.

Consider each product of the joint distribution:

$$\begin{aligned} &p(x_n \mid \theta) p(z_n \mid x_n, \theta) \\ &= LD_n(\theta) [\frac{LD_n(\theta) - LC_n(\theta)}{LD_n(\theta)}]^{z_n} [\frac{LC_n(\theta)}{LD_n(\theta)}]^{1-z_n} \\ &= \begin{cases} LD_n(\theta) - LC_n(\theta) & \text{if } z_n = 1 \\ LC_n(\theta) & \text{if } z_n = 0 \end{cases}. \end{aligned} \tag{10}$$

Note that the full precision likelihood $LD_n(\theta)$ now only appears in those data (we call "bright data") for which $z_n = 1$. At a given iteration, the data points for which $z_n = 0$ are dark: we can run the MCMC by computing their likelihoods only in

Code 2 FPGA Firefly MCMC Algorithm with mixed precision

Input: initial setting θ_0 and $\{z_n\}_{n=1}^{N}$, 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, $s^2 I_D$);
3:      $L(\theta') = 1$;
4:      for $n = 1$ to $N$ do
5:          $u \sim$ Uniform(0,1);
6:          if $z_n = 1$ then
7:              $L(\theta') = L(\theta') * (LD_n(\theta') - LC_n(\theta'))$;
8:              if $1 - LC_n(\theta')/LD_n(\theta') \leq u$ then
9:                  $z_n = 0$;
10:             else
11:                 $z_n = 1$;
12:             end if
13:         else
14:             $L(\theta') = L(\theta') * LC_n(\theta')$;
15:             if $n\%ResampleFraction = 0$ then
16:                 if $1 - LC_n(\theta')/LD_n(\theta') \leq u$ then
17:                     $z_n = 0$;
18:                 else
19:                     $z_n = 1$;
20:                 end if
21:             else
22:                 $z_n = 0$;
23:             end if
24:         end if
25:     end for
26:     Compute $a = \frac{L(\theta')}{L(\theta_{i-1})}$;
27:     $u \sim$ Uniform(0,1);
28:     if $u \leq a$ then
29:         $\theta_i = \theta'$;
30:     else
31:         $\theta_i = \theta_{i-1}$;
32:     end if
33: end for
```

reduced precision. Therefore, the full likelihood is now given by:

$$L(\theta) = \prod_{i}^{z_i=1} (LD_i(\theta) - LC_i(\theta)) * \prod_{j}^{z_j=0} LC_j(\theta) \qquad (11)$$

The low precision likelihood can be seen as a close and tight bound to the true likelihood as in [11]. Therefore, the bright data are only a small subset in the whole data set. Thus we have just shifted the computational burden from evaluating the $LD_n(\theta)$ to evaluating the $LC_n(\theta)$. Compared to the idea presented in [11], the main benefit lies in that the low precision likelihood is naturally a tight bound to the true likelihood. We don't need to choose the bound to have a convenient form, which as reported in [11] depends on the application.

This FPGA firefly algorithm (here still called firefly MCMC to be in correspondence with [11]) is shown in Code 2. As described in [11], the resampling of the z_n, which is shown in Code 3, can sample from the Bernoulli distribution for a random fixed-size subset of the data. This results from the fact that at every iteration most of the binary variables keep unchanged. In this paper, the sampling step for z_n is shown in lines 8-12 and 15-23 of Code 2, which is run 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, we can reuse these values and resample all the bright variables without extra computational cost. Since most updates to $z_n = 0$ will leave it unchanged, we resample the dark variables at a fixed fraction rate (1/ ResampleFraction as shown in line 15) to avoid computing the full precision likelihoods for all the dark data, which will be given more details in Section IV.

Code 3 Explicit z_n sampling for Firefly MCMC Algorithm in the reference [11].

```
1: for $j = 1$ to $N \times ResampleFraction$ do
2:     $n \sim RandInteger(1, N)$;
3:     if $1 - LC_n(\theta)/LD_n(\theta) \leq u$ then
4:         $z_n = 0$;
5:     else
6:         $z_n = 1$;
7:     end if
8: end for
```

IV. Proposed Hardware Architecture

The FPGA-mapped MCMC sampler contains two main blocks as shown in Figure 1: a hardware block for the generic MCMC operations and a block to compute the full likelihood $L(\theta)$ in log (to avoid numerical instability [18]). Because likelihood evaluations dominate the computational cost, the key for MCMC accelerator design is to increase the throughput of the likelihood evaluator block.

When the likelihood can be decomposed into sub-components (which is assumed in this paper), FPGA implementations typically use a likelihood evaluation block which consists of parallel likelihood modules [19]. We denote the number of modules by P and we aim at maximizing this P through the use of reduced precision. Accordingly the data memory is partitioned into P blocks, thus each evaluator block processes one block of data. Finally the total sum (i.e. the full log likelihood) is computed using an adder tree. An example of the conventional double-precision floating point design at $P = 4$ is shown in Figure 2. However, double-precision floating operators prevent high parallelism factor (i.e. degree P) to be achieved because double precision arithmetic operators consume a lot of FPGA areas (see Section VI). This motivates the idea in this paper to implement low precision data path in order to save computational resources and achieve more parallelism.

Figure 3 shows the architecture of the FPGA firefly MCMC algorithm in Code 2 as it is mapped on FPGAs. One high-precision data path and P low-precision data paths are implemented for the likelihood evaluation, while the data are stored in P memories and each data memory is attached with a BM and DM memory to store the indexes of the bright and dark data points respectively. An efficient data structure to store and update the auxiliary variable is presented here. We propose to store the indexes of the bright and dark binary variables (rather than the binary value of each z_n) separately in two independent memories denoted as BM and DM respectively.

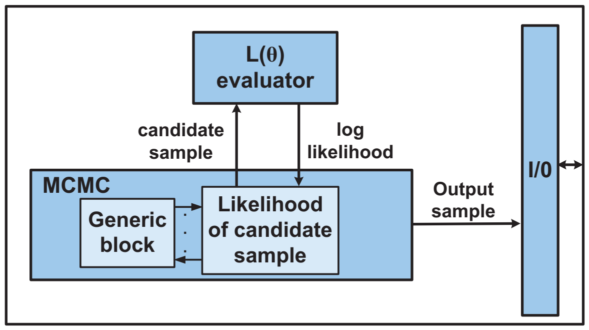


Fig. 1. The overall architecture of the FPGA-mapped MCMC sampler which mainly contains the generic and likelihood $L(\theta)$ evaluator block.

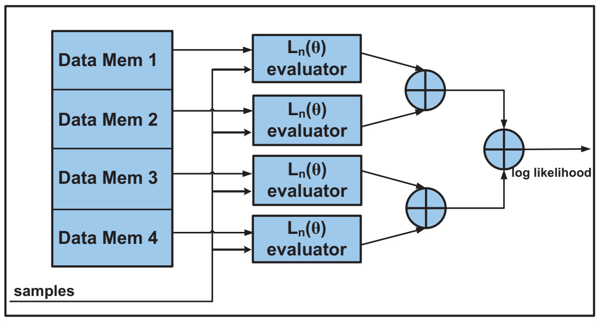
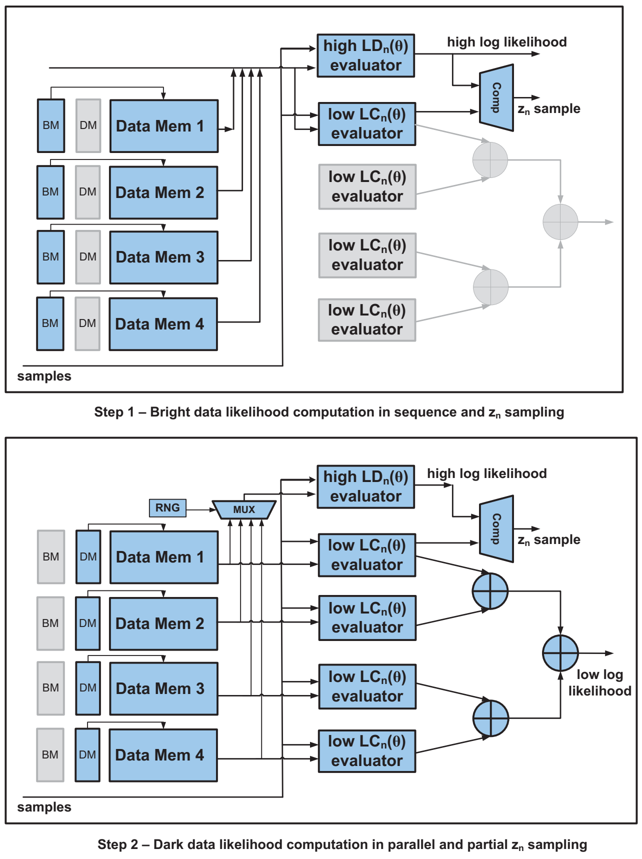


Fig. 2. The architecture of double-precision floating point likelihood $L(\theta)$ evaluator design with the conventional parallel implementation at $P = 4$.



high LDn(θ) evaluator
high log likelihood
low LCn(θ) evaluator
Comp
zn sample
BM
DM
Data Mem 1
Data Mem 2
Data Mem 3
Data Mem 4
samples
Step 1 – Bright data likelihood computation in sequence and zn sampling
RNG
MUX
low log likelihood
Step 2 – Dark data likelihood computation in parallel and partial zn sampling

Fig. 3. The architecture of the parallel FPGA firefly MCMC algorithm using mixed precision design at $P = 4$. The full likelihood is computed and the binary variables are updated by two steps: 1) Bright data likelihood computation in sequence and z_n sampling; 2) Dark data likelihood computation in parallel and partial z_n sampling. The dark blocks indicate they remain idle at the corresponding step.

For each memory we keep track of the number of total bright and dark points. With the proposed data structure for storing the variables z_n, we can ensure that both full and low precision data paths are fully pipelined.

In each iteration, Steps 1 and 2 in Figure 3 (corresponding to lines 6-12 and 13-24 respectively in Code 2) which process the likelihood computation and z_n sampling need to be performed to generate one sample. First, we read the P BM memories one by one to use the bright data index to access the bright data in the corresponding data memory sequentially, then pass them to the high-precision data path 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 low-precision data path to compare the difference between these two values $LD_n(\theta)$ and $LC_n(\theta)$ to update z_n for next iteration. Therefore, the other $(P-1)$ low data paths remain idle (appear dark in the figure) during this step. After the first step, the P DM memories are read in parallel to access the dark data points in parallel, then pass through the P low-precision data paths and get the sum of the log likelihood of each path. Accordingly, at every cycle one of the P dark data will be chosen randomly to go through the high-precision data path for updating the corresponding z_n. Therefore only $1/P$ of the dark variables are resampled, and all the data paths are fully used at the second step. Since the dark variables can be resampled at a fixed fraction rate as mentioned previously, here we sample the dark data at the fraction rate $1/P$ based on the degree of parallelism (P) we achieved for the low precision path, in order to keep all data paths busy and maximize utilization.

Finally the full log likelihood is obtained as the sum of the log likelihood at each step. The execution time (in clock cycles) of the architecture to generate one MCMC sample in each iteration is the total clock cycles to run the two steps which is given by:

$$T_{FPGAfirefly} = N\alpha + N(1-\alpha)/P \qquad (12)$$

where N is the number of the data, α is the proportion of bright data and P is the parallelism of the low-precision path. The traditional double-precision design shown in Figure 2 needs $T = N/P$ clock cycles in total for generating one sample where P is the number of double-precision likelihood evaluator blocks. The above execution time refers to raw speed of the MCMC sampler. As mentioned before, the resulting sampling efficiency to compare different MCMC algorithms needs to include the effect of sample dependence using ESS given in (5). This is measured by dividing the ESS by the execution time i.e. ESS/T.

V. Case Studies

Two popular problems in Bayesian inference: 2-class classification with small and large parameter dimensionality and data size that utilize a logistic regression model are evaluated in this section. 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.

A. *Synthetic Example*

We first demonstrate the workings of our algorithm by performing logistic regression on a synthetic data set, a two-class classification problem in two dimensions (and one bias dimension) as presented in [11]. Here the linear model in (13) 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 (14), 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.

$$y = sign(\beta_0 + \beta_1 x_1 + \beta_2 x_2) \tag{13}$$

$$L_n(\theta) = \frac{1}{1 + exp\{-y_n \theta^T x_n\}} \tag{14}$$

B. *MNIST Classfication*

We also applied the proposed firefly MCMC algorithm to a real problem, which is the logistic regression task described in [4]. 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 [20]. 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, which is quite a big problem in current MCMC applications. Each likelihood still takes the form shown in (14) where x_n is the set of features for the nth data point. However the parameter dimension increase to 12 plus a bias. Also the total number of likelihood terms increases from 500 to 2000 compared to the first example.

VI. Evaluation and Experiments

The proposed architecture 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 are implemented in double floating point and reduced custom precisions respectively, using floating point operators generated by FloPoCo [21]. All designs ran on a single 150 MHz clock and fully utilized the FPGA's resources. All results are post place and route.

A. *Quality of MCMC Samples*

We first assess the quality of MCMC sample results. For both problems, we compare three algorithms: regular MCMC using double-precision floating point by default, custom MCMC which uses the reduced precision for the computation of all likelihood terms, our FPGA firefly MCMC in Code 2 using mixed precision to compute the likelihoods. For convenience, the notation sAeB is used in this paper to

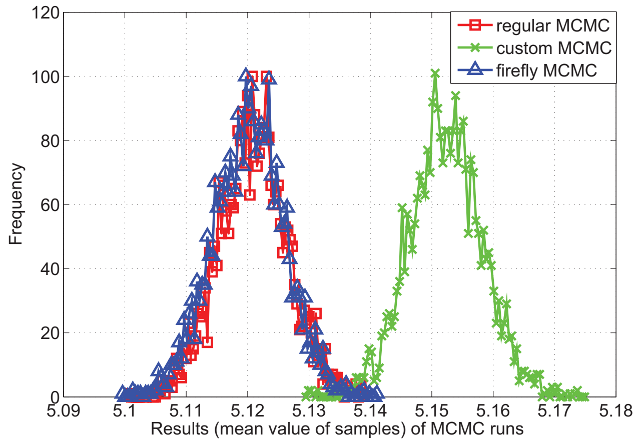


Fig. 4. Distribution of the mean value of the first parameter samples in the 2-dimension problem with 3k runs of the regular MCMC in double precision s52e11, custom and firefly MCMC in reduced precision s8e8.

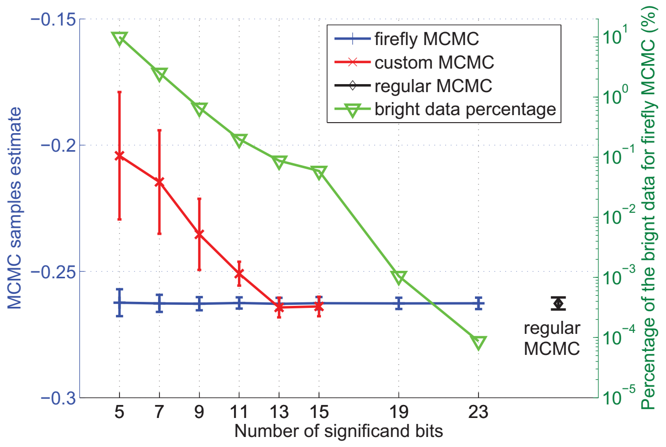


Fig. 5. The bias and its variance estimates of the parameter in the MNIST problem, where regular MCMC runs in double precision (s52e11), custom and firefly MCMC algorithms run in the precision with the number of significant bits from 5 to 23. The green line indicates the average percentage of the bright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

denote a floating point representation, where A is the number of significand bits and B is the number of exponent bits.

Figure 4 shows the distributions of the first parameter β_1 in the synthetic example 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 custom and firefly algorithms are run with the precision s8e8. As the results indicate, when reduced precision data-paths are used for custom MCMC simulation, the mean value of the parameter has a bias and also a larger variance compared to the regular MCMC results, showing no difference compared to the conclusions in [13], [16]. However, the firefly MCMC sample distributions have the same variance and mean as the regular MCMC samples, which demonstrates that the proposed algorithm removes any bias introduced in the results due to low-precision computations.

Figure 5 shows the bias and the variance of its estimates for the first parameter in the MNIST problem, using a range of

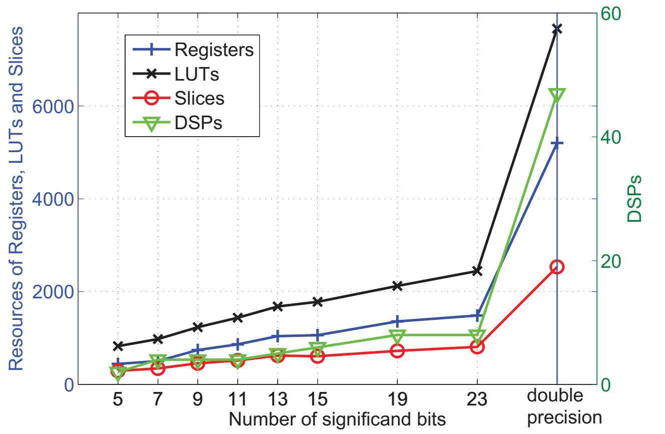


Fig. 6. The resource utilization of a single likelihood evaluation block of the two-dimension (plus a bias) problem with custom precision where the significand bits range from 5 to 23 and a fixed exponent bits at 8, and also double precision.

reduced precisions (with the number of significand bits from 5 to 23 and a fixed exponent bits number 8) in custom and firefly MCMC with comparison to regular MCMC simulations (where double precision with the significand 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 when the precisions become lower for custom MCMC, while firefly MCMC samples always keep almost the same mean value and deviation as regular MCMC samples regardless of how much the precision is reduced. Figure 5 also shows the proportion of bright data point of the data set for firefly MCMC under different reduced precisions. Since the low precision likelihood is a tight bound to the double-precision likelihood, the proportion of bright data is only in the range of 0.0009% (with single-precision s23e8) to 10% (with a very small precision s5e8) among the evaluated precisions.

B. Resource Utilization and Sampling Efficiency

The proposed architecture in Figure 3 and the traditional double-precision design (regular MCMC) in Figure 2 are both implemented in the target device. Besides the resources of the parallel likelihood evaluation blocks, the firefly architecture also needs some extra resources to implement the z_n sampling. To achieve the maximum throughput, full utilization of the FPGA logic resources is assumed for both architectures. Therefore, a maximum number of parallel log likelihood evaluation block is obtained for each sampler. The execution time for generating one sample of regular MCMC architecture is given by $T = N/P$, and that of the FPGA firefly MCMC architecture can be obtained by equation (12) using the proportion of the bright data which is presented in Figure 5 and the maximum parallelism number P.

Table I gives the resource utilization (Registers, LUTs, Slices etc.) of the generic MCMC block using double floating point arithmetic operators, and also of one log likelihood evaluation block using double-precision for the two case studies.

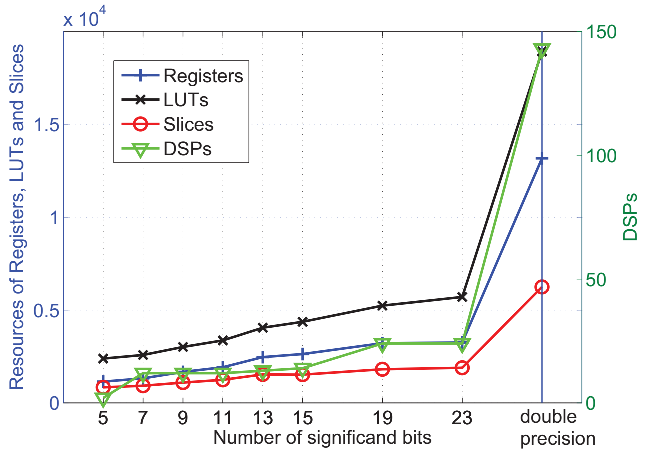


Fig. 7. The resource utilization of a single likelihood evaluation block of the MNIST problem with custom precision where the significand bits range from 5 to 23 and a fixed exponent bits at 8, and also double precision.

TABLE I. RESOURCES OF THE GENERIC BLOCK AND ONE LOG LIKELIHOOD EVALUATION BLOCK USING DOUBLE FLOATING POINT ARITHMETIC OPERATORS ON XILINX VIRTEX-6 LX240T.

Resources (double-precision)		Slices Registers	LUTs	Slices	DSP48E1s
Generic block		2853	4837	1722	11
Log Likelihood Evaluation block	Synthetic Example	5203	7666	2533	47
	MNIST	13168	18919	6242	143

Figures 6 and 7 show the resource utilization of a single likelihood evaluation block of the synthetic example and MNIST application under different reduced precisions respectively. The double-precision block's resources in Table I are also plotted in this figure. The resources increase largely when the utilised precision increases to 64-bit double floating point, which means much more parallel blocks to compute the likelihood can be obtained in the target device under reduced precisions.

Finally, we compare the speedups of our proposed firefly MCMC accelerator over the double-precision design in terms of the resulting sampling efficiency, which is normalized using the ESS divided by the execution time i.e. ESS/T. The results for both problems are shown in Figure 8. The speedups of the two problems with the evaluated precisions are in the order of 0.16x-4.21x and 0.33x-4.76x respectively. When precision is reduced we can achieve more parallelism to implement the low-precision likelihood blocks and thus increase the throughput. At the same time, the bright data proportion will increase, which in turn introduces more running time as the computation for bright data likelihood is executed in sequence. Therefore, there is a trade-off when choosing the precision to achieve the maximum throughput of the firefly MCMC architecture. Besides, there is a large reduction in the effective sample size (ESS) as the precision is reduced to be very small. This is why the sampling efficiency of the design with significand bits smaller than 5 is slower than double-precision design. The optimal precisions for each problem size are (s, e) = (11, 8) and (9, 8) respectively, with the corresponding speedup of 4.21x and 4.76x.

Table II summarise all the results of the regular MCMC

TABLE II. SUMMARY OF THE FIREFLY MCMC RESULTS COMPARED TO THE REGULAR MCMC RESULTS INCLUDING THE SAMPLES' MEAN VALUE, ITS STAND DERIVATION, THE PROPORTION OF BRIGHT DATA AND THE SPEEDUP OF FIREFLY ARCHITECTURE COMPARED TO REGULAR MCMC IMPLEMENTATION.

		Optimal precision	Mean value	STD	Proportion of bright data	Speedup
2-Dim problem	Regular MCMC	-	5.120	0.004975	-	-
	Firefly MCMC	s11e8	5.1225	0.005345	0.2%	4.21x
MNIST problem	Regular MCMC	-	-0.262691	0.002067	-	-
	Firefly MCMC	s9e8	-0.262767	0.00228	0.66%	4.76x

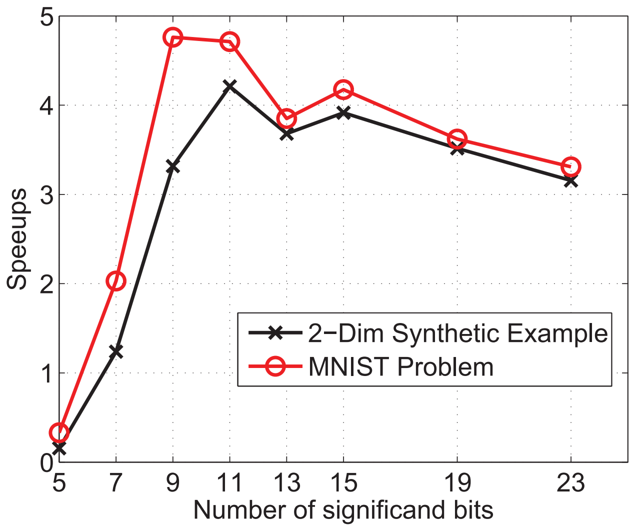


Fig. 8. The Speedups in terms of the resulting sampling efficiency of the proposed firefly MCMC architecture over the double-precision MCMC implementations on the target device for both two problems.

and firefly MCMC for both case studies under the optimal precision in terms of speedups, including the MCMC samples in terms of its mean value and the stand derivation (STD) for each MCMC algorithm (with the same simulation configurations as described previously), the proportion of bright data for firefly MCMC and the speedups over regular MCMC implementations. The optimal speedup results match or exceed the custom precision design presented in [13], [16]. However, our MCMC accelerator provides unbiased samples and has no requirement of bias correction.

VII. CONCLUSION

This paper presents an MCMC accelerating method that exploits the custom precision support on FPGAs to minimize the likelihood computation cost of the MCMC sampler in resources and execution time. The key contribution of this work is that by introducing a set of auxiliary binary variables, our MCMC accelerator generates unbiased MCMC samples which is important for applications that cannot tolerate bias in the estimates. Experimental results show that notable speedups over double-precision designs can be achieved with our proposed architecture. Future work will focus on the on-line selection of the optimal precision, and applying this methodology to large data set problems which should involve off-chip memory access, since this method is also appropriate for data sets which are too large to fit into on-chip memory on FPGAs.

REFERENCES

[1] J. S. Liu, *Monte Carlo strategies in scientific computing*. Springer, 2001.
[2] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.
[3] *Handbook of Markov Chain Monte Carlo*, 1st ed. Chapman and Hall/CRC, May 2011.
[4] 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.
[5] 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.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] M. Quiroz, M. Villani, and R. Kohn, "Speeding up MCMC by efficient data subsampling," *arXiv preprint arXiv:1404.4178*, 2014. [Online]. Available: http://arxiv.org/abs/1404.4178
[13] 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.
[14] X. Tian and C.-S. Bouganis, "A Run-Time Adaptive FPGA Architecture for Monte Carlo Simulations," in *Proc. FPL*, 2011, pp. 116 –122.
[15] 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.
[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] 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.
[18] D. Sorensen and D. Gianola, *Likelihood, Bayesian, and MCMC methods in quantitative genetics*. Springer Science & Business Media, 2002.
[19] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel resampling for particle filters on FPGAs," in *Proc. FPT*, Dec 2014, pp. 191–198.
[20] 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.
[21] 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.