2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines

Communication-Aware MCMC Method for Big Data Applications on FPGAs

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

***Abstract*—Markov Chain Monte Carlo (MCMC) based methods have been the main tool for Bayesian Inference for some years now, and recently they find increasing applications in modern statistics and machine learning. Nevertheless, with the availability of large datasets and increasing complexity of Bayesian models, MCMC methods are becoming prohibitively expensive for real-world problems. At the heart of these methods, lies the computation of likelihood functions that requires access to all input data points in each iteration of the method. Current approaches, based on data subsampling, aim to accelerate these algorithms by reducing the number of the data points for likelihood evaluations at each MCMC iteration. However the existing work doesn't consider the properties of memory hierarchies in modern computational systems, but treats the memory as one monolithic storage space. This paper proposes a communication-aware MCMC framework that takes into account the underlying performance of the memory subsystem during the sampling process, leading to faster execution times. The framework is based on a novel subsampling algorithm that utilises an unbiased likelihood estimator based on Probability Proportional-to-Size (PPS) sampling, allowing information on the performance of the memory system to be taken into account during the sampling stage. The proposed MCMC sampler is mapped to an FPGA device and its performance is evaluated using the Bayesian logistic regression model on MNIST dataset. The proposed system achieves a 3.37x speed up over a highly optimised MCMC-based FPGA design, and therefore the risk in the estimates based on the generated samples is largely decreased.**

I. Introduction

Over recent years, Bayesian methods have become increasingly popular due to their ability to analyse data of complex structures using flexible models [1], and they have been proven to be effective in a wide range of statistical applications such as computational physics, population genetics and statistical classifications [2]–[4].

Markov chain Monte Carlo (MCMC) is a class of algorithms for estimating expectations with respect to distributions which can be intractable like most posterior distributions arising in Bayesian inference. In each iteration of the MCMC algorithm, a likelihood function is evaluated, that is the probability of the parameters to explain the given data, and a pass through the whole dataset is required. Modern inference problems utilise large dataset and the current trend is for these datasets to grow fast [5]. Moreover, the availability of large datasets allow the construction of complex models, leading to computationally expensive likelihood functions. As such, the application of MCMC algorithms to modern problems start becoming prohibited and many researchers and practitioners work on the acceleration of MCMC algorithms.

Two main research directions can be found in the literature for MCMC acceleration. The first direction focuses on the acceleration of the likelihood computation through approximation models [6], and/or parallelising its evaluation based on the assumption of *i.i.d* data [7]. The former approach has the difficulty of selecting a suitable approximation model that has the desired properties of fast evaluation and at the same time obeys certain assumptions on the quality of the approximation [6]. The latter approach has been explored by various works using multi-core CPU and GPU devices, and eventually the system's performance is limited by the available memory bandwidth.

The second direction focuses on reducing the number of data points that need to be processed in every MCMC iteration, and effectively addresses the memory bandwidth problem. Most of the methods in this category propose a sampling scheme to generate an unbiased estimate of the likelihood function based only on a small sample of the dataset [8]–[10]. However, the existing works treat the memory as one monolithic storage space, and are oblivious on the performance characteristics of the various memory technologies of modern memory system hierarchies. As such, the actual latency of accessing a data point from the memory is not taken into account in the MCMC construction, and all the "accesses" are considered to have the same cost (i.e. latency, power).

The work proposed in this paper belongs to the second set of work, and it proposes a sampling-based algorithm that aims to reduce the memory accesses, but it exposes the performance of the memory sub-system to the MCMC algorithm in order to guide the sampling process, constructing as such a communication-aware MCMC algorithm. The key idea is the use of Probability Proportional-to-Size (PPS) sampling, where the inclusion probability of each data point is proportional to its approximate contribution to the likelihood function, allowing the system to reason during run-time on how often a specific data point will be accessed, and as such it can decide on its suitable storage location across the memory hierarchy.

The main contributions of this work are:

- A communication-aware MCMC algorithm based on PPS sampling is proposed, that takes into account the perfor-

978-1-5386-4037-1/17 $31.00 © 2017 IEEE
DOI 10.1109/FCCM.2017.9

mance characteristics of the underlying memory hierarchy. The proposed algorithm reduces the data transfer overheads among memories, compared to the regular MCMC and other subsampling-based algorithms, leading to faster execution times;
- An optimized hardware architecture tailored for FPGA implementation that implements the proposed algorithm and efficiently utilises the on-chip memory blocks;

It should be noted, that FPGAs are particular suited for the proposed algorithm due to the customisation of the use of the on-chip memory blocks, as well as due to the available high on-chip memory bandwidth that allows the full utilisation of of multiple processing elements for the likelihood evaluation, converting as such the MCMC implementation into a memory-bounded problem for most real-life MCMC applications.

II. Background

A. Markov Chain Monte Carlo

Markov Chain Monte Carlo [11] is a family of stochastic algorithms which are designed to draw samples from arbitrary probability distributions $p(\theta)$. The above is necessary in order to solve a variety of problems in Bayesian modelling, e.g. prediction and model comparison, making MCMC a fundamental tool in modern statistics. The samples generated from MCMC are typically used to estimate the expectation of a function $f(\theta)$ with respect to $p(\theta)$, i.e.

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

By collecting N_s samples from a MCMC run, the integral $I(f)$ is approximated by $\tilde{I}(f)$ as follows:

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

which is an asymptotically unbiased estimator [11].

The error in the estimate of (2) due to the execution of the MCMC algorithm for a finite number of steps can be quantified by the risk in the estimate, which is defined as the mean squared error in the estimate of (2), i.e. $R = \mathbb{E}[(I - \tilde{I})^2]$, where the expectation is taken over multiple executions of the Markov chain [8]. This risk can be decomposed as the sum of squared bias and variance, which capture the bias in the estimate and the uncertainty around the produced estimate respectively. It should be noted that the variance is reduced by the generation of more samples. The objective of MCMC in practice is to obtain estimates with lower risk in a given time budget. In approximate MCMC algorithms, a small bias is often allowed in the stationary distribution of the Markov chain. By doing so, a larger number of samples can be collected in a given time and therefore the variance in the estimate is reduced. The design of high performance MCMC algorithms for big data often comes down to the trade-off between bias and variance, and therefore the performance of different algorithms can be studied using the risk of the estimators for a given time budget.

In more details, the MCMC methods are used to generate samples from the Bayesian posterior probability distribution $p(\theta \mid \{x_n\}_{n=1}^N)$ of the unknown parameters θ, given the data of observations $\{x_n\}_{n=1}^N$. A Markov chain with the stationary distribution $p(\theta \mid \{x_n\}_{n=1}^N)$ can be constructed using the Metropolis algorithm (Metropolis MCMC [11]) as described in Code 1, assuming the parameters of θ have D-dimensions. In each iteration of the algorithm, a move in the parameter space is proposed (a Gaussian random walk proposal is presented in line 2) that is stochastically accepted or rejected (in line 4-9). The posterior probability distribution needs to be computed in every iteration as shown in line 3. Assuming that the data points are independent (which is often assumed in real applications), this distribution breaks down into the product of the likelihood of each data point i.e. $p(x_n \mid \theta)$ as:

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

As such, when dealing with applications that utilise large amounts of data, evaluating the above equation can be very costly in terms of both computation and memory requirements.

Code 1 Metropolis 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 $a = \dfrac{p(\theta' \mid \{x_n\}_{n=1}^N)}{p(\theta_{i-1} \mid \{x_n\}_{n=1}^N)}$;
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**

B. Related Work

Several works have been proposed to overcome the computational problems brought by the big data. A detailed review of current MCMC methods used for large datasets can be found in [12]. In [13]–[15], they propose to speed up the required computations by parallelising the likelihood evaluations exploiting the data independence property.

Other approaches can be broadly classified into two groups: Consensus Monte Carlo (CMC) and subsampling-based algorithms. The CMC approaches [16] divide the initial dataset into batches, run MCMC on each batch separately and then combine the results to obtain an approximation of the posterior. However, the strategy to efficiently combine the batch posterior approximations is difficult to obtain and it has no theoretical guarantees for convergence [12].

Subsampling approaches [6], [8]–[10] use subsets of data to provide a faster estimation of the likelihood in which only a fraction of the whole dataset is employed to estimate the likelihood. [8] introduce a statistical hypothesis test relying on the central limit theorem to accept/reject samples with high confidence using only a fraction of the data. [9] propose an adaptive subsampling technique that only requires evaluating the likelihood on a random subset of the data. Both algorithms rely on a bound for the difference between the log-likelihood contributions at the proposed and current sample, and that of the control variates [17]. Recently, [10] demonstrate that the simple random sampling (SI) used in the previous methods often leads to a highly inefficient MCMC chain with extremely poor mixing. [10] propose a subsampling of the data based on the contribution on each likelihood term. This algorithm needs to build a surrogate of the true likelihood, using either a Gaussian process or a spline approximation. As such, it introduces another costly requirement of computing the surrogate likelihood for all data before running the subsampling step. [6] present an auxiliary variable MCMC algorithm that queries the likelihoods of a small subset but achieves exact posterior distribution. The fundamental assumption is that each likelihood term can be approximated from below by another function that is faster to compute. The drawback of this method lies in the construction of these functions as it depends on the target distribution.

Previous work on accelerating MCMC methods on GPUs and FPGAs are very limited. The GPU implementations are mainly focused on parallelising the likelihood computations as in [15]. Most FPGA-based accelerators exploit the reduced precision data-paths to allow for more parallelism for the likelihood computations. The authors in [13] and [14] propose a mixed precision MCMC algorithm, accelerating the computations of the likelihood function guaranteeing at the same time an asymptotically unbiased estimate, where the authors in [4] propose an optimised FPGA architecture for Population-based MCMC. Furthermore, [18] propose a framework for identifying an optimum custom precision number representation in the MCMC architecture, targeting a specific bias-variance ratio at the output.

However, to the best of the authors' knowledge, none of the above works that target GPUs or FPGAs has considered to reduce the memory accesses required by the MCMC algorithm. The proposed work addresses exactly the memory-bound problem in the MCMC construction, opening the way for applying the MCMC algorithm to large scale datasets.

III. Communication-Aware MCMC Method

This work proposes a novel subsampling-based MCMC method which utilises an intelligent way to sample the dataset in order to evaluate the likelihood partially. The main focus of the work is on the approximation of the regular Metropolis test step in line 3 to line 9 in Code 1. As the code illustrates, this is equivalent to compare two values: the reformulated random number u_0 and the average difference μ in the log-likelihoods of θ' and θ_{i-1} (the computations are performed in the log-domain).

$$u_0 = \frac{1}{N}\log(u), \text{where } u \sim \text{Uniform[0,1]} \tag{4}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} l_i, \text{where } l_i = \log p(x_i \mid \theta') - \log p(x_i \mid \theta_{i-1}) \tag{5}$$

If $\mu > u_0$, the proposed sample θ' is accepted, otherwise it is rejected. In this work, the above Metropolis step is approximated similar to [8], and it is casted as a hypothesis test. Given the random sample $\{l_{i_1}, ..., l_{i_n}\}$ drawn from the population $\{l_1, ..., l_N\}$, a statistical hypothesis test is developed to decide whether the population mean is greater or less than u_0 with a user defined confidence. The standard deviation s of the sample mean $\bar{l}$, together with $\bar{l}$, is used to compute the test statistic $t = (\bar{l} - u_0)/s$ which follows a standard Student-t distribution with $n-1$ degrees of freedom. Then $\delta = 1 - \Phi_{n-1}(|t|)$ is computed, where $\Phi_{n-1}(|t|)$ is the cdf of the standard Student-t distribution, an it is compared with a fix threshold (user defined) ϵ, in order to determine the level of confidence for a decision to be taken.

In [8], the authors propose to use equal probability sampling, SI, and the results show that it doesn't work well as a large sampling fraction is needed to take a decision. On contrary, Probability Proportional-to-Size sampling (PPS) is an unequal sampling method where each sample is selected with a probability (which is often called "inclusion probability") that is proportional to its contribution towards the quantity under estimation (5). Compared to SI, PPS can largely reduce the variance in the estimate of the mean, leading to a MCMC chain with many more efficient draws for a given time budget compared to a regular MCMC on the full dataset. Moreover, in unequal sampling, the data points that have been assigned high probabilities will have higher probability to be chosen across iterations than the samples with low probabilities. The proposed work exploits the above property of PPS, in order to reason on the actual storage of the data points during the execution of the algorithm. Thus, the data points with high inclusion probabilities are kept in the on-chip memories where data points with low inclusion probabilities are left in the slower access off-chip memory. Please note that the inclusion probability of a data point depends on the values of the current sample, creating the need to dynamically reallocating data point across the memory hierarchy during the execution of the algorithm. However, due to the smoothness of the likelihood function, and the small distance between the current and proposed samples, the inclusion probabilities do not have to be evaluated in every iteration. In more details, to design a sampling scheme with unequal probabilities, we first construct sampling weights $\omega_i = |l_i|$. Then for a given target size m of the subset $\mathcal{S}$, i.e., $|\mathcal{S}| = m$, each unit is selected with inclusion probability $\pi_i = c\omega_i$ where c is a positive constant satisfying $\sum_{i=1}^{N} \pi_i = m$. The population mean and its variance can be

estimated using the Horvitz-Thompson (HT) estimator[1]:

$$\bar{l} = \frac{1}{N} \sum_{i \in \mathcal{S}} l_i / \pi_i \tag{6}$$

$$var(\bar{l}) = \frac{1}{N^2} \sum_{i \in \mathcal{S}} (1 - \pi_i)(l_i / \pi_i)^2 \tag{7}$$

Given the above estimates, the algorithm follows the same flow as in SI, where t is computed and compared to ϵ in order to determine if a decision can be made.

The proposed communication-aware MCMC (CA-MCMC) algorithm is shown in Code 2. The PPS sampling design is based on the Poisson sampling proposed in [19]. The advantage of Poisson sampling is its simple implementation, and on the simplicity in estimating the variance of the HT-estimator as shown in (7). There are some key points in the proposed algorithm: Firstly, this method doesn't need to update π_is at every iteration. This would require access on the whole dataset, leading to the same cost as in the regular Metropolis step. The algorithm only updates the probabilities when it cannot use the subset to make a decision (defined by the parameter *flag_regular_mcmc* to 1 in line 22). If a decision cannot be taken, a regular Metropolis step is performed. This is shown in line 5 to 9, where the on-chip dataset is also built. Secondly, an adaptive method is proposed to determine the size of the subset M_target which is initialised at m in line 1. The variable $conseq\_done$ is used to record the progression of the algorithm, and it is used to guide the selection of the sample size M_adjust. This is shown in line 24 to 29.

The advantage of the CA-MCMC algorithm compared to the random sampling based MCMC (RS-MCMC) algorithm proposed in [8] is that it reduces the variance in the estimator of the likelihood using PPS sampling. Thus the average size of the subset is reduced. Moreover, as the algorithm is now based on an unequal sampling technique, reasoning on the "optimum" storage location of the data points can be performed. The proposed algorithm can store the data points with high inclusion probabilities in the on-chip memories, with the potential to reduce considerably the data transfer times from the off-chip memory, pushing further the memory-bound problem and allowing more samples to be drawn in a given time budget.

Code 2 Communication-Aware MCMC Algorithm

Input: initial setting θ_0, number of samples N_s, parameters: ϵ, m, M_adjust, $conseq\_thres$;
Output: samples of parameter $\theta_i, i = 1, ..., N_s$;

1: Initialization: $i = 1, flag\_regular\_mcmc = 1$, $M\_target = m, conseq\_done = 0$;
2: **while** $i < N_s$ **do**
3: Propose $\theta' \sim \theta_{i-1}$+Normal(0, $s^2 I_D$);
4: $u \sim$ Uniform(0,1);
5: **if** $flag\_regular\_mcmc == 1$ **then**
6: Compute the log-likelihood term l in (5) for the whole data;
7: Update π_is and build the on-chip dataset;
8: Perform the regular Metropolis step;
9: $i = i + 1; flag\_regular\_mcmc = 0$;
10: **else**
11: Draw a subset $\mathcal{S}$ with $|\mathcal{S}| = M\_target$ from Poisson Sampling, and access the data either from on-chip dataset or the main memory;
12: Estimate $\bar{l}$ and its variance using HT estimator;
13: Compute $\delta = 1 - \Phi_{n-1}(|t|)$;
14: **if** $\delta \leq \epsilon$ **then**
15: **if** $u_0 \leq \bar{l}$ **then**
16: $\theta_i = \theta'$; //accept
17: **else**
18: $\theta_i = \theta_{i-1}$; //reject
19: **end if**
20: $i{+}{+}; flag\_regular\_mcmc = 0; conseq\_done{+}{+}$;
21: **else**
22: $flag\_regular\_mcmc = 1; conseq\_done - -$;
23: **end if**
24: **if** $conseq\_done \geq conseq\_thres$ **then**
25: $M\_target = M\_target - M\_adjust$;
26: **end if**
27: **if** $conseq\_done \leq (-1 * conseq\_thres)$ **then**
28: $M\_target = M\_target + M\_adjust$;
29: **end if**
30: **end if**
31: **end while**

IV. Hardware Mapping

A. IP Architectures and FPGA system Integration

In this section, an FPGA-based architecture is proposed for the implementation of the proposed CA-MCMC sampler, together with two other architectures for the implementation of the regular (traditional) MCMC shown in Code 1 and the RS-MCMC algorithm proposed in [8].

The high-level view of the proposed system is shown in Figure 1. It consists of the MCMC IP (which represents one of the above three algorithms), an AXI bus interface, an ARM Cortex processor and a DMA controller. The ARM processor is responsible to initialise the system, and to read the generated MCMC samples from DDR for further processing. The DMA controller is used by both the ARM processor and the MCMC IP to access the off-chip DDR memory on the board.

A run of the MCMC algorithm proceeds as follows: initially, the implemented MCMC IP of each algorithm is initialised and started by the processor. The IP generates the required number of samples using the dataset which is stored in the off-chip memory (DDR). The generated samples are written in the off-chip memory, and at the end of the processing, the ARM core reads the data for further evaluation.

The architectures of the MCMC IPs and the likelihood datapaths are optimized aiming to speed up the data transfers

[1]Please note that the HT estimator has these expressions under a design of Poisson sampling [19].

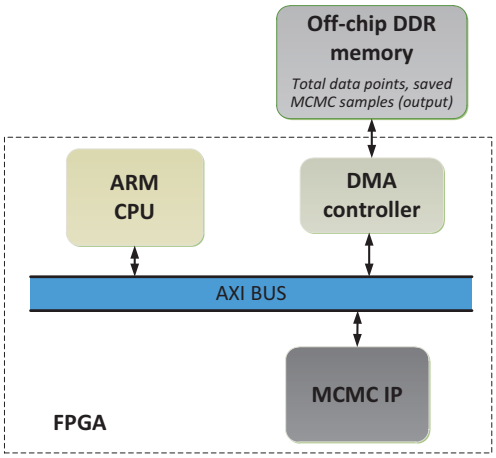

Fig. 1. The architecture of the FPGA system integrated with MCMC IP.

between the off-chip and FPGA device. For each MCMC IP, we built a on-chip memory with the size of M data points using the available BRAMs in the FPGA to reduce the data transfer times from off-chip memory to the IPs. Figure 2 shows the block diagram of the architectures of the three algorithms. Assuming the total data size is N with each component x_n as a vector of D-dimension (consisting of D single precision floating point numbers), DDR stores the whole dataset and the on-chip memory can store M data points. For the regular MCMC, the on-chip memory performs as temporary storage. When computing the likelihood, we first transfer a subset of the data in off-chip to the on-chip memory and constantly process these data points. This is performed repeatedly until we processed all the data points in off-chip memory. For the random sampling based MCMC (RS-MCMC), the on-chip memory performs as a FIFO. In order to draw a subset from the whole dataset, in every cycle a data point is drawn randomly without replacement from off-chip to on-chip memory. At the same time, the likelihood datapath can process the data points from the on-chip memory without stalling. As the data points are randomly chosen, the DDR memory needs to be accessed in every data point acquisition, which is a disadvantage of the RS-MCMC algorithm. Please note that the sampling stage is done in parallel with the computation of the likelihoods (the subset does not depend on the result of the likelihood evaluation), and the existence of the on-chip memory guarantees that the datapaths can be fully pipelined and utilised.

In the case of the proposed CA-MCMC architecture, the on-chip memory performs as a FIFO but also as working memory. In each iteration, when we update π_is, we store a subset of data from the dataset on the on-chip memory using Poisson sampling. As such, the data points with high probability to be selected in the next subset are already in the on-chip memory (which is determined by the "On-chip flag" shown in Figure 2(c)). As the data in the on-chip memory maintain similar inclusion probability values for several iterations, the off-chip access is largely reduced.

To further improve the performance of the systems, the CA-MCMC and RS-MCMC architectures have been parameterised to access a mini-batch of data points B (contiguously stored in memory) in each read operation from the off-chip memory instead of reading one data point each time. The benefit of increasing the mini-batch size B is the reduced latency of accessing B data point from the off-chip memory, but this is in the expense of introducing a more coarse sampling of the dataset. This will be further discussed in Section V.

B. Performance Model

TABLE I
SYSTEM AND PROBLEM PARAMETERS.

N	Total Number of data points.
D	Dimensions of each data point.
N_s	Number of samples to be generated for each MCMC IP.
ϵ	the fixed threshold to control the bias of the RS-MCMC and CA-MCMC IPs.
M	the number of the data points which can be stored in the on-chip BRAMs of the three IPs when mapped on an FPGA.
B	the size of the mini-batch to be transferred from the DDR to FPGA each time for the RS-MCMC and CA-MCMC algorithms when mapped on an FPGA.

As the work is focused on pushing the memory bandwidth bound, the derived performance models capture the required memory accesses to the external memory. As the latency of the on-chip memory to the IP can be designed to be one clock cycle per data point (D floating numbers), as we have shown in Figure 2, and since all datapaths are fully pipelined, the overall performance of the system is dictated by the transfer of data points from the off-chip memory to the IP. Table I provides a summary of the problem and system parameters.

Let N be the total number of data points, where each point is a D-dimensional vector whose elements are under single precision floating point representation. Assuming that the algorithms perform N_s iterations, then the regular MCMC architecture which requires access to all the data points in each iteration, needs N_{MCMC} external memory access in total, which is given by:

$$N_{MCMC} = N_s N D \tag{8}$$

assuming a 32-bit wide external memory.

In the case of the RS-MCMC architecture, the number of data used for the likelihood evaluation is reduced. Due to random sampling, each sampled data-point needs to be transferred from the off-chip memory to the IP. Assuming that on average each iteration requires M_{RS} data points, the expected off-chip memory access is given by:

$$N_{RS-MCMC} = N_s M_{RS} D \tag{9}$$

In the case of the proposed CA-MCMC architecture, most of the data points used for the likelihood evaluation will consist of the data points with high inclusion probabilities. Let's assume

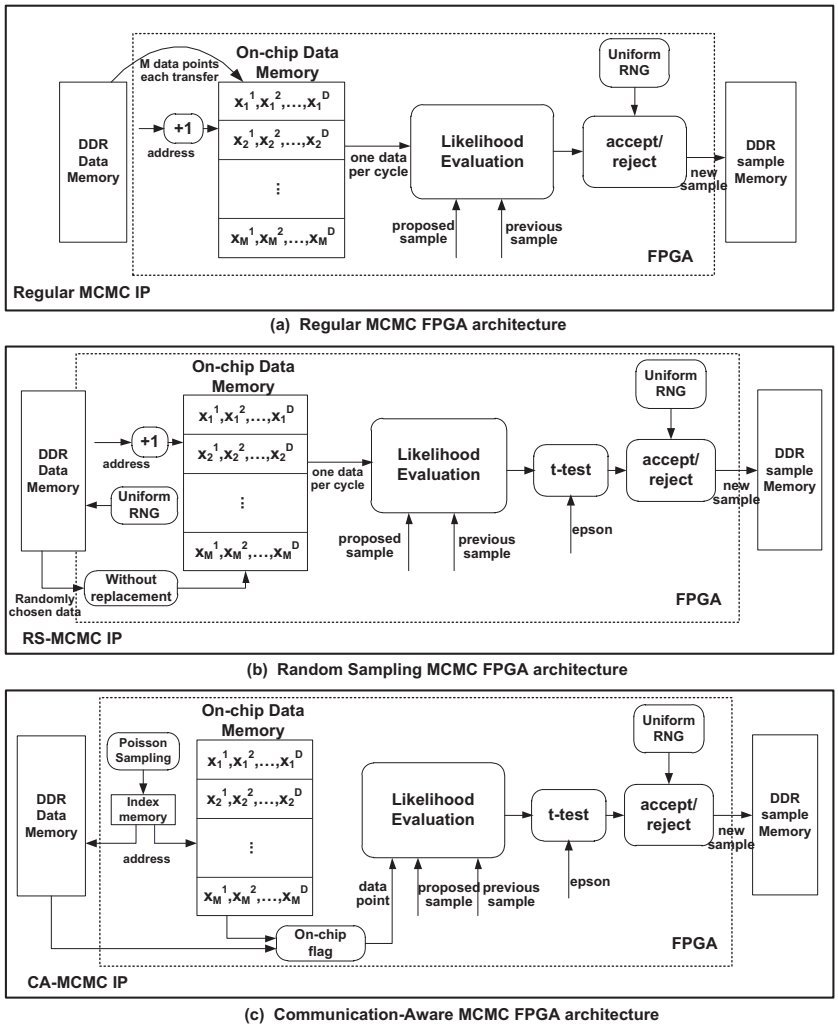

Fig. 2. FPGA architectures of the three implemented IPs: (a) regular MCMC; (b) RS-MCMC; (c) CA-MCMC.

that in average M_{CA} data points used in each iteration (this is different from M_{RS} as the two architectures utilise different sampling techniques), and on average a percentage, α, of these points will be available in the on-chip memory. Please note that α will increase as the size of the on-chip memory (M) increases. Then the off-chip memory access times will be $(1-\alpha)M_{CA}$. Assuming the rate to perform the regular Metropolis step and update the inclusion probabilities π_is is on average β, the total number of off-chip memory accesses for N_s iterations is given by:

$$N_{CA-MCMC} = \beta N_s N D + (1-\beta) N_s M_{CA} * (1-\alpha) \quad (10)$$

V. Evaluation

A. Case Study

As the case study to assess the performance of the proposed system, a Logistic regression problem widely used in the MCMC literature is considered [8], [9], [13]. Logistic regression is used in many fields, including medical and social sciences. In this case study, the target distribution is the posterior for a logistic regression model trained on the MNIST dataset for classifying digits 7 vs 9. The logistic regression likelihood is given by:

$$p(x_n|\theta) = \frac{1}{1 + exp\{-y_n \theta^T x_n\}} \quad (11)$$

where $x_n \in \mathbb{R}^D$ is the set of features for the nth data point and $y_n \in \{-1, 1\}$ is its class. The dataset in the study consists of 10,000 data points chosen from the total 12,214 data points in the database, and the first 12 principal components (and one bias) from PCA are used as features, i.e., $D = 12$.

B. Parameter Selection

In the following sections, an investigation is performed to assess the impact of the various parameters to the overall performance of the algorithms. In the proposed implementation, the initial values for the parameters are as following: $m = 500, M\_adjust = 100, conseq\_thres = 10$. Please note that no actual tuning has been performed in the selection of these parameters, and their values were mainly guided to

be similar to the parameters used in RS-MCMC [8]. The performance of the proposed algorithm under different values of ϵ, and the size of mini-batch size B is investigated in the following sections.

C. Resource Utilization

The three MCMC IPs proposed in Section IV were implemented using Xilinx Vivado HLS, and single precision floating point number representation was used for all datapaths. The overall system was mapped on a Xilinx ZC702 ZedBoard with the system clock (which is also used in the IPs) set to 100 MHz. All the reported results are measured results from the board except if it is stated otherwise.

Table II shows the results of the FPGA resource utilisation for the three MCMC samplers, with the same architecture parameter $M = 1000$ which is limited by the number of BRAMs available in the ZedBoard. The regular and RS-MCMC use fewer BRAMs than the CA-MCMC, as the two algorithms do not substantially benefit from storing a subset of the data on-chip [2]. In the case of the proposed CA-MCMC architecture, extra on-chip memory is needed to store the inclusion probabilities, one-bit flag for each data to remark if the data is on-chip or not, and the address of the on-chip data in off-chip memory, as shown in Figure 2. Please note, as the dimensionality D of the data increases, the above overheads diminish as their storage requirements do not scale with the dimensionality of the data. Also, the logic resources such as LUTs and DSPs for RS-MCMC and CA-MCMC are very close and both utilise more logic resources than the regular MCMC architecture. This is expected as both subsampling-based architectures need to perform extra computations for calculating the mean and variance of the likelihood estimator in order to make a decision at each iteration. It should be noted that all systems fit easily in this relative small FPGA device, indicating the potential use of FPGA devices in this application domain.

TABLE II
RESOURCE UTILISATION OF THE THREE MCMC FPGA-MAPPED SAMPLERS WITH THE ARCHITECTURE PARAMETER $M = 1000$.

IP Block	LUTs	FFs	DSPs	BRAMs
Regular	25639 (48%)	15623 (14%)	105 (47%)	24 (9%)
RS-MCMC	34718 (65%)	20240 (19%)	138 (62%)	36 (13%)
CA-MCMC	33618 (63%)	20756 (19%)	151 (68%)	86 (30%)
Total	53200	106400	220	280

D. Obtained Risk and Speedup

Figure 3 shows how the Risk in estimating the mean parameter of the MNIST decreases as a function of the execution time for the different architectures, having fixed the design parameters as follows: $B = 1$ and $\epsilon = 0.1$. The experiment configuration is the same as in [8]: the true value of the mean is

[2]A small gain can be realised in these two algorithms by having a small proportion of the data on chip. However as the size of the dataset increases, this gain diminishes, which is not the case for the proposed CA-MCMC.

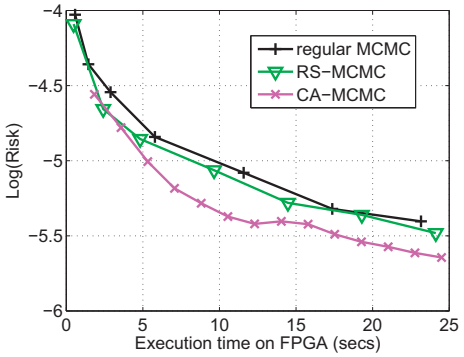

Fig. 3. The Risks in the estimate of the mean value of the first parameter of the logistic model with the design parameters: $B = 1$ and $\epsilon = 0.1$.

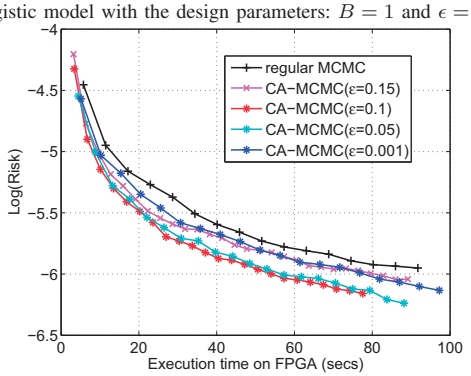

Fig. 4. The Risks of CA-MCMC in the estimate for different settings of ϵ with the design parameter $B = 1$.

estimated using a long run of the regular MCMC algorithm, and then this values is used as a reference point. Multiple estimates are then computed from the three architectures, and the Risk is calculated. In the provided results, the risk is calculated over 200 runs of each algorithm. The figure demonstrates that the proposed architecture largely reduces the risk compared to that of the regular and RS- MCMC, by drawing more samples and reducing the variance faster within the same time period. The obtained speed-ups for the CA-MCMC and RS-MCCM over the regular MCMC are 1.58x and 1.19x respectively. The speed-up of the proposed system CA-MCMC compared to the single precision optimised CPU implementation of the regular MCMC executed in an Intel i7-3770 CPU (single thread) is 13.5x.

The performance of the proposed algorithm was also investigated under different values of ϵ, which determines the required confidence in the estimation of the likelihood for taking a decision in the Metropolis step. The obtained results are shown in Figure 4. In all the tested cases, the proposed algorithm outperform the regular MCMC architecture, with an optimal setting for ϵ to be 0.1. As ϵ is increased, the algorithm needs fewer data to sample in order to make a decision, and thus more samples are generated per unit of time, leading to a reduction in the variance of the estimate. However the bias in the estimates also increases. It should be noted, that as more samples are generated, the risk will be dominated by the bias, leading to a monotonic relationship between ϵ and risk. Nevertheless, for the given execution times the best performance is obtained for $\epsilon = 0.1$.

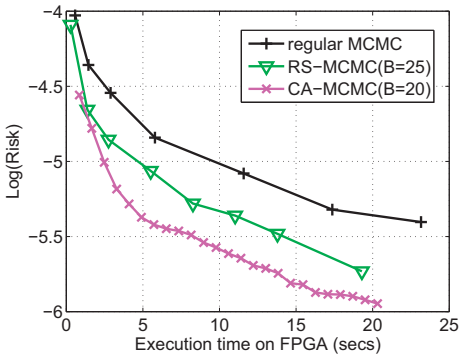

Fig. 5. The Risks in the estimate of the mean value of the first parameter with the optimal design parameters.

Moreover, we investigated the impact of the mini-batch size B to the performance of the algorithm having fixed ϵ to 0.1. The speedups of the two architectures normalised over the performance of the regular MCMC [3] for different values of B are shown in Table III. The obtained results show that the optimal speedups for RS-MCMC and CA-MCMC are 2.10x at $B = 25$ and 3.37x at $B = 20$ respectively. Figure 5 captures how the Risk in the estimate reduces as a function of wall-clock time for these two optimally tuned designs.

TABLE III
SPEEDUPS OF RS- AND CA- MCMC FOR DIFFERENT VALUES OF B. NORMALISED OVER REGULAR MCMC.

B	1	5	10	20	25	50
RS-MCMC	1.19x	1.48x	1.72x	1.84x	2.10x	1.52x
CA-MCMC	1.58x	2.39x	3.21x	3.37x	3.35x	2.70x

Finally, the performance model introduced in Section IV-B is utilised to make predictions of the speedups of CA-MCMC architecture for different FPGA devices that utilise larger on-chip memory storage space. The predicted speed-ups are shown in Figure 6, assuming the default design $B = 1$. As more on-chip BRAMs are available, the data reuse percentage α in (10) is increased and thus the performance of the proposed architecture improves. However, as M increases, the execution time of the regular and RS- MCMC has little improvement as we have shown in (8) and (9), and effectively diminished for large datasets.

VI. CONCLUSIONS

This paper presents a novel communication-aware and subsampling based MCMC framework, CA-MCMC, to push further the memory bandwidth bottleneck in current MCMC applications for big data. The key contribution of this work is that by introducing the PPS sampling to sample the dataset, the proposed design can largely reduce the off-chip memory access times across iterations, and therefore a lower risk in the estimate can be achieved for a given time budget. Experimental results show that notable speedups and reduced risks over other highly optimized FPGA designs can be achieved with

[3]Please note that the performance of the regular MCMC architecture is not affected by B, as it requires all the data to be streamed to the FPGA.

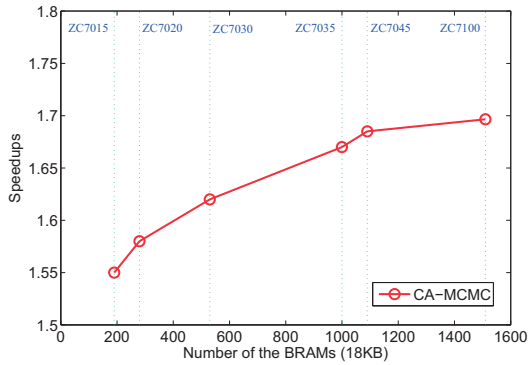

Fig. 6. Scaling of Speedups of CA-MCMC compared to the regular one when varying the device (number of BRAM blocks), at design point $B = 1$, $\epsilon = 0.1$.

our proposed architecture. Even though the presented results are problem specific, the methodology can be applied to other problems with potential gains as it couples the "contribution" of the data with its storage location. Future work will focus on the automated tuning of the algorithm's parameters for achieving maximum performance under any input problem.

REFERENCES

[1] R. D. Payne and B. K. Mallick, "Bayesian Big Data Classification: A Review with Complements," *arXiv preprint arXiv:1411.5653*, 2015.
[2] J. S. Liu, *Monte Carlo strategies in scientific computing*. Springer, 2001.
[3] C. Geyer, "Introduction to Markov Chain Monte Carlo," *Handbook of Markov Chain Monte Carlo*, pp. 3–48, 2011.
[4] 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.
[5] E. Angelino, M. J. Johnson, R. P. Adams *et al.*, "Patterns of scalable bayesian inference," *Foundations and Trends® in Machine Learning*, vol. 9, no. 2-3, pp. 119–247, 2016.
[6] 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.
[7] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel resampling for particle filters on FPGAs," in *Proc. FPT*, Dec 2014, pp. 191–198.
[8] A. Korattikara, Y. Chen, and M. Welling, "Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget," in *Proc. ICML*, 2014, pp. 181–189.
[9] R. Bardenet, A. Doucet, and C. Holmes, "Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach," in *Proc. ICML*, 2014, pp. 405–413.
[10] M. Quiroz, M. Villani, and R. Kohn, "Speeding up MCMC by efficient data subsampling," *arXiv preprint arXiv:1404.4178*, 2014.
[11] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.
[12] R. Bardenet, A. Doucet, and C. Holmes, "On Markov chain Monte Carlo methods for tall data," *arXiv preprint arXiv:1505.02827*, 2015.
[13] S. Liu, G. Mingas, and C.-S. Bouganis, "An exact MCMC accelerator under custom precision regimes," in *Proc. FPT*, Dec 2015, pp. 120–127.
[14] ——, "An Unbiased MCMC FPGA-based Accelerator in the Land of Custom Precision Arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 5, November 2016.
[15] A. F. da Silva, "cudaBayesreg: Bayesian computation in CUDA," *The R Journal*, vol. 2, no. 2, pp. 48–55, 2010.
[16] S. L. Scott, A. W. Blocker, F. V. Bonassi, H. Chipman, E. George, and R. McCulloch, "Bayes and big data: The consensus Monte Carlo algorithm," in *Proceedings of the Bayes 250 conference*, vol. 16, 2013.
[17] M. Quiroz, M. Villani, and R. Kohn, "Exact Subsampling MCMC," *arXiv preprint arXiv:1603.08232*, 2016.
[18] G. Mingas, F. Rahman, and C. S. Bouganis, "On optimizing the arithmetic precision of mcmc algorithms," in *Proc. FCCM*, April 2013, pp. 181–188.
[19] Y. Tillé, *Sampling algorithms*. Springer, 2011.