A HIGH THROUGHPUT FPGA IMPLEMENTATION OF MARKOV CHAIN MONTE CARLO METHOD FOR MIXTURE MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CANER BOZGAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2019

Approval of the thesis:

A HIGH THROUGHPUT FPGA IMPLEMENTATION OF MARKOV CHAIN MONTE CARLO METHOD FOR MIXTURE MODELS

submitted by **CANER BOZGAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Tolga Çiloğlu
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. İlkay Ulusoy
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ali Ziya Alkar
Electrical and Electronics Engineering, Hacettepe University _____

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaççı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Emre Özkan
Electrical and Electronics Engineering, METU _____

Date: 29/01/2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Caner Bozgan

Signature :

ABSTRACT

A HIGH THROUGHPUT FPGA IMPLEMENTATION OF MARKOV CHAIN MONTE CARLO METHOD FOR MIXTURE MODELS

Bozgan, Caner
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. İlkay Ulusoy

January 2019, 92 pages

Markov Chain Monte Carlo (MCMC) is a class of algorithms which can generate samples from high dimensional and multimodal probability distributions. In many statistical and control applications, MCMC algorithms are employed widely thanks to their ability to draw sample from arbitrary distribution regardless of dimension or complexity. However, as the complexity of the Bayesian models and the computational load of the MCMC algorithm increase, performing MCMC inference becomes impractical or too time consuming for the real applications with large scale data sets. Motivated by this problem, this thesis proposes a low latency, scalable and high throughput hardware architecture for Parallel Tempering method, which is a MCMC algorithm to sample from multimodal distributions. The work demonstrates that the implementation of the Parallel Tempering method on Field Programmable Gate Array (FPGA) provides significant speedups compared to respective CPU and GPU implementations when performing Bayesian inference for a mixture model. The proposed work also adapts the architecture to the big data MCMC problems by eliminating the external memory related performance losses that arise in the MCMC hardware

implementations.

Keywords: Markov Chain Monte Carlo (MCMC), Sampling, Mixture Model, Field Programmable Gate Array (FPGA)

ÖZ

KARIŞIM MODELLERİ İÇİN MARKOV ZİNCİRLİ MONTE CARLO YÖNTEMİNİN YÜKSEK İŞLEM HACİMLİ FPGA UYGULAMASI

Bozgan, Caner
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İlkay Ulusoy

Ocak 2019 , 92 sayfa

Markov Zincirli Monte Carlo (MZMC) çok boyutlu ve çok tepeli dağılımlardan örnekleme yapabilen bir algoritma sınıfıdır. Dağılımın karmaşıklığına ve boyutuna aldırmaksızın örnek çekebilme yetenekleri sayesinde birçok istatiksel uygulamada yaygın şekilde kullanılırlar. Ancak, uygulanan MZMC metodunun numerik yükü ve hedef alınan modelin karmaşıklığı arttıkça, büyük veri içeren problemler için MZMC uygulamaları giderek zorlaşmakta ve algoritmanın çalışma süreleri oldukça uzamaktadır. Bu problemden yola çıkarak, çok tepeli problemlerden örnekleme yapabilen Parallel Tempering metodu için düşük gecikmeli, ölçeklenebilir ve yüksek veri hacimli bir donanım mimarisi önerilmektedir. Yapılan çalışma, karışım modelleri için Bayesci çıkarımda, FPGA tabanlı Parallel Tempering mimarisinin çok çekirdekli işlemci (CPU) ve grafik işlemcilere (GPU) kıyasla ciddi hız artışları sağladığını göstermiştir. Ayrıca bu çalışma MZMC donanım gerçeklemelerinde harici hafızaya erişim sırasında gözlenen performans kayıplarının önüne geçerek, mimariyi büyük veri içeren problemlere uyugun hale getirmiştir.

Anahtar Kelimeler: Markov Zincirli Monte Carlo (MZMC), Örnekleme, Karışım Modelleri, Alanda Programlanabilir Kapı Dizinleri

To my family

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Prof. Dr. İlkay Ulusoy, for her guidance and continuous support of my research. The door to her office was always open whenever I ran into a trouble spot. The most thing I admire and appreciate is that she gave me a great freedom in my research.

I would like to thank ASELSAN A.Ş. for providing necessary hardware and software environments to develop my studies. Also, all seniors from workplace deserve sincere thanks for their contributions on the improvement of my engineering skills.

I also have to thank my parents Şengül Bozgan and Yaşar Bozgan and my brother Alper Bozgan. They dedicate their life to my education and without them I would not be what I am.

Lastly, I wish to thank my dear wife İlknur Bozgan who gave me a great support from the beginning.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

CLK	Clock
CPU	Central Processing Unit
DSP	Digital Signal Processor
ESS	Effective Sample Size
$FIFO$	First In First Out
$FPGA$	Field Programmable Gate Array
GPU	Graphics Processing Unit
$GRNG$	Gaussian Random Number Generator
IO	Input Output
IP	Intellectual Property
$I2C$	Inter-Integrated Circuit
$i.i.d.$	Independent and Identically Distributed
LUT	Look Up Table
MC	Monte Carlo
$MCMC$	Markov Chain Monte Carlo
PT	Parallel Tempering
RAM	Random Access Memory
RNG	Random Number Generator
SIS	Sequential Importance Sampling
$SISR$	Sequential Importance Sampling Resampling
SMC	Sequential Monte Carlo
$URNG$	Uniform Random Number Generator

CHAPTER 1

INTRODUCTION

In Bayesian statistics, sampling from an arbitrary probability distribution is an important task. Mainly, generated samples are used to estimate the complex integrals, apply stochastic optimization and draw conclusions about model parameters and the system. Including Importance Sampling and Acceptance Rejection method, many well known methods have been proposed to generate independent random samples from a given probability distribution. Although they can be applied for any probability distribution, most of the methods that generate independent samples are inefficient when they face with complex and high dimensional distributions.

On the other hand, Markov Chain Monte Carlo (MCMC) is a generic method that generates dependent samples from any arbitrary probability distribution regardless of its multi-modality and dimension. Main idea of MCMC methods is forming Markov Chain whose stationary distribution matches the target distribution. The success of MCMC methods over multi modal and high dimensional distributions has been proved in the past years in many different scientific disciplines such as statistics, finance, engineering, biology, chemistry, physics and control [43]. The mixture models, which are the models used in this thesis, show also multi-modality that make it a target for MCMC methods.

Despite their all advantages, MCMC methods can be too time consuming when they face with real-world Bayesian problems. Advanced MCMC algorithms with intensive computational demands and the use of large scale data sets for the extremely complex likelihood computations make the MCMC method slow for any practical use. For example, models used in genetics or cosmology require the usage of massive data sets [71,76] so the excessive run times can be observed. Besides, the need of external

memory in the MCMC computing systems arises with the usage of massive data sets, which may limit the system's performance. Since the data transfer rate of the external memory may not meet the demands of MCMC processing units.

Consequently, much efforts have been devoted in recent years to accelerate MCMC algorithms and to adapt them to Bayesian inference with massive data sets. One of the research directions is reducing the number of individual data processed at each MCMC iteration by proposing sampling scheme [8, 9, 45, 57]. The second research direction focuses on the data partitioning, which is based on splitting whole data into sub-groups then running each groups separately (generally parallel) and finally combining the results of each groups [32, 35, 55, 74]. Another research direction is accelerating the likelihood evaluations by parallel computations with the help of multi-core Central Processing Units (CPUs) and Graphics Processing Units (GPUs) [7,37,40]. Since the parallel computations in parallel hardware architectures achieved satisfactory results for the acceleration of MCMC algorithms, Field-Programmable Gate Arrays (FPGAs) have shown to be a promising hardware to meet increasing speed demands of MCMC applications in the literature [45, 50–52].

Why the use of FPGAs is considered as good solution for MCMC acceleration relies on the following facts: FPGAs are re-configurable devices that provide the chance of implementing customizable and massively parallel architectures. With their deep pipelining capabilities, FPGAs are also suitable for sequential architectures like most of Markov Chain Monte Carlo methods. Another important advantage of FPGAs is their flexibility to operate in any custom arithmetic precision format [45]. By reducing arithmetic precision in the wisely selected parts of the architecture, fewer resource usage and thus more parallelism and higher throughput are possible [50,52]. Besides, the power consumption is the major concern in the mobile systems like robots. When compared to other parallel computing platforms (GPUs, CPUs), the power consumptions of FPGAs are very low and this makes them a good candidate for the sampling applications in mobile systems.

Motivated by the facts given above, this thesis aims to develop and implement a FPGA architecture for the MCMC algorithms and to bring these algorithms closer to real time applications. To this end, the MCMC architecture proposed by G. Min-

gas [50–52] are taken as reference work and implemented in a completely different FPGA brand. With this architecture which forms a deep pipeline system in FPGA by employing parallel processing blocks, the significant performance improvements are achieved when compared to state-of-the-art CPU and GPU implementations of the same problem. The obtained results validates the performance improvements achieved by FPGA accelerators proposed in the closely related works [50–52] for the MCMC problems. In this thesis, performance metrics like sampling throughput, the latency, the resource utilization and the power consumptions are provided and compared with closely related works [37, 50–52]. The scalability of the architecture with varying FPGA resource and data size is also investigated.

In contrast to closely related works, this thesis also investigates the effects of external memory usage on the system's performance. Since, memory issue becomes bottleneck in MCMC applications with big data and thus impacts the system's performance significantly. While closely related works [50–52] do not consider memory related problems and focus only likelihood acceleration, this work takes into account the performance of external memory and proposes an architecture to prevent performance losses caused by external memory access latencies. In this architecture, whole data is partitioned into two sub-groups such that the processing unit is fed with these subgroups respectively without a break. The results obtained in this work show that it is possible to achieve speedups of 24x and 4x against the respective state-of-the-art implementations in CPUs and GPUs even if an external memory is added to system for big data problems.

The rest of the thesis is organized as follows:

In Chapter 2, a brief background information about the theory of Bayesian inference is given. Strategies for sampling a random variable from given probability distributions are described. Including the Parallel Tempering method, basic principles of most common Monte Carlo and Markov Chain Monte Carlo methods are explained. Moreover, the reasons of long run times in MCMC applications and possible solutions are discussed. In a separate section, the literature overview on MCMC acceleration and especially closely related works on GPUs and FPGAs which accelerated the MCMC algorithms are presented.

By exploiting inherent features of FPGAs, the way of acceleration of MCMC algorithms are sought in Chapter 3. For this purpose, a FPGA architecture for parallel tempering method, which is a population based MCMC method for drawing samples from multi-modal probability distributions, is designed. All functional elements of the architecture and the details of their implementation in FPGA are described. Moreover, a mixed precision MCMC architecture is provided. How this architecture saves FPGA resources and thus gives chance to add more parallel blocks to obtain more performance achievements are explained.

Chapter 4 starts with the details of the case study (Gaussian mixture model) implemented to evaluate FPGA architectures. Then, evaluation platforms and simulation tools for verification of the algorithms are introduced. Moreover, achieved throughput, latency, power consumption results obtained from real hardware and FPGA resource utilization of the FPGA architecture are presented. All these performance results are compared with other studies [37, 50–52] and the way the architecture scales with data size and chain number are also investigated. At the end of this chapter, important studies that do not exist in closely related works are presented: the performance analysis of the external memory and a FPGA design with the external data memory for MCMC acceleration. This proposed design does not allow stalling in the pipelines of the processing unit and thus saves the achieved throughput.

Finally, Chapter 5 gives the summary of this thesis and the possible future works for MCMC acceleration.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Background

The main idea of the Bayesian modeling is to represent uncertainty or unobserved parameter in the model. Many real life problems such as genetics [59, 77, 78], physics [43, 72], chemistry [19, 43], machine learning [7, 27], economics [22], nonlinear dynamic systems (e.g. target tracking) [43] use modeling to make predictions about unknown quantities, anticipate outcomes of the system, compare alternative models, and learn model parameters from given observations [27]. When prior knowledge about unknown parameters with observed data sets is available, inferring the posterior probability of the unknown variables given the observed data is possible. The well known rule of the probability theory, Bayes rule, expresses the probability of uncertainty based on the prior knowledge of variable and the observed data, which forms the basis of Bayesian inference.

$$p(\theta|\mathbf{y}) = \frac{p(\mathbf{y}|\theta)p(\theta)}{p(\mathbf{y})} \tag{2.1}$$

Here, $\mathbf{y}$ and θ correspond to observation values and unobserved parameter in the model respectively. In the above formula, there exist some important terms that form the basis of the Bayesian Inference.

- $p(\theta)$ is the **prior probability** of the parameter θ that represents the information about the model before some evidence is taken into account. It is an unconditional probability distribution that may be determined from past observations or experiments.

- $p(\mathbf{y}|\theta)$ is the **likelihood function** that describes the possibility of observation based on the known model. In other words, when the model is known, it simply gives the likely observation .

- $p(\theta|\mathbf{y})$ is the **posterior distribution** of the unknown parameter, which is formed after the observations **y** and the prior are combined.

- The term in the denominator of the formula 2.1 is **marginal likelihood function** and it can be very hard to compute.

$$p(\mathbf{y}) = \int p(\mathbf{y}, \theta) d\theta = \int p(\mathbf{y}|\theta) p(\theta) d\theta \tag{2.2}$$

Luckily, in MCMC methods, normalizing constant $p(\mathbf{y})$ need not to be computed as it will be explained in later sections. However, the computation of integral is the heart of the many scientific problems. Instead of evaluating the integral analytically, which is sometimes impossible, an approximation to given integral can be obtained by drawing samples from the given function of interest [43]. As described later, MCMC is a generic method to draw sample from any given probability distribution. Simply, the key task that need to be performed in Bayesian inference is evaluating the following integral by generating samples from the posterior probability $p(\theta|\mathbf{y})$.

$$\mathbb{E}_{p(\theta|\mathbf{y})}[f(\theta)] = \int f(\theta) p(\theta|\mathbf{y}) d\theta \tag{2.3}$$

Here, $f(\theta)$ is any desired function and the integral 2.3 is the expectation of that desired function under the posterior probability distribution. If $f(\theta)$ is set to θ then it will become the estimation of the mean of the parameter θ. If $f(\theta)$ is set to $p(\mathbf{y'}|\theta)$, it will become the prediction of future observation [45]. In the following sections, the most popular ways of estimating integrals via generated samples will be described.

2.2 Monte Carlo Approximation

As it is indicated in the previous section, computing integrals is a challenging but unavoidable task in many scientific problems.

$$I = \int_R g(x)dx \tag{2.4}$$

In the integral 2.4, g(x) is the complex target function of interest and R is usually high dimensional space. When independent identically distributed (i.i.d.) random samples $x^{(1)}, x^{(2)}, ..., x^{(m)}$ are generated from the space R, the approximation to I can be obtained as [43]

$$I_m = \frac{1}{m} g(x^{(1)}) + ... + g(x^{(m)}) \tag{2.5}$$

According to the *law of large numbers*, the average of many independent random variables approaches to their common mean; that is,

$$lim_{m \to \infty} I_m = I \text{ with probability 1.} \tag{2.6}$$

Also, by the *central limit theorem* (CLT), its convergence rate is defined as normal Gaussian distribution with zero mean.

$$\sqrt{m}(I_m - I) \xrightarrow{d} N(0, \sigma^2), \text{ where finite } \sigma^2 = var(g(x)) \tag{2.7}$$

The above properties show that the sample size determines the variance of the estimator. In other words, as the sample size (m) goes to infinity, the estimate converges to the true value so the accuracy of the approximation increases. Many simple algorithms exist to estimate for a few dimensional integrals by using the above idea and Rejection Sampling and Importance Sampling are the most famous ones.

2.2.1 Rejection Sampling

Rejection sampling is a method for drawing samples from a distribution $p(x)$ by using another distribution $q(x)$ from which we can easily sample. In this method, there exists a constant c such that $cq(x) > p(x)$ for all $x \in X$ [54].

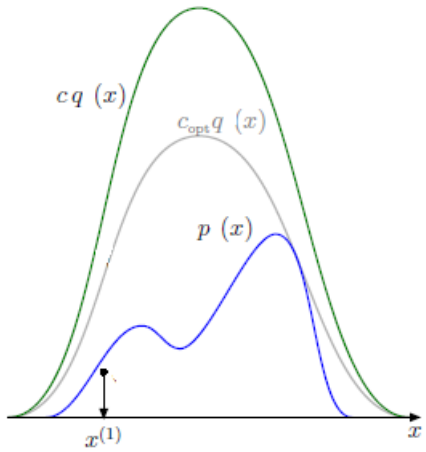

Figure 2.1: Rejection Sampling [54]

Samples are drawn uniformly underneath a simpler curve $cq(x)$ and points between $p(x)$ and $cq(x)$ are rejected. The remaining points come uniformly from underneath $p(x)$ so they are valid samples. The constant c is chosen such that $cq(x)$ is always above $p(x)$. A drawback of the Rejection sampling is that it is usually very hard to apply the algorithm on high dimensional problems. There are some techniques [30, 43, 54, 58] to find optimal c_{opt} and adjust $q(x)$ in order to reduce the number of rejected samples but it is not always possible. Actually, the rejection rate is decreased by updating $q(x)$ as points from $p(x)$ are evaluated.

2.2.2 Importance Sampling

Throwing samples away along with all the associated computations seems wasteful [54]. Importance sampling avoids such rejections. Our aim is computing the

Algorithm 1 Rejection Sampling

Inputs: target distribution p(x) and simple distribution q(x),

Outputs: x samples from p(x)

$i = 0$

while $i \neq N$ **do**

  $x^{(i)} \sim q(x)$

  $u \sim Uniform(0, 1)$

  if $u < \dfrac{p(x^{(i)})}{cq(x^{(i)})}$ **then**

    accept $x^{(i)}$

  else

    reject $x^{(i)}$

  end if

  $i \Leftarrow i + 1$

end while

following integral.

$$u = \int f(x)p(x)dx \tag{2.8}$$

We can sample from a known and easy distribution $q(x)$ such that $x^{(i)} \sim q(x)$ and q that has the same support as p. Importance weight will be:

$$w(x^{(i)}) = \frac{p(x^{(i)})}{q(x^{(i)})} \tag{2.9}$$

Now integral 2.8 can be approximated by using importance weight $w(x^{(i)})$ and samples from $q(x)$.

$$\begin{aligned} u &= \int f(x)p(x)dx = \int f(x)\frac{p(x)q(x)}{q(x)}dx, \\ &= \int f(x)w(x)q(x)dx, \\ &\approx \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)})w(x^{(i)}), \text{ where } x^{(i)} \sim q(x) \end{aligned} \tag{2.10}$$

In order to reduce the variance of our estimator, $q(x)$ is chosen as close in shape to $f(x)p(x)$ as possible. Analytically, the variance of the estimator is set to minimum if $q(x) = \frac{f(x)p(x)}{\int f(x)p(x)dx}$ by the below formula.

$$var_q(f(x)p(x)) = E_{x\sim q}[f^2(x)w^2(x)] - E_{x\sim q}[f(x)w(x)]^2 \geq 0 \tag{2.11}$$

Of course setting such q is NP hard in general, but at least it shows a direction for good approximation.

2.3 Markov Chain Monte Carlo (MCMC)

Both Rejection Sampling and Importance Sampling require tractable distribution $q(x)$. Indeed, Rejection sampling will rarely return samples and Importance Sampling will have large variance for non-standard forms of probability distribution $p(x)$. On the other hand, MCMC methods can be used to sample from complex and high dimensional distributions that have unknown normalization [54].

In MCMC methods, the requirement of independent samples is relaxed and dependent samples associated with target density can be used. MCMC generates samples from the posterior by constructing a Markov chain. A Markov chain generates a correlated sequence of states. Each step in the sequence is generated using a transition kernel T($x^{i+1} \leftarrow x^i$), which gives the probability of moving from state x^i to state x^{i+1}. Markov chains have property that the transition probability depends on only the current state, x^i. Running the chain for some number of times, the stationary distribution of the chain will equal to the target distribution regardless of the initial sample under some regulatory conditions. These condition are *irreducibility*, the ability to reach any x where $p(x) > 0$ in a finite number of steps and *aperiodicity*, which is about returning any state at any time [54]. These two conditions are sufficient for detailed balance. For more detail, see [70] or [28].

When compared to Monte Carlo methods, MCMC has the ability to draw samples from any complex distribution. Intractable integrals in real life problems can be evaluated by advanced MCMC methods. Also, a big advantage of MCMC is that when

evaluating the posterior 2.1, there is no need to compute the normalizing constant as it will be shown later. This reduces the computational workload in Bayesian inference problems. As a result, MCMC methods have become the most mainstream tools to perform Bayesian inference during the last two decades [49]. Rapid and continuous advancements in MCMC methodology have also created new application areas for Bayesian framework. In the next section, some of the well known MCMC methods will be described.

2.3.1 MCMC Algorithms

Starting from Metropolis method [56], which was introduced in 1953, various MCMC methods have been proposed to get better results in evaluating distributions. [61] and [54] give a brief history of the advancements in MCMC methodology. In the following sections, important MCMC algorithms used in this thesis will be described.

2.3.1.1 Metropolis Algorithm

Metropolis algorithm [56] is the cornerstone of the Markov Chain Monte Carlo methods. The algorithm is simply as follows:

- By using symmetric proposal distribution q, generate a sample θ^*.
- Evaluate target probability density $p(\theta)$ with the new proposed sample and the previous sample. Then, calculate the acceptance ratio α. In Bayesian inference problems, target density $p(\theta)$ equals $p(\theta|\mathbf{D})$. Since α is a ratio of two posteriors, there is no need to calculate the normalizing constant in posterior 2.1. This is one of the biggest advantage of using MCMC in Bayesian inference problems.
- Compare the ratio α with uniform random number μ in the range [0, 1]. If α is greater than μ, accept the proposed sample θ^*. Otherwise, reject it and use the previous sample in next iteration.

Algorithm 2 Metropolis Algorithm

Inputs: initial setting θ_0, number of samples N,

for i = 1 to N **do**

propose $\theta^* \sim q(\theta^*|\theta_{i-1})$

evaluate $\alpha = \dfrac{p(\theta^*)}{p(\theta_{i-1})}$

generate uniform random number $\mu \sim Uniform[0, 1)$

if $\alpha > \mu$ **then**

accept θ^*

else

reject θ^* and $\theta_i \Leftarrow \theta_{i-1}$

end if

end for

2.3.1.2 Metropolis-Hasting Algorithm

In 1970, W. K. Hastings relaxed the requirement of the symmetric proposal distribution by modifying acceptance ratio [31]. M-H algorithm is valid for any proposal distribution but generally simple distributions such as Gaussian are chosen. If a symmetric proposal distribution is used $q(\theta_{i-1}|\theta^*)$ will be equal to $q(\theta^*|\theta_{i-1})$ and the acceptance ratio will take the form in Metropolis algorithm.

Algorithm 3 Metropolis - Hasting Algorithm

Inputs: initial setting θ_0, number of samples N,

for i = 1 to N **do**

propose $\theta^* \sim q(\theta^*|\theta_{i-1})$

evaluate $\alpha = \dfrac{p(\theta^*)q(\theta_{i-1}|\theta^*)}{p(\theta_{i-1})q(\theta^*|\theta_{i-1})}$

generate uniform random number $\mu \sim Uniform[0, 1)$

if $\alpha > \mu$ **then**

accept θ^*

else

reject θ^* and $\theta_i \Leftarrow \theta_{i-1}$

end if

end for

2.3.1.3 Gibbs Sampling

Gibbs sampling [24] has been proposed by S. Geman and D. Geman in 1984. It is a special case of Metropolis Hasting method where the proposal distribution is based on the conditional distributions and the acceptance probability is always one [6]. Basically, by decomposing the random variable into d components $x = (x_1, x_2, ..., x_d)$, randomly or systematically, the selected coordinate i is updated with a new sample $x_i^{'}$ drawn from the conditional distribution $p(x_i|x_{-i}^t)$ while leaving the other coordinates as unchanged. The iteration is made one at a time [43].

- Sample $x_i^{'} \sim p(x_i|x_{-i}^t)$
- Set $x^{t+1} = (x_1^t, ..., x_i^{'}, ..., x_d^t)$

where x_{-i}^t refers to all coordinates in x^t except x_i. In Bayesian network models, Gibbs sampling is widely used since, in these models, the posterior is naturally expressed in terms of the full conditional distributions of each component [49].

2.3.1.4 Parallel Tempering

Metropolis and Metropolis-Hasting (M-H) are the basic MCMC methods that are applicable to any problem. However when multi-modal distributions are targeted they are inefficient to search posterior, resulting slow convergence and slow mixing [33, 49]. Since multi-modal distributions have more than one modes in their support and difficult distributions to sample for classical MCMC methods. However, population based MCMC methods like parallel tempering offer a solution to this problem by constructing multiple chains that sample from slightly different probability distributions around the original target distribution.

The idea of Parallel Tempering, or Replica Exchange, was first introduced by Swendsen and Wang [72], then extended and formulated by Geyer [26] in 1991 [19]. Basically, Parallel Tempering is simulating the replicas of the given distribution that are at different temperatures. Usually, replicas at high temperatures quickly search the state space, whereas replicas at low temperature search the local region and

Algorithm 4 Parallel Tempering [52]

Inputs: M(Number of chains), N(run time), $\theta^0_{1:M}$(initial samples of all chains), p (target density), q(proposal density)

Outputs: $\theta_1^{0:N}$ (samples of chain 1)

for i = 1 to N **do**

  for j = 1 to M **do**

    propose $\theta^* \sim q(\theta^*|\theta_j^{i-1})$

    evaluate acceptance ratio $\alpha_1 = \dfrac{p_j(\theta^*)}{p_j(\theta_j^{i-1})}$

    generate uniform random number $\mu \sim Uniform[0, 1)$

    if $\alpha_1 > \mu$ **then**

      accept θ^* and $\theta_j^i \Leftarrow \theta^*$

    else

      reject θ^* and $\theta_j^i \Leftarrow \theta_j^{i-1}$

    end if

  end for

  select exchange pairs ((1,2),(3,4)...) or ((2,3), (4,5)...) in turn

  for selected chains r and q **do**

    evaluate exchange ratio $\alpha_2 = \dfrac{p_q(\theta_r^i)p_r(\theta_q^i)}{p_q(\theta_q^i)p_r(\theta_r^i)}$

    generate uniform random number $\mu \sim Uniform[0, 1)$

    if $\alpha_2 > \mu$ **then**

      exchange the samples θ_q^i and θ_r^i

    else

      Do not exchange the samples

    end if

  end for

  Return the sample of chain 1, θ_1^i

end for

samples from distributions closer to the original target. By exchanging samples between replicas, a good mixing is accomplished. This is why parallel tempering method is widely chosen for multi-modal systems. Psudocode of the parallel tempering algorithm is given in Algorithm 4 where $p_j(\theta) = p^{1/Temp_j}(\theta)$ and $1 = Temp_1 < Temp_2 < .... < Temp_M$.

At each iteration of the PT algorithm two stages are performed. Firstly, the current samples of all chains are updated by using simple Metropolis Kernel. Secondly, the adjacent chains interact with each other and exchange their samples according to exchange kernel. The exchange pairs are selected in a rotating order such that pairs are $((1,2),(3,4),(M-1,M))$ or $((2,3),(4,5),(M-2,M-1))$. Only the samples of first chain are stored and the samples of other chains are discarded. Some initial samples (burn-in samples) are also discarded as will be described in the following sections.

Parallel Tempering vs Metropolis Algorithm

In this thesis, Parallel Tempering algorithm is implemented on FPGA platform to sample from a Gaussian mixture model. This Gaussian mixture model has 4 components with mean $(\mu_1, \mu_2, \mu_3, \mu_4)$ and supports 24 different modes. The observation data $(D_{1:32})$ is simulated for $\mu_{1:4} = (-3, 0, 3, 6)$. The only unknown parameter in the problem is mean μ of the components. The details of the case study are given in section 4.1. Before this case study is implemented on FPGA, it is implemented in software (Matlab) by using Metropolis sampler and Parallel Tempering samplers with different number of chains to compare the algorithms.

Figure 2.2 and Figure 2.3 show the estimated marginal posterior density $p(\mu_{1:2}|D)$ and the histogram of μ_1 when Metropolis algorithm is applied to the problem. The marginal density $p(\mu_{1:2}|D)$ in Figure 2.2 has 5 out of 12 modes in $\mathbb{R}^2$. It is important to note that the joint density $p(\mu_{1:4}|D)$ traverses 10 out of 24 modes in $\mathbb{R}^4$. Indeed, Figure 2.10 shows the number of generated samples from each modes. These results show that the Metropolis algorithm is inefficient to search whole modes in the posterior distribution since it mainly gets stuck in some of the modes and rarely draws samples from other modes in sample space.

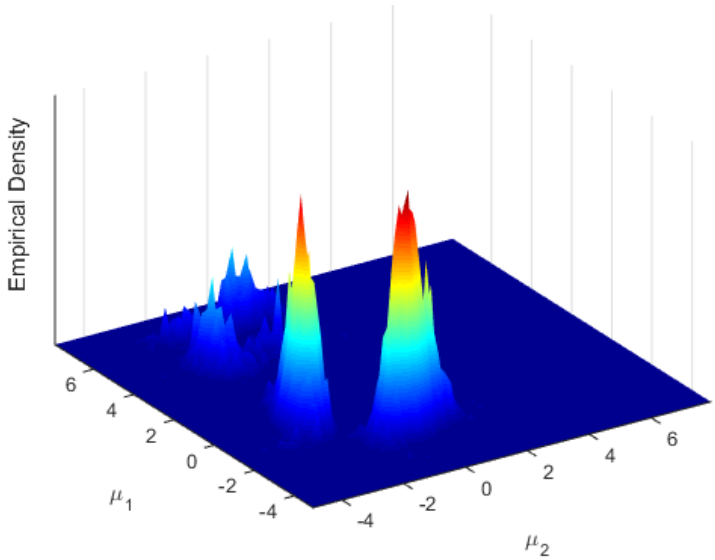

Figure 2.2: Estimated marginal posterior density $p(\mu_{1:2}|D)$ from 60000 samples generated via Metropolis algorithm.

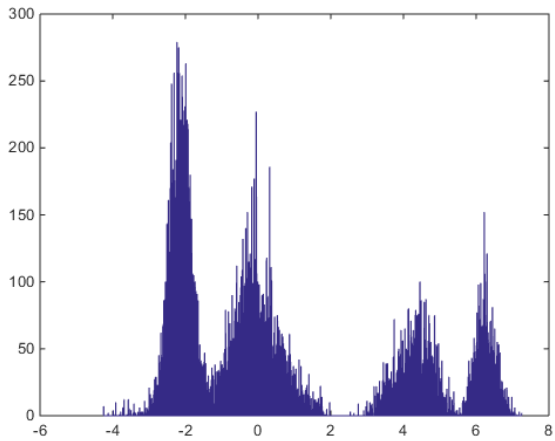

Figure 2.3: The histogram of 60000 samples generated via Metropolis algorithm for μ_1.

On the other hand, Parallel Tempering method can easily discover all existing modes in sample space. Figure 2.4 and Figure 2.5 show the estimated marginal posterior density $p(\mu_{1:2}|D)$ and the histogram of μ_1 when Parallel Tempering with four(4) chains is applied to the problem. The marginal density $p(\mu_{1:2}|D)$ in Figure 2.4 has 12 modes in $\mathbb{R}^2$. On the other side the joint density $p(\mu_{1:4}|D)$ traverses all modes in $\mathbb{R}^4$. As the number of chains increases, the better mixing and better mean approximation are

achieved as shown in Figure 2.6 and Figure 2.8.

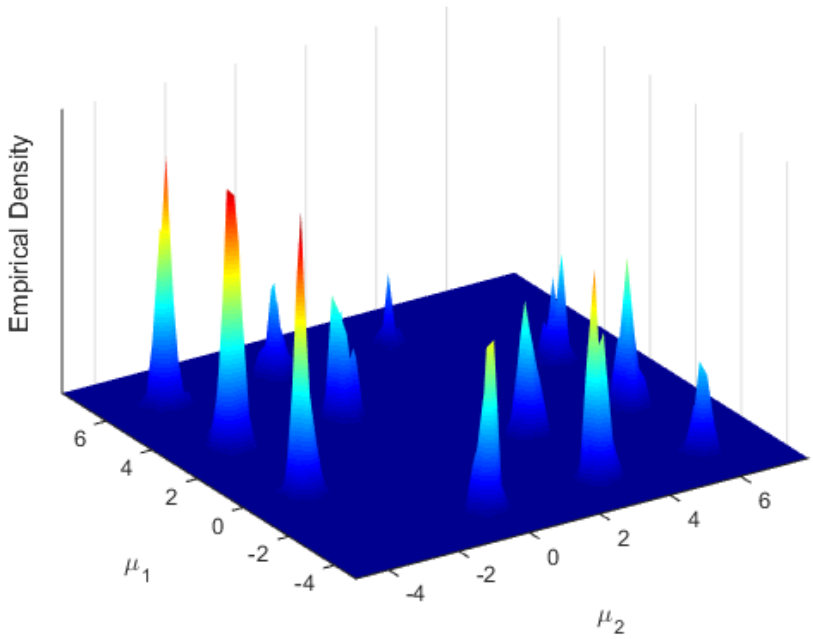

Figure 2.4: Estimated marginal posterior density $p(\mu_{1:2}|D)$ from 60000 samples generated via Parallel Tempering method with 4 chains.

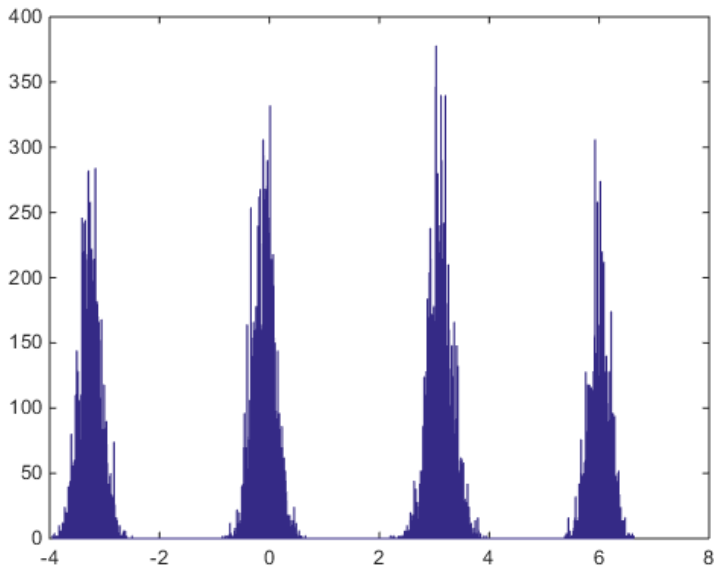

Figure 2.5: The histogram of 60000 samples generated via Parallel Tempering method with 4 chains for μ_1.

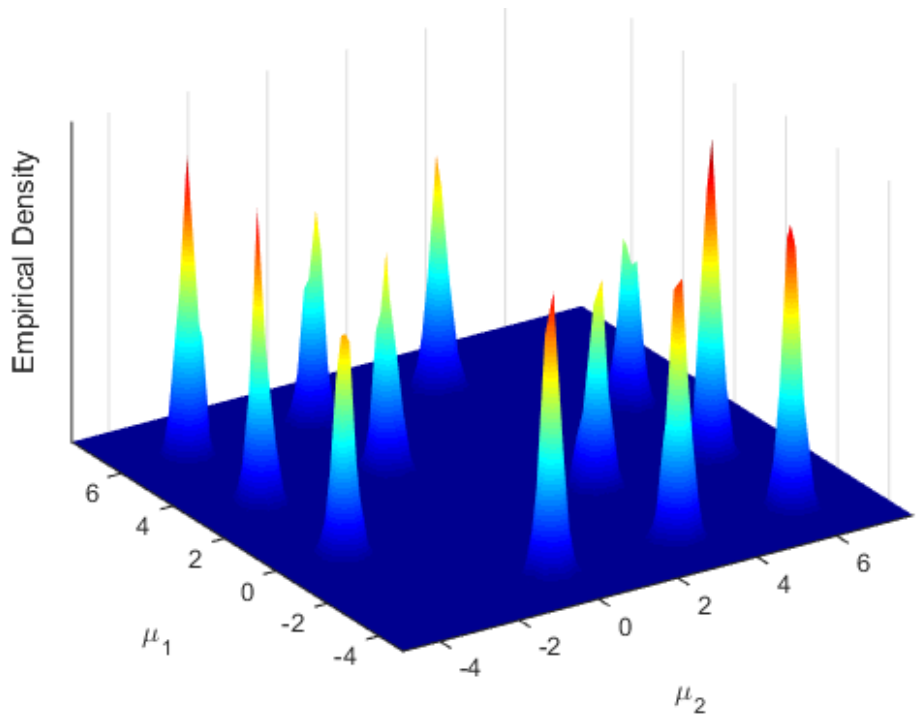

Figure 2.6: Estimated marginal posterior density $p(\mu_{1:2}|D)$ from 60000 samples generated via Parallel Tempering method with 32 chains.

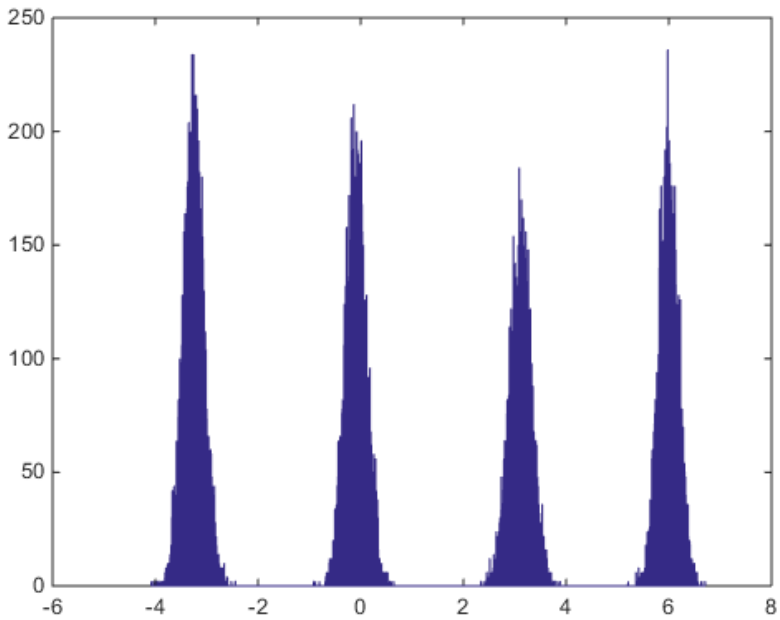

Figure 2.7: The histogram of 60000 samples generated via Parallel Tempering method with 32 chains for μ_1.

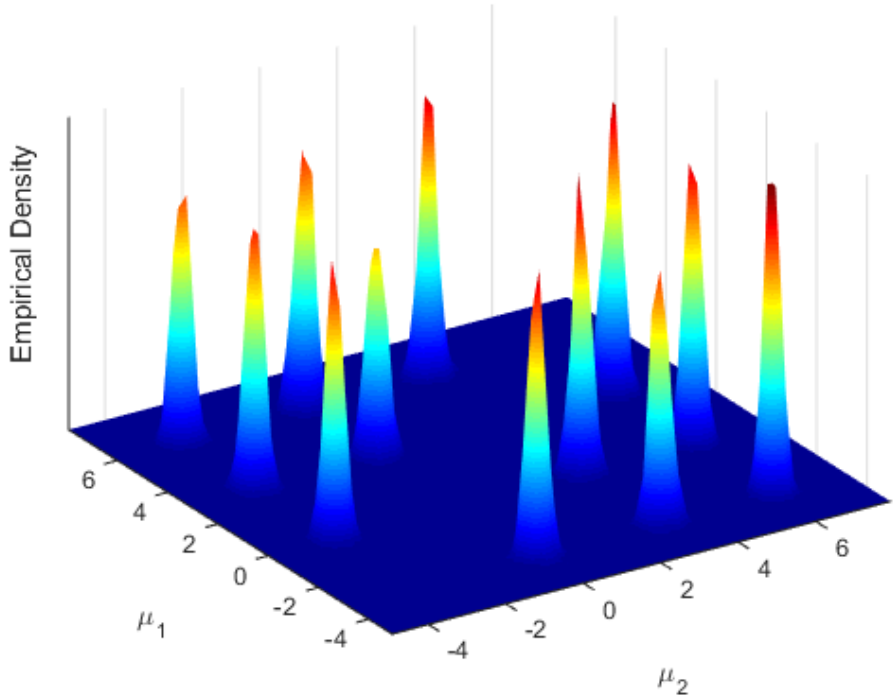

Figure 2.8: Estimated marginal posterior density $p(\mu_{1:2}|D)$ from 60000 samples generated via Parallel Tempering method with 512 chains.

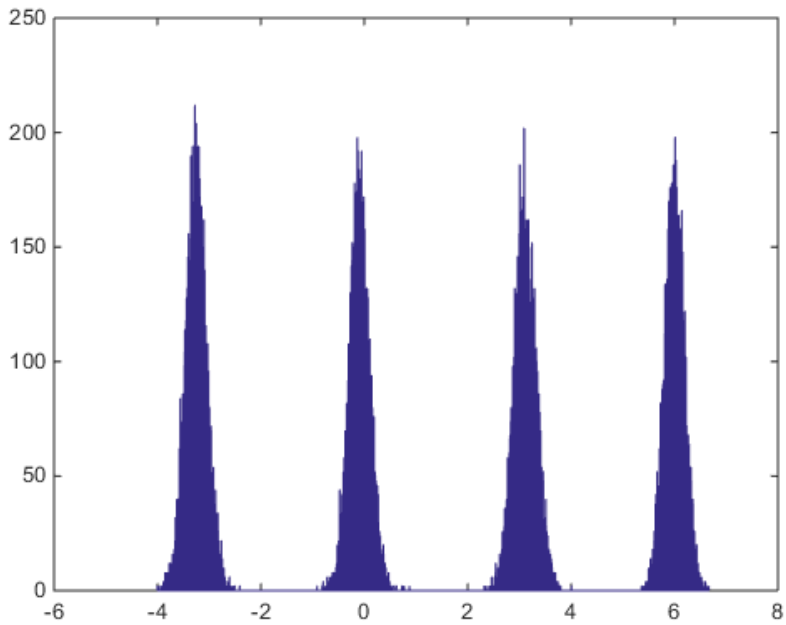

Figure 2.9: The histogram of 60000 samples generated via Parallel Tempering with 512 chains for μ_1.

Figure 2.10 shows the number of samples in each mode generated by different algorithms. While the metropolis algorithm is getting stuck in some of the modes, Parallel Tempering method draws samples from every mode in the sample space. As the chain number(M) increases, better sample distribution is achieved. Figure 2.11 shows the average number of iterations to traverse all modes for the different number of chains (M). These scores reflect the average of 20 independent runs for each case. For the values of M between 4 and 128, the average iterations to traverse all modes are in between 42000 and 5200. However it is worth noting that for the cases where the chain number is larger than 128, the traversal time does not change significantly.

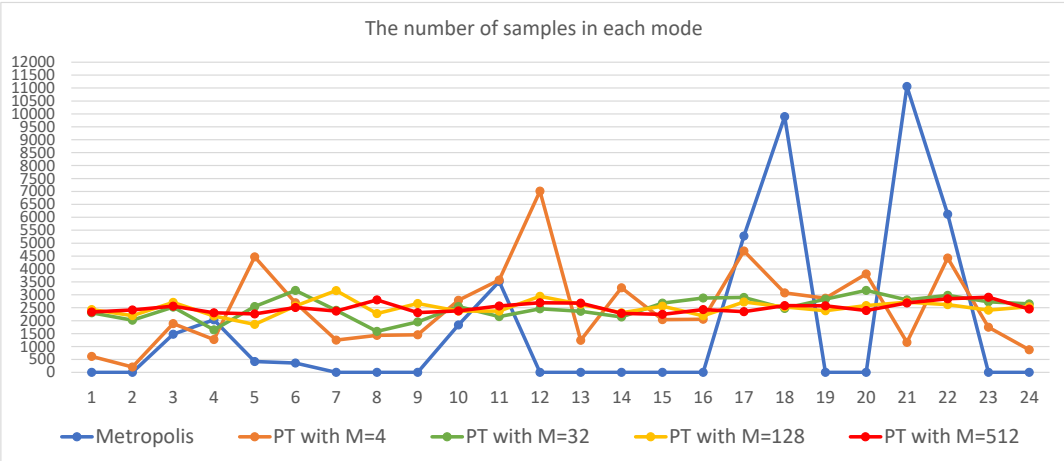

Figure 2.10: The number of samples in each mode for 60000 iteration.

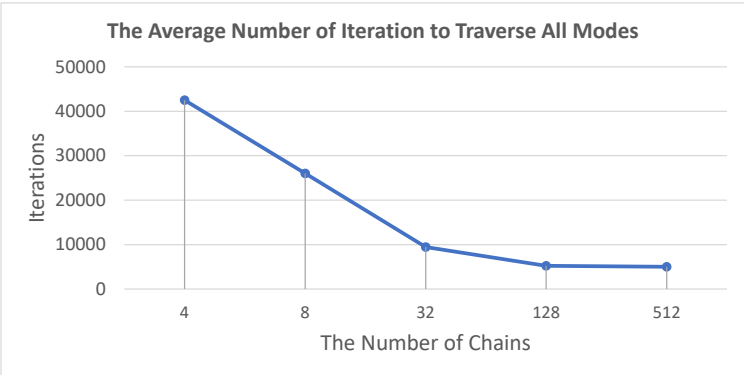

Figure 2.11: The average number of iterations to traverse all modes in sample space.

2.3.1.5 Other MCMC Algorithms

Apart from the methods described in previous sections, many MCMC algorithms have been developed and used up to now. Here, a list of some important algorithms in the literature. For more detail, the reader can look [4, 30, 41, 43] and [61].

- Hamiltonian Monte Carlo
- Adaptive MCMC
- Slice Sampling
- Multiple-try Metropolis
- Reversible-jump
- Hybrid Monte Carlo

2.3.2 The Output Analysis of MCMC

The convergence is an issue in the analysis of MCMC outputs. Since the early samples are strongly affected by the initial sample, they are not typically distributed according to the correct posterior. The Markov chain in MCMC will gradually converge to stationary distribution [49]. Therefore, some erroneous samples, which are generated before the convergence occurs, should be discarded. These samples are called *burn-in samples*. Although there is no exact rule for determining the number of burn-in samples, various techniques such as Gelman and Rubin's test [11,23] and Geweke's Test [25] have been proposed to decide whether the convergence has occurred.

After the burn in period, the chain starts generating samples from stationary distribution of the chain which matches the target distribution. When compared to Monte Carlo methods given in the previous section, MCMC produces dependent samples because of Markov property. This dependency causes an increase in a variance of the outputs of MCMC. The correlation between the samples also affects the mixing speed which can be defined as the speed of exploration [28]. When mixing speed is high (samples are not highly correlated), sampler quickly scans the support of the

posterior. This efficiency can be measured by the Effective Sample Size(ESS) [45] ratio.

$$ESS = \frac{M}{1 + 2\sum_{j=1}^{k} \rho(j)} \tag{2.12}$$

where M 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 auto-correlations. ESS is a good way of measuring the efficiency of MCMC algorithm since it gives the reduction in true number of samples compared to i.i.d. samples [60].

2.4 Sequential Monte Carlo

In Bayesian inference problems, the posterior distribution should be updated in some cases since the observations may arrive sequentially in time. Besides, choosing a good proposal distribution in Monte Carlo methods and MCMC methods can be non-trivial and thus building up proposal distribution sequentially may be useful strategy. For these cases including dynamic models with time-varying parameters, Sequential Monte Carlo (SMC) is a convenient alternative sampling method to MCMC. Over the recent years, SMC has found many application areas such as target tracking, missile guidance, terrain navigation, machine learning, robotics, computational biology, industrial process control and others [16]. Sequential Importance Sampling (SIS) forms the basis of Sequential Monte Carlo and several closely related algorithms such as bootstrap filters, particle filters and Monte Carlo filters have emerged from SMC.

2.4.1 Sequential Importance Sampling (SIS)

Sequential Importance Sampling is the sequential version of the Importance Sampling method given in subsection 2.2.2. Let X_n be a sequence of random variables from some random space χ_n. We are interested in approximating the following integral in which p_n is sequence of distributions known up to normalizing constant and defined

on χ_n and ϕ_n is the sequence of real valued functions.

$$p_n(\phi_n) = E_{p_n}(\phi_n(X_{1:n})) = \int p_{1:n}(x_{1:n})\phi(x_{1:n})dx_{1:n} \tag{2.13}$$

Suppose proposal distribution (importance distribution) $q(x)$ is constructed as follow where $q(x_1)$ is some initial density which is easy to sample from:

$$q(x_{1:n}) = q_1(x_1)q_1(x_2|x_1)q_n(x_n|x_1, ..., x_{n-1}) \tag{2.14}$$

Corresponding importance weight w_n and self normalized importance weight W_n^i are constructed as follows:

$$\begin{aligned} w(x_{1:n}) &= \frac{p_n(x_{1:n})}{q_n(x_{1:n})} \\ &= \frac{p_n(x_{1:n})p_{n-1}(x_{1:n-1})}{q_{n-1}(x_{1:n-1})q(x_n|x_{1:n-1})p_{n-1}(x_{1:n-1})} \\ &= w_{n-1}(x_{1:n-1})\frac{p_n(x_{1:n})}{q(x_n|x_{1:n-1})p_{n-1}(x_{1:n-1})} \end{aligned} \tag{2.15}$$

$$W_n^i = \frac{w_n(X_{1:n}^i)}{\sum_{i=1}^N w_n(X_{1:n}^i)} \tag{2.16}$$

As an approximation to p_n, the weighted empirical distribution p_n^N is calculated from the normalized particles W_n^i, where δ is the Dirac delta function. This leads to obtain the approximation of the expectation given in equation 2.13 as $p_n^N(\phi_n)$ [79].

$$p_n^N(x_{1:n}) := \sum_{i=1}^N W_n^i \delta_{X_{1:n}^i}(x^{1:n}) \tag{2.17}$$

$$p_n^N(\phi_n) = \sum_{i=1}^N W_n^i \phi_n(X_{1:n}^i) \tag{2.18}$$

The general framework of SIS algorithm is described in Algorithm 5.

Algorithm 5 Sequential Importance Sampling [79]

for n = 1,2,... **do**
  for i = 1 to N **do**
    if $n = 1$ **then**
      sample $X_1^i \sim q(.)$, (*Sampling Step*)
      calculate $w_1(X_1^i) = \dfrac{p_1(X_1^i)}{q_1(X_1^i)}$ (*Weight Update*)
    else
      sample $X_n^i \sim q(.|X_{1:n-1}^i)$ and set $X_{1:n}^i = (X_{1:n-1}^i, X_n^i)$ (*Sampling Step*)
      calculate $w_n^i(X_{1:n}) = w_{n-1}^i(X_{1:n-1}^i)\dfrac{p_n(X_{1:n}^i)}{q(X_n^i|X_{1:n-1}^i)p_{n-1}(X_{1:n-1}^i)}$ (Weight Update)
    end if
  end for
  for i = 1 to N **do**
    calculate $W_n^i = \dfrac{w_n(X_{1:n}^i)}{\sum_{i=1}^N w_n(X_{1:n}^i)}$ (*Normalizing Step*)
  end for
end for

2.4.2 Sequential Importance Sampling Resampling (SISR)

The major disadvantage of SIS is the increase in the variance of the importance weight in over time. In other words, if the proposal distribution is not close to the target distribution, after a small number of iterations, only a small part of weights will have very large weights while remaining weights are tend to be zero. This problem is called weight degeneracy.

To solve weight degeneracy problem, an additional step called resampling or selection is introduced in SIS algorithm. Resampling removes particles of small weights and creates the equally weighted particles by using weighted empirical distribution. This method is called Sequential Importance Sampling Resampling or Particle Filter and its general framework is given in Algorithm 6.

Indeed, the resampling strategy given above is called multinomial resampling. Apart from multinomial resampling, there exist many resampling strategies such as Stratified resampling, Systematic resampling, Metropolis resampling and Rejection resam-

Algorithm 6 Sequential Importance Sampling Resampling [79]

for n = 1,2,... **do**
  if $n = 1$ **then**
    for i = 1 to N **do**
      sample $X_1^i \sim q_1(.)$, (*Sampling Step*)
      calculate $W_1^i \propto \dfrac{p_1(X_1^i)}{q_1(X_1^i)}$ (*Weight Update*)
    end for
  else
    Resample from $X_{1:n-1}^i$ according to the weights W_{n-1}^i to obtain resampled and equally weighted particles $\tilde{X}_{1:n-1}^i$ (*Resampling Step*)
    for i = 1 to N **do**
      sample $X_n^i \sim q(.|\tilde{X}_{1:n-1}^i)$ and set $X_{1:n}^i = (\tilde{X}_{1:n-1}^i, X_n^i)$ (*Sampling Step*)
      calculate $W_n^i \propto \dfrac{p_n(X_{1:n}^i)}{q(X_n^i|\tilde{X}_{1:n-1}^i)p_{n-1}(\tilde{X}_{1:n-1}^i)}$ (*Weight Update*)
    end for
  end if
end for

pling in the literature [45]. It is important to note that *Importance step* and *Sampling step* in the SIS and SISR algorithms can be implemented in parallel for each particle. However, it is not trivial to create parallel designs for *Resampling step* in SISR.

One of the main application field of SMC is nonlinear filtering problems in dynamic systems such as the state space models. A state space model has two major parts: the state equation, which is represented by Markov process and the observation equation that gives the observation under hidden states.

- the state equation $x_k = f(x_{k-1}, u_k)$
- the observation equation $y_k = g(x_k, v_k)$

Such a model is sometimes called as Hidden Markov Model (HMM) in some applications such as computational biology and speech recognition [43]. The main task is the estimating hidden states and computing posterior distribution of the states of the model given sequential observation with noises. For this purpose, SMC methods can

be utilized in the cases where HMM is nonlinear or non Gaussian (For the details, see [17, 43, 79]). With simple modifications, general frameworks given in Algorithm 6 and 5 can be applied under the names of bootstrap filter to solve nonlinear filtering problems. When the targeted state space model is linear Gaussian state space, then exact analytical solutions via recursion is possible. This recursion is called Kalman filter [34].

2.5 The Acceleration of MCMC

Thanks to the recent improvements on MCMC and SMC methods, these sampling methods have found a wide variety of applications areas including computational biology, physics, financial econometrics, machine learning and image processing, target tracking, robotics and other control fields [12,43]. However, the speed of MCMC and SMC sampling are still not enough for real time applications in mobile systems. Runtime of the algorithms can exceed days, weeks or even more when they are applied to the complicated real life problems [7, 20, 29, 64, 68, 77, 78, 81]. By considering SMC acceleration as a future work, this thesis focuses on the one of the population based MCMC application and seeks the way of acceleration of the algorithm. To this end, the main reasons of long run-times in MCMC applications and the acceleration techniques of MCMC methods are summarized below:

- Computationally intense algorithms like population based MCMC and particle MCMC cause long run-times. For example, parallel tempering method needs to evaluate dozens or hundreds of chains for one iteration. This means likelihood evaluation is repeated for all chains to produce one sample. Also, apart from parallel tempering method, many computationally expensive MCMC algorithms which are dedicated to specific problems exist in the literature.

- High dimensional big data is another reason for long run-times in MCMC applications. Common scientific areas like genetic, physics, chemistry that use MCMC for inference have large scale of datasets. At each iteration of the algorithm, whole dataset should be swept over for likelihood evaluation. In these applications, practitioners face with not only large volume of data but also high

dimensional data (in the order of 10^7 in [59]). Both of these factors make the evaluation of the likelihood density computationally very expensive and so too time consuming.

- Another reason for long run-time in MCMC applications is the uncertainty of convergence. Despite there exist no exact rule for convergence test, practitioners want to guarantee the convergence as possible with long MCMC runs. Also, because of multi-modality, the mixing speed can be slow. In these cases, a long MCMC run improves the mixing by exploring state space more.

As a result of reasons given above, much efforts have been devoted to accelerate MCMC algorithms. These works can be categorized into four groups:

- *Data sub-sampling*: The aim of data sub-sampling is to reduce the size of data to be processed in every MCMC iteration. Instead of using whole data, a fraction of randomly selected data is used for inference. This method is used when a large scale i.i.d data exists. Bias in the outputs is one drawback of this method. [8, 45, 57] and [47] are some examples of works related to the data sub-sampling in the literature.

- *Data partitioning*: It is mainly based on splitting whole data into sub-groups then running each groups separately(generally parallel) and finally combining the results of each groups. This method needs i.i.d. data and parallel architecture. Like sub-sampling methods, biased results can be obtained after combining the results [13, 35]. However, [53, 73] proposed unbiased data partitioning techniques.

- *Parallel MCMC methods*: In the literature, there exist MCMC methods that have parallel nature to be used for increasing sampling speed or mixing speed. For example, population-based MCMC algorithms [33] utilizes more than one chain to increase mixing speed and thus reduce compilation time via fast convergence. Besides, some MCMC algorithms such as Multiple-Try Metropolis algorithm [44], Hamiltonian Monte Carlo [10] and Slice sampling [46] exhibit parallel nature that can be exploited by parallel hardware platforms (multi-core CPU, GPU and FPGA).

- *Parallel computations on parallel hardware*: In order to reduce run-times of MCMC algorithms, parallel computational hardware platforms such as multi-core CPUs, GPUs and FPGAs have been used in recent years. Multi-core CPU can provide multiple processing units for parallel operations that results in speed gain. However, parallel resources that is provided by CPU can be limited for MCMC algorithms when it is compared to GPU and FPGA. On the other side, GPUs have evolved to high performance computing machine due to the their highly parallel structure. Many advanced and computationally intensive algorithms can be easily suited to GPUs when compared to CPUs. Especially, for certain class of MCMC algorithms such as population based MCMC, GPUs offer massively parallel simulations [37]. However, an optimized programming is required to get better performance from GPUs.

  Apart from multi-core CPUs and GPUs, FPGAs are also suitable hardware platforms for MCMC applications. The FPGA is a re-configurable device that consists of massive size of programmable logic elements, memories, IOs, multipliers, DSP blocks and even CPU. These elements are combined in parallel such that any complex computations and tasks can be implemented. Massive parallelism property and availability of deep pipeline structure make FPGA a promising platform for accelerated MCMC applications. By dividing the algorithm into parallel tasks that can be executed at the same time and mapping them into FPGA properly, significant performance gain can be obtained when compared to other hardware platforms [45]. Another big advantage of FPGAs is the flexibility of arithmetic precision. Not only double/single-precision floating point arithmetic, which are only available precisions in CPUs and GPUs, but also custom precision is possible for FPGAs. By reducing arithmetic precision, fewer resources can be consumed and this leads to a chance of adding more parallel computation blocks. Furthermore, the memory consumption can be decreased with reduced precision (Memory is important in big-data problems). It is important to note that a tradeoff between accuracy and arithmetic precision exists and that means the decrease in the arithmetic precision of the algorithm causes biased outputs. However, the arithmetic precision of wisely selected parts of the algorithm can be reduced without sacrificing the sampling accuracy (mixed precision) [50, 52]. Lastly, the power efficiency is another

advantage of FPGAs when compared to CPUs and GPUs.

2.6 The Approach of this Thesis

In this thesis, we focus on the implementation of population based MCMC algorithms on FPGA platform in order to reduce the execution time. For this purpose, all special characteristics of FPGAs given above are exploited. Parallel pipelined sub-blocks are created to speed up likelihood computations. The performance is increased with the mixed precision design technique without sacrificing sampling accuracy. Besides, this thesis also considers the performance losses related with the external memory. Since if the bandwidth of the external memory is not enough to feed the processing unit of MCMC algorithm, the performance of processing unit can degrade. Therefore, to avoid pipeline stalls caused by memory inefficiency, data partitioning based feeding structure is created. As a summary, this thesis presents the combination of following ideas in order to solve run time problems in MCMC applications:

- Parallel computations on parallel hardware
- Parallel MCMC method
- Mixed precision
- Modified data partitioning

2.7 Related Works

As previously declared, this thesis aims to accelerate a parallelizable MCMC method by using parallel hardware platform while employing the mixed precision and data partitioning techniques. When the literature is examined, a limited number of works which are related to this thesis can be found.

Parallel computing on parallel hardware

Parallel computing on parallel hardware is very common approach in the MCMC literature. The parallel nature of some MCMC algorithms are exploited with the use

of parallel hardware platforms (multi-core CPU, GPU and FPGA). It is important to note that the forms of the parallelism and the applied techniques vary with the MCMC method, model and utilized hardware. Here, a list of some works that implement parallel MCMC algorithms on parallel hardware is presented.

In multi-core CPU field, the works [62] and [75] improve mixing speeds and thus reduce convergence time by processing multiple chains in parallel. In the work [40], Li et all. focused on the specific overhead caused by inter-processor communication in replica exchange in PT. With decentralized scalable exchange method, running time improvements were obtained when compared to centralized PT. In the work [18], CPU idle time is reduced by proposing a scheme to optimize the allocation of replicas in PT algorithm.

Tibbits et al. [46] implemented parallel multivariate slice sampling on a GPU with the aim of acceleration of the construction of the hypercube and achieved a speedup of 5-6x compared to a CPU. Da silva [63] parallelized the likelihood computation via GPU and achieved 60x speedup compared to CPU in fMRI data analysis. Beam et al. [10] accelerated the Bayesian inference on multinomial logistic regression model with Hamiltonian Monte Carlo; achieving great speedups over a CPU implementation. An important research was conducted by Lee et. al. [37] in field of population based-MCMC applications on GPU platform. GPU implementation of PT achieves one to two orders of magnitude acceleration over a CPU by assigning each chain to one separate GPU thread. With clearly given performance metrics, this work is suitable for us to compare our FPGA implementation of parallel tempering method.

Apart from multi-core CPU and GPU, some researchers investigated FPGA based MCMC Acceleration. Lin et. al. [42] created a high throughput computing machine with FPGA that can be used for Bayesian inference problems. Asadi et al. [7] implemented the combination of the MCMC and Bayesian network learning techniques on multi-FPGA system. Liu [45] focused on the design of mixed precision MCMC implementation, by allowing more parallelism and low latency and guaranteeing unbiased estimates. Besides, [82] and [5] used large degree of available parallelism and deep pipelined structure of FPGAs to reduce execution time in phylogenetic likelihood evaluations. The achieved performance gains against CPU were encouraging

for practitioners to use FPGA in MCMC applications.

Data partitioning

Another way of MCMC acceleration is data partitioning which has three stages to perform: dividing the data set into sub-groups, running a separate and parallel MCMC for each groups and combining the results to obtain the approximation of the target distribution. Consensus Monte Carlo approach [35] applies this strategy to overcome big data problems in MCMC applications. Averaging samples generated by each separate MCMC runs causes approximate results in the outputs. [32] proposes the utilization of Importance Sampling to combine the results coming from partitioned groups. Nevertheless, this approach does not guarantee the convergence theoretically. Data partitioning is exploited in [74] for MCMC implementation of Bayesian inference for latent spatial Gaussian models. Although a remarkable speedup is achieved in this work, the error in the outputs due to the approximations is drawback of this approach. On the other hand, [55] offers a combining procedure that yields asymptomatically exact samples from whole data set posterior by assuming i.i.d. data and small burn in period. In contrast to common data partitioning approaches, in our work, no separate run is performed and all sub groups are processed in order in same processing unit. Actually, the aim of data partitioning in our thesis is to avoid external memory latencies and it has no impact on output accuracy.

Mixed Precision

Some researchers gave special attention on reducing precision to increase sampling performance. This approach is useful when the algorithm runs on the FPGA platform. Since utilizing reduced precision-arithmetic operators saves FPGA area and thus more parallel processing units can be added to the system. However there is a tradeoff between arithmetic block precision and sampling accuracy. Here, some existing works that investigate the consequences of precision reduction on sampling accuracy and system's performance are given.

For Monte Carlo based simulations, Xiang and Bouganis [69] proposed a FPGA design with both high precision and reduced precision data paths to compute cumulative distribution functions (CDFs). The precision of the reduced precision data path

is adapted by comparing two CDFs using Kolmogorov-Smirnoff metric in runtime so that the distance between high precision data path's samples and reduced precision data path's samples are below the threshold. Chow et. al. [15] also proposed a precision a optimization method including mixed precision run and low precision run for Monte Carlo simulation in FPGA. The mixed precision run is used to correct the bias in the output of low precision run. The authors in [49–52] propose population based MCMC method on FPGA and investigate its performance in the custom precision land. Actually, these are closely related works for this thesis.

Closely Related Works

G. Mingas proposed novel FPGA architectures for population based MCMC method in his works [49], [51], [50] and [52]. In the work [51], a significant speed up is achieved compared to CPU for Bayesian inference on multi-model Gaussian mixture model. In the work [50], the author proposes the use of mixed precision arithmetic for population-based MCMC methods such that auxiliary chains in Parallel Tempering method are evaluated in reduced precision and more parallel processing units are added to the system to increase the system's performance without introducing bias in the output. Furthermore, [52] proposes an optimized FPGA architecture for weighted PT Method such that custom precision is used for all chains and importance weights are assigned to the samples generated by the first chain to correct the first chain's bias. [49] combines all these works and applies also these approaches to FPGA based Particle Filter. Even though huge speedups were achieved when compared to CPU and GPU implementations of the same problem, his works do not consider the properties of memory hierarchies in modern computational systems. As described earlier, modern Bayesian problems have huge data sets and thus in-chip memories of FPGAs are not enough for this applications. Furthermore, the external memory in FPGA systems can limit the system's performance due to the limited bandwidth of data transfer and latency in memory accesses.

In this work, we also focus on the acceleration of population based MCMC methods by utilizing mixed precision technique in FPGA design without sacrificing accuracy. To this end, we apply the underlying idea in [49–52] to the mixed precision FPGA design for population based MCMC. By using completely different FPGA vendor, while

we were validating the performance achievements of FPGA accelerators in existing works, we also developed existing proposed works with data partitioning method to avoid memory limitations on the system performance. Thanks to the double buffers that feed the processing unit continuously with the data extracted from DDR3 memories, no stall is introduced in the system, which keeps the performance gains achieved by likelihood acceleration. To the best of our knowledge, this is the first MCMC accelerator that presents the combination of the ideas on data partitioning method and parallel computations of population based MCMC. In contrast to existing work that estimates power consumption of FPGA, this is also the first work that provides real power consumption measurements taken from hardware in run time.

CHAPTER 3

HARDWARE ARCHITECTURES FOR PARALLEL TEMPERING ALGORITHM

MCMC algorithms that have parallel nature such as population based MCMC [33] are suitable algorithms for FPGA platforms. In population based methods, instead of running single chain, multiple chains are operated so multiple samples are produced in every iteration. The main idea of running multiple chain is improving mixing property of Markov Chain Monte Carlo methods. Indeed, the tempered chains move freely among the state space and they interact with the original distribution by exchange mechanism. These methods are more successful when compared to the basic MCMC samplers (Metropolis Hasting) for multi modal and high dimensional problems.

Parallel Tempering, or Replica Exchange, method which is given in 2.3.1.4 is one of the population based MCMC methods. [50–52] and [37] applied to Parallel Tempering method to a Gaussian mixture model by using parallel hardware (FPGA and GPU) and obtained remarkable accelerations when compared to CPU implementation of the same problem. In this chapter, we also implement Parallel Tempering method in FPGA by using the idea proposed by G. Mingas [51, 52]. Firstly, basic Parallel Tempering method is implemented in Intel Stratix V FPGA platform by utilizing double floating point arithmetic. Besides, single floating point arithmetic and mixed precision technique [50] are also implemented and their results(throughput, latency, ESS and resource usage) are compared with double floating point case. Then, we seek to find the way of area savings with architectural optimizations. Sub-blocks such as Gaussian random number generator and exchange block are optimized for extra area savings. All MCMC architectures provided in this work are evaluated with a Gaussian mixture model used in the works [37, 51, 52] and obtained results are compared

with these works. Also, the effects of the number of chains, the size of the dataset and the size of the FPGA device to the performance of the algorithm are also examined in this work. In later sections, external memory related performance losses are discussed and a FPGA architecture that adds data partitioning method to our existing architecture is proposed to solve memory related problems.

3.1 Parallel Tempering

As described in section 2.3.1.4, parallel tempering method has two iterative stages. In the update stage, a new sample is generated for every chain by using Metropolis kernel. In this work, a Gaussian distribution that has centered around the previous sample is selected as proposal distribution. The candidate sample generated from the proposal distribution is accepted with the probability α.

$$\alpha = min(1, \frac{p(\theta^*)}{p(\theta_{i-1})}) \tag{3.1}$$

In the exchange stage, an exchange between neighboring chains(q, r) occurs with the probability e. A different exchange strategy such as *the exchange between randomly selected chains* can be applied but it may change mixing speed and throughput of the system. In this work, we apply the same strategy used in [51, 52] in order to compare our result with them in the same conditions. It is important to note that $p(\theta)$ refers to posterior distribution $p(\theta|D)$ in this section.

$$\begin{aligned} e &= min(1, \frac{p_q(\theta_r^i)p_r(\theta_q^i)}{p_q(\theta_q^i)p_r(\theta_r^i)}) \\ (q, r) &= (1, 2), (3, 4)...(M-1, M) \text{ or} \\ &= (2, 3), (4, 5)...(M-2, M-1) \text{ alternatively} \end{aligned} \tag{3.2}$$

One of the important parameters in the Parallel Tempering algorithm is temperature $Temp$. It controls how the target distribution is smoothed and affects the mixing speed. The distribution of the chains is equal to $p_i(x) = p(x)^{1/Temp_i}$ where $1 =$

$Temp_1 < Temp_2 < ... < Temp_M$ and M is the number of chains. In this work, $Temp_i$ is selected as $(\frac{M}{M-1+i})^2$ which is again offered in [37, 51, 52].

The structure of the algorithm permits the parallel computations if the hardware platform possesses sufficient resources. Indeed, the parallel computations depend on the size of the hardware platform and the type of the likelihood function. One parallelism exists between the chains in the update stage. In order to generate a sample, a chain requires only its previous sample and it does not communicate with any other chain in the update stage. (The chain interactions occur in only exchange stage). Therefore, all chains can generate their own samples in parallel. Besides, the likelihood density can also be evaluated in parallel when i.i.d. data is used. In that case, the likelihood density will be equal to the product of sub densities of all data which allows the evaluation of sub-densities in parallel.

Apart from the parallelism in the architecture, pipelined system is another factor that can increase the throughput (the number of generated samples in one second). Since the same procedure(update and exchange) is applied to all chains, a long pipeline is possible. In this design, all computation blocks constitute the pipelined system. For example, while the probability density of third chain is being evaluated, the exchange between first and second chain can be performed at the same clock cycle. Actually, one drawback of the long pipeline is high latency. In this work, too much effort is given to reduce the latency, while considering the resource usage. It is important to note that as the latency of a computation block decreases, the resource utilization of that block increases.

In the following sections, FPGA architectures for parallel tempering method are presented. With utilizing parallel pipelines for probability density evaluations, high throughput is aimed. In the implementations, single precision and double precision are employed respectively. Mixed precision technique offered by Mingas [50] is also implemented. The performance of the each implementation is evaluated by comparing the results from different hardware platforms. The results obtained in this section demonstrate that a low latency and high throughput MCMC is possible without sacrificing the sampling accuracy.

Algorithm 7 Parallel Tempering [52]

Inputs: M(Number of chains), N(run time), $\theta^0_{1:M}$(initial samples of all chains), p (target density), q(proposal density)

Outputs: $\theta_1^{0:N}$ (samples of chain 1)

for i = 1 to N **do**

  for j = 1 to M **do**

    propose $\theta^* \sim q(\theta^*|\theta_j^{i-1})$

    evaluate acceptance ratio $\alpha_1 = \dfrac{p_j(\theta^*)}{p_j(\theta_j^{i-1})}$

    generate uniform random number $\mu \sim Uniform[0, 1)$

    if $\alpha_1 > \mu$ **then**

      accept θ^* and $\theta_j^i \Leftarrow \theta^*$

    else

      reject θ^* and $\theta_j^i \Leftarrow \theta_j^{i-1}$

    end if

  end for

  select exchange pairs ((1,2),(3,4)...) or ((2,3), (4,5)...) in turn

  for selected chains r and q **do**

    evaluate exchange ratio $\alpha_2 = \dfrac{p_q(\theta_r^i)p_r(\theta_q^i)}{p_q(\theta_q^i)p_r(\theta_r^i)}$

    generate uniform random number $\mu \sim Uniform[0, 1)$

    if $\alpha_2 > \mu$ **then**

      exchange the samples θ_q^i and θ_r^i

    else

      Do not exchange the samples

    end if

  end for

  Return the sample of chain 1, θ_1^i

end for

3.2 System Architectures

3.2.1 The Standard-Precision PT Architecture

In this subsection, a hardware architecture which utilizes standard precision (double floating point) for Parallel Tempering method is presented. All sub-blocks in the design work in double floating point arithmetic precision. The same implementation is also repeated for single floating point arithmetic precision and the performance metrics (sampling throughput, latency, ESS and resource usage) are evaluated in each cases. The proposed hardware architecture consists of three main blocks (Proposal block, Update block and Exchange block). All these blocks have interface with the system memories (Sample memory, Data memory, Probability memory, Temperature memory). *Sample memory* stores the last generated sample for all chains. The data used for likelihood evaluation is stored in the *Data memory*. *Probability memory* which is used in Accept/Reject stage of Update block stores the probabilities of current samples for all chain. Lastly, since the temperature value of each chain ($Temp_j = (\frac{M}{M+1-j})^2$) is constant and does not change in run time, the temperature values are initially written in the *Temperature memory* and used where needed. This approach reduces the computational load of the architecture.

The strict pipelining is applied to the architecture so that the chains are processed in all computational blocks consecutively which can be thought as streaming application as seen in Figure 3.7. In other words, the same computations on different data are repeated iteratively in all sub-blocks for all chains. In the example shown in Figure 3.7, while proposal block is preparing the samples of Chain-50 and Chain-51 respectively, the Probability Evaluation block, the Accept/reject block and the Exchange block deal with the computations of other chains. Besides, parallel pipelines are implemented in the *Update block* to calculate probability density of proposed sample. To instantiate more parallel pipelines increases the sampling throughput of the system. However, the number of parallel pipelines depends on only the available resources in FPGA. Especially, the number of floating point DSP blocks in FPGA decides the number of parallel pipelines and so the sampling throughput. The system architecture can be seen in figure 3.1.

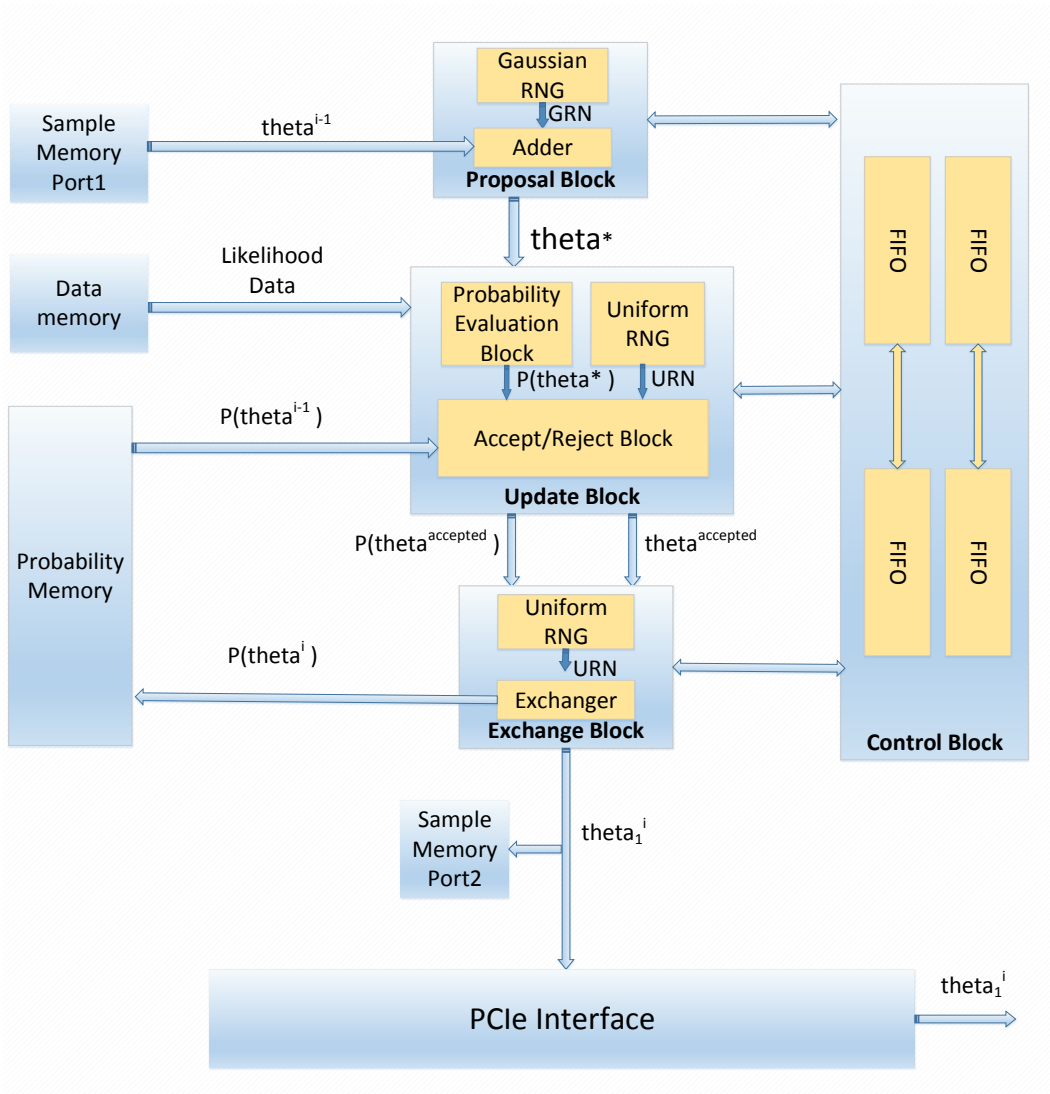

Figure 3.1: FPGA architecture of PT algorithm: In-chip FPGA memories are used for Data, Sample(Dual port) and Probability memories. The Accept/Reject block in the Update block is continuously fed by the Probability Evaluation block and the Uniform Random Number generator. The Control block controls the data flow between Proposal, Update and Exchange blocks. All system components support double floating point arithmetic precision.

Proposal Block

Proposal block is responsible for generating random sample for the each chain. The last generated sample of a chain j is read from the sample memory. Then, a Gaussian random number which is generated from the GRNG (Gaussian Random Number Generator) is added to the previous sample. The obtained value is the candidate sample θ_j^* for the chain j. This process is repeated for all chains. For the multi-modal problems, multi-samples are generated with the aid of parallel GRNGs.

To generate Gaussian random numbers is an important task for our work. Many studies have been conducted recently for random number generation in FPGA platforms [14, 38, 39, 48, 65, 80].

- In *Ziggurat method* [80] which is based on the rejection method, to produce one sample per cycle is not guaranteed. This inability can cause stalls in the MCMC application when no random number is generated.
- *Wallace method* [38] does not require uniform numbers as an input. However, it produces correlated outputs because of the feedback nature of the algorithm.
- In the work [14] *Inversion method* is applied to generate Gaussian distributed random numbers. In order to obtain samples with high statistical quality, high degree of polynomials and the large numbers of DSP blocks should be used in inversion method.
- D. Lee [39], offered FPGA architecture for *Box Muller method*, which is a transformation based Gaussian random number generator. Despite satisfactory statistical quality is obtained for the generated samples, a large number of DSP blocks is utilized in the implementation of Box Muller method on FPGA. However, with Box Muller method, two samples can be generated in one cycle which is preferable when applied to multi-dimensional problems.
- *Central Limit Theorem (CLT)* is applied with smart corrections to generate high quality Gaussian random numbers in the work [48]. The proposed algorithm tackles the problem of poor accuracy in tail regions of the probability density function which is common in CLT applications by adding error correction function to the algorithm.

- Lastly, *Piecewise-CLT algorithm* is implemented by D. Thomas [65] who offers very high statistical accuracy. Both [48] and [65] are resource efficient and able to produce one sample per cycle.

The details of the strategies given above can be found in [67]. Also D.B. Thomas made comparison of such strategies in different hardware platforms such as CPUs, GPUs and FPGAs [66].

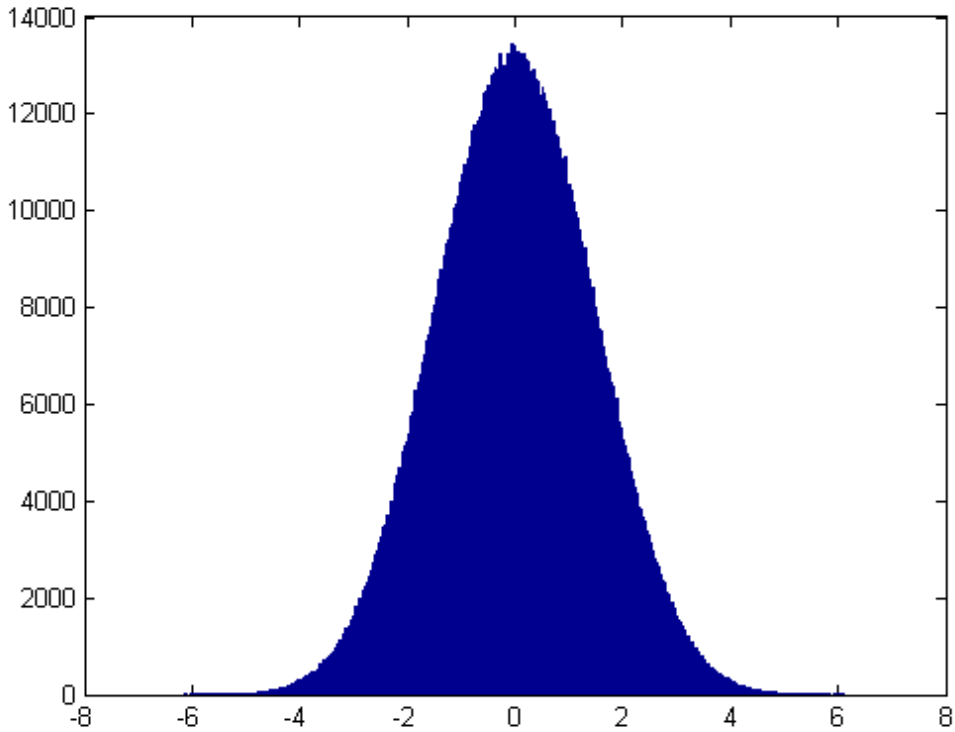

Figure 3.2: The histogram of CLT based GRNG for two million samples

In our work, we focus on resource efficient GRNG that can produce at least one sample per cycle. Also the flexibility for fixed point to floating point conversion is another important factor to choose GRNG method. To this end, CLT is applied to generate Gaussian distributed random number because of low resource usage and its simplicity. In order to increase the statistical accuracy, the sum of sixty-four parallel uniform random numbers are used in our CLT based GRNG.

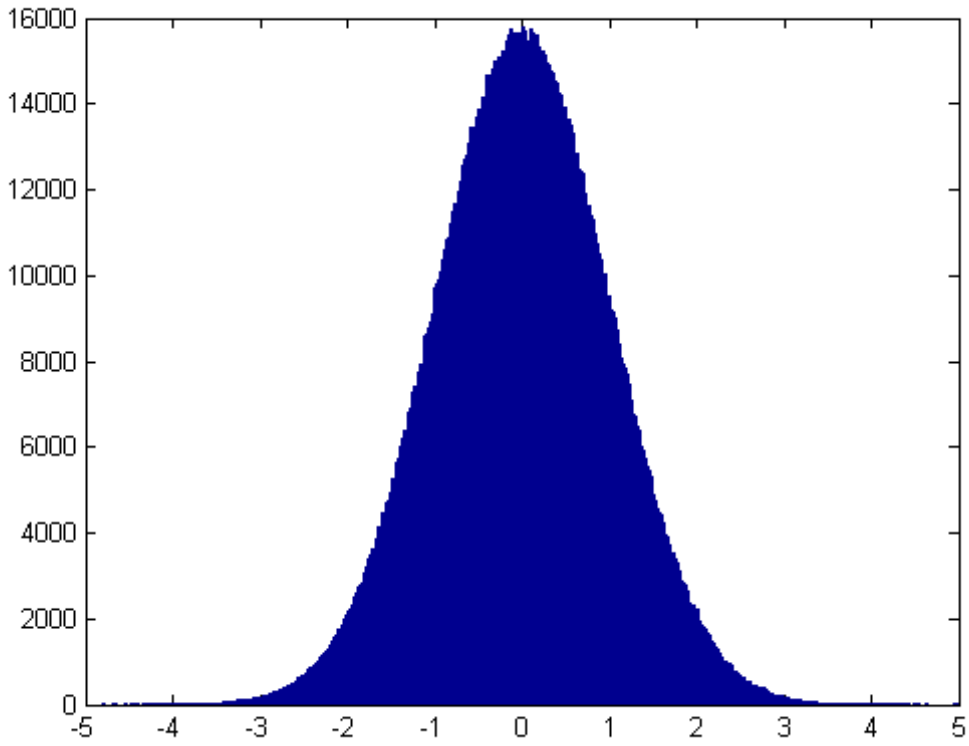

Figure 3.3: The histogram of Box Muller based GRNG for two million samples

Apart from CLT based GRNG, Box Muller method is also implemented as an alternative approach in this thesis. Despite the its disadvantage of resource usage (it utilizes DSP blocks), Box Muller based GRNG has the advantage of two times higher throughput and higher statistical accuracy when compared to CLT based GRNG. The formula for generated samples x_0 and x_1 is given in 3.3. In the formula 3.3, u_0 and u_1 are uniform random numbers generated by Tausworthe algorithm which is one of the most common techniques to generate uniform random numbers [36]. The histogram of the obtained samples by CLT based GRNG and Box Muller based GRNG are shown in Figure 3.2 and 3.3 respectively.

$$\begin{aligned} x_0 &= \sqrt{-2\ln(u_0)} * sin(2 * \pi * u_1) \\ x_1 &= \sqrt{-2\ln(u_0)} * cos(2 * \pi * u_1) \end{aligned} \tag{3.3}$$

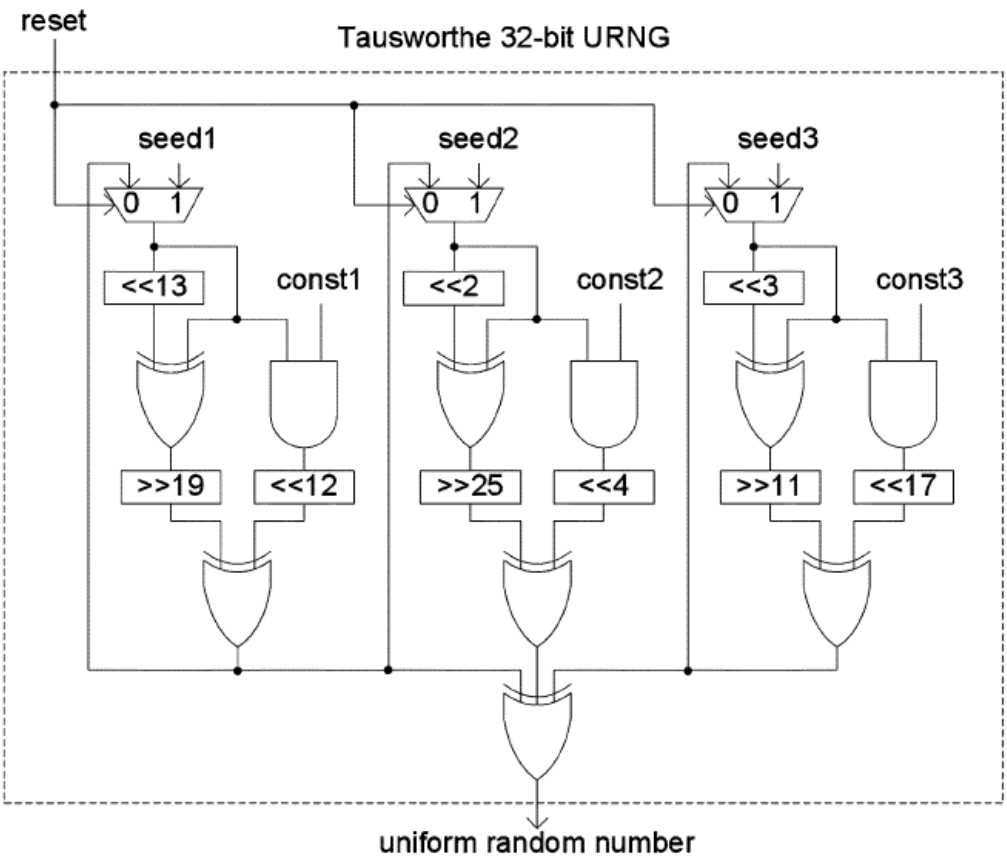

Figure 3.4: Architecture of the Tausworthe URNG [14]

Update Block

Update block is totally problem dependent. An appropriate probability evaluation block is formed and the samples proposed by Proposal Block are evaluated by utilizing Metropolis kernel. Hence, the Update block can be considered as two stages: probability evaluation stage and accept/reject stage.

The probability evaluation stage which computes the probability of candidate sample $p(\theta^*)$ is the most computationally demanding part of the algorithm. In our work, as a case study, a Gaussian mixture model which obeys the i.i.d. principle is evaluated. Hence, the implemented probability density is the product of sub-densities (see section 4.1). Figure 3.5 illustrates the pipelined sub-density evaluator block which calculates each components of Gaussian mixture density in parallel. In order to reduce resource consumption, probabilities are evaluated in logarithmic scale and the

division, multiplication and power operators are replaced with subtraction, summation and multiplication operators respectively [51]. Therefore, the probability density implemented in probability evaluation stage is actually the sum of the log-densities. The computational load of the probability evaluation stage is directly proportional to the number of i.i.d. data. To this end, parallel pipelines are utilized to evaluate probability density of the candidate sample. It is important to note that the number of parallel pipelines is dependent to FPGA-size. For the real world problems that have the large number of data and complex likelihood, fully parallel probability evaluation is not possible, thus one probability is generated in more than one clock cycles. Figure 3.6 illustrates an example for the process in the Probability Evaluation block. In this example, there exist four parallel pipelines in Probability Evaluation block and thus in every clock cycle four sub densities are evaluated. In $n/4$ clock cycles (n is the total data size), the probability of a chain for the proposed sample is calculated.

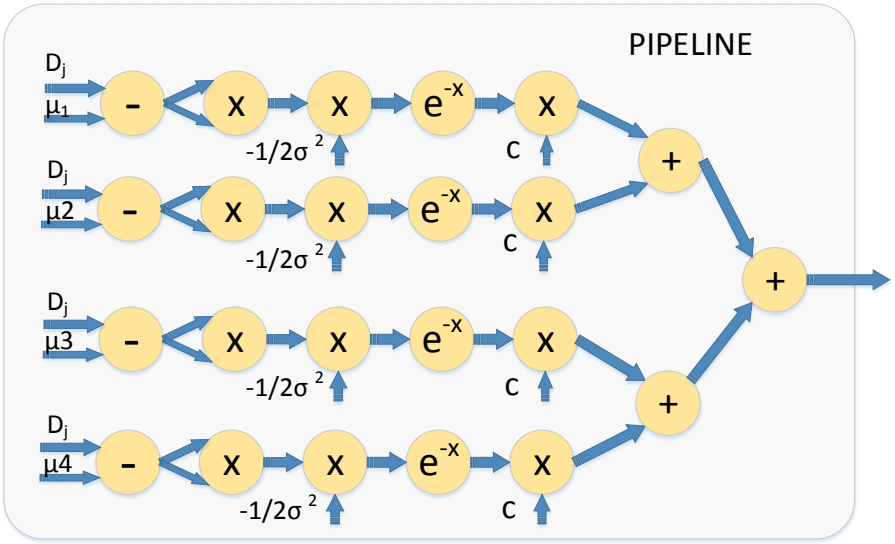

Figure 3.5: Pipelined subdensity evaluator block that calculates the four(4) components of the mixture model in parallel.

The second stage of Update block is Accept/Reject step. In this stage, Metropolis ratio, which is found by using the candidate probability $p_j(\theta^*)$ and previous probability $p_j(\theta_j^{i-1})$ of chain j, is compared with uniform random number generated by Tausworthe URNG to decide the acceptance of candidate sample θ^*. Accept/Reject module can reach the current probabilities of each chain with *Probability memory*.

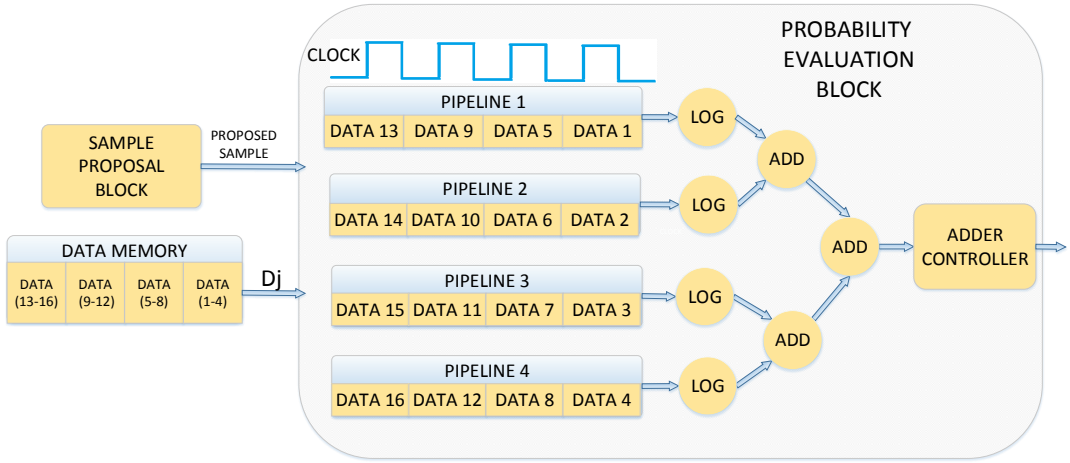

Figure 3.6: Parallel pipelines in Probability Evaluation block: There are four pipelines in the Probability Evaluation block and for a cycle four(4) sub densities are evaluated. For data size n=16, the likelihood density is generated in every four cycles for a chain.

Exchange Block

The samples generated from Proposal block and evaluated by Update block are now forwarded to Exchange block for mixing operation. Due to the nature of pipelined architecture, the Exchange block does not need to wait for the end of the update operation of all chains. When two neighboring chains are updated and forwarded to Exchange block successively, the exchange operation can be performed between them. While Exchange block is performing mixing operation between the two successive chains, Proposal block and Update block prepare the next chains for exchange operation. This pipeline ability provides significant speedups when compared to non-pipelined implementations in CPU and GPU platforms.

Many exchange strategy can be applied in this stage. For example, the exchange pairs can be selected randomly or pre-selected chains can perform the exchange at every iteration. The chosen strategy has effects on the sampling throughput, the latency and the mixing speed. In this work, the exchange between neighboring chains is applied to avoid stall in the system pipeline so the sampling throughput is maximized. Figure 3.8 illustrates the update and the exchange stage for eight chains.

In the Exchange stage, exchange rate e 3.4 is evaluated compared with uniform ran-

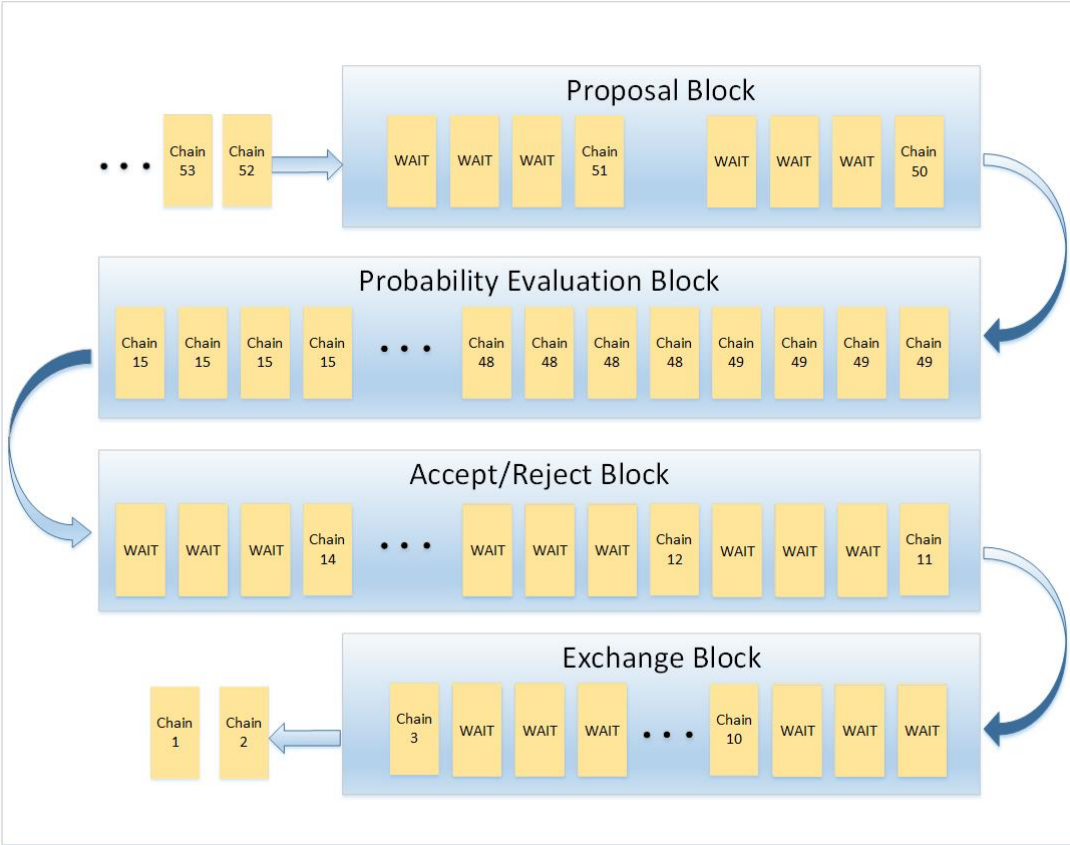

Figure 3.7: An example of chain streaming for the PT architecture. There are four pipelines in the Update block and for data size n=16, the likelihood density is generated in every four cycles. Since other blocks (Proposal, Accept/Reject and Exchange) fulfill their own tasks in one clock cycle, they are unoccupied for three clock cycles of every four cycles (wait state).

dom number generated by Tausworthe URNG. The new samples and their probability values are stored in Sample memory and Probability memory respectively. The new sample of the first chain is also given as the output in order to be transmitted by serial interface. When all stages are completed for all chains, a new iteration starts for the algorithm.

$$
\begin{aligned}
e &= min(1, \frac{p_q(\theta_r^i)p_r(\theta_q^i)}{p_q(\theta_q^i)p_r(\theta_r^i)}) \\
(q,r) &= (1,2),(3,4)...(M-1,M) \text{ or} \\
&= (2,3),(4,5)...(M-2,M-1) \text{ alternatively}
\end{aligned}
\tag{3.4}
$$

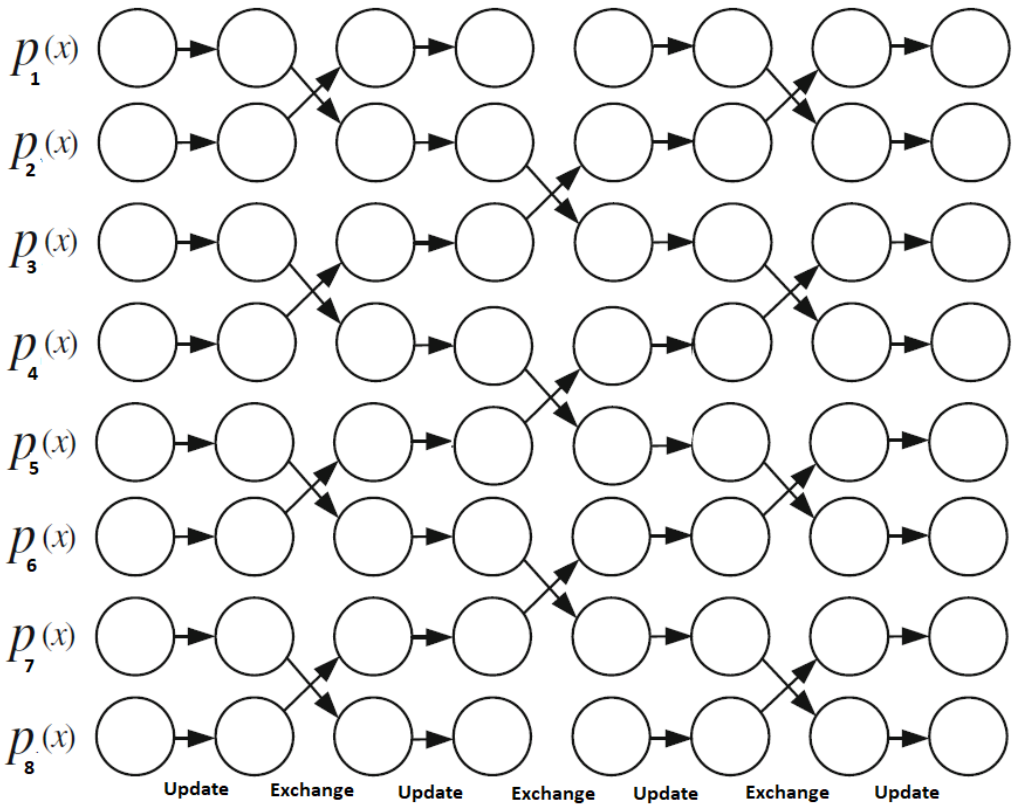

Figure 3.8: Successive update and exchange stages for eight chains

3.2.2 The Mixed-Precision PT Architecture

In the previous sub-section, only one precision(single or double) is utilized in all sub blocks of the architecture. As it will be seen later, utilizing single floating point precision in the architecture consumes less DSP block and enables more parallel pipelines and higher throughput when compared to double floating point case. However, reducing the precision causes altered probability densities and reduces the accuracy of the system.

To reduce the resource usage without sacrificing the sampling accuracy, G. Mingas [50] used the mixed precision technique in FPGA. In this approach, which is actually based on the idea of sampling from the probabilistic approximations for auxiliary chains [21], the probabilities of auxiliary chains are evaluated in reduced precision (approximated distributions) while the first chain is evaluated at double precision (true distribution). This strategy is based on the fact that only the samples of

first chain is kept and the samples of auxiliary chains are used to improve mixing before they are dropped in PT algorithm. Since the target distribution of the first chain is unchanged (the proof is given by G. Mingas in the work [49]), the output accuracy is also unchanged. On the other hand, the mixing speed of the algorithm is affected due to the fact that sample exchanges between the first and the second chains are less likely to succeed after the reduction in precision of the second chain [50].

- The first chain samples from true distribution $p_1(\theta) = p(\theta)$
- The auxiliary chains $j \epsilon (2, ..., m)$ samples from reduced precision approximation $p_j = \widetilde{p}(\theta)^{1/Temp_j}$

The exchange rate is also modified to cover two different precision domains. For the chains $q > 1$ and $r > 1$, exchanges are accepted with the probability:

$$e = min(1, \frac{\widetilde{p}_q(\theta_r^i)\widetilde{p}_r(\theta_q^i)}{\widetilde{p}_q(\theta_q^i)\widetilde{p}_r(\theta_r^i)}) \tag{3.5}$$

For the chains $q = 1$ and $r = 2$, the exchange is accepted with the probability:

$$e = min(1, \frac{p_1(\theta_2^i)\widetilde{p}_2(\theta_1^i)}{p_1(\theta_1^i)\widetilde{p}_2(\theta_2^i)}) \tag{3.6}$$

The architecture for mixed-precision is given in Figure 3.9. In Intel FPGAs, the lowest floating point arithmetic precision that is supported is single floating point. Therefore, our mixed precision design utilizes single floating point arithmetic blocks for the probability evaluation of auxiliary chains and double floating point for the rest of the design. When compared to the standard-precision(double precision) PT architecture, mixed precision architecture provides higher throughput with same sampling accuracy since it utilizes more parallel pipelines in the Probability Evaluation block. The mixed precision architecture also provides better sampling accuracy with same throughput when compared to single precision PT architecture since mixed precision architecture uses higher precision for the probability evaluation of the first chain.

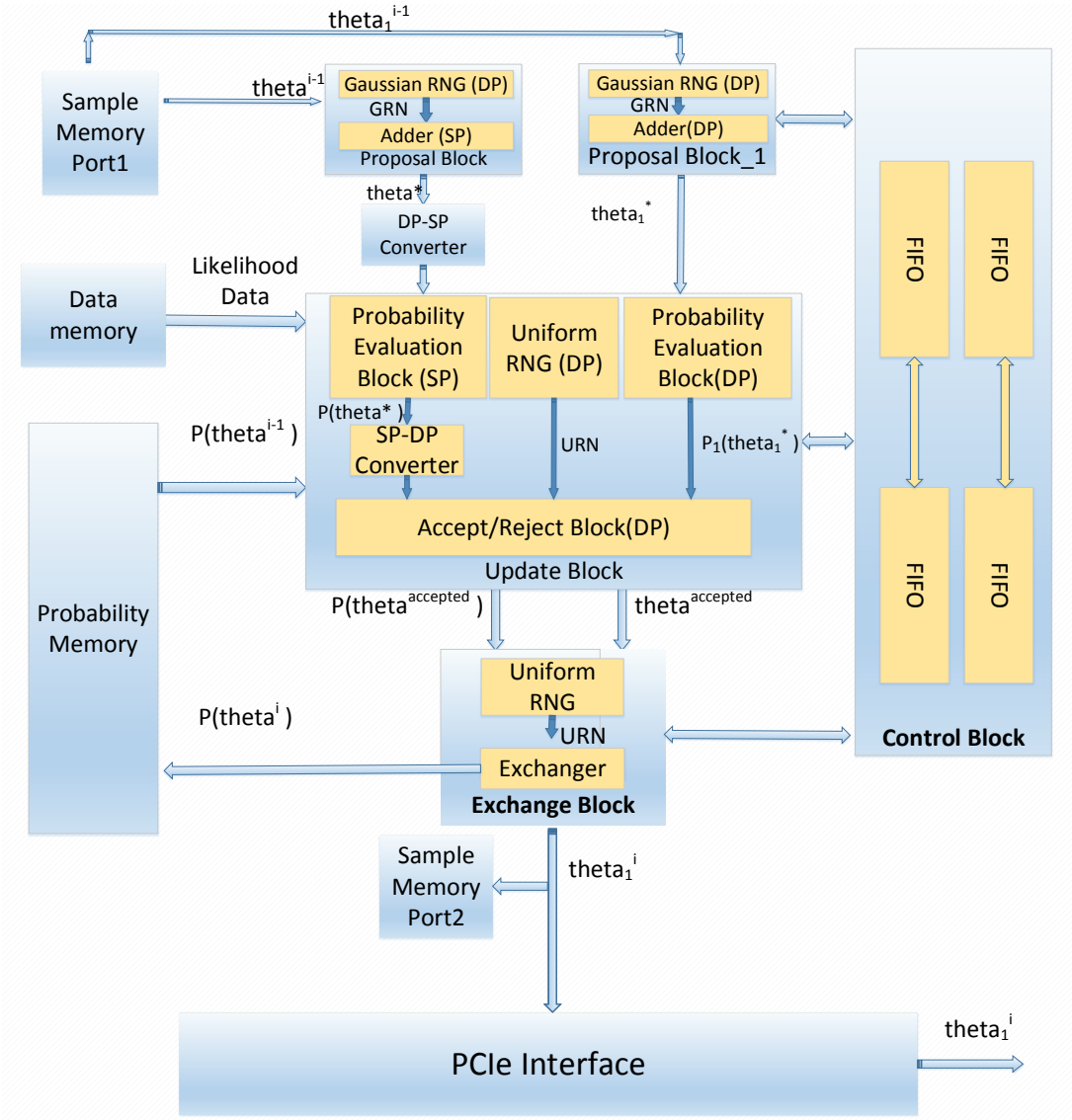

Figure 3.9: FPGA architecture of mixed precision PT algorithm: SP and DP stand for single precision and double precision respectively. For auxiliary chains, there exists a single precision-Probability Evaluation block. All other sub-blocks support double floating point arithmetic precision format. The precision of a data is converted to via SP-DP converter or DP-SP before sending to the next block that runs in different precision format.

CHAPTER 4

PERFORMANCE EVALUATION

4.1 Case Study

In order to demonstrate the hardware architectures presented in the previous chapter, Bayesian inference is performed on a Gaussian mixture model taken from [37]. Mixture models are used to model heterogeneous data and multi-modality often exists in mixture modeled problems [33].

Let $D_1, ..., D_n$ denote the observed data $D_l \in R$ where $l = 1, ..., n$ assuming that D_l obeys the i.i.d principle with the density:

$$p(D_l|\mu, \sigma, w) = \sum_{i=1}^{k} w_i f(D_l|\mu_i, \sigma_i) \tag{4.1}$$

In the equation 4.1, f refers to the univariate Gaussian distribution, k is the component number and μ_k, σ_k and w_k are component specific parameters such that $\sum_{i=1}^{k} w_i = 1$.

The known and fixed parameters are given in the Table 4.1 and only unknown parameter is μ which has k-dimensional uniform prior distribution. Data set $D_{1:n}$ is simulated using $\mu_{1:4} = (-3, 0, 3, 6)$.

To this end, we aim to infer mean $\mu_{1:4}$ via MCMC algorithm given in the previous chapter. The resulting posterior density is the product of the prior density and the likelihood density, which is also a product of sub-densities. There is no need to

Table 4.1: The Parameters of Mixture Model

Number of components (k)	Number of Likelihood Data (n)*	Dimension of data (d)	Variance σ_i	Weight w_i
4	100	1	0,55	0.25

* scalable parameter.

evaluate the normalizing constant due to the nature of used MCMC method.

$$\begin{aligned} p(\mu_{1:4}|D_{1:n}) &= p(D_{1:n}|\mu_{1:4})p(\mu_{1:4}) \\ &= \prod_{j=1}^{n} p(D_j|\mu_{i:4}, \mu_{1:4}, w_{1:4})p(\mu_{1:4}) \end{aligned} \tag{4.2}$$

4.2 Evaluation Platforms

The architecture proposed in previous chapter was implemented on a device of the Intel Stratix V FPGA series which can be considered as a medium-end FPGA. As a hardware platform, a custom FPGA board shown in Figure 4.1 is used which has one Intel Stratix V - 5SGXA7 FPGA, two 64 bits DDR3 memories and high speed interfaces like PCIe and Ethernet. Also, for real time power measurements, Altera Signal Integrity Development Board (SI) [3] shown in Figure 4.2 which has the same FPGA device as custom FPGA board (Intel Stratix V - 5SGXA7 FPGA), general I/O pins, a flash memory, high speed serial interfaces, power monitor devices (LTC2978) and temperature sensors. Apart from Stratix V device, the performance of one Intel Stratix 10 device is also evaluated. It is important to note that real runs were performed only for the Stratix V device. The results for the Stratix 10 device are obtained from performance estimates which depend on device resources and post-place and route resource utilization.

In order to compare the performance of the FPGA system with different platforms, the studies published by Lee et. al. [37] and G. Mingas [49, 52] are used as reference. In the work [37], the PT acceleration for the same target distribution is carried out on

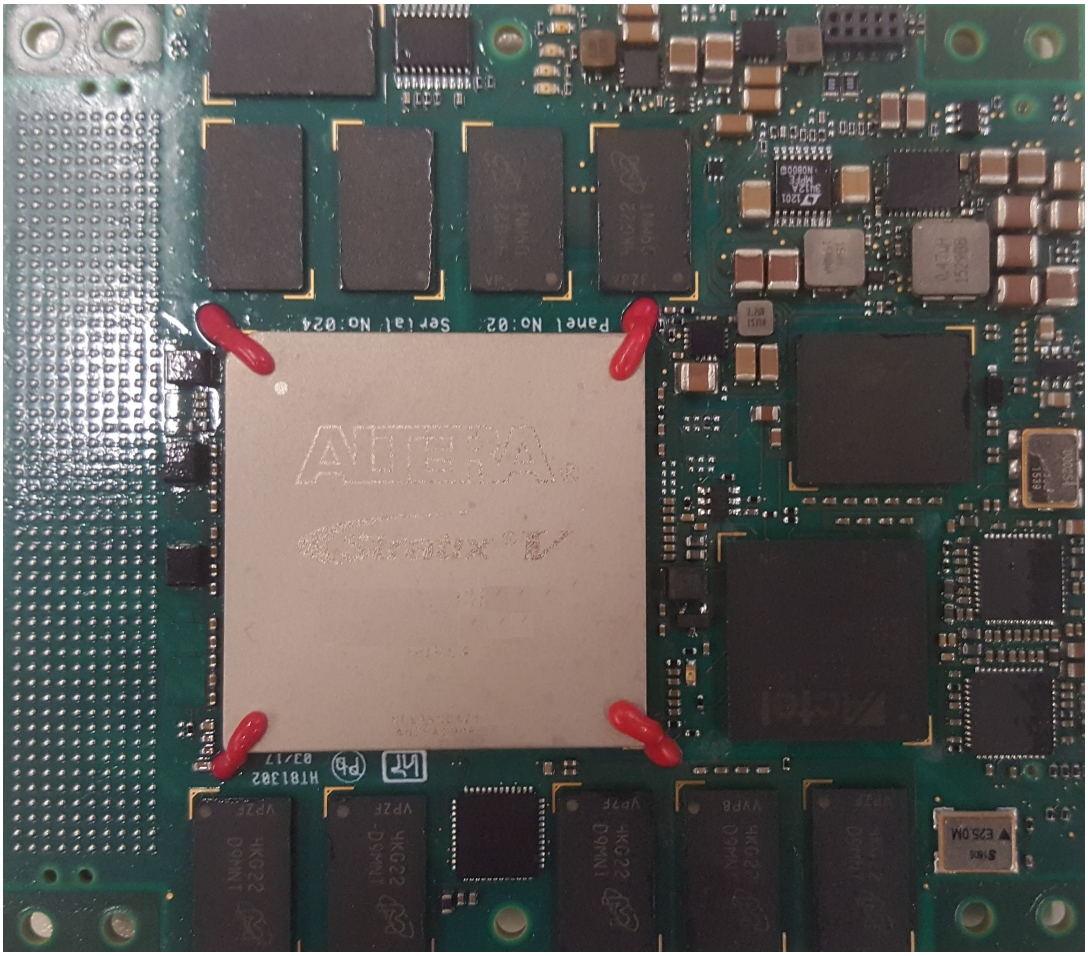

Figure 4.1: The custom FPGA board includes Intel Stratix V - 5SGXA7 FPGA, two 64 bits DDR3 memories, PCIe and Ethernet Interfaces.

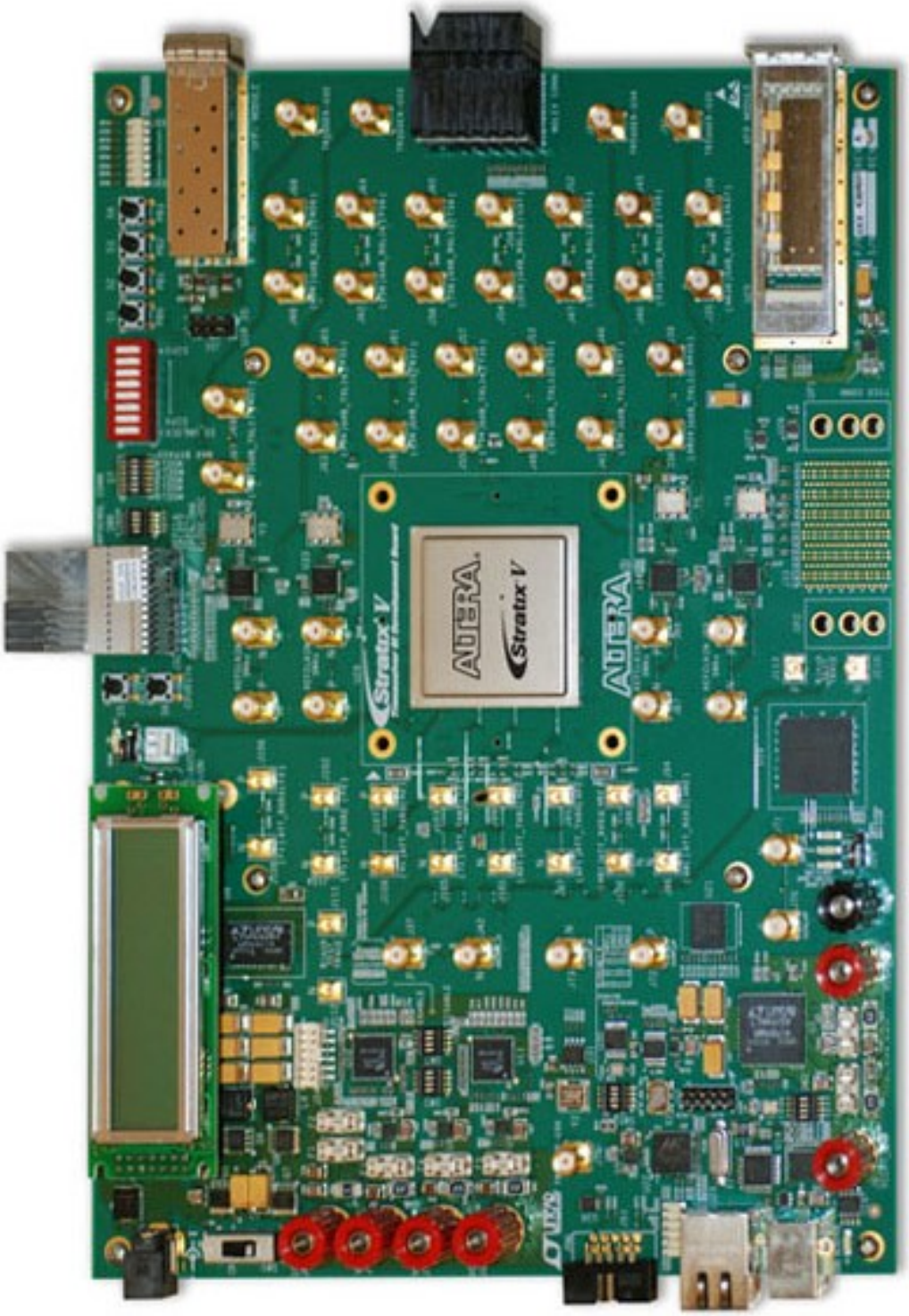

Figure 4.2: Altera Signal Integrity Development Board (SI) [3] includes Intel Stratix V - 5SGXA7 FPGA, power monitors and temperature sensors. It is used to measure real time power consumption of core FPGA.

Nvidia 8800GT and Nvidia GTX280 GPU platforms. Lee et al. [37] also presented the performance results of sequential CPU implementation of PT algorithm for the Intel Xeon E5420 device.

On the other hand, G. Mingas [49, 52] uses double precision floating point arithmetic and mixed precision in PT acceleration on three different platforms for the same target distribution. For the GPU platform, the devices of the Nvidia GeForce-285 and Nvidia GeForce-480 series are utilized in his works. Intel Xeon E5 series devices with varying number of cores are the representative of the multi-core CPU in his works. He also presented the performance results of PT acceleration for Xilinx Virtex 6 and Virtex 7 series FPGAs. All these performance data provided by G. Mingas [49,52] are evaluated as reference to evaluate our FPGA design. For fair comparisons, the same target distribution is evaluated and PT algorithm is identically tuned in all platforms.

4.3 FPGA Implementation

The PT architectures given in previous sections are designed via VHDL, simulated by QuestaSim, which is an advanced simulation tool for VHDL verification and lastly implemented on Intel Stratix V FPGA device (5SGXA7), which is a medium-end device of Intel FPGA families. Despite, the actual runs are performed in Intel Stratix V FPGA on the hardware platform, the results for Intel Stratix 10 GX1100 FPGA, which is one of the highest performance FPGA family of Intel FPGAs brand, are also presented. Indeed, the results for the Intel Stratix 10 GX1100 FPGA are estimated by looking resource utilization of the PT architectures and Stratix 10 device resources.

The generated samples by the FPGA are transferred to the host processor via PCI Express interface. Transfer rate of the interface is over 100 MB/sec, which is enough for transferring all samples without any losses. The clock frequency of single precision PT architecture is set to 225 MHz. For the other PT architectures, the clock frequency is set to 200 MHz which is obtained by internal PLL of the FPGA driven by 50 MHz reference clock. It is important to note that the clock frequency is mainly dependent on the performance of floating point arithmetic IPs and timing performance of the FPGA design. Since totally synchronous design technique is employed in the

FPGA architecture, only limiting factor for the clock frequency is the performance of floating point arithmetic IPs which is related to the precision of the IPs. For single precision floating point IPs, it is possible to obtain higher clock frequency.

The number of burn in samples (5000), the number of likelihood data(n), the number of chains (M), the temperature of the chains ($Temp_M$), initial samples, clock speed and other constants such as variance (σ_k), component number (k) and dimension of samples (d) are fixed and set before synthesize to reduce resource usage. The system starts to work with power on reset and outputs of the PT algorithm (burn-in samples are discarded) are sent via PCIe to the host processor.

4.4 Performance Metrics

This section describes the outcomes of FPGA implementation of PT algorithm. The given architectures will be evaluated in terms of sampling throughput, the latency, the resource utilization and the power consumption.

Sampling Throughput

The throughput of the architecture can be considered as the number of generated samples per second. In our design, the sampling throughput is totally related to the chain number, the data size, the clock speed and the number of parallel pipeline in the Probability Evaluation block.

$$SamplingThroughput = \frac{frequency}{M\frac{n}{p}} \tag{4.3}$$

Here, *M* refers to the chain number, *n* is the size of the data for likelihood evaluation, *p* is the number of parallel pipelines in the Probability Evaluation block and *frequency* is the clock speed of the FPGA.

Latency

Latency can be considered as the delay between starting point of the algorithm and the point where the first sample is generated. The latency of floating point arithmetic blocks will decide the latency of whole system. The exact formulas for the latency of

the system are given in Table 4.2.

Table 4.2: Latency formula for PT blocks

Block	**Latency formula**
Proposal Block	Lat(add)
Probability Evaluation Block	4*Lat(mult) + Lat(exp) + 7*Lat(add) + Lat(log) + Lat(control)
Accept/Reject Block	Lat(mult) + Lat(add) + Lat(comp)
Exchange Block	Lat(mult) + 2*Lat(add) + Lat(comp) + Lat(exchanger)
Total System	6*Lat(mult) + 11*Lat(add) + Lat(exp) + Lat(log) + 2*Lat(comp) + Lat(control) + Lat(exchanger)

Lat(add), Lat(mult), Lat(log), Lat(comp), Lat(exp) are the latencies of the floating point add/subtract IP, the floating point multiplier IP, the floating point logarithm IP, the floating point comparator IP and the floating point exponent IP, respectively. Lat(control) refers to the latency caused by the data control operation inside the probability evaluation block. In mixed precision architecture, there exist also latencies caused by SP/DP converter and DP/SP converter. It is important to note that the precision of the arithmetic blocks affects their latency.

Power Consumption

The power consumption results of the PT architecture on FPGA platform can be obtained with two different ways. The first way is using Intel PowerPlay Early Power Estimator(EPE) tool. EPE estimates power consumption of designs without implementation on real hardware. When the targeted FPGA family with total resource utilization and clock frequency of the design are provided, EPE tool estimates power consumption with a good accuracy [1]. The second method to measure power consumption is using power monitor chip on Altera Signal Integrity Development Board, named as LTC2978 [2]. LTC2978 measures the current drawn by core voltage rail(0.85 V) of FPGA. Although, there exist more than one LTC2978, which monitor the transceiver voltages, I/O voltages of FPGA and the voltages of other components on the board, our primary focus is the power consumed by the core voltage rail of FPGA. LTC2978 measures the current value over the sense resistor connected to the

core voltage power rail of FPGA. By multiplying this value with core voltage (0.85 V), power consumption value in Watts is obtained. The data transfer between FPGA and LTC2978, including the configuration of LTC2978 and reading measured values is established via I2C interface. In this thesis, both methods are employed. Real-time results attained from LTC2978 device are compared with estimations of EPE-tool in order to verify power consumption of the PT implementation on FPGA. Also, these results are compared with the nominal thermal power design of other platforms (CPU and GPU) for different conditions (data size and chain number).

4.5 Performance Results

In this section, the PT accelerators (CPU, GPU and FPGA) are compared in terms of throughput and power efficiency. How the power consumption and throughput scale with the number of chains and the size of likelihood data is also examined. Besides, FPGA resource utilization and the latency of PT architecture under different precision are presented. As described earlier, the performance results of PT architecture on CPU and GPU for the same case study are taken from the works [37] and [49, 52]. Before giving the performance results of the architecture, the distribution of the samples generated by FPGA when the chain number is 32 and 128 are given in Figure 4.3 and Figure 4.5 respectively.

The marginal density $p(\mu_{1:2}|D)$ in Figure 4.3 and 4.5 has 12 modes in $\mathbb{R}^2$. On the other side, the joint density $p(\mu_{1:4}|D)$ traverses all modes in $\mathbb{R}^4$ for all cases where M is equal or greater than 8. In other words, PT sampler on FPGA has the ability to draw samples from all modes in the sample space with small variance and almost true mean. Figure 4.4 and Figure 4.6 show the histogram of the 60000 samples generated by PT sampler on FPGA for μ_1. Since the statistical quality of random number generators affects the accept reject rate and exchange rate, better inference are possible with the use of random number generators that have more statistical quality.

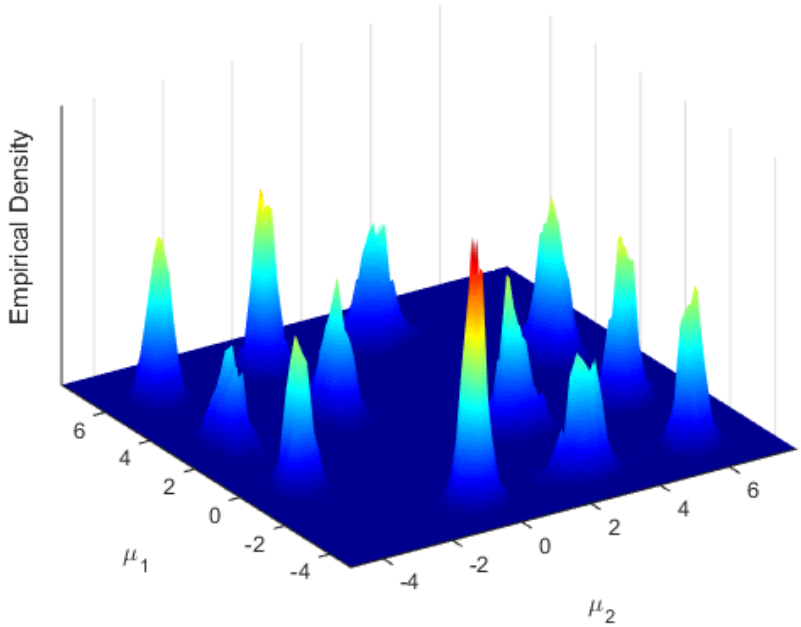

Figure 4.3: Estimated marginal posterior density $p(\mu_{1:2}|D)$ from 60000 samples generated via PT method on FPGA with 32 chains.

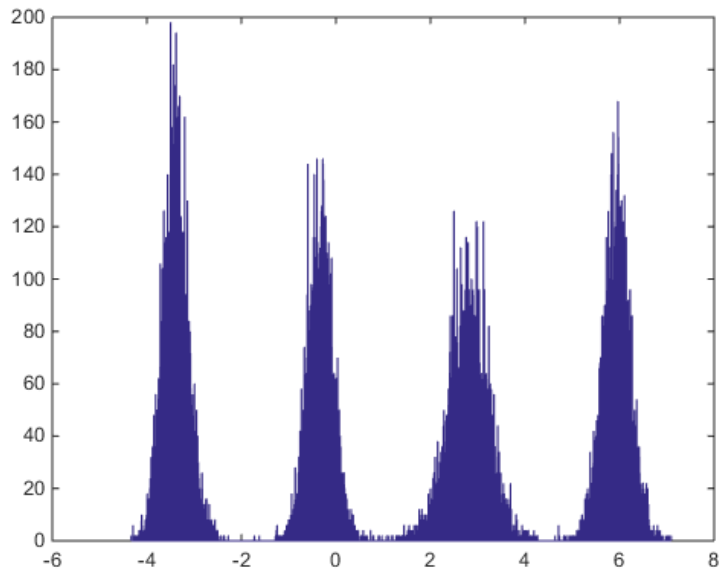

Figure 4.4: The histogram of 60000 samples generated via PT method on FPGA with 32 chains for μ_1.

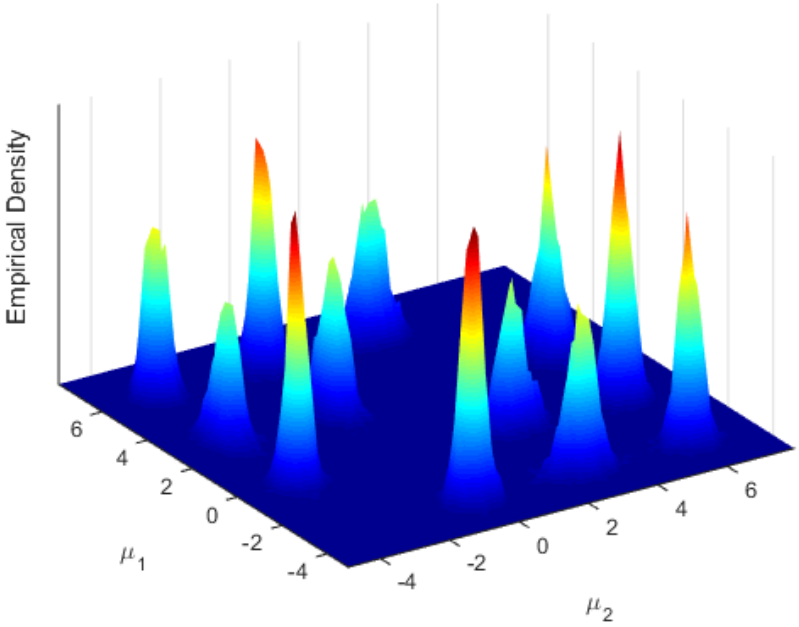

Figure 4.5: Estimated marginal posterior density $p(\mu_{1:2}|D)$ from 60000 samples generated via PT method on FPGA with 128 chains.

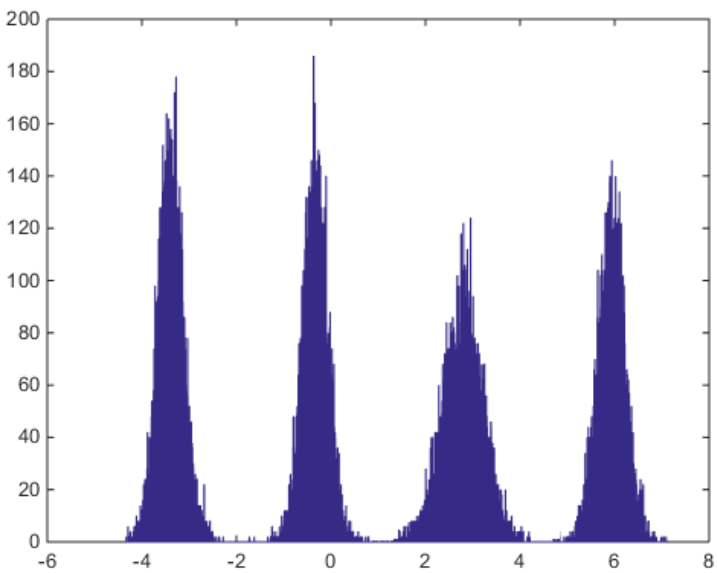

Figure 4.6: The histogram of 60000 samples generated via PT method on FPGA with 128 chains for μ_1.

4.5.1 Throughput Analysis

As given in equation 4.3, the throughput of the PT architecture depends on the number of chains(M), the data size(n), the number of parallel pipelines in the probability evaluation block(p) and the clock frequency of the system. Figure 4.7 illustrates the functional simulation of Probability Evaluation block for $n = 16$ and $p = 4$. In that case, Probability Evaluation block calculates the posterior probability of a chain in every ($\frac{n}{p} = 4$) clock cycles. If the total number of chains is 128, the current samples and probabilities of the first chain, which are the output of the algorithm, are calculated in every ($M\frac{n}{p} = 512$) clock cycles and the sampling throughput will equal to ($\frac{frequency}{M\frac{n}{p}} = 390625$). This example is illustrated in figure 4.8.

Here, the number of parallel pipelines is the important parameter for the sampling throughput and its limiting factor is the number of DSP blocks in the FPGA. A single precision pipeline in the probability evaluation block consumes 19.1% of the existing DSP blocks in Stratix V (5SGXEA7) device. On the other hand, the double precision pipeline consumes 44.5% of the existing DSP blocks. The total resource usage for double precision PT architecture is given in Table 4.9.

Besides, the clock frequency is dependent to the performance of the floating point DSP IPs. For the single precision floating point PT architecture, the floating point adder IP supports 228 MHz clock frequency at maximum. For the double precision floating point PT architecture, the maximum supported clock frequency is limited by the floating point logarithm IP, which is 211 Mhz. In order to stay in the safe side, the clock frequency of the system is selected as 200 Mhz and 225 MHz for the double precision PT architecture and the single precision PT architecture respectively.

Despite the fact that the throughput of the system is inversely proportional to the number of chains employed in Parallel Tempering simulations, the number of chains should be large enough to ensure good mixing. Indeed, utilizing dozens of chains seems acceptable for even complex problems in PT literature. However, in order to investigate how the throughput scales with the number of chains in different platforms, various different numbers of chains were used while setting the amount of data to fixed number (128) in this work.

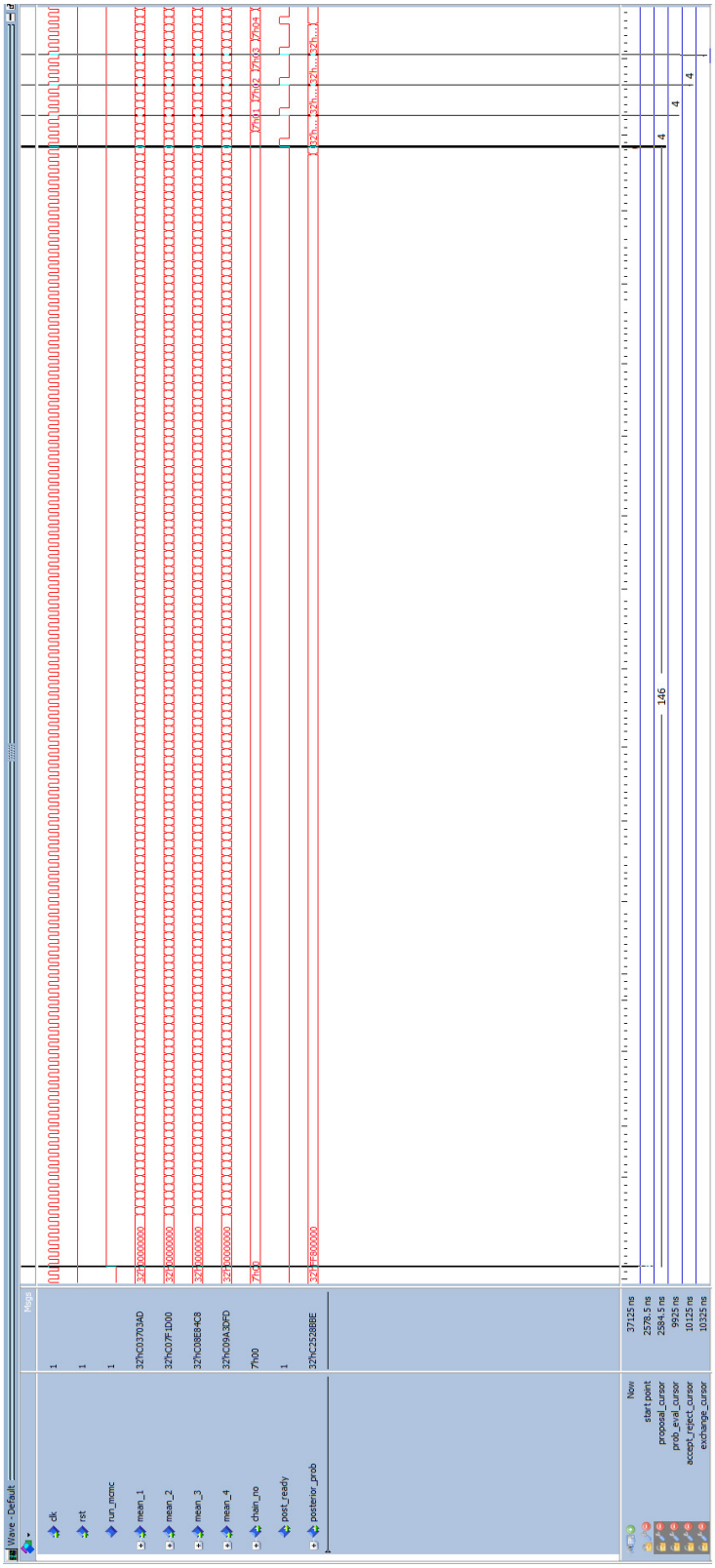

Figure 4.7: Functional simulation of Probability Evaluation block for n=16 and p=4: The Probability Evaluation Block starts to produce the posterior probability of chains in every four clock cycle after 146 clock cycles delay from the start of the algorithm.

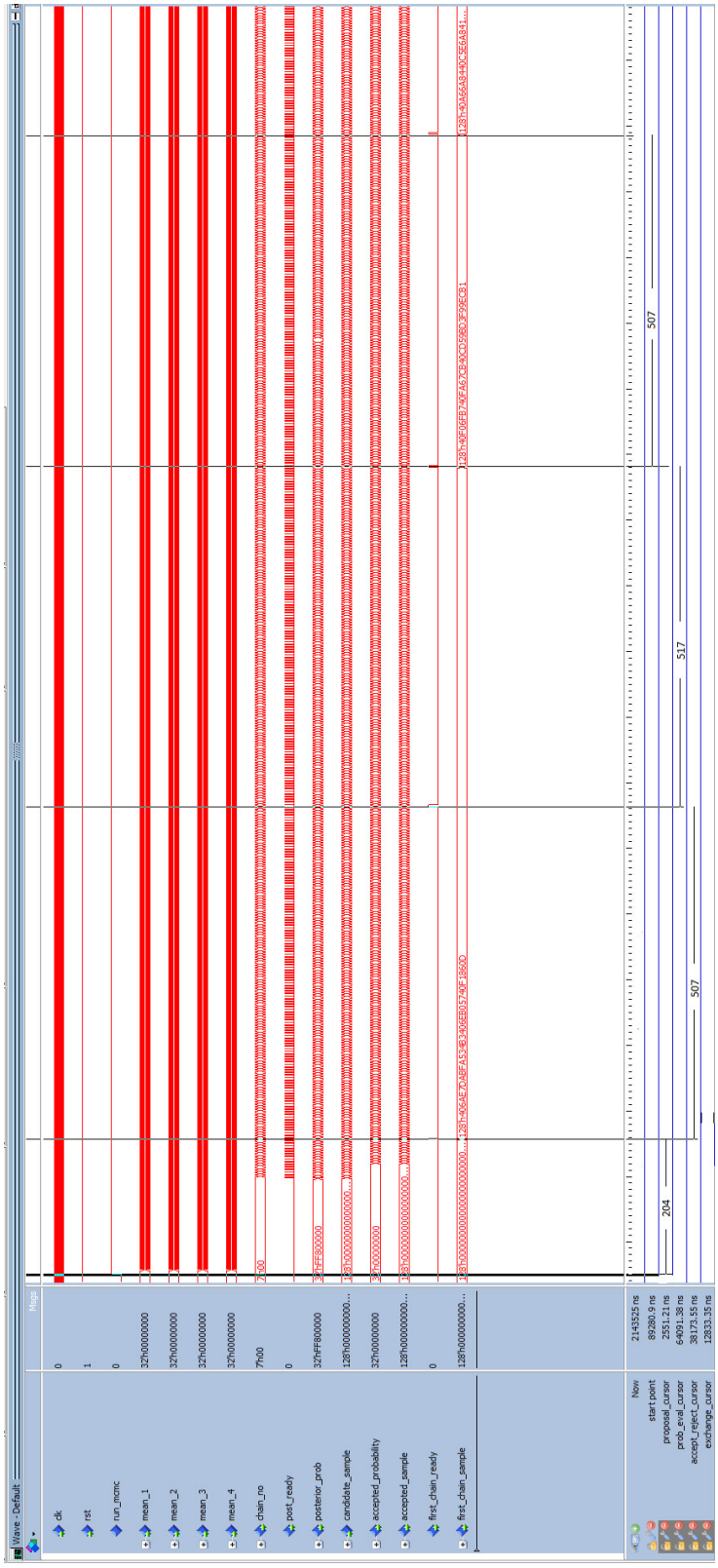

Figure 4.8: Functional simulation of PT architecture for M=128, n=16 and p=4: The PT architecture starts to produce the current samples and probabilities of the first chain in every 512 clock cycles.

The throughput achieved by PT accelerators on FPGA for the fixed data size (128) and various numbers of chains are given in Table 4.3. It is important to note that only real runs are performed for Intel Stratix V-5SGXA7 device. The throughput of Intel Stratix 10-GX1100 device are estimated by using resource utilization of the PT architectures and the device resources. The throughput of the FPGAs given in Table 4.3 are compared with sequential reference implementation in C++ on Intel Core i7-2600 device with one core activated [49] [52] in Figure 4.9. In Figure 4.9, there also exist other speedups achieved by GPU and multi-core CPU implementation of the same problem. The throughput and the speedups given in Table 4.3 and Figure 4.9 reflect the measurements taken from the double precision accelerators (CPU, GPU and FPGA).

Table 4.3: Throughput (samples/sec) of the PT on FPGAs for different chain numbers M when n=128.

Platform	**Device**	**M = 8**	**M = 32**	**M = 128**	**M = 512**	**M = 2048**
FPGA	Intel Stratix V-5SGXA7	390625	97656	24414	6103	1525
FPGA	Intel Stratix 10-GX1100	3515625	878906	219726	54931	13732

For multi-core CPU, G. Mingas achieves speedup at most 13.8x with respect to sequential reference implementation [49] [52]. The peak performance is achieved with more than 2048 chains. In his works [49] [52], GPUs outperform multi-core CPU for almost all numbers of chains. For a few hundred chains, the achieved speedups are in the range of 1.5x-50x. Like multi-core CPU, GPUs reach their peak performance for thousands of chains thanks to the fully utilization of GPU with adequate parallelism. However, GPU implementation of Lee et al. [37] outperforms the multi-core CPU [49] [52] for $M > 512$ due to the fact that the state-of-the-art implementation of Lee et al. [37] needs massive number of chains to fully utilize GPU.

Our FPGA accelerators achieve constant speedup against sequential reference implementation for all cases. Intel Stratix V - 5SGXA7 device has throughput 24x higher than the reference. This value is close to the FPGA(Xilinx Virtex-6) implementation of G. Mingas [49] [52]. When a device of upper family(Intel Stratix 10-GX1100)

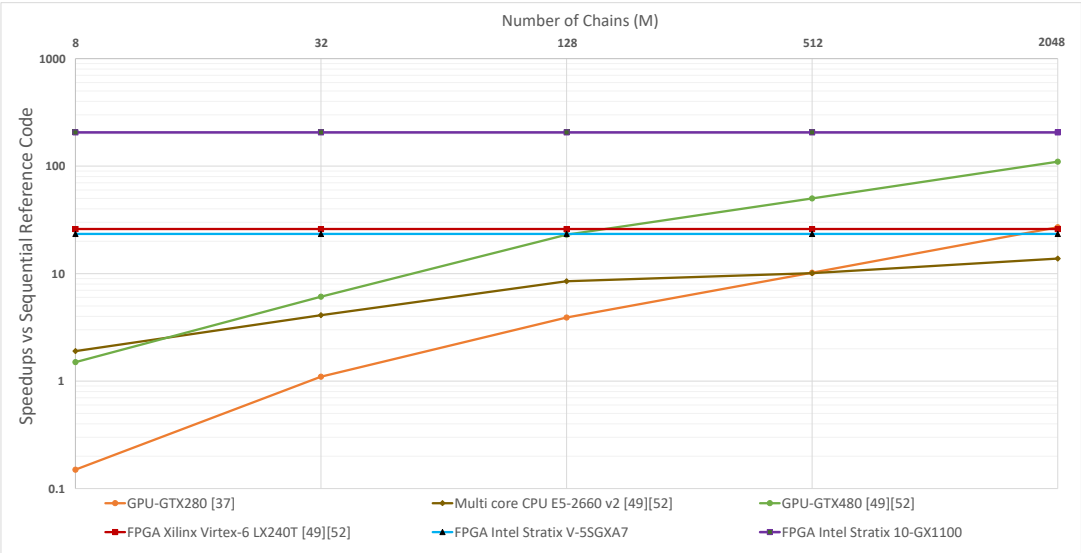

Figure 4.9: The speedups of FPGAs, Multi-core CPU [49] [52] and GPUs [37]cite5 [52] vs the sequential reference implementation on Intel Core i7-2600 CPU [49] [52] for different numbers of chains M.

is used, the achieved speedup increases to 206x which is higher than the all of the state-art-implementation of G.Mingas [49] [52] and Lee et. al. [37], regardless of the chain number. Also, for the reasonable number of chains(up to 128), even the implementation on Intel Stratix V - 5SGXA7 device is faster than GPU samplers and multi-core CPU sampler. This shows that FPGA sampler provides better throughput than the multi-core CPU and GPU for the realistic cases.

The results given in Table 4.3 and Figure 4.9 show that the number of chains linearly down-scales the throughput of the FPGA sampler and result in a constant speedup against the sequential reference implementation. The peak speedup for the FPGA is reached even for M equals 8. On other side, multi-core CPU and GPUs [37, 49, 52] reach their peak results for unrealistic scenarios (utilizing thousands of chains). This is due to the fact that the created parallelism in FPGA is related to the pipelines for probability evaluation. If a small number of likelihood data and big FPGA like Stratix-10 are used, inter-chain parallelism is also possible. By processing each chain in parallel, faster samplers can be designed with FPGAs.

Another parameter that scales the throughput of FPGA sampler is the amount of data. In order to investigate the way the throughput scales with the data size, various num-

bers of data sets are used while number of chains is set to 32. Table 4.4 shows the throughput achieved by Intel Stratix FPGA implementations for various numbers of data sets. Figure 4.10 shows the scaling of speedups with number of data for different platforms (CPU, GPU and FPGA) against the sequential reference implementation in C++ on Intel Core i7-2600 device with one core activated. Multi-core CPU sampler implemented by G. Mingas achieves speedup 16.1x at most against the sequential reference implementation. Also, speedups for the GPUs are between 5x and 99x. When compared to Figure 4.9, this performance drop is expected since GPUs perform better for large values of chain numbers and for this case the chain number is only 32 [49] [52].

On the other side, our FPGA samplers keep their constant performance achievements for also varying numbers of data sets. Intel Stratix V-5SGXA7 device reaches to 24x speedups against the sequential reference implementation on CPU. When Intel Stratix 10-GX1100 device is used, this speedup reaches to 206x. This shows that our FPGA sampler performs stable performance for different system parameters (chain and data size), while the multi-core CPU and GPU performances reduce with small numbers of chains. The main reason is that the FPGA takes advantage of all kinds of parallelism (the parallelism between chains and the parallelism inside the chain) in the system while the performance of multi-core CPUs and GPUs depend mostly on the parallelism between chains. However, it is important to note that the parallelism inside the chains is due to the i.i.d. data usage for the problem given in Section 4.1. In other words, FPGA exploits the parallelism between sub-densities which are multiplied later to calculate the density and thus the way of performance scale with data size can not be generalized for the non i.i.d. problems. Nevertheless, the FPGA is able to take advantage of both parallelism and it is possible to obtain performance achievements with the parallelism between chains for non i.i.d. data cases.

4.5.2 Power Analysis

With the help of the power monitor device, named LTC2978, the power consumption of the PT architecture is analyzed on the real platform. For this purpose, Altera Signal Integrity (SI) Development Board, which can be seen in Figure 4.2, is used. On this

Table 4.4: Throughput (samples/sec) of the PT on FPGAs for different number of data (n)

Platform	Device	n = 32	n =128	n = 512	n = 2048	n = 8192
FPGA	Intel Stratix V-5SGXA7	390625	97656	24414	6103	1525
FPGA	Intel Stratix 10-GX1100	3515625	878906	219726	54931	13732

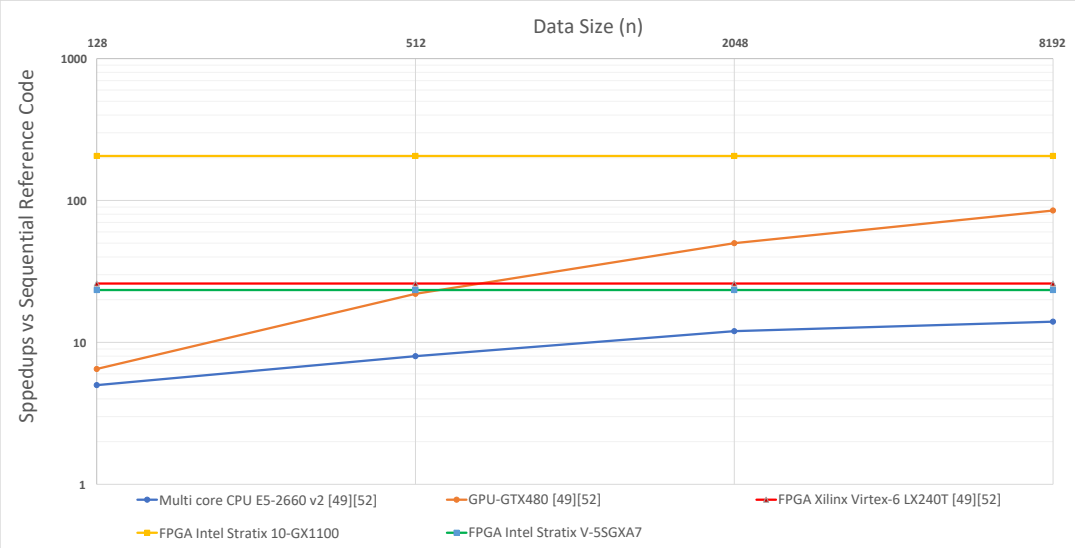

Figure 4.10: The speedups of FPGAs, Multi-core CPU [49] [52] and GPUs [49] [52] vs the sequential reference implementation on CPU [49] [52] for different number of data (n) when M=32.

development board, one Intel Stratix V - 5SGXEA7 FPGA device and two LTC2978 power monitor devices exist with memories and high speed components [3]. The data transfer from LTC2978 device to FPGA is accomplished via I2C interface. Since the configuration LTC2978 device is out of scope in this thesis, the details of I2C transfer is not given. While analyzing the power consumption of PT architecture on FPGA, the current drawn by the core voltage (0.85 V) is considered primarily.

The consumed power by core voltage rail is measured for three different (M, n) combinations and given in Table 4.6. Apart from actual measurements obtained from the board, power consumption values estimated by Intel Early Power Estimator Tool

for PT architecture are also given in Table 4.5. It is important to note that EPE tool estimates the power consumption by using resource utilization of targeted FPGA, the clock frequency of the design and the percentage of read-write mode for on-chip memories.

Real measurements taken from the platform and EPE estimations reveal that PT architecture without external memory (DDR3) on Intel Stratix V FPGA consumes power between *5.5 Watts* and *7 Watts*. The power consumption of the FPGA scales with the number of chains and the data size due to the fact that more on-chip memory and ALUTs are utilized for more chains and more data size. However, power consumption of FPGA board (custom FPGA board that has Intel Stratix-V FPGA and DDR3 interface) reaches *13.3 Watts* when external memories are enabled. This increase in power consumption is due to the usage of transceivers in FPGA and power consumption of DDR3 itself. Note that this power consumption includes not only the power consumption of the core voltage rail but also the power consumption of total FPGA board.

On the other side, power consumption of CPU and GPU accelerators which are designed by G. Mingas [49] [52] are between *95 Watts* and *260 Watts*. These values are the nominal thermal power taken from each device specifications. In order to compare platforms fairly, FPGA is considered as fully utilized despite the fact that this is not valid for our FPGA designs. Even with this assumption, FPGA accelerator which consumes *16 Watts* achieves 6x-16x power savings when compared to other platforms(CPU and GPU).

Table 4.5: Power consumption of PT estimated by EPE tool for different number of chains(M) and data size(n) combinations

	(M,n) =(128,16)	**(M,n) =(128,128)**	**(M,n) =(2048,128)**	**(M,n) =(128,2048)**
Power (W)	5.76	5.92	6.43	6.74

Table 4.6: Power consumption of PT measured by LTC2978 on Altera SI Development Board for different number of chains(M) and data size(n) combinations

	(M,n) =(128,16)	**(M,n) =(128,128)**	**(M,n) =(128,2048)**
Power (W)	5.56	6.13	6.98
Current(A)	6.54	7.21	8.21

Table 4.7: Power consumption of PT estimated by EPE tool for M=128 and n=16

INPUT PARAMETERS	
Family	Stratix V
Device	5SGXEA7N
Package	F40
Temperature Grade	Commercial -2L/-3/-4 (0.85V)
Power Characteristics	Typical
THERMAL POWER (W)	
Logic	1.291
RAM	0.703
DSP	1.819
I/O	0.002
PLL	0.014
CLOCK	0.787
XCVR	0.073
PCS and HIP	0.209
Power static	0.860
TOTAL	5.758

4.5.3 Latency Analysis

Another performance metric evaluated in this work is the latency. Table 4.2 includes a formula for the latency of PT architecture. The latency of the Proposal block is the latency of floating point adder IP (Lat(add)) which adds Gaussian random number to the previous sample. The latency of the Probability Evaluation block consists of the latency of sub-density evaluation, the latency of floating point logarithm IP (Lat(log)), the latency of adders that sums log sub-densities and the latency of data arbiter (Lat(control)) which depends on the data size(n).

In Accept/Reject block, a floating point multiplier IP (Lat(mult)) to multiply log densities with the temperature of the chain, a floating point adder IP (Lat(add)) to calculate the acceptance ratio and lastly a floating point comparator IP (Lat(comp)) to compare acceptance ratio with the logarithm of a uniform random number are utilized. The latency of Exchange block is the sum of the latency of floating point multiplier IP, the latency of floating point adder IP (needed to calculate terms in the exchange ratio), the latency floating point comparator IP (Lat(comp)) and the latency of exchanger core (Lat(exchanger)).

The latency of PT architecture is totally dependent to the latency of floating point IPs and the data size(n). For some floating point IPs, such as adder/subtracter, the latency is adjustable. However, there exist a tradeoff between the latency of floating point IPs and the maximum speed of the system clock such that the latency of some floating point IPs restrict the maximum achievable clock speed in the system, so the throughput of the architecture. Therefore, an optimization procedure was applied to obtain the floating point IPs that operate at maximum clock speed and provide lowest latency which in turn costs FPGA resources. The floating point precision also changes the latency of floating point arithmetic IPs. The latency of PT architecture in terms of clock cycle for single and double floating point precision can be seen in Table 4.8. Another factor that affects the latency is the data size(n). As the data size increases, the latency of Probability Evaluation block increases since more floating point adder IPs are instantiated to sum more sub-densities. On the other side, the number of chains has no effect on the latency of the system. Figure 4.11 demonstrates the latency of single precision-PT architecture for M=128, n=16 and p=4.

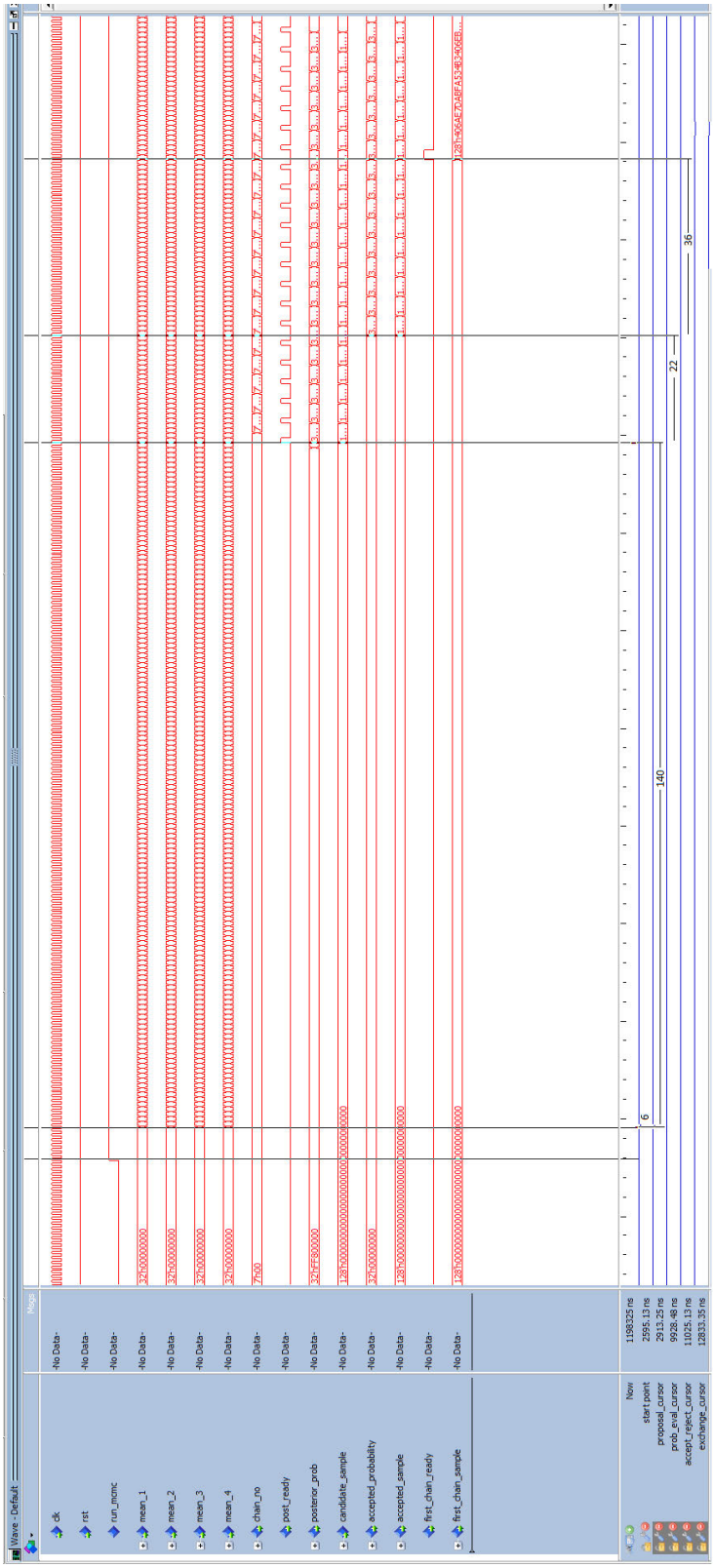

Figure 4.11: Functional simulation of successive PT stages for M=128, n=16 and p=4: The values between cursors correspond to the latency of each stage in terms of clock cycle.

Table 4.8: The latency (clock cycle) of main PT blocks for single and double floating point precision for M=128, n=16 and p=4

Block	**Latency formula**	**Single-precison PT**	**Double-precison PT**
Proposal Block	Lat(add)	6	6
Probability Evaluation Block	4Lat(mult) + Lat(exp) + 7Lat(add) + Lat(log) + Lat(control)	140	162
Accept/Reject Block	Lat(mult) + Lat(add) + Lat(comp)	22	22
Exchange Block	Lat(mult) + 2*Lat(add) + Lat(comp) + Lat(exchanger)	36	36
Total System	6Lat(mult) + 11Lat(add) + Lat(exp) + Lat(log) + 2Lat(comp) + Lat(control) + Lat(exchanger)	204	226

4.5.4 FPGA Resource Utilization of PT

The PT architecture is implemented on Intel Stratix V FPGA. Basically, Intel FGPAs consist of Adaptive Logic Modules(ALMs) that contains Adaptive Look-up Tables (ALUTs), adders and registers to implement logic functions, Digital Signal Processor (DSP) blocks and embedded memory blocks(M20K, MLAB). Table 4.9 shows the resource utilization of double precision PT architecture and its internal blocks in terms of ALMs, ALUTs , registers, DSPs and block memory bits.

Since the resource utilization of blocks is dependent to the floating point precision, using lower precision reduces the resource utilization of FPGA. Table 4.10 gives the comparison of double precision PT and single precision PT architecture in terms of the resource utilization. Moreover, the number of chains (M) and the data size (n) scale the resource utilization in a way that total block memory bits and registers are increased due to the increase in probability memory size and sample memory size.

Table 4.9 and Table 4.10 indicate that PT architecture consumes low ALMs and block memory bits. However, Digital Signal Processor (DSP) blocks are the limiting re-

source that limit the number of parallel pipelines(p) in the Probability Evaluation block. It is again important to note that higher number of pipeline(p) leads to higher throughput as described before. Therefore, in the next section, we seek and present the ways of reducing DSP usage and the ways of adding more parallel pipelines to the system in order to obtain higher throughput.

Table 4.9: FPGA resource utilization of double precision PT stages for M=128 and n=16

Block	**ALM**	**ALUTs**	**Registers**	**Block Memory Bits**	**DSP**
Proposal Block	4744 (2%)	8150	8807	32768 (0.06%)	0
Probability Evaluation Block	60352 (25.7%)	103215	91051	56896 (0.1%)	228 (89%)
Accept/Reject Block	5214 (2.2%)	9685	12831	132144 (0.25%)	12 (4.7%)
Exchange Block	10753 (4.6%)	17886	18959	25468 (0.05%)	16 (6.3%)
PT Resource Utilization	81063 (34.5%)	138936	131648	247456 (0,5%)	256 (100%)

4.5.5 Precision Analysis

Single and double floating point arithmetic precisions are commonly used precisions in MCMC applications. Also, the highest precision supported by hardware platforms(CPU, GPU and FPGA) is double precision which can be considered as infinite precision. For this reason, all performance comparisons were made between double precision - PT architecture on CPU, GPU and FPGA in previous sections. However, FPGA resource utilization and consequently the system performance are strictly dependent to the precision of the architecture. Hence, PT architectures with different precision are implemented and investigated in this section.

Single Precision-PT

Single precision PT is implemented with the architecture given in section 3.2.1. In-

stead of double precision, all operands in the system are configured to single precision floating point arithmetic. Lowering precision from double to single saves significant amount of FPGA resources, especially DPS blocks. Table 4.10 shows the area savings of single precision PT when compared to double precision PT.

Thanks to the area savings of single precision PT, 2x more parallel pipeline can be instantiated in the Probability Evaluation block and thus 2.25x higher throughput is achieved when compared to double precision PT for all (M,n) cases (see Table 4.11). Furthermore, the power consumption of PT architecture is reduced with the usage of single precision due to FPGA area savings (see Table 4.12).

Despite all these performance gains, utilizing lower precision in whole system leads to errors in the output since it samples from approximate distribution with approximate MCMC kernel. Actually, the perturbations caused by reduced precision may be lower than the output variance which are due to randomness of Monte Carlo estimates. Depending on the application needs, MCMC practitioners may also tolerate output errors due to reduced precision in some cases. However, it is possible to produce samples from true distribution by using reduced precision in wisely selected parts of the algorithm (Mixed-precision).

Mixed Precision-PT

As mentioned above, reducing precision saves FPGA resources which in turn performance boost with lost in output accuracy unfortunately. Mixed precision-PT architecture introduced by G. Mingas [52] [50] solves this problem by utilizing reduced precision blocks in the wisely selected parts of the algorithm such that probability evaluations for auxiliary chains are performed with reduced precision while the Probability Evaluation block for first chain and other generic blocks employ double precision format. Due to the fact that the target distribution of the first chain remains unchanged, mixed precision-PT produces outputs from true distribution as desired. The details of this design and how it preserves the output accuracy are described in 3.2.2 and the works [52] [50].

For this purpose, the architecture given in Figure 3.9 was implemented. As mentioned earlier, probability evaluation of the auxiliary chains are handled in reduced

precision(single precision) that gives chance to add more parallel pipelines to the system. The resource utilization of mixed precision PT architecture when *M=128* and *n=16* is shown in Table 4.10. It is clear from Table 4.10 that the same amount of DSP blocks are utilized in mixed precision-PT when compared to double precision PT even though two more parallel probability evaluation pipelines are added to system. This extra parallel pipelines leads to throughput improvements as given in Table 4.11.

Table 4.10: FPGA resource utilization for double, single and mixed precision PT in terms of when M=128 and n=16. There exist three 3 single precision pipelines and one double precision pipeline in Mixed-Precision-PT

	Single Precision PT	Double Precision PT	Mixed Precision PT
ALM	43014 (18.3%)	81063 (34.5%)	87189 (37%)
ALUTs	58386	138936	144006
Registers	79584	131648	137251
Memory Bits	129994 (0,25%)	247456 (0.5%)	283101 (0.6%)
DSP	219 (85.5%)	256 (100%)	256 (100%)
Pipeline Number	4	2	3+1

Table 4.11: Throughput(samples/sec) for double, single and mixed precision PT when M=128 and n=16

	Single Precision PT	Double Precision PT	Mixed Precision PT
Throughput	439453	195312	293225

It is important to note that Table 4.11 includes raw throughput values, which corresponds to produced MCMC samples per second. However, the exploration speed

of mixed precision - PT is lower than the double-precision PT due to the usage of reduced precision in the update stage of auxiliary chains. In other words, the less effective samples (samples that explore the distribution slowly) are generated in mixed precision design when it is compared to double precision design. Indeed, sample exchanges between first and second chains occur *13%* less frequently in mixed precision design than double precision design. This value was obtained from the average of *10* free runs of both strategies.

Apart from the throughput, the power consumption are also altered with mixed precision architecture. Due to the minor increase in total resource utilization, power consumption of mixed precision PT is higher than the double precision-PT (see Table 4.12). It is important to note that although resource utilization of reduced precision probability evaluation pipeline is less than double precision probability evaluation pipeline, total resource utilization of mixed design is higher due to the extra pipelines in mixed precision design.

Table 4.12: Power consumption for double, single and mixed precision PT when M=128 and n=16

	Single Precision PT	Double Precision PT	Mixed Precision PT
Power (W)	5.38	5.76	5.97

Since the lowest floating point arithmetic precision that can be used in Intel FPGAs is single precision, auxiliary chain calculations are performed with single precision in this work. On the other side, Xilinx FPGAs have DSP blocks which can be configured to precision formats which are lower than single precision. Hence, it is possible to design more optimized mixed precision PT architectures that offer greater speedups in Xilinx FPGAs as G. Mingas [52] [50] achieved.

4.5.6 Memory Analysis

When large scales of data sets or many chains MCMC problems are targeted, memory can become the bottleneck of the FPGA system. Due to the fact that on-chip memories of FPGAs can be accessed at every clock cycle, memory bandwidth is not an issue

for PT architectures that use on-chip FPGA memories. On the other hand, the size of memory inside the FPGA is limited (6.5 MB on-chip memory for Intel Stratix V - 5SGXEA7 device). Although there exist sufficient on-chip memory inside the Stratix V FPGA family for all (M,n) cases investigated in this thesis, it is not possible to implement all of the memories used in PT architecture (Data Memory, Sample Memory and Probability Memory) into the FPGA for large values of chain numbers and big data sets. In that case, some of these memories should be implemented externally.

When using external memories in PT architecture, the latency of data transfers between processing blocks and external memory and limited bandwidth can degrade the system performance. Actually, these are not issue for external Probability Memory or external Sample Memory. Since PT architecture accesses to Probability Memory and Sample Memory $2 * M$ times for each PT iteration (M times for reading and M times for updating). Numerically, as a worst case, PT architecture with $M = 32768$, $n = 32$ accomplishes *380* iterations in one second, which needs *760* MB/sec total throughput with external Sample Memory and *190* MB/sec total throughput with external Probability Memory. For more realistic scenario, PT architecture with $M = 128$, $n = 32768$ requires *0.76* MB/sec and *0.19* MB/sec total throughput with external Sample Memory and external Probability Memory respectively. When compared to throughput of the DDR3 memory (6.4 GB/sec) in custom FPGA board given in Figure 4.1, these are easily achievable requirements for external memory throughput.

On the other hand, memory bandwidth and memory efficiency issues can be faced with depending on the size of the data sets if the Data Memory is moved off FPGA. Indeed, the number of accesses to Data Memory in one PT iteration equals to $M * n$ for the mixture model implemented in this thesis. For example, PT architecture with $M = 128$, $n = 327684$ accomplishes *95* iterations in one second, which requires *3.13* GB/sec external memory bandwidth. Despite the fact that this is achievable throughput value for DDR3 memory interface used in custom FPGA board, the need for continuous feeding of Probability Evaluation block imposes a big challenge on our MCMC application. In other words, the mandatory latency of memory accesses leads to pipeline stall in probability evaluation and so performance degrades.

In order to avoid pipeline stalls, the architecture shown in Figure 4.12 is implemented.

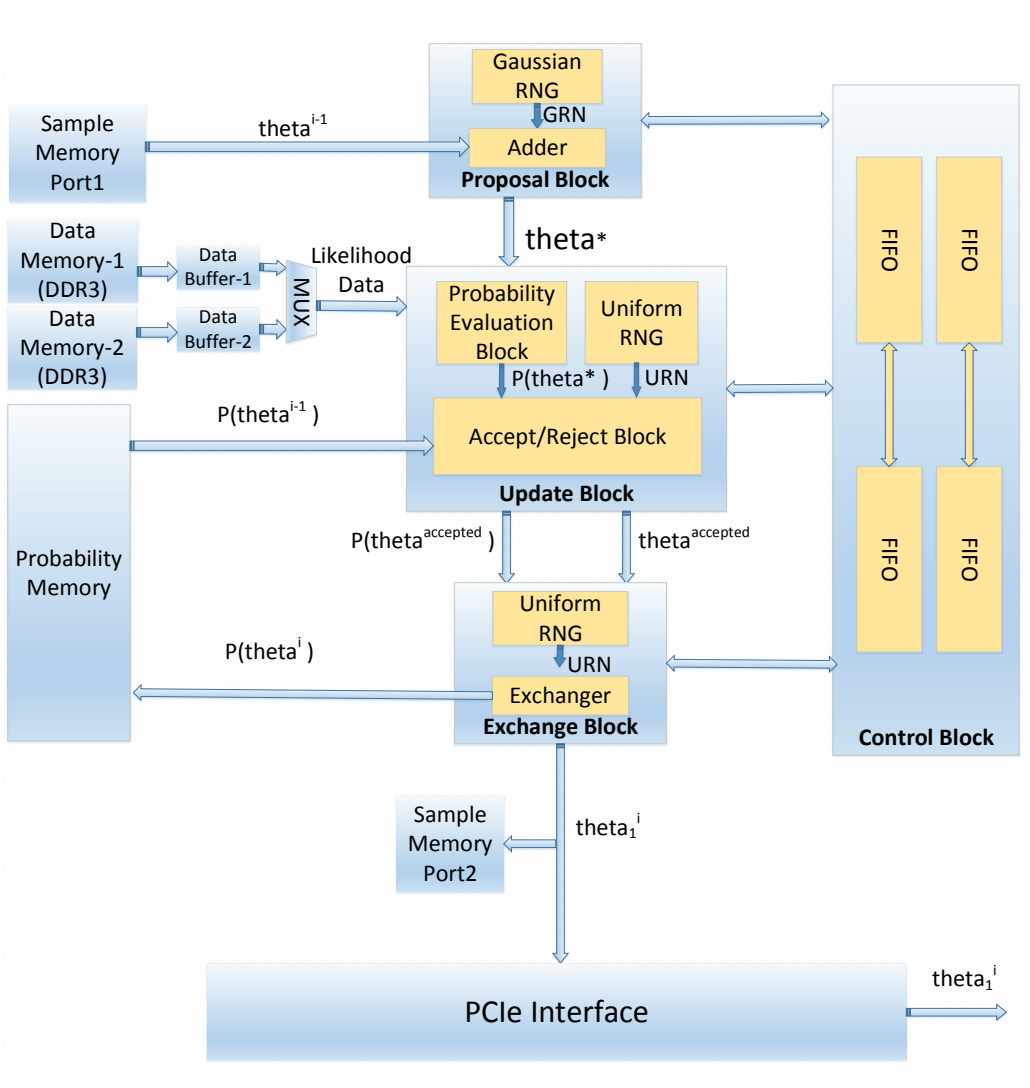

Figure 4.12: Double precision PT architecture with external Data memories.

In this architecture, the whole data is split into two equal groups such that a half of data is stored in first DDR3 memory and the other half of data is stored in second DDR3 memory. The common approach of data partitioning methods in MCMC literature is splitting data into subgroups and combining the results after processing each group separately (in separate core, processor or hardware). However, PT architecture with two DDR3 data memories splits data into two groups and processes these two groups in the same processing unit (Probability Evaluation block) respectively.

The flow of data from external memories to Probability Evaluation block is illustrated in Figure 4.13. While the second DDR3 memory is accumulating the second buffer

with some part of its content (it depends on whole data size and burst read length), the Probability Evaluation block is fed by the first buffer which was filled before with some part of data by first DDR3 memory. This duration is called *Phase-1* in which first DDR3 memory gets ready for next access (no operation) and read operation is performed for the second DDR3 memory. The Second phase (*Phase-2*) starts immediately after the end of *Phase-1*. In this phase, the first DDR3 memory accumulates the first buffer with some part of its data and the Probability Evaluation block is fed by the second buffer which was filled in *Phase-1* by the second DDR3 memory. *Phase-1* and *Phase-2* continue in order till all data is transferred to the Probability Evaluation block. All these successive phases are repeated without a break for all chains until the end of MCMC run. It is important to note that Probability Evaluation block can be fed continuously at each clock cycle since there always exist a filled data buffer in the system.

The Probability Evaluation block demands *128* bits wide data at every clock (*200* MHz) so that *3.13* GB/sec throughput is needed for $M = 128$, $n = 327684$ case. On the other side, since each 64-bit DDR3 memory is operated with *400* MHz clock rate, provided throughput by external memories is almost four times of required throughput. Besides, with double buffer mechanism, the negative effect of latency imposed by data transfer is removed so parallel pipelines run without stall.

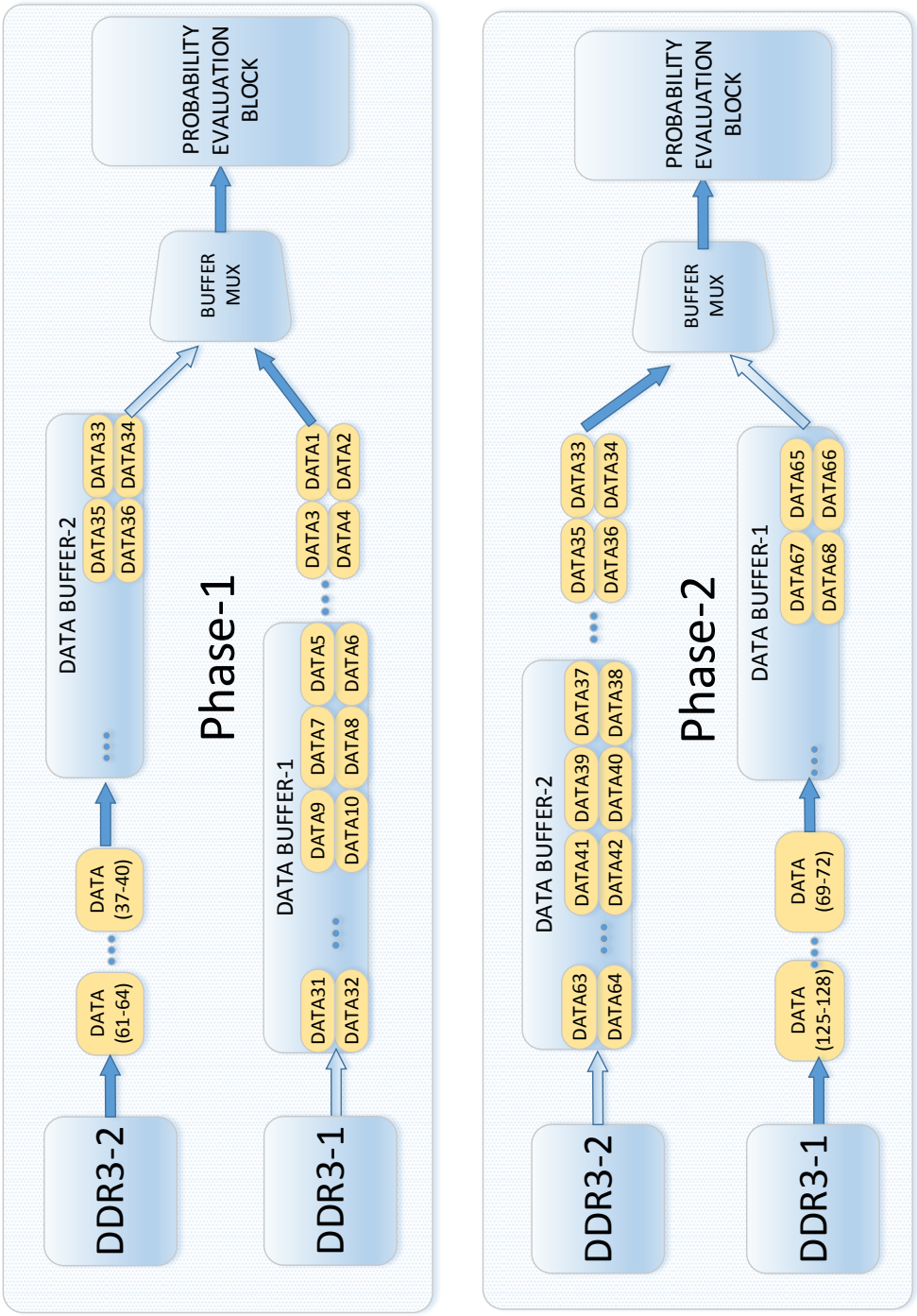

Figure 4.13: Data flow from external data memory to the Probability Evaluation Block: Dark blue arrow indicates active flow in related phases. Light blue arrow means no-flow.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In recent years, Markov Chain Monte Carlo has become the mainstream tool to investigate the massive data sets when Bayesian models are employed. Despite its ability to analyze multi modal and high dimensional distributions, long execution times of MCMC algorithms can restrict the usage of MCMC in the fields of applied statistics. The reasons for excessive run times in MCMC applications are mainly large data sets and computationally intensive MCMC algorithms. Hence, the interest in accelerating MCMC methods has gained momentum recently. The use of multi-core CPUs, GPUs and FPGAs to increase sampling throughput, data partitioning methods and subsampling methods to deal with massive data sets are the trend topics in the literature for MCMC acceleration. In this thesis, we also focused on these approaches and combined them to design a MCMC accelerator for Bayesian inference on multi-modal posteriors.

To this end, a FPGA based hardware architecture for the population based MCMC is implemented by taking G. Mingas' works [50–52] as reference. This architecture exploits all the capabilities of FPGAs such as deep pipelining, mixed arithmetic precision, massive parallelism and fast inter-circuit communication. Speedups of 24x to 4x over the respective sequential CPU and GPU implementations were achieved using the proposed double precision design on FPGAs on a Gaussian mixture model case study. From this aspect, these results verify the achievements of G. Mingas' studies [50–52]. Also, thanks to the mixed precision method which uses reduced precision in the calculations of auxiliary chains, further improvements on the system's performance are achieved without sacrificing the sampling accuracy. The main limitation of mixed precision architecture is related to FPGA brand used in thesis. The

lowest floating point precision supported by Intel FPGAs is single precision. Probably, with the use of lower precisions than single precision for the calculations of auxiliary chains, further performance improvements or optimizations (mixing speed is altered) are possible.

In contrast to closely related works, this thesis also investigates the memory related performance losses in FPGA-based MCMC applications. All the performance results presented above are in the situation where all data sets fit into on-chip FPGA-memory. However, when the problems with massive data sets are targeted, the need for the usage of external memory arises. In this case, due to the limited memory bandwidth or the latencies between external memory accesses, the processing unit cannot be fed with data in every clock cycle, which leads to break in the pipeline and thus loss of performance. To solve this problem, a FPGA architecture with novel external memory access strategy is presented. This architecture employs data partitioning method in order to reduce memory accesses for an external memory. While one external memory feeds its data buffer, the other external memory gets ready for the next access. This process is repeated recursively until all data is extracted from the external memories. The proposed design keeps all speedups achieved with FPGA based PT architecture with on-chip memory.

Overall, the main contribution of this thesis is that we deal with both the memory and computation bound problems for MCMC in the big data regime using FPGAs. This is accomplished by exploiting the unique capabilities of FPGAs such as highly-parallel and pipelined architecture and custom precision support. Besides, with the proposed MCMC accelerator improved via novel memory access strategy, the intractable or computationally intensive tasks in Bayesian inference can be solved.

This work can be improved by targeting several approaches that exist in the MCMC literature. Employing sub-sampling algorithms and heterogeneous computing devices in MCMC applications or targeting other sampling methods are the possible future researches for this work. Despite the enormous efforts, in some cases, it is not possible to deal with whole data sets due to the memory or execution time restrictions. For these cases, the algorithms based on data sub-sampling can be employed so that instead of the full data set, a random sub-set of the data are processed in every MCMC

iterations. Firefly Monte Carlo method [47] is a successful example of these approaches, which has been already mapped to FPGA platform with custom precision techniques [45].

Since utilizing single device may not be enough to deal with the big data problems, the use of multiple FPGAs or heterogeneous computing devices such as GPUs and FPGAs may become more common in the future. Via this approach, the memory bottlenecks can also be eliminated by partitioning data into sub groups. In contrast to our data partitioning strategy, the partitioned sub-groups are processed in different computing devices in parallel which leads to further improvements on sampling throughput. In order to avoid the sampling bias or to reduce the sampling bias, a good strategy for combining the results of sub groups can be searched as future work.

In this thesis, a population based Markov Chain Monte Carlo method, Parallel Tempering, is accelerated via hardware accelerator. However, in the literature, there exist many MCMC methods such as Hamiltonian Monte Carlo, Gibbs Sampling, Multiple Try Metropolis, Slice Sampling and Adaptive MCMC, which are suitable candidates for FPGA acceleration. Apart form MCMC methods, alternative sampling methods like Sequential Monte Carlo (SMC) for dynamic models can also be mapped into FPGAs to extend the use of FPGAs in Bayesian inference. The inherent parallelism in the steps of SMC method can be exploited by FPGA to speedup the algorithm. Also, different resampling algorithms can be mapped into parallel FPGA architecture to extend SMC usage in mobile systems via reconfigurable hardware.

REFERENCES

[1] Altera early power estimator tool. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_epe.pdf.

[2] Ltc2978 - octal digital power supply manager with eeprom. http://www.linear.com/product/LTC2978.

[3] Stratix si development kit. https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-transceiver-si-stratix-v.html.

[4] *Handbook of Monte Carlo Methods*. Chapman and Hall/CRC, 2011.

[5] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis. Exploring fpgas for accelerating the phylogenetic likelihood function. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, May 2009.

[6] E. Angelino, M. J. Johnson, and R. P. Adams. Patterns of scalable bayesian inference. *Foundations and Trends in Machine Learning*, 9:119–247, 2016.

[7] N. B. Asadi, T. H. Y. Meng, and W. H. Wong. Reconfigurable computing for learning bayesian networks. In *FPGA*, 2008.

[8] R. Bardenet, A. Doucet, and C. C. Holmes. Towards scaling up markov chain monte carlo: an adaptive subsampling approach. In *ICML*, 2014.

[9] R. Bardenet, A. Doucet, and C. C. Holmes. On markov chain monte carlo methods for tall data. *Journal of Machine Learning Research*, 18:47:1–47:43, 2017.

[10] A. Beam, S. Ghosh, and J. Doyle. Fast hamiltonian monte carlo using gpu computing. *Journal of Computational and Graphical Statistics*, 1402, 02 2014.

[11] S. P. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998.

[12] J. M. R. Byrd, S. A. Jarvis, and A. H. Bhalerao. Reducing the run-time of mcmc programs by multithreading on smp architectures. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[13] T. A. Castoe, T. Doan, and C. L. Parkinson. Data partitions and complex models in bayesian analysis: the phylogeny of gymnophthalmid lizards. *Systematic biology*, 53 3:448–69, 2004.

[14] R. C. C. Cheung, D. U. Lee, W. Luk, and J. D. Villasenor. Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(8):952–962, Aug 2007.

[15] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. W. Leong, and D. B. Thomas. A mixed precision monte carlo methodology for reconfigurable accelerator systems. In *FPGA*, 2012.

[16] A. Doucet, N. De Freitas, and N. Gordon. *Sequential Monte Carlo methods in practice*. Springer New York ; London, 2001.

[17] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10, 04 2003.

[18] D. J. Earl and M. W. Deem. Optimal allocation of replicas to processors in parallel tempering simulations. *The Journal of Physical Chemistry B*, 108(21):6844–6849, 2004.

[19] D. J. Earl and M. W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics (Incorporating Faraday Transactions)*, 7:3910, 2005.

[20] D. C. K. Fan. *Bayesian Inference of Vascular Structure from Retinal Images*. PhD thesis, University of Warwick, 2006.

[21] M. Fielding, D. J. Nott, and S.-Y. Liong. Efficient mcmc schemes for computationally expensive posterior distributions. 53:16–28, 02 2011.

[22] T. Flury and N. Shephard. Bayesian inference based only on simulated likelihood: Particle filter analysis of dynamic economic models. *Econometric Theory*, 1:1–24, 01 2011.

[23] A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992.

[24] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov 1984.

[25] J. Geweke. Evaluating the accuracy of sampling-based approaches to calculating posterior moments. In J. M. Bernardo, J. Berger, A. P. Dawid, and J. F. M. Smith, editors, *Bayesian Statistics 4*, pages 169–193. Oxford University Press, Oxford, 1992.

[26] C. J. Geyer. Markov chain monte carlo maximum likelihood. In *Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface*, pages 156–163, 1991.

[27] Z. Ghahramani. Bayesian non-parametrics and the probabilistic approach to modelling. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 371(1984):20110553, February 2013.

[28] W. Gilks and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC, 1996.

[29] J. Gross, W. Janke, and M. Bachmann. Massively parallelized replica-exchange simulations of polymers on GPUs. *Computer Physics Communications*, 182:1638–1644, Aug. 2011.

[30] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Chapman and Hall, 1964.

[31] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[32] Z. Huang and A. Gelman. Sampling for bayesian computation with large datasets. 2003.

[33] A. Jasra, D. A. Stephens, and C. C. Holmes. On population-based simulation for static inference. *Statistics and Computing*, 17:263–279, 2007.

[34] R. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of basic Engineering*, 82:35–45, 01 1960.

[35] S. L. Scott, A. W. Blocker, F. V. Bonassi, H. A. Chipman, E. George, and R. McCulloch. Bayes and big data: The consensus monte carlo algorithm. 11:1–11, 02 2016.

[36] P. L'Ecuyer. Maximally equidistributed combined tausworthe generators. *Math. Comput.*, 65(213):203–213, Jan. 1996.

[37] A. Lee, C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *ArXiv e-prints*, May 2009.

[38] D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. W. Leong. A hardware gaussian noise generator using the wallace method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(8):911–920, Aug 2005.

[39] D. U. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong. A hardware gaussian noise generator using the box-muller method and its error analysis. *IEEE Transactions on Computers*, 55(6):659–671, June 2006.

[40] Y. Li, M. Mascagni, and A. Gorin. A decentralized parallel implementation for parallel tempering algorithm. *Parallel Computing*, 35(5):269 – 283, 2009.

[41] F. Liang, C. Liu, and R. J. Caroll. *Advanced Markov Chain Monte Carlo Methods*. Wiley, 2010.

[42] M. Lin, I. Lebedev, and J. Wawrzynek. High-throughput bayesian computing machine with reconfigurable hardware. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, pages 73–82, New York, NY, USA, 2010. ACM.

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

[44] J. S. Liu, F. Liang, and W. H. Wong. The multiple-try method and local optimization in metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000.

[45] S. Liu. *Acceleration of MCMC-based Algorithms Using Reconfigurable Logic*. PhD thesis, Imperial College London, 2017.

[46] M. M. Tibbits, M. Haran, and J. C. Liechty. Parallel multivariate slice sampling. *Statistics and Computing*, 21:415–430, 07 2011.

[47] D. Maclaurin and R. P. Adams. Firefly Monte Carlo: Exact MCMC with Subsets of Data. *ArXiv e-prints*, Mar. 2014.

[48] J. S. Malik, J. N. Malik, A. Hemani, and N. D. Gohar. Generating high tail accuracy gaussian random numbers in hardware using central limit theorem. In *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*, pages 60–65, Oct 2011.

[49] G. Mingas. *Algorithms and architectures for MCMC acceleration in FPGAs*. PhD thesis, Imperial College London, 2015.

[50] G. Mingas and C. S. Bouganis. A custom precision based architecture for accelerating parallel tempering mcmc on fpgas without introducing sampling error. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 153–156, April 2012.

[51] G. Mingas and C.-S. Bouganis. Parallel tempering mcmc acceleration using reconfigurable hardware. In *Proceedings of the 8th International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, ARC'12, pages 227–238, Berlin, Heidelberg, 2012. Springer-Verlag.

[52] G. Mingas and C. S. Bouganis. Population-based mcmc on multi-core cpus, gpus and fpgas. *IEEE Transactions on Computers*, 65(4):1283–1296, April 2016.

[53] S. Minsker, S. Srivastava, L. Lin, and D. Dunson. Scalable and robust bayesian inference via the median posterior, 06 2014.

[54] I. Murray. *Advances in Markov chain Monte Carlo methods*. PhD thesis, University of London, 2007.

[55] W. Neiswanger, C. Wang, and E. Xing. Asymptotically exact, embarrassingly parallel mcmc. *Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014*, 11 2013.

[56] M. NS, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. J. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 01 1953.

[57] M. Quiroz, R. Kohn, M. Villani, and M.-N. Tran. Speeding Up MCMC by Efficient Data Subsampling. *ArXiv e-prints*, Apr. 2014.

[58] W. R. Gilks and P. Wild. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, 41:337–348, 01 1992.

[59] O. J. L. Rackham, P. Dellaportas, E. Petretto, and L. Bottolo. Wgbssuite: simulating whole-genome bisulphite sequencing data and benchmarking differential dna methylation analysis tools. In *Bioinformatics*, 2015.

[60] G. Reinert. Markov chain monte carlo and applied bayesian statistics. http://www.stats.ox.ac.uk/ reinert/mcmc/mcmc07.pdf. Accessed on 2007.

[61] C. Robert and G. Casella. A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data. *ArXiv e-prints*, Aug. 2008.

[62] J. S. Rosenthal. Parallel computing and monte carlo algorithms. *Far East Journal of Theoretical Statistics*, 4:207–236, 1999.

[63] A. Silva. cudabayesreg: Bayesian computation in cuda. *The R Journal*, 2, 12 2010.

[64] M. A. Suchard, Q. Wang, C. Chan, J. Frelinger, A. Cron, and M. West. Understanding gpu programming for statistical computation: Studies in massively parallel massive mixtures. *Journal of computational and graphical statistics : a joint publication of American Statistical Association, Institute of Mathematical Statistics, Interface Foundation of North America*, 19 2:419–438, 2010.

[65] D. B. Thomas. Fpga gaussian random number generators with guaranteed statistical accuracy. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 149–156, May 2014.

[66] D. B. Thomas, L. Howes, and W. Luk. A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, pages 63–72, New York, NY, USA, 2009. ACM.

[67] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor. Gaussian random number generators. *ACM Comput. Surv.*, 39(4), Nov. 2007.

[68] E. Thonnes, A. Bhalerao, W. Kendall, and R. Wilson. A bayesian approach to inferring vascular tree structure from 2d imagery. In *Proceedings. International Conference on Image Processing*, volume 2, pages II–937–II–940 vol.2, 2002.

[69] X. Tian and C. Bouganis. A run-time adaptive fpga architecture for monte carlo simulations. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 116–122, Sep. 2011.

[70] L. Tierney and A. Mira. Some adaptive monte carlo methods for bayesian inference. *Statistics in medicine*, 18:2507–15, 09 1999.

[71] R. Trotta. Bayes in the sky: Bayesian inference and model selection in cosmology. *Contemp. Phys.*, 49:71–104, 2008.

[72] J. Wang and R. H. Swendsen. Replica Monte Carlo Simulation (Revisited). *Progress of Theoretical Physics Supplement*, 157:317–323, 2005.

[73] X. Wang and D. B. Dunson. Parallelizing MCMC via Weierstrass Sampler. *ArXiv e-prints*, Dec. 2013.

[74] S. P. Whiley, Mattand Wilson. Parallel algorithms for markov chain monte carlo methods in latent spatial gaussian models. *Statistics and Computing*, 14(3):171–179, Aug 2004.

[75] D. Wilkinson. Parallel bayesian computation, 12 2005.

[76] K. Wreczycka, A. Gosdschan, D. Yusuf, B. A. Grüning, Y. Assenov, and A. Akalin. Strategies for analyzing bisulfite sequencing data. *Journal of biotechnology*, 261:105–115, 2017.

[77] X.-L. Wu, T. Beissinger, S. Bauck, B. Woodward, G. Rosa, K. A Weigel, N. de Leon Gatti, and D. Gianola. A primer on high-throughput computing for genomic selection. 2:4, 02 2011.

[78] X.-L. Wu, C. Sun, T. Beissinger, G. Rosa, K. A Weigel, N. de Leon Gatti, and D. Gianola. Parallel markov chain monte carlo - bridging the gap to high-performance bayesian computation in animal breeding and genetics. 44:29, 09 2012.

[79] S. Yıldırım. Monte carlo: Simulation methods for statistical inference. `http://people.sabanciuniv.edu/sinanyildirim/Lecture_notes.pdf`.

[80] G. Zhang, P. H. W. Leong, D.-U. Lee, J. D. Villasenor, R. C. C. Cheung, and W. Luk. Ziggurat-based hardware gaussian random number generator. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 275–280, Aug 2005.

[81] Y. Zhao, J. Kang, and Q. Long. Bayesian multiresolution variable selection for ultra-high dimensional neuroimaging data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(2):537–550, March 2018.

[82] S. Zierke and J. Bakos. Fpga acceleration of the phylogenetic likelihood function for bayesian mcmc inference methods. *BMC bioinformatics*, 11:184, 04 2010.