MPT: Multiple Parallel Tempering for High-Throughput MCMC Samplers

Morteza Hosseini
Computer Science and Electrical Engineering Department
University of Maryland, Baltimore County
Maryland, USA
hs10@umbc.edu

Rashidul Islam
Computer Science and Electrical Engineering Department
University of Maryland, Baltimore County
Maryland, USA
dr97531@umbc.edu

Lahir Marni
Computer Science and Electrical Engineering Department
University of Maryland, Baltimore County
Maryland, USA
mlahir1@umbc.edu

Tinoosh Mohsenin
Computer Science and Electrical Engineering Department
University of Maryland, Baltimore County
Maryland, USA
tinoosh@umbc.edu

***Abstract*—This paper proposes "Multiple Parallel Tempering" (MPT) as a class of Markov Chain Monte Carlo (MCMC) algorithm for high-throughput hardware implementations. MCMC algorithms are used to generate samples from target probability densities and are commonly employed in stochastic processing techniques such as Bayesian inference, and maximum likelihood estimation, in which computing large amount of data in real-time with high-throughput samplers is critical. For high-dimensional and multi-modal probability densities, Parallel Tempering (PT) MCMC has proven to have superior mixing and higher convergence to the target distribution as compared to other popular MCMC algorithms such as Metropolis-Hastings (MH). MPT algorithm, proposed in this paper, imposes a new integer parameter, D, to the original algorithm of PT. Such modification changes one MCMC sampler into multiple independent kernels that alternatively generate their set of samples one after another. Our experimental results on Gaussian mixture models show that for large values of D, the auto-correlation function of the proposed MPT falls comparably to that of a PT sampler. A fully configurable and pipelined hardware accelerator for the proposed MPT, as well as PT are designed in Verilog HDL and implemented on FPGA. The two algorithms are also written in C language and evaluated on Multi-core CPU from the TX2 SoC. Our implementation results indicate that by selecting an appropriate value for D in our case study the sampling throughput of the MPT can raise from 4.5 Msps in PT to 135 Msps on average, an amount near maximum achievable frequency of the target FPGA, which is about 1470$\times$ higher than when implementing on fully exploited Multi-core CPU.**

***Index Terms*—MCMC, Mixture model, Parallel Tempering, High-Throughput Sampler, Hardware Accelerator**

I. Introduction

MCMC is one of the essential tools for stochastic processing techniques and is mainly used for solving Bayesian inference problems found in several fields such as modern machine learning [1], probabilistic mixture models [2], genetics [3] and prediction of time series events [4][5]. Stochastic estimators such as MCMC are regularly becoming methods of choice as deterministic numerical techniques are inefficient for large dimension data. In probabilistic inference problem, we often need to compute expectation of a function $g(x)$ for a random variable x by solving the integral:

$$E[g(x)] = \int g(x)p(x)dx \tag{1}$$

where $p(x)$ is the target probability distribution. MCMC consists of two steps: Monte Carlo, that indicates that the process is based on random events, and Markov Chain, that implies the sequence of the events are causal, which means every current event is a result of previous event (or events). In Monte Carlo integration, a set of samples, $x^{(t)}$ where t=1...N are drawn out from a target distribution to approximate an expected value as calculated below:

$$\hat{E}[g(x)] = \frac{1}{N}\sum_{i=1}^{N} g(x^{(i)}) \tag{2}$$

Generally, the accuracy of the MCMC sampler can be improved by increasing number of samples, "N". However, there is no unique way to measure the convergence of MCMC samples [6]. Common techniques include checking the auto-correlation of the samples from the underlying sampler, or using several independent sampling kernels initialized to different states that should eventually generate similar results once converged [6][2].

Markov chain is an iterative stochastic process which is initialized to a starting state, $x^{(1)}$, and determines the next states based on a transition probability, where the transition probability can be described by a 1^{st} order Markov chain as shown in equation 3:

$$p(x^{(i)}|x^{(i-1)}, x^{(i-2)}, ..., x^{(1)}) = p(x^{(i)}|x^{(i-1)}) \tag{3}$$

MCMC contains iterative interdependent kernels which makes hardware acceleration challenging. To overcome this challenge, FPGA-based research has focused on parallel implementation of Markov chains to achieve acceleration [7][8][9][10][11][12][13].

In this paper, we propose "Multiple Parallel Tempering" (MPT) MCMC algorithm which is a special case of D^{th} order Markov chain and probability transition of the MPT can be described by:

$$p(x^{(i)}|x^{(i-1)}, ..., x^{(i-D)}, ..., x^{(1)}) = p(x^{(i)}|x^{(i-D)}) \quad (4)$$

where D is an integer greater or equal to 1. Equation 4 is representable by D independent partitioned subsets of the set of variable x, and can be described by D independent 1^{st} order Markov chains. The bottleneck of the sequential behavior when implementing an MCMC algorithm on hardware, can be resolved by changing a 1^{st} order MCMC algorithm to the D^{th} order MCMC algorithm, with an appropriate selection of the parameter D. This modification is inspired from hardware designer's perspective, with the intention of achieving an ultra high-throughput sampler. In this paper, we evaluate the impact of imposing such parameter in the algorithm, validate the new algorithm by showing the auto-correlation and expected value graphs, and implement the hardware of our proposed MPT MCMC as well as the traditional MCMC algorithms, Parallel Tempering (PT) and Metropolis Hastings (MH) on the FPGA.

Main contributions to this paper include:

- Two most popular MCMC algorithms are studied in terms of application, limitation, and complexity.
- A modification in PT MCMC algorithm is proposed, named MPT, and the mixing and convergence behavior of the new algorithm over a given $p(x)$ is investigated with respect to the number of samples.
- A high-throughput fully scalable hardware accelerator of the proposed MPT along with the original PT sampler is designed and implemented on Artix-7 Xilinx FPGA for chain numbers of 1, 2, 4 and 8 and is compared with PT.
- The original PT and the proposed MPT algorithms are implemented on the Multi-core CPU and the result of which is compared with the FPGA implementations.

II. Multiple Parallel Tempering (MPT)

This section briefly introduces two popular MCMC algorithms, Metropolis-Hastings (MH) and Parallel Tempering (PT), and will propose Multiple Parallel Tempering (MPT) algorithm from which both former algorithms are deducible.

MH algorithm, one of the most basic variations of MCMC algorithms, initiates a random sample from a probability density space, and thereafter continues a sequence of random steps to generate other samples that will eventually approximate the given probability distribution. Each random step at every iteration that generates the next sample is taken based on an accept/reject probability criteria given in equation 5, where x^* sample is generated by using another proposal distribution and sample $x^{(t-1)}$ is generated in the $(t-1)^{th}$ iteration.

$$\alpha = min(1, \frac{p(x^*)}{p(x^{(t-1)})}) \quad (5)$$

The proposal distribution, $q(x|x^{(t-1)})$, is a symmetrical pdf that decides the next random jump from the current state, and is usually a Gaussian distribution (Line 9 in Algorithm 1).

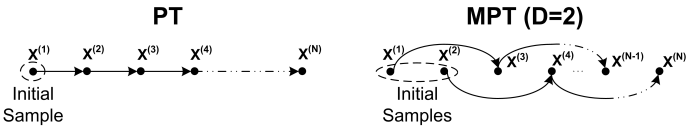

Fig. 1. Proposed sampling technique for MPT: In PT, next sample can only be derived when previous sample is available, but in MPT every next sample is generated upon the previous D^{th} sample. In the figure, for D=2, two independant kernels are illustrated that alternatively generate and interleave their set of chain samples.

For multi-modal distributions, MH gets trapped in isolated modes [2] and suffers from slow convergence. To overcome this drawback, PT was proposed by [14] that uses parallel Markov chains in which each chain i samples from a tempered version of the original distribution. A tempered version of $p(x)$ is denoted by $p_i(x)$ and formulated by equation 6:

$$p_i(x) = p(x)^{1/T_i}; i \in \{1 \dots M\} \quad (6)$$

where M is the number of the chains and T_i represents the temperature of the i^{th} chain. The 1^{st} chain is the principal chain whose temperature is equal to 1 and from which the final samples are extracted. Temperature selection for other chains depends on the target distribution and the algorithm configuration. In this work, however, we use a geometric progression of temperatures suggested by [14]. It has been shown in [15] that choosing temperatures for every two consecutive chains i and j such that $\frac{T_i}{T_j}$ is a constant value results in equal acceptance ratios between chains. In PT algorithm, $p_1(x)$ is the main target distribution and the remaining distributions are the tempered versions of $p(x)$. After samples are generated from chains in parallel, samples from different pairs of chains are swapped with a probability condition; the larger probability of a proposed sample of a tempered chain, the more likely it is swapped with a colder chain. The objective of this swap is to push samples, that are more likely meeting isolated modes in the $p_1(x)$, from the high temperature chain to the coldest chain, thereby mixing and redemption from getting stuck in a particular mode. PT with one chain is the same as MH.

MPT, proposed in this paper, is devised by adding a new parameter, D, to the original PT algorithm. In MPT the next sample, instead of being contingent upon the current sample, is derived from the previous D^{th} sample as shown in the Figure 1. MPT with D equal to 1 is the same as PT. Algorithm 1 shows the pseudo-code of the proposed MPT. Throughout the rest of this paper, we denote PT-m and MPT-m as PT and MPT that employ m number of chains.

III. Case Study: Gaussian Mixture Model

In order to evaluate and compare the three sampling algorithms, we investigate their sampling performance over i.i.d. multivariate multi-modal Gaussian mixture models (GMM) which are weighted summation of Gaussian densities. This model is used to establish a likelihood function, which is used in a Bayesian Inference problem, where the parameters of posterior is tuned upon the likelihood and prior density

Algorithm 1 Multiple Parallel Tempering (MPT)

Inputs:
Density $p(x)$, chain temperatures $T_{2:m}$, parameter D, initial samples ($x_{1:M}^{(1)}$ ... $x_{1:M}^{(D)}$), mean μ, proposal variances ($\sigma_1^2 : M$), number of the required samples (N).
Output:
Samples taken from $\mathbf{x}_1$
Sampling:
for $i = D + 1$ to N **do**
  Sample update:
  for $j = 2$ to M **do**
    $\mathbf{x}_j^* \leftarrow \mathbf{x}_j^{(i-D)}$ + $\sigma_j N(0,1)$
    Do $\mathbf{x}_j^{(i)} \leftarrow \mathbf{x}_j^*$ with probability:
    $accept(\mathbf{x}_j^{(i-D)}, \mathbf{x}_j^*)$ = $min(1, (\frac{p(\mathbf{x}_j^*)}{p(\mathbf{x}_j^{(i-D)})})^{\frac{1}{T_j}})$
    Otherwise do $\mathbf{x}_j^{(i)} \leftarrow \mathbf{x}_j^{(i-D)}$
  end for
  Sample Exchange:
  Choose chain pairs in a binary fashion: ((1,2),(3,4), ...) in stage 1, chain pairs ((1,3),(5,7),...) in stage 2, ..., and chain pair (1, $\frac{M}{2} + 1$) in stage $log(M)$.
  for every chosen pair (u,v) in consecutive stages **do**
    Swap samples $\mathbf{x}_u^{(i)}$ and $\mathbf{x}_v^{(i)}$ with probability:
    $swap(\mathbf{x}_u^{(i)}, \mathbf{x}_v^{(i)})$ = $min(1, (\frac{p(\mathbf{x}_v^{(i)})}{p(\mathbf{x}_u^{(i)})})^{(\frac{1}{T_u} - \frac{1}{T_v})})$
  end for
end for

functions, or in Markov random fields where the exact maximum likelihood estimation (MLE) is impossible. A likelihood function that employs i.i.d GMMs can be generically denoted by equation 7:

$$p(\mathbf{x}) = \prod_{i=1}^{m} (\sum_{j=1}^{k} (\mathbf{w}_{i,j} \exp^{-\frac{(x_i - u_{i,j})^2}{2\sigma_{i,j}{}^2}})) \tag{7}$$

where, m, k, σ and μ are respectively the number of dimensions, number of mixture modes per dimension, variances, and means of the GMM. w_j is the weight of the subclass i, $0 < w_j < 1$ for all subclasses and $\sum_{j=1}^{k} w_{i,j} = 1$ given any i. The model employed for our work to compare the three algorithms adapts to $k = 4$, $m = 4$, $\sigma_{1,} = \sigma_{2,} = \sigma_{3,} = \sigma_{4,} = (0.7, 0.7, 0.7, 0.7)$ and $\mu_{1,} = \mu_{2,} = \mu_{3,} = \mu_{4,} = (10, 30, 50, 70)$ which is a 4 dimensional i.i.d. GMM with 4 modes per dimension.

We performed several experiments on the given case study to display how the three algorithms mix generated samples from the GMM. Figure 2 shows a traceplot that represents the mixing properties of the three MCMC samplers. MH sampler gets trapped in isolated modes of the distribution, thus indicating poor mixing. PT shows good mixing, and although switching between different modes is quite frequent, it is evident that the random walking from the sampler gets temporarily trapped, but eventually hops to other modes more frequently. MPT with $D = 51$, on the other hand, displays more mixing than both of the latter as a result of having 51 kernels that independently sample from the target probability density one after another.

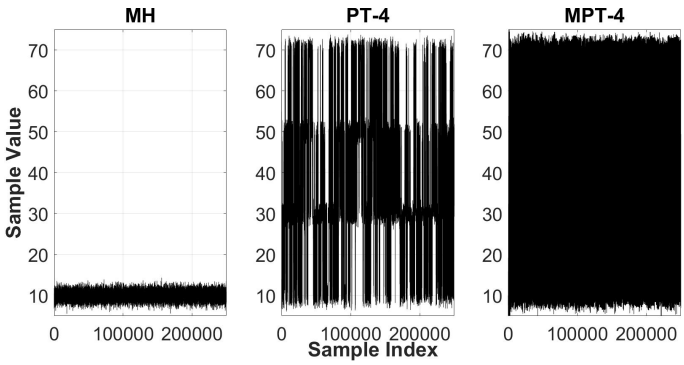

Fig. 2. Trace-plot of samples for one variable of 4D distribution with 4 modes with mean values equal to 10,30, 50, and 70. MH gets trapped in a mode, PT-4 can sample from all four modes, but the mixing quality is low and MPT-4 with D=51 represents better mixing properties to sample from all the 4 modes.

We also investigate the convergence to the target distribution by evaluating auto-correlation between drawn samples from PT and MPT algorithms. The auto-correlation function [16] is a figure of merit that measures correlation between neighboring observations in a series of samples, and is calculated with the following equation:

$$ACF(k) = \frac{\sum_{i=k+1}^{N} (x_i - \overline{x})(x_{i+k} - \overline{x})}{\sum_{i=1}^{N} (x_i - \overline{x})^2} \tag{8}$$

where ACF is the auto-correlation function that spans between -1 and + 1, x_i are the drawn samples ($i = 1...N$), $\overline{x}$ is the average value of all the drawn samples, and k is the lag. The lagged series, x_{i+k}, is simply the original series shifted k units backward in time. ACF identifies the non-randomness of a series of data and can be used to evaluate the mixing property of the drawn samples; the quicker the auto-correlation curve falls w.r.t the lag, the less correlated, and hence the more random the drawn samples are. Figure 3 depicts the auto-correlation function versus lag over the drawn samples for the case study with various values of D. Figure 3 indicates that PT $(D = 1)$ and MPT with $D = 2$ respectively have the least and the most correlations between their drawn samples. However, increasing D in MPT increases the randomness of the drawn samples as a result of D many independent kernels, and therefore, results in more analogous auto-correlation curves of MPT to that of PT.

Figure 4 depicts the normalized expected value calculated on one of the 4 dimensions of the actual samples generated by MPT-8 and PT-8 with respect to the samples generated. It shows that the expected value for one of the dimensions of the case study in PT-8 fluctuates around the true value, whereas the expected value for MPT-8 with D=54 has a more smooth curve that quickly averaged to the actual result within a defined confidence range. The smoother curve of the latter is

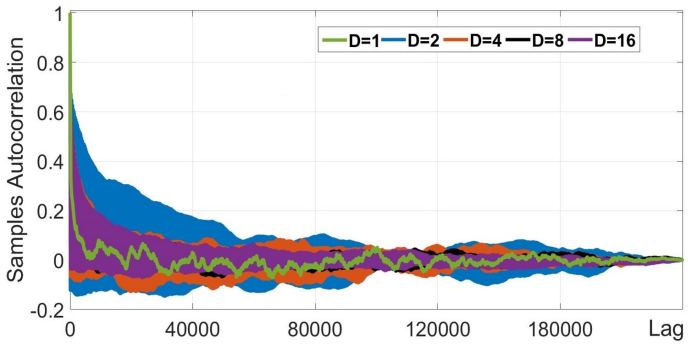

Fig. 3. Auto-correlation function versus lag for MPT (D>1) evaluated by 8 chains indicates that for large values of D, the correlation between the drawn samples can be as little as the correlation between samples drawn by PT (D=1). Periodic ripples within the auto-correlation curves of MPT, that make the curves look solid for D>1, indicate that there are periodic inter-dependencies between generated samples as a result of multiple independent samplers.

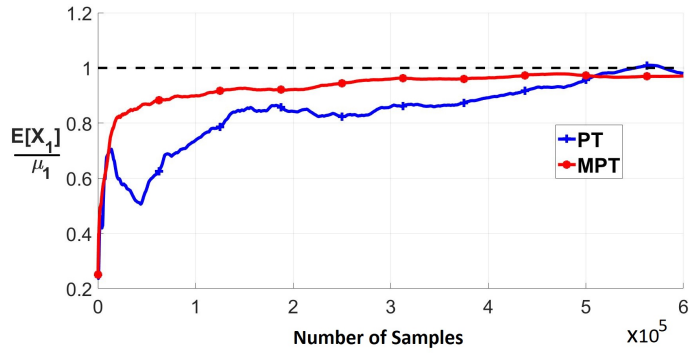

Fig. 4. Comparison between normalized expected value of one of the output dimensions of PT and MPT (D=54), both evaluated by 8 chains, with respect to the generated samples

due to multiple independent kernels in the MPT that has more degree of freedom to explore the target probability density.

IV. Hardware Architecture

In PT MCMC algorithm, shared chain architecture experiences trade-off between throughput and number of chains, however number of chains will add idle cycles due to pipeline structure of the architecture [17]. We propose a high-throughput architecture for MPT algorithm with limited number of chains, which can be efficiently implemented on tiny Artix-7 Xilinx FPGA for a given distribution. Figure 5 depicts the hardware architecture of MPT Algorithm (algorithm 1) with two chains that generates samples from the given $p(x)$. We consider this architecture as a baseline for a PT and a high-throughput MPT sampler over the given $p(x)$. The architecture has two streams of data that can be assumed as a clockwise circular flow: forward and backward data flows.

The proposed architecture is reconfigurable for PT and MPT algorithms depending on the value selected for parameter D: in case of $D = 1$ it is configured to PT algorithm, and when $D > 1$ it acts as an MPT algorithm. The architecture is initially designed from a PT point of view with a purely combinational logic for the forwarding data flow. Sample values are in 18-bit fixed-point format, and the probability of each sample is calculated in the logarithmic domain. Fixed point representation reduces hardware complexity, while logarithmic representation converts the exponential, multiplication and division in Algorithm 1 to multiplication, addition, and subtraction respectively. The proposed architecture consists of four kernels: 1. Algorithm Initialization (INIT), 2. Random Number Generators (RNG), 3. Chain Update, and 4. Exchanger. INIT kernel initializes samples $(x_{1:M}^{(1)} \ \ldots \ x_{1:M}^{(D)})$ by user and are carefully chosen. The Random Number Generators (RNG) consists of Uniform and Gaussian Random Number Generators (URNG and GRNG.) We implement Tausworthe URNG, and GRNG is implemented by using Inverse Cumulative Distribution Function (ICDF) [18]. The Chain Update is the computational block that chooses between the current sample and the proposed sample by comparing the ratio of their probabilities with a generated URNG sample. The proposed sample is made by summing a GRNG sample and the current sample and the chosen sample will be the next sample (line 9 to 12 of Algorithm 1). Probability Estimation module in each chain update kernel calculates the probability of each incoming proposed sample, and is the most computationally intensive module among all modules of the design. It comprises logarithm and exponential calculators, implemented using 16 cascaded blocks on behalf of 16 unfolded iterations of CORDIC algorithms. Every hardware configuration and application specific data, such as chain temperatures, GRNGs' standard deviations, initialization values, μ's and weights of the GMM model are stored in Block RAMs and are subject to the change of the application and the configuration. For the sample exchange between chains, we choose a binary fashion policy so that with the least required number of stages all chains have the chance to mix with the 1^{st} chain. In order to mix chains in an 8 chain design for example, 4 parallel exchangers handle chain pairs (1,2), (3,4), (5,6) and (7,8) at the first stage, then again two parallel exchangers handle (1,3) and (5,7), and finally a last exchanger mixes chain pair (1,5).

The PT architecture can be converted to an MPT by applying D-pipeline stages to the forwarding data flow. The conceptual Pipeliner block defines the number of pipeline stages that divide the forwarding data flow for MPT into D chunks. We are interested in the smallest D that results in the maximum system frequency. By doing so, both the system throughput and each independent PT sampler within the MPT will reach to their maximum throughput. Ideally, the best D for MPT can be inferred by dividing the maximum achievable frequency of the FPGA by the clock frequency of the baseline PT, from which MPT is derived. The updated and exchanged sample of each chain will finally loop back to the Initializing block of its affiliated chain. By using a counter that counts only once from 1 to $D + 1$, the INIT block sends in the first D initialization samples to the forwarding data flow upon the system's reset (line 1 of the algorithm) and, from there,

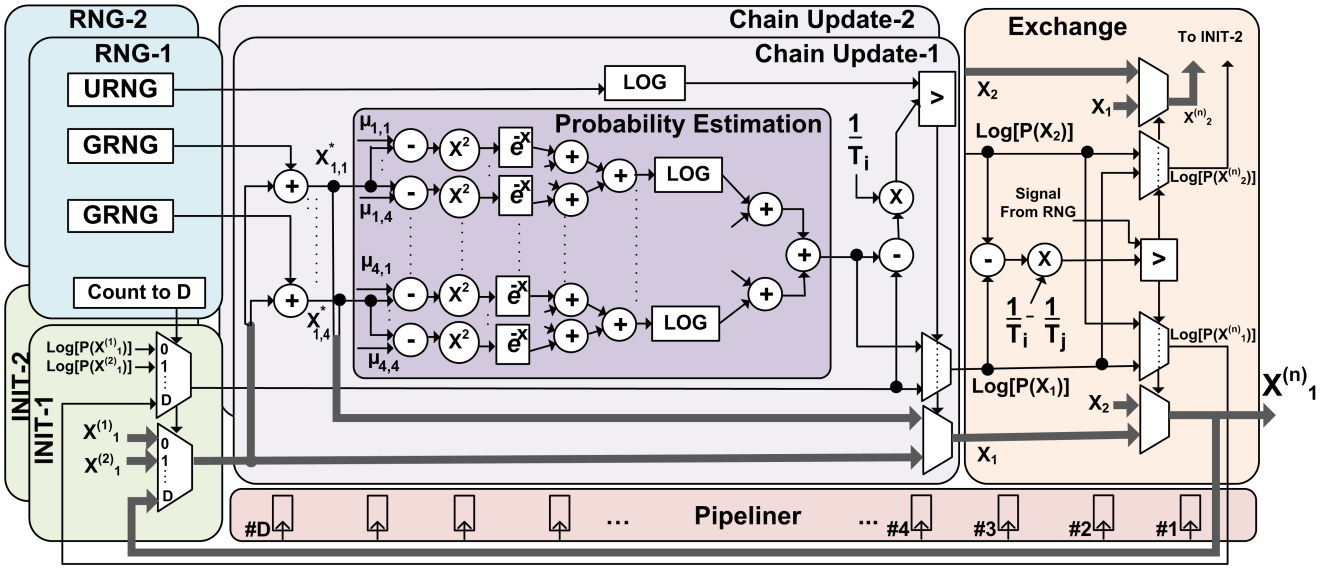

Fig. 5. Hardware architecture design of PT and MPT with two chains (PT-2 and MPT-2) for four dimensional multi-modal GMM on FPGA. For every proposed sample, probability is evaluated by the Probability Estimation block in the logarithm domain, that for our case study in this work is evaluated as $log(p(x)) = \sum_{i=1}^{4} log(e^{-(x_i-\mu_{i,1})^2} + e^{-(x_i-\mu_{i,2})^2} + e^{-(x_i-\mu_{i,3})^2} + e^{-(x_i-\mu_{i,4})^2})$. The proposed sample may get accepted conditioned by a URNG sample. The accepted sample may similarly get swapped by means of the Exchange block with the sample generated by the other chain. $\mu_{i,j}$ represents the mean of j^{th} mode in the i^{th} dimension of the probability density. $X_{i,j}$ and X_j^n respectively represent the j^{th} dimension and the n^{th} sample in the sequence of series generated by the i^{th} chain. For the sake of simplicity, weights of the GMM aren't shown. T_i represents temperature of chain i. The conceptual Pipeliner block implies that, in order to convert a PT into an MPT, the forwarding data flow should get pipelined into D chunks. For a 1 chain design, only Chain Update-1, RNG-1, and INIT-1 are required and no Exchanger is needed. For 4 chain configuration, the exact 2 chain architecture, before establishing the backward data, is duplicated and an extra Exchanger mixes chain pair (1,3). Similarly, an 8 chain configuration is made out of two 4 chain architectures, as well as an extra Exchanger that mixes chain pair (1,5).

TABLE I
POST PLACE AND ROUTE IMPLEMENTATION RESULTS ON ARTIX-7 FOR PT AND MPT ARCHITECTURES WITH CHAIN NUMBERS EQUAL TO 1, 2, 4, AND 8, IN TERMS OF RESOURCE UTILIZATION, THROUGHPUT AND POWER DISSIPATION.

Architecture	PT-1 (MH)	**MPT-1**	PT-2	**MPT-2**	PT-4	**MPT-4**	PT-8	**MPT-8**
Slice Count (%)	2,757 (8%)	**3,083 (9%)**	6,597 (18%)	**6,948 (20%)**	12,748 (36%)	**13,390 (39%)**	23,241 (70%)	**25,755 (76%)**
Register Count (%)	66 (∼0%)	**14,726 (5%)**	133 (∼0%)	**29,851 (11%)**	418 (∼0%)	**59,726 (22%)**	969 (∼0%)	**119,439 (44%)**
Block RAMs	1	**1**	2	**2**	4	**4**	8	**8**
Mul 18×18	12	**12**	25	**25**	51	**51**	103	**103**
No. of Pipelines (D)	1	**45**	1	**48**	1	**51**	1	**54**
Dynamic Power (W)	0.07	**1.61**	0.13	**3.05**	0.27	**5.82**	0.54	**11.71**
Static Power (W)	0.22	**0.23**	0.22	**0.25**	0.22	**0.29**	0.23	**0.39**
Total Power (W)	0.29	**1.84**	0.35	**3.30**	0.49	**6.11**	0.77	**12.10**
Throughput (Msps)	5.0	**156.5**	4.7	**147.4**	4.3	**125.7**	4.1	**114.6**
Throughput Improvement	Base	**31.1**×	Base	**31.4**×	Base	**29.5**×	Base	**28.0**×

provides a direct link between backward and forwarding data flows. Finally, only samples from the first chain will be taken out of the hardware through four 18-bit outputs to feed any interface, e.g. an accumulator for calculating equation 2, that follows the sample generator for further process.

A. FPGA Implementation Results

The reconfigurable PT and MPT architecture with 1, 2, 4, and 8 number of chains is implemented using Verilog HDL, and synthesized and placed & routed on tiny and low power Artix-7 FPGA using Xilinx ISE tools. We start each architecture from a PT algorithm point of view for each number of chain configurations, and then, designing the respective MPT required finding the optimal value for D. The optimal D results in the highest throughput of the MPT, as a whole, and the highest throughput of independent PT kernels inside MPT. Such value for each case study and algorithm configuration can be found experimentally; with a trial and error method of placing pipeline stages at different locations of the initial PT architecture, we found the near optimum values of D for each configuration. Table I shows the chosen D, device utilization, total power dissipation and throughput for each configuration. At the expense of respectively 5, 11, 22, and 44 percent extra utilization of register resources inside the FPGA (for implementing the pipeline) for chain numbers equal to 1, 2, 4, and 8, the MPT yields an average speed-up of 30 × in throughput as compared to the analogous PT with the same chain configuration.

B. TX2 Implementation Results

In order to evaluate the PT and MPT on other Commercial Off-The-Shelf (COTS) products, the two algorithms for our GMM case study were written in C and run on the Multi-core CPU from TX2 SoC. NVIDIA's Jetson TX2 features an integrated 256-core NVIDIA Pascal GPU, a CPU complex combining a dual-core NVIDIA Denver 2 alongside a quad-core ARM Cortex-A57, and 8GB of LPDDR4 memory with a 128-bit interface. Every chain in PT and MPT was mapped to a core for parallel processing, and message passing interface (MPI) was used to communicate data for the swap phase in their algorithms. The frequency of all cores were set to 1728 MHz and floating point operations were used. In each experiment $100,000$ samples were generated. We measured that at the given frequency, the PT algorithm generates samples on par with the MPT algorithm, regardless of the value of D, with a throughput rate of 0.12, 0.11, 0.09, and 0.06 million samples per second (Msps) respectively for chain number 1, 2, 4, and 8. It is noteworthy that mapping respectively 1 and 2 chains to the quad-core ARM processor is not as efficiently resource-exploiting as mapping 4 and 8 chains.

V. Conclusion

In this work, we imposed a new parameter, D, to the Parallel Tempering algorithm, a powerful class of MCMC algorithms, and proposed Multiple Parallel Tempering algorithm that, by choosing an appropriate integer value for D, allows to increase the sampling throughput over a given $p(x)$, to near the maximum frequency achievable by the targeted FPGA. The optimal value selection for D in every application depends on the hardware complexity of the implemented probability density, i.e. the longer the critical path of a probability estimation, the more number of pipeline stages will result in the maximum throughout, and thus requiring larger D. For an employed 4 dimensional i.i.d GMM likelihood function in this work, and by choosing an average value of 48 for D, the MPT sampler approximately yields $30\times$ speedup in sample generation throughput for chain number 1, 2, 4 and 8 configurations when compared to the PT sampler with equivalent number of chains. When compared to the implementation on ARM Cortex A57, as a Multi-core CPU, for the same case study and algorithm configuration, MPT on the FPGA achieves averagely $1470\times$ speedup in terms of throughput. The new algorithm not only gains in terms of throughput on FPGA, but also, for large values of D, generates samples whose auto-correlation and randomness are analogous to that of samples drawn out of an equivalent PT sampler.

References

[1] A. D. C. Andrieu, N.D. Freitas and M. Jordan, "An introduction to mcmc for machine learning," *Machine Learning*, no. 50, pp. 5–43, 2003.

[2] G. Mingas and C. S. Bouganis, "Population-based mcmc on multi-core cpus, gpus and fpgas," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 65, no. 4, pp. 1283–1296, 2016.

[3] C. J. Ter Braak, "A markov chain monte carlo version of the genetic algorithm differential evolution: easy bayesian computing for real parameter spaces," *Statistics and Computing*, vol. 16, no. 3, pp. 239–249, 2006.

[4] F. Harl, F. Chatelain, C. Gouy-Pailler, and S. Achard, "Bayesian model for multiple change-points detection in multivariate time series," *IEEE Transactions on Signal Processing*, vol. 64, no. 16, pp. 4351–4362, 2016.

[5] R. Azencott, V. Muravina, R. Hekmati, W. Zhang, and M. Paldino, "Automatic clustering in large sets of time series," in *Contributions to Partial Differential Equations and Applications*. Springer, 2019, pp. 65–75.

[6] B. Darvish Rouhani, M. Ghasemzadeh, and F. Koushanfar, "Causalearn: Automated framework for scalable streaming-based causal bayesian learning using fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 1–10.

[7] G. M. Shuanglong Liu and C.-S. Bouganis, "An unbiased mcmc fpga-based accelerator in the land of custom precision arithmetic," *IEEE TRANSACTIONS ON COMPUTERS*, vol. PP, no. 99, pp. 1–1, 2016.

[8] N. B. Asadi *et al.*, "Reconfigurable computing for learning bayesian networks," in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. ACM, 2008, pp. 203–211.

[9] 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*, April 2012, pp. 153–156.

[10] L. Marni, M. Hosseini, and T. Mohsenin, "Mc3a: Markov chain monte carlo manycore accelerator," in *ACM Proceedings of the 28th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2018.

[11] G. Mingas and C.-S. Bouganis, "Population-based mcmc on multi-core cpus, gpus and fpgas," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1283–1296, 2016.

[12] R. Hekmati, R. Azencott, W. Zhang, and M. Paldino, "Machine learning to evaluate fmri recordings of brain activity in epileptic patients."

[13] S. Liu, G. Mingas, and C.-S. Bouganis, "An unbiased mcmc fpga-based accelerator in the land of custom precision arithmetic," *IEEE Transactions on Computers*, no. 5, pp. 745–758, 2017.

[14] D. J. Earl and M. W. Deem, "Parallel tempering: Theory, applications, and new perspectives," *Physical Chemistry Chemical Physics*, vol. 7, no. 23, pp. 3910–3916, 2005.

[15] D. A. Kofke, "Erratum:on the acceptance probability of replica-exchange monte carlo trials[j. chem. phys. 117, 6911 (2002)]," *The Journal of chemical physics*, vol. 120, no. 22, pp. 10 852–10 852, 2004.

[16] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[17] G. Mingas and C.-S. Bouganis, "Parallel tempering mcmc acceleration using reconfigurable hardware," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2012, pp. 227–238.

[18] R. Gutierrez *et al.*, "Hardware architecture of a gaussian noise generator based on the inversion method," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 8, pp. 501–505, Aug 2012.

[19] C. Shea, A. Page, and T. Mohsenin, "Scalenet: a scalable low power accelerator for real-time embedded deep neural networks," in *ACM Proceedings of the 28th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2018.

[20] A. Jafari, M. Hosseini, C. P. A.Kulkarni, and T. Mohsenin, "Binmac: Binarized neural network manycore accelerato," in *ACM Proceedings of the 28th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2018.

[21] M. Hosseini, R. Islam, A. Kulkarni, and T. Mohsenin, "A scalable fpga-based accelerator for high-throughput mcmc algorithms," in *IEEE Symposium on Field- Programmable Custom Computing Machines (FCCM)*. IEEE, 2017.

[22] A. Kulkarni, A. Page, N. Attaran, A. Jafari, M. Malik, H. Homayoun, and T. Mohsenin, "An energy-efficient programmable manycore accelerator for personalized biomedical applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, 2017.

[23] N. Attaran, A. Puranik, J. Brooks, and T. Mohsenin, "Embedded low-power processor for personalized stress detection," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, no. 99, pp. 1–1, 2018.

[24] A. Jafari, , and T. Mohsenin, "Sensornet: A scalable and low-power deep convolutional neural network for multimodal data classification," *IEEE Transactions on Circuits and Systems*, vol. PP, pp. 1–12, 2018.

[25] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with fft on embedded hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–24, 2018.