A Custom Precision Based Architecture for Accelerating Parallel Tempering MCMC on FPGAs Without Introducing Sampling Error

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

***Abstract*—Markov Chain Monte Carlo (MCMC) is a method used to draw samples from probability distributions in order to estimate - otherwise intractable - integrals. When the distribution is complex, simple MCMC becomes inefficient and advanced, computationally intensive MCMC methods are employed to make sampling possible. This work proposes a novel streaming FPGA architecture to accelerate Parallel Tempering, a widely adopted MCMC method designed to sample from multimodal distributions. The proposed architecture demonstrates how custom precision can be intelligently employed without introducing sampling errors, in order to save resources and increase the sampling throughgput. Speedups of up to two orders of magnitude compared to software and 1.53x-76.88x compared to a GPGPU implementation are achieved when performing Bayesian inference for a mixture model.**

I. Introduction

Markov Chain Monte Carlo (MCMC) is a class of stochastic algorithms used in numerous fields ([1] for an overview). Its aim is to sample from a given probability distribution, typically in order to solve the following problem:

Problem: Estimate the expectation of $f(\mathbf{x})$ under $p(\mathbf{x})$:

$$E_p[f(\mathbf{x})] = \int f(\mathbf{x})p(\mathbf{x})\mathbf{dx} \quad (1)$$

where $\mathbf{x}$ is the random variable of interest, $p(\mathbf{x})$ is the probability distribution of $\mathbf{x}$ and $f(\mathbf{x})$ is a function of interest (e.g. mean, moment). MCMC draws dependent samples from $p(\mathbf{x})$ (the target distribution) using a Markov chain [1], in contrast to other Monte Carlo methods which draw independent samples. Integral (2) is then approximated by:

$$\tilde{E_p}[f(\mathbf{x})] = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}^{(i)}) \quad (2)$$

where $\mathbf{x}^{(i)}$, $i \in \{1, ..., N\}$ are the dependent samples.

The choice of the transition kernel of the Markov chain (which defines how each new sample is generated from the previous one) is crucial to ensure rapid convergence to $p(\mathbf{x})$ and good mixing (fast movement around the support of $p(\mathbf{x})$). In cases of multimodal target distributions, simple kernels tend to get "stuck" in one of the modes [1] and advanced MCMC methods, like Parallel Tempering (PT) [1], are used. PT employs a population of Markov chains and this makes it computationally intensive (see [2] and [3]).

Lately, several studies on accelerating expensive MCMC methods (like PT) have appeared in the literature. Until now, FPGA-based research has focused on exploiting the form of the target distribution to achieve acceleration, rather than efficiently mapping generic MCMC methods. [4] proposes an FPGA architecture for MCMC inference in Bayesian networks, which parallelizes intra-chain calculations. [5] uses FPGAs to accelerate inference in Markov Random Fields by splitting the problem into sub-problems and running parallel samplers. More generic work is found in [3] and [6], where GPGPU and cluster implementations of PT are presented.

In this work, a custom precision FPGA architecture which aims at accelerating PT is proposed, based on the streaming architecture of [7]. The architecture is generic and can be used to sample from any distribution. It simultaneously takes advantage of the nature of PT and the characteristics of FPGAs (deep pipelining, fast inter-circuit communication). Moreover, the capability of FPGAs to use custom precision is exploited in a novel way, making it possible to entirely avoid sampling errors related to reduced precision. This can translate into significant speedups, making the proposed system up to two orders of magnitude faster than software and 1.53x-76.88x faster than an existing GPGPU implementation when performing Bayesian inference for a mixture model.

The rest of the paper includes a description of PT (Section II), a presentation of the architecture and the evaluation model (Sections III, IV), a presentation of the results (Section V) and some concluding remarks (Section VI).

II. Parallel Tempering

Parallel Tempering uses a population of Markov chains to improve mixing when the target distribution is multimodal. Each chain j samples from a different distribution $p_j(\mathbf{x})$:

$$p_j(\mathbf{x}) = p(\mathbf{x})^{1/T_j}, j \in \{1, ..., M\} \quad (3)$$

where T_j (with $1 = T_1 < T_2 < ... < T_M$) is the temperature of the chain. Distribution $p_1(\mathbf{x})$ is equal to the target $p(\mathbf{x})$ and the remaining distributions are smoothed versions of

978-0-7695-4699-5/12 $26.00 © 2012 IEEE
DOI 10.1109/FCCM.2012.34

$p(\mathbf{x})$, which are easier to sample from (closer to uniform). The degree of smoothing is increased with higher j. The target distribution is called the "coldest" distribution ($T_1 = 1$) and distributions get "hotter" for $j > 1$.

At time (iteration) i, the state of PT comprises the samples $\{\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, ..., \mathbf{x}_M^{(i)}\}$ from all M chains. PT updates these samples independently (using separate transition kernels according to (3)). An update at time i for chain j happens as follows: 1) A candidate sample $\mathbf{y}_j^{(i)}$ is drawn from a normal distribution centred around the previous sample $\mathbf{x}_j^{(i)}$. 2) The candidate sample is accepted as the next sample ($\mathbf{x}_j^{(i+1)} = \mathbf{y}_j^{(i)}$) with probability $a(\mathbf{x}_j^{(i)}, \mathbf{y}_j^{(i)})$:

$$a(\mathbf{x}_j^{(i)}, \mathbf{y}_j^{(i)}) = min\Big(1, \frac{p_j(\mathbf{y}_j^{(i)})}{p_j(\mathbf{x}_j^{(i)})}\Big) \quad (4)$$

If the candidate is rejected, the previous sample $\mathbf{x}_j^{(i)}$ becomes the next sample ($\mathbf{x}_j^{(i+1)} = \mathbf{x}_j^{(i)}$).

The hot chains move quickly in the space while the cold chains might get "stuck". Periodically, sample exchanges between chains are proposed, which push samples from the hot to the colder chains and eventually to the coldest chain, helping it escape from isolated modes. An exchange between chains q and r is accepted with probability $e(\mathbf{x}_q, \mathbf{x}_r)$ (the time step index i is omitted for clarity):

$$e(\mathbf{x}_q, \mathbf{x}_r) = min\Big(1, \frac{p_q(\mathbf{x}_r)p_r(\mathbf{x}_q)}{p_q(\mathbf{x}_q)p_r(\mathbf{x}_r)}\Big) \quad (5)$$

Here, exchanges only between neighbouring chains (chains $(1,2), (3,4), ..., (M-1, M)$ or chains $(2,3), (4,5), ..., (M-2, M-1), (M, 1)$ alternatively) are performed. Finally, to compute (2), only samples from the coldest chain are kept.

III. System Architecture

The tempered chains can run independently and only communicate during exchanges. The most computationally demanding task within each chain j is the evaluation of the probability $p_j(\mathbf{y}_j)$ of the proposed sample $\mathbf{y}_j$. Due to the form of (3), this computation is the same for all chains (apart from the temperature and the proposed sample). To exploit this, the architecture treats PT as a streaming application, taking advantage of both parallelism and pipelining. The core of the design comprises multiple pipelines which implement the density $p(\mathbf{x})$. Each pipeline is assigned a subset of the independent chains' density evaluations until it is fully utilized. The temperature scaling is applied at a later stage.

Figure 1 shows an overview of the architecture, where the probability evaluation block contains the parallel pipelines. The remaining modules are also pipelined but they are not critical in terms of the throughput of the system. The sample proposal block reads the current sample of every chain from the memory and generates a proposed sample, using numbers from a Gaussian Random Number Generator (RNG) [8]. The probability evaluation block reads each proposed sample and computes its proposed probability. The update block receives the proposed probability (together with the current probability and temperature of the respective chain and a uniform random number) and accepts or rejects the proposed sample using (4). This happens until all chains pass through the system and then the sequence is repeated.

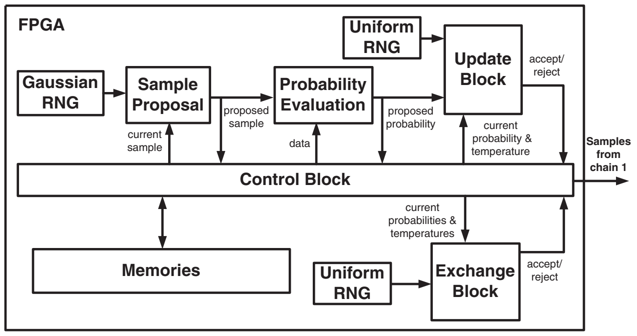

Figure 1. Overview of the architecture.

The exchange step is executed for a pair of chains in parallel to the above update operations. Exchanges are proposed between neighbouring chains only, in order to avoid stalls in the probability evaluation pipelines (chains 2, 3 are exchanged while chains 4, 5, etc are updated).

The bottleneck of the system is the probability evaluation block. The throughput depends on how fast this block can evaluate the probabilities of incoming samples. This in turn depends on how many parallel pipelines (which implement $p(\mathbf{x})$) can fit into the FPGA. In order to instantiate more pipelines, [7] propose the use of custom arithmetic precision when implementing $p(\mathbf{x})$ in order to save area. This leads to sampling from an altered distribution and [7] show that this perturbation does not affect the quality of estimates (up to a break point). Here, a more advanced method which guarantees correct sampling and estimates regardless of the employed precision is proposed. It is based on the fact that only samples from the coldest chain are kept; samples from the hotter chains are only utilized to improve mixing and do not affect the sampled distribution. In order to guarantee correct sampling, the density $p_1 = p$ used in the coldest chain is implemented in single precision (considered high precision). The remaining chains sample from tempered versions of $\tilde{p}$ (the low precision implementation of p):

$$\tilde{p_j}(\mathbf{x}) = \tilde{p}(\mathbf{x})^{1/T_j}, j \in \{2, ..., M\} \quad (6)$$

Exchanges between chains $q > 1$ and $r > 1$ are not accepted with probability given by (5) but with probability:

$$e(\mathbf{x}_q, \mathbf{x}_r) = min\Big(1, \frac{\tilde{p_q}(\mathbf{x}_r)\tilde{p_r}(\mathbf{x}_q)}{\tilde{p_q}(\mathbf{x}_q)\tilde{p_r}(\mathbf{x}_r)}\Big) \quad (7)$$

Exchanges that include the first (coldest) chain and chain r (here $r = 2$ always) are accepted with probability:

$$e(\mathbf{x}_1, \mathbf{x}_r) = min\Big(1, \frac{p_1(\mathbf{x}_r)\tilde{p_r}(\mathbf{x}_1)}{p_1(\mathbf{x}_1)\tilde{p_r}(\mathbf{x}_r)}\Big) \quad (8)$$

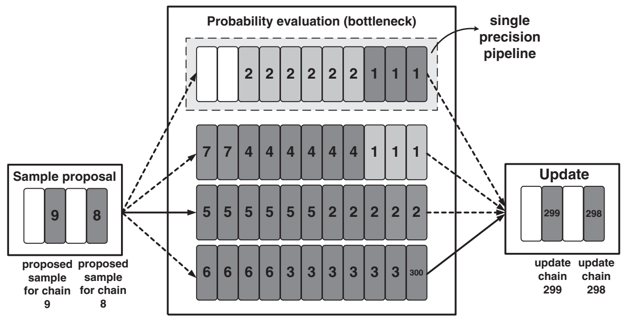
Probability evaluation (bottleneck)
single precision pipeline
Sample proposal
Update
proposed sample for chain 9
proposed sample for chain 8
update chain 299
update chain 298

Figure 2. Streaming of chains through the sample proposal, probability evaluation and update blocks. One single precision and three reduced precision pipelines comprise the probability evaluation block and 300 tempered chains are processed. The numbers represent the chains that occupy each stage. Each pipeline generates a new probability every 6 cycles and the whole probability evaluation block generates a probability every 2 cycles. Thus the sample proposal and update blocks are half-utilized. The solid arrows show which pipeline will be used next. The different colours of the stages are explained in the main text.

The idea of using inexpensive approximations of the tempered distributions for the hot chain updates and using the true expensive distribution only for the coldest chain updates has been previously introduced by [2]. By altering the exchange acceptance probability for the exchanges that include the coldest chain, the stationary distribution (see [1]) of every chain is retained [2]. The use of limited precision implementations as approximations to the true density has not been proposed in existing literature.

The price for not using full precision in hotter chains is that mixing might be affected because sample exchanges between the first and the second chain are less likely to succeed (the authors in [2] fail to clearly comment on this behaviour). Therefore, mixing speed is reduced while throughput is increased (because of the less area-expensive pipelines). This tradeoff is investigated in Section V.

Figure 2 shows the parallel pipelines. The single precision pipeline is only responsible for the probability evaluations used in the first chain's updates (dark grey stages) and the evaluation of $p_1(\mathbf{x}_2)$ in (8) for exchanges between chains 1 and 2 (light grey stages). The low precision pipelines handle the hotter chains' probability evaluations during updates (dark grey stages) and the evaluation of $\tilde{p_2}(\mathbf{x}_1)$ in (8) for exchanges between chains 1 and 2 (light grey stages).

The architecture is designed so that sampling from any $p(\mathbf{x})$ is possible. Only the appropriate probability evaluation pipelines, implemented using libraries of floating point operators (e.g. FloPoCo [9]), need to be plugged in.

IV. Bayesian Inference for Mixture Models

Bayesian inference is a statistical inference method used to draw conclusions about unobserved or unobservable quantities, given a model and some observed data [1]. Mixture models are a powerful family of models used in numerous fields. Multimodal posterior distributions often appear when performing inference for these models. Hence, they are representative of problems that PT is used to solve.

To test the architecture's performance, a Gaussian mixture model taken from [3] is used: A set of independent observations (data) $\mathbf{d} = d_{1:Q}$, where $d_q \in \Re$ for $q \in \{1, ..., Q\}$, is given. Each observation is distributed according to:

$$p(d_q|\mu_{1:k}, \sigma_{1:k}, a_{1:k-1}) = \sum_{i=1}^{k} a_i f(d_q|\mu_i, \sigma_i) \qquad (9)$$

Here, f is the density of a univariate Gaussian, k is the number of mixture components and $\mu_{1:k}$, $\sigma_{1:k}$ and $a_{1:k-1}$ are the parameters of the model (means, variances and weights).

The model configuration of [3] is used: $k = 4$, $\sigma_i = \sigma = 0.55$ and $a_i = a = 1/k$ for $i \in \{1, ..., k\}$. The posterior distribution of $\mu = \mu_{1:k}$ needs to be sampled. The prior distribution on μ is 4-dimensional uniform. The data $\mathbf{d} = d_{1:m}$ (with $m = 100$) are simulated using $\mu = (-3, 0, 3, 6)$. The likelihood function is given by:

$$p(\mathbf{d}|\mu) = \prod_{j=1}^{100} p(d_j|\mu, \sigma_{1:k}, a_{1:k-1}) \qquad (10)$$

If $p(\mu)$ is the prior, the posterior distribution for μ is:

$$p(\mu|\mathbf{d}) = p(\mathbf{d}|\mu)p(\mu) \qquad (11)$$

This posterior distribution is the target distribution of PT and it is multimodal (24 modes).

V. Results

The proposed architecture was implemented in VHDL and the targeted FPGA board was a Xilinx ML605, which contains a Virtex 6 XC6VLX240T FPGA. The FPGA was clocked at 212 MHz. Communication with the host PC was done using n Xilinx System Generator Ethernet interface.

In contrast to [7], the proposed architecture does not suffer from sampling errors related to precision. Here, the sampling distribution remains unaffected but the mixing speed changes with precision. Thus, more samples are necessary to achieve the same variance in the estimates. Effective Sample Size (ESS) [1] is the most widely adopted method to measure mixing speed. It gives an estimate of how many independent samples the correlated MCMC samples are equivalent to. The mean ESS from 10 independent runs of 10^6 samples (using $M = 128$ chains) for all precision configurations is shown in Figure 3. ESS varies but remains close to the single precision ESS until precision drops to (8,5). This demonstrates that using low precision in hot chains does not significantly affect the mixing speed of the coldest chain. The figure also shows the speedup of each configuration compared to the single precision version in terms of samples/sec (raw sampling throughput) when the FPGA is fully utilized. Speedup increases due to the cheaper floating point operators which allow the instantiation of more pipelines.

In order to compare the performance of the architecture for different precision configurations, the Effective Samples

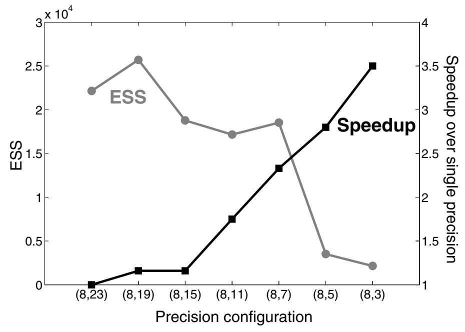

Figure 3. ESS from 10 runs of 10^6 samples for every precision configuration (Exponent bits, Mantissa bits) using $M = 128$ chains and samples/sec speedup over the single precision version.

Table I
ES/SEC SPEEDUP FOR ALL PRECISION CONFIGURATIONS COMPARED TO THE SINGLE PRECISION VERSION WHEN RUNNING $M = 128$ CHAINS.

Precision	(8,23)	(8,19)	(8,15)	(8,11)	(8,7)	(8,5)	(8,3)
ES/sec speedup	-	1.34	1.02	1.35	1.95	0.44	0.33

per second (ES/sec) that each one can generate is used. This is a form of "effective" throughput which is more interesting to an MCMC practitioner than the raw sampling throughput. Table I shows the ES/sec speedup vs. the single precision configuration for all precision configurations when running $M = 128$ chains. The peak performance is reached for the (8,7) configuration, which has raw sampling throughput 2.33x higher than single precision but its mixing speed (ESS) is 1.19x lower than that of single precision. Combining the two numbers, the ES/sec is 1.95x higher than the ES/sec of the single precision architecture. The same procedure can be applied for any employed M in order to find the optimal precision configuration which maximizes ES/sec.

Finally, the performance of the architecture is compared to the performance of a CPU and a GPGPU implementation of PT [3] for the same target. The scaling of the performance with the problem size (chains M) is also investigated. The CPU code was written in C++ and ran on a Xeon E5420 (2.5 GHz). The GPGPU code was written in CUDA and ran on an Nvidia GTX280 in single precision. Table II lists the ES/sec speedups of the GPGPU and FPGA compared to the CPU. For small M, the speedups are significantly lower than their peak values since there are not enough chains to fill the pipelines (or the GPGPUs' resources). The FPGA outperforms the GPGPU for all M (up to 76.88x faster). The fluctuation of the FPGA's speedup for $M > 128$ is due to the varying effect that precision has on mixing (ESS) when M changes. It must be noted that the probability evaluation is not parallelized in the GPGPU implementations (each thread is assigned one chain). In the FPGA, the use of pipelining takes advantage of the independent likelihood components. Even when taking this into account, for $M = 131072$ the FPGA architecture's ES/sec speedup is 1.53x higher than the GPGPU's ES/sec speedup (peak speedup of the GPGPU).

Table II
ES/SEC SPEEDUP OF FPGA (LX240T) AND GPGPU (GTX280) [3] VS CPU (E5420) [3] FOR DIFFERENT NUMBERS OF CHAINS M.

Nbr. of chains (M)	8	32	128	512	2048	8192	32768	131072
LX240T speedup	69.2	211.9	897.0	903.1	865.0	860.9	895.7	880.6
GTX280 speedup	0.9	4	14	51	175	430	527	572

VI. CONCLUSION

This work presents a generic FPGA architecture that accelerates PT, a computationally expensive MCMC method, by taking advantage of the characteristics of FPGAs and the nature of the method. Custom precision is employed in such a way that no sampling error is introduced. The proposed architecture is shown to be faster than existing CPU and GPGPU implementations.

REFERENCES

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

[2] M. Fielding, D. J. Nott, and S.-Y. Liong, "Efficient MCMC Schemes for Computationally Expensive Posterior Distributions," *Technometrics*, vol. 53, no. 1, pp. 16–28, 2011.

[3] 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," *Journal of Computational and Graphical Statistics*, vol. 19, no. 4, pp. 769–789, 2010.

[4] N. B. Asadi, T. H. Meng, and W. H. Wong, "Reconfigurable computing for learning Bayesian networks," in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, ser. FPGA '08. New York, NY, USA: ACM, 2008, pp. 203–211.

[5] V. K. Mansinghka, E. M. Jonas, and J. B. Tenenbaum, "Stochastic Digital Circuits for Probabilistic Inference," Massachussets Institute of Technology, Technical Report MIT-CSAIL-TR-2008-069, 2008.

[6] Y. Li, M. Mascagni, and A. Gorin, "A decentralized parallel implementation for parallel tempering algorithm," *Parallel Comput.*, vol. 35, pp. 269–283, May 2009.

[7] G. Mingas and C.-S. Bouganis, in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, O. Choy, R. Cheung, P. Athanas, and K. Sano, Eds. Springer Berlin Heidelberg, 2012, vol. 7199, pp. 227–238.

[8] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, "Gaussian random number generators," *ACM Comput. Surv.*, vol. 39, November 2007.

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