Algorithm and Hardware Co-Design for FPGA Acceleration of Hamiltonian Monte Carlo Based No-U-Turn Sampler

Yu Wang
Dept. of Electrical and Computer Engineering
University of California
Santa Barbara, CA, United States
yu95@ucsb.edu

Peng Li
Dept. of Electrical and Computer Engineering
University of California
Santa Barbara, CA, United States
lip@ucsb.edu

***Abstract*—Monte Carlo (MC) methods are widely used in many research areas such as physical simulation, statistical analysis, and machine learning. Application of MC methods requires drawing fast mixing samples from a given probability distribution. Among existing sampling methods, the Hamiltonian Monte Carlo (HMC) utilizes gradient information during Hamiltonian simulation and can produce fast mixing samples at the highest efficiency. However, without carefully chosen simulation parameters for a specific problem, HMC generally suffers from simulation locality and computation waste. As a result, the No-U-Turn Sampler (NUTS) has been proposed to automatically tune these parameters during simulation and is the current state-of-the-art sampling algorithm. However, application of NUTS requires frequent gradient calculation of a given distribution and high-volume vector processing, especially for large-scale problems, leading to drawing an expensively large number of samples and a desire of hardware acceleration. While some hardware acceleration works have been proposed for traditional Markov Chain Monte Carlo (MCMC) and HMC methods, there is no existing work targeting hardware acceleration of the NUTS algorithm. In this paper, we present the first NUTS accelerator on FPGA while addressing the high complexity of this state-of-the-art algorithm. Our hardware and algorithm co-optimizations include an incremental resampling technique which leads to a more memory efficient architecture and pipeline optimization for multi-chain sampling to maximize the throughput. We also explore three levels of parallelism in the NUTS accelerator to further boost performance. Compared with optimized C++ NUTS package: RSTAN, our NUTS accelerator can reach a maximum speedup of 50.6X and an energy improvement of 189.7X.**

***Index Terms*—Hamiltonian Monte Carlo, No-U-Turn Sampler, FPGA, Hardware Acceleration**

I. INTRODUCTION

Monte Carlo (MC) method is a powerful algorithm to approximate a target numerical integration (generally as $\int f(\theta) \cdot p(\theta) d\theta$), especially in the cases where the closed form of the integration is intractable. The efficient application of MC method requires drawing fast mixing samples from the given distribution $p(\theta)$. Unfortunately, in many applications such as Bayesian inference, the distributions cannot be solved analytically, making the deterministic method not available. Therefore, more generally applicable methods based on Markov process named Markov Chain Monte Carlo (MCMC) are proposed. The MCMC family consists of many algorithms such as Metropolis-Hastings [1], Gibbs sampling and Hamiltonian Monte Carlo (HMC) [2]. Among all these algorithms, HMC is the most generally applicable algorithm with highest efficiency. HMC gets the idea from quantum physics simulation by introducing an auxiliary momentum variable and converting the sampling from a target distribution to a step by step simulation of Hamiltonian dynamics [2]. Unlike its counterparts in MCMC family which usually suffers from low efficient random walk behaviors [3] (Metropolis-Hasting), or the strict requirement of full conditional distributions (Gibbs sampling) [4], HMC utilizes gradient information during the simulation and can choose the optimal direction for each step, allowing the algorithm to draw fastest mixing samples in theory.

Unfortunately, the general application of HMC is limited by its requirement of at least two properly tuned parameters, the step size and the step length of the simulation. Improper choice of these parameters can dramatically reduce the algorithm's efficiency. For instance, small step size or step length will generally result in simulation locality, and produce slow mixing samples. Large step lengths on the other side, will cause the simulation trajectory to trace back to its initial state and waste a lot of computation. While some methods have been proposed to tune step size on the fly [5], tuning step length is difficult since the optimal value differs in various problems or even in drawing different samples from a same distribution.

To tackle this problem, the No-U-Turn Sampler (NUTS) was recently proposed to eliminate the need to hand tune step length [3]. NUTS can work at least as efficient as HMC without requiring the user to specify the optimal simulation parameters and thus is the current state of art sampling algorithm. NUTS has also been incorporated into many mainstream sampling packages such as STAN [6] and Pyro [7].

Despite the highest efficiency of NUTS, the algorithm itself is complex to implement and requires frequent gradient calculation, computation of the distribution values and high volume vector processing. Using NUTS to draw larger number of samples from large scale problem is expensive and painful and thus comes the demand to accelerate the algorithm with

advanced architectures. However, the complexity of the algorithm also brings challenges to designing the corresponding accelerator.

On the embedded system side, past works have seen implementation of many MCMC algorithms on hardware platforms [8] [9], while very few of them are focusing on the implementation of algorithms in the HMC family. CausaLearn [10] serves as the first HMC accelerator for the analysis of time series data utilizing stochastic gradient HMC (SGHMC) algorithm [11] for Gaussian process. While CausaLearn focuses on the application and acceleration of traditional HMC algorithm and provides better memory management for problem with different batch sizes, advanced HMC algorithms which could further improve sample quality are not further discussed in that paper. Specifically, evaluation of the NUTS algorithm, as well as the related algorithm hardware co-optimization, has yet been explored by the hardware design community.

In this paper, we present the first NUTS accelerator design on FPGA with algorithm hardware co-optimization. Our main contributions targeting the challenges in accelerating the NUTS algorithm can be summarized as:

- Introducing multi-chain sampling in a single PE pipeline to increase the throughput and hardware utilization.
- Proposing incremental resampling technique for efficient memory utilization and management.

The rest of the paper is organized as follows: In section II, conceptual introduction to MCMC, HMC and NUTS are presented. In section III, analysis of the computation challenges, as well as our proposed solutions will be discussed. The accelerator overall architecture is also provided. In section IV, we will discuss additional issue for implementing NUTS accelerator on FPGA. In section V, we'll introduce the experiment setup and in section VI, results and analysis will be provided. We then will conclude the whole paper in section VII.

II. BACKGROUND

A. Markov Chain Monte Carlo

Monte Carlo methods approximate a numerical integration as shown in eq. (1):

$$\int f(\theta)\cdot p(\theta)\cdot d\theta = \frac{1}{n}\sum_{i=1}^{n} f(\theta_i) \qquad \theta_i \sim p(\theta), i.i.d \tag{1}$$

Implementing MC methods requires drawing large amount of samples subject to a given distribution $p(\theta)$. In many applications, especially real time inference tasks [12], drawing fast mixing samples is the bottleneck to improve the speed and accuracy in those algorithms.

However, with the increase of problem size and the complexity in target distributions, deterministic sampling methods such as reject-accept sampling cannot draw samples efficiently. Therefore, Monte Carlo Markov Chain (MCMC) sampling method is proposed to utilize Markov chain to sequentially draw correlated samples which will finally converge to the stationary target distribution. Many different algorithms have been proposed in the MCMC family such as Metropolis Hasting, Gibbs sampling, slice sampling, Hamiltonian Monte Carlo (HMC) and the recently proposed No-U-Turn Sampler (NUTS). Among them, NUTS is the current state of the art sampling method which are adopted in mainstream software solutions such as STAN [6] and Pyro [7].

B. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) introduces an additional auxiliary momentum (r) that is independent to the sampling variable (θ) and converts the sampling from the target distribution ($p(\theta)$) to the simulation of "state" variables (θ, r) of an equivalent frictionless physical system as shown in Fig. 1 (left), where θ donates the position of the virtual ball and the auxiliary momentum r donates its velocity. Provided an initial position θ^0 and velocity r^0, the total energy $H(\theta^0, r^0)$ of the Hamiltonian system is conserved. The virtual ball will slide back and forth on the frictionless plane while samples are drawn from the simulation trajectory. Meanwhile, in the energy (H) space, the trajectory will stay on the same energy level, which is shown in Fig. 1 (right).

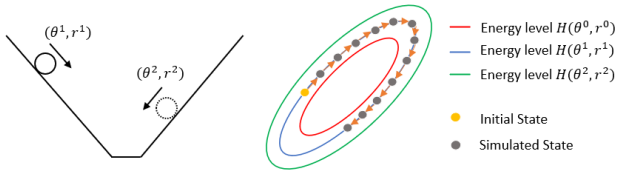

Fig. 1. A conceptual explanation of HMC. Left picture illustrates the frictionless system. Right picture demonstrates energy conservation of the simulation trajectory in H space.

The simulation of Hamiltonian dynamics is based on the discretized leapfrog method. A single leapfrog update is done by eq. (2).

$$\begin{aligned} r' &= r + \frac{\epsilon}{2}\cdot\nabla_\theta H(\theta, r) \\ \theta' &= \theta + \epsilon\cdot\nabla_r H(\theta, r') \\ r' &= r' + \frac{\epsilon}{2}\cdot\nabla_\theta H(\theta', r') \end{aligned} \tag{2}$$

Due to the discretized error introduced in leapfrog method, the energy is not always conserved after steps of simulation. To preserve detailed balance, an additional Metropolis correction step has to be introduced to accept the new sample θ' with a given probability: $\min(1, \frac{H(\theta', r')}{H(\theta, r)})$.

Clearly, this single simulation trajectory will not traverse the whole sample space of the given distribution. For instance, the virtual ball in Fig. 1 (left) will never slide beyond the height of its starting point if the initial velocity is 0. As a consequence, after a new sample is drawn from the simulation, the auxiliary momentum r will be resampled from a multivariate normal distribution ($\mathcal{N}(0, 1)$) as the new initial value for the next simulation.

To conclude, the key computation steps for drawing one sample in HMC are quite straight forward:

a. Specify simulation parameters (ϵ, L).

b. Run leapfrog methods for L steps: $(\theta^{'}, r^{'}) = leapfrog(\theta, r)$.

c. Accept $\theta^{'}$ with probability: $\min(1, \frac{H(\theta^{'}, r^{'})}{H(\theta, r)})$.

d. Redraw $r \sim \mathcal{N}(0, 1)$.

The performance of HMC relies heavily on the user specified parameters: step size ϵ and step length L. If ϵ or L is too small, the new samples suffer from locality and results in slow mixing. If ϵ is large, the simulation error will be too large and the rejection rate will increase dramatically. If L is too large, the simulation trajectory will trace back to initiate state or already visited states, which wastes a lot of computation. The examples for the three cases are illustrated as in Fig. 2.

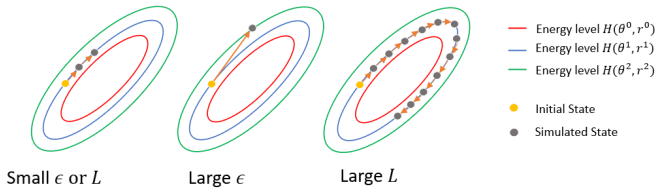

Fig. 2. Three cases of improperly chosen simulation parameters.

C. The No-U-Turn Sampler

The optimal simulation length for drawing a new sample in HMC should satisfy two requirements: first, it should be sufficiently long enough to prevent locality; second, it should not be too long to make the trajectory trace back, resulting in computation waste. NUTS combines slicing sampling and HMC to eliminate the need of tuning step length by first defining "U-Turn".

1) U-Turn Definition:

NUTS defines a "U-Turn" as when the Euclidean norm between the newly simulated state (θ^{new}, r^{new}) and the starting state (θ^{old}, r^{old}) begins to decrease, i.e. its derivative is smaller than 0, as shown in eq. (3):

$$\frac{1}{2}d((\theta^{new} - \theta^{old})^2) = (\theta^{new} - \theta^{old}) \cdot r^{new} < 0 \quad (3)$$

U-turn detection in NUTS is in fact a vector multiplication and comparison as illustrated in a 2-D example in Fig. 3. When the position and momentum vector are making an angle of more than 90 degree, a U-turn is said to be encountered.

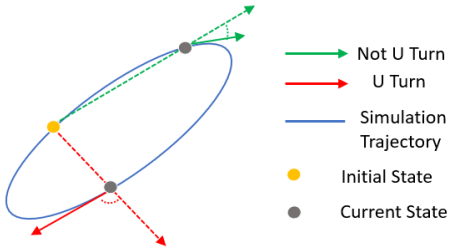

Fig. 3. Example of U-turn during the leapfrog simulation.

Unfortunately, directly applying this criteria to HMC will not satisfy time reversibility which violates detailed balance. Thus, a balanced tree structure is introduced to help correctly sample from the target distribution.

2) Balanced Tree:

A balanced tree $(\mathcal{B})$ is constructed recursively by running leapfrog method in random forward or backward directions, while doubling the length in each direction, until a stopping criteria is met to preserve time reversibility. An example of building a balanced tree is shown in Fig. 4.

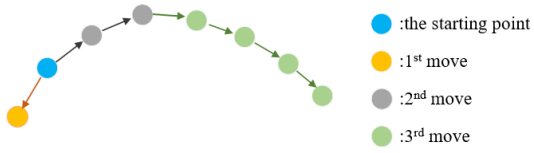

Fig. 4. An example of tree doubling process. Each leaf in the balanced tree corresponds to a simulated state by leapfrog method. First moves left for 1 step, then right for 2 steps, then right for 4 steps. The direction is randomly chosen.

In the balanced tree with height i, there are i subtrees and a top full balanced tree, each has its left most and right most leaf state as (θ^-, r^-) and (θ^+, r^+). U-turn in this case is defined as, when any of the subtrees or the top tree satisfies the condition in eq. (4):

$$\begin{aligned}(\theta^+ - \theta^-) \cdot r^- < 0, or \\ (\theta^+ - \theta^-) \cdot r^+ < 0\end{aligned} \quad (4)$$

3) Slice Sampling and Valid Sample Set:

A valid sample set $(\mathcal{C})$ is constructed by samples from $(\mathcal{B})$ which satisfies the requirement of slice sampling. Given the root node of $(\mathcal{B})$ as (θ^0, r^0), and the logarithm of the distribution $(\mathcal{L}(\theta))$, a slice variable (u) is first drawn as in eq. (5):

$$u \sim Uniform[0, exp(\mathcal{L}(\theta^0) - \frac{1}{2}r^0 \cdot r^0)] \quad (5)$$

For any state (θ, r) in $\mathcal{B}$, if eq.(6) is satisfied, the state will be added to $\mathcal{C}$.

$$log(u) \leq \mathcal{L}(\theta) - \frac{1}{2}r \cdot r \quad (6)$$

4) Uniform Sampling from Valid Sample Set:

Once the valid sample set $(\mathcal{C})$ is constructed, the final sample can be drawn uniformly from $\mathcal{C}$. However, the size of valid sample set grows exponentially with the balanced tree depth i and may consume unacceptable amount of memory. Therefore, in practice, NUTS samples from the valid set during the process of constructing $\mathcal{B}$ by recursively draw samples from each subtree with weights (which is the number of valid samples in that subtree). This is proved to be equal to drawing a sample uniformly from the final valid sample set. A detailed example will be provided in later section.

5) Stopping Criteria:

The tree stops expanding when either one of the two conditions are met. First, a U-turn is detected in $\mathcal{B}$ as in eq.

(4). Second, the simulation error is sufficiently large as shown in eq. (7), where Δ_{max} is a large positive value.

$$log(u) - \Delta_{max} \leq \mathcal{L}(\theta) - \frac{1}{2} r \cdot r \tag{7}$$

6) Adaptively Tuning ϵ:

NUTS also adaptively tunes ϵ during the burn-in period before the Markov chain converges to its stationary distribution with dual averaging algorithm [5]. The algorithm is provided as in eq. (8):

$$\begin{aligned} H_m &= (1 - \frac{1}{m + t_0}) H_{m-1} + \frac{1}{m + t_0}(\delta - \frac{\alpha}{n_\alpha}) \\ log(\epsilon_m) &= \mu - \frac{\sqrt{m}}{\gamma} H_m \\ log(\epsilon_m^{'}) &= m^{-\kappa} log(\epsilon_m) + (1 - m^{-\kappa}) log(\epsilon_{m-1}) \end{aligned} \tag{8}$$

Where m is simulation steps, $t_0, \delta, \gamma, \kappa$ are empirical parameters, α, n_α are values derived during building $\mathcal{B}$. Since it is not the major computation overhead in NUTS, optimizing this algorithm is not within the scope of our accelerator design.

As a summary, after ϵ is tuned, the key steps for drawing one sample in the NUTS algorithm can be summarized as below:

a. Draw slice variable u with initial state (θ^0, r^0).
b. Run multiple leapfrog simulations and recursively build a balanced tree ($\mathcal{B}$) while repeatedly checking stopping criteria.
c. Construct valid sample set ($\mathcal{C}$) from the balanced tree.
d. Uniformly draw a new sample $\theta^{'}$ from the valid sample set.
e. Redraw momentum $r \sim \mathcal{N}(0, 1)$.

While NUTS eliminates the need to manually tuning parameters and achieves highest efficiency. The complexity of this state of art algorithm, both in its sequential computation nature and high memory requirement, brings challenges to its accelerator design on FPGA. Detailed analysis of these difficulties and corresponding solutions are discussed in the next section.

III. Algorithm and Hardware Co-optimization of NUTS

Samples in the NUTS algorithm are drawn from a Markov chain where a certain new sample has dependency on the previous sample. This sequential data processing flow brings challenge to the general parallel acceleration of the algorithm. One of the common practices is to run multiple Markov chains. While some algorithms require the data sharing and exchanging between different chains running at different temperature [9], the NUTS benefits from the data independency between different chains and simplifies multi-chain sampling. On the FPGA platform, however, it is not suitable to directly adopt the idea, since we are encountering unique challenges which we summarize as two key points:

- Non-linear pipeline problem due to feedback dataflow.
- Inefficient memory utilization and management for storing intermediate vectors.

To deal with these challenges, we propose three solutions for the design of NUTS accelerator:

- Fitting multiple sampling chains in a single PE pipeline to increase throughput and hardware utilization.
- Utilizing our proposed incremental resampling technique for efficient memory utilization and management.
- Leveraging three levels of parallelism to further increase the throughput.

The overview of the NUTS accelerator architecture with our proposed ideas is shown in Fig. 5. The NUTS accelerator consists of a processing element (PE) array where each PE contains multiple computational modules and local memories with multiple different chains running in a same pipeline. The leapfrog module utilizes eq. (2) to update θ and r. The sample validation module uses eq. (6) to determine whether the current sample is valid or not. The U-turn detection module checks if a U-turn is already encountered with eq. (4). The intermediate memory temporarily stores the newly simulated samples and momentum vectors requested by the U-turn detection module. The valid sample memory efficiently stores the valid samples with our proposed incremental resampling technique. The temporary newly simulated samples will continuously be sent back to the leapfrog module after sample validation and U-turn detection until the stopping criteria is met. After that, a sample from a certain chain will be uniformly drawn from the valid sample memory with our proposed incremental resampling technique.

The design challenges and our proposed solutions will be discussed with more details in the later section.

For better explanation, we use a reduced vanilla NUTS PE which simulates a single Markov chain with only key components that are relevant to data flow and memory issue as shown in Fig. 6. The sample evaluation module combines the sample validation module and the U-turn detection module in Fig. 5 together. We assume the processing latency of leapfrog module and the evaluation module is t_l and t_e.

A. Non-linear pipeline problem due to feedback dataflow

In the vanilla PE architecture, the time for simulating and evaluating a new sample is $t_l + t_e$. The leapfrog module cannot take any new inputs before the current sample is evaluated. As a result, the whole pipeline in the PE is functioning at low efficiency with a period of $t_l + t_e$. In this case, if we simply adopt the multi-chain sampling idea by duplicating multiple PEs to run independent chains, all PEs will not be functioning at maximum throughput, resulting in extremely low utilization of computational resources. To address this issue, we propose that since different chains in NUTS are data independent from each other while the computation flows are exactly the same, the computation of different chains can be fitted into a single pipeline. In other words, while the pipeline input (the leapfrog module) is waiting for the new sample to be simulated and evaluated, samples from other chains can be taken in and processed. The pipeline can continuously take inputs as simulating a new Markov chain until the new sample of the first chain is evaluated and sent back to the

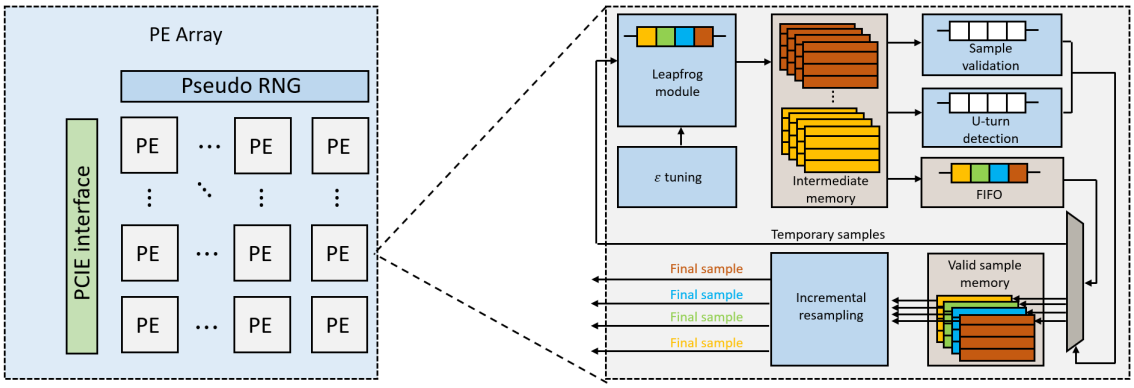

Fig. 5. Overview of the NUTS accelerator architecture with our proposed ideas. In the PE architecture, square blocks with different colors represent samples from different chains.

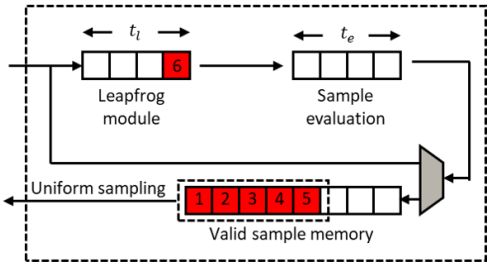

Fig. 6. The vanilla PE architecture for NUTS accelerator. The red block indicates a sample in the current chain and the number represents the sequence of samples.

input. As shown in Fig. 7 (left). The benefit of this multi-chain sampling within a single pipeline is that it increases the pipeline throughput and sample generation rate while utilizing the same amount of computational resources. However, it also brings the requirement for additional storage of valid sample sets from all different chains, which brings another need for efficient memory utilization.

B. Inefficient memory utilization and management for storing intermediate vectors

The valid sample set in the NUTS algorithm stores all valid samples from the balanced tree during the simulation until the stopping criteria is met. Since the tree depth i is usually growing with the problem dimension D, the total number of valid samples can be as large as 2^i, resulting in exponential growing in the memory requirement. As a result, NUTS utilizes its balanced tree structure to reduce the number of intermediate vectors that needed to be stored to $O(i)$. Here, we use a balanced tree with 8 samples, i.e. the tree depth is 3 to illustrate how NUTS samples from the valid sample set as shown in Fig. 8 (left). The top tree with 8 samples is divided into two smaller subtrees, which are further divided into 4 smaller subtrees with 2 samples each. The higher level trees will perform weighted resampling from the lower level trees as explained in the bottom of Fig. 8. During the simulation, the samples are coming sequentially and the actual sampling process of NUTS in memory is shown in Fig. 9.

However, the sampling method of NUTS is not suitable for FPGA implementation. On the one hand, the number of samples to be stored is $O(i)$, while the tree heights vary from different distributions or different simulation points within a same distribution. Previous experiment results show that the maximum tree height can be different from 2 to 20 [3], with no upper bound guaranteed. This randomness in tree height prevents allocating optimal memory resources and brings challenge in the potential over-utilization of on-chip memory, especially when we are utilizing multi-chain sampling. On the other hand, while we cannot benefit from the recursive balanced tree on FPGA, this method is also bringing challenge in the expensive memory management for large number of high dimensional vectors. As shown in Fig. 9, to perform the method in NUTS, the accelerator needs to continuously keep track of the following information: the valid sample numbers (weights) of all the subtrees, the available address of the memory where the new samples should be stored. Additionally, the number of weighted resampling actions is also different when different sample arrives, varying from 0 to 3 in the example. These are all bringing difficulties in the design of control logic and management of the temporary memory. The PE architecture with multi-chain sampling in a single pipeline and the NUTS method to sample from valid sample set is shown in Fig. 7 (middle).

To deal with the difficulty mentioned above, we propose an incremental resampling technique for the required uniform sampling from the valid sample set to provide additional memory efficiency and better memory management. The key idea is that we don't want to perform uniform sampling until the valid sample set is fully constructed, which typically requires huge amount of memory, but to keep and drop samples during the simulation of the Markov chain while making sure that every sample should have an equal chance of being selected as long as it is valid. To achieves this, suppose the number of valid samples are n before a new sample arrives, if we accept the new sample with probability $\frac{I}{I+n}$, where I is 1 if it's valid or 0 if not. It is easy to prove that if we follow this process across the simulation, all valid samples will have an equal chance of being selected. An illustration of our proposed idea

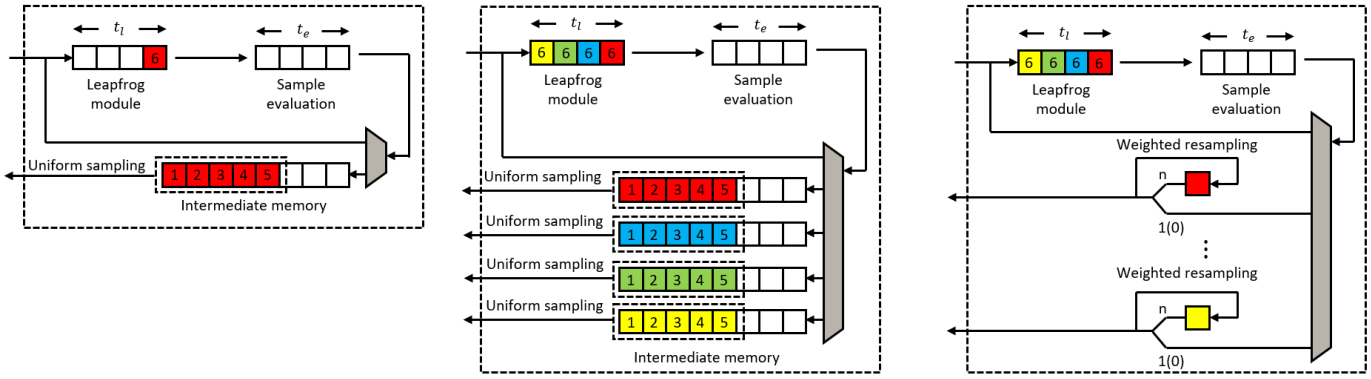

Fig. 7. Different PE architectures. The left adopts multi-chain sampling within a single pipeline. The middle adopt multi-chain sampling and uses the NUTS sampling method to sample the valid set. The right adopt multi-chain sampling and our incremental resampling technique to sample the valid set. The blocks in different colors imply samples from different chains.

for the same previous example is shown in Fig. 8 (right). Our proposed method not only requires to store only 1 temporary sample per chain, but also only needs to keep track of the total valid sample numbers, which eliminates the need for additional complex logic for memory management. The PE architecture with multi-chain sampling in a single pipeline and our proposed method to sample from the valid sample set is shown in Fig. 7 (right).

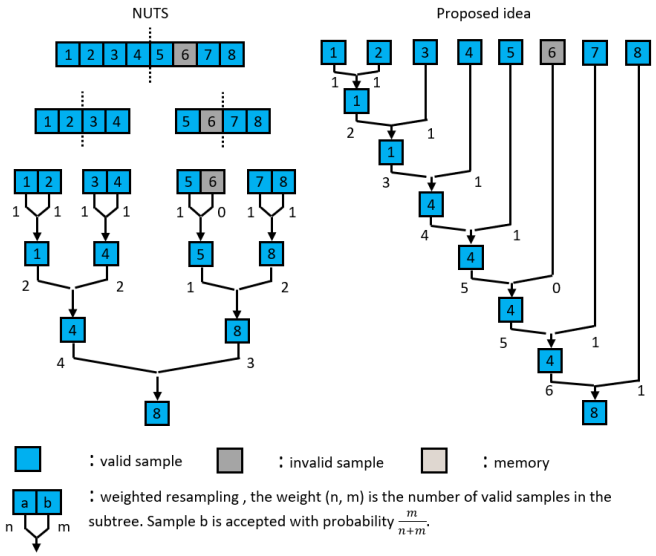

Fig. 8. The sampling techniques to uniformly draw samples from the valid sample set in NUTS (left) and in our proposed idea (right).

C. Three levels of parallelism in NUTS accelerator design

To accelerate the NUTS algorithm for high dimension problems, the design effort should also focus on the high volume vector processing including vector and vector / matrix multiplication, vector and vector addition, etc. With this, we can extract three levels of parallelism in the design of NUTS accelerator. The lowest level includes acceleration of vector processing by unrolling different dimensions, utilizing multiplier-adder trees, etc. The second level of parallelism fits

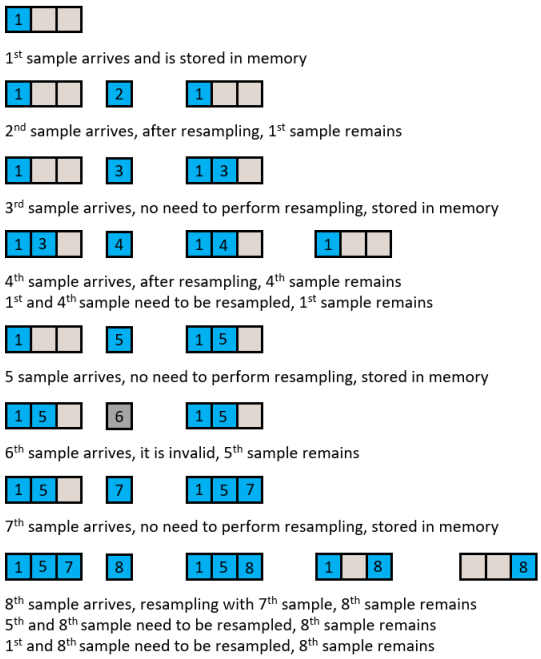

Fig. 9. The sampling process in NUTS for the example in Fig.8 with a memory that can hold 3 samples.

multiple chains in a single pipeline within the PE to maximize the hardware utilization. On the top level, multiple PEs are running independently to further increase the throughput.

IV. HARDWARE IMPLEMENTATION

While the overall architecture is shown in Fig. 5, for the hardware implementation on FPGA, the rich parallel computation resources and block RAM (BRAM) can be further utilized to improve the performance. The high volume vector processing requirement can be speed up by unrolling multiple dimensions for parallel processing or utilizing the multiplier-adder tree structure. The large number of intermediate vector results should also be stored in multiple small blocks of

BRAMs to allow parallel accessing of different dimensions of the vectors and further reduce the data accessing latency.

V. EXPERIMENTAL SETUP

For high dimensional distributions, sampling is becoming extremely painful due to the increasing exploration difficulty. Drawing samples from multivariate Gaussian distributions is also becoming a challenging task. As a result, they are usually utilized to evaluate the efficiency of different sampling algorithms [3]. In our paper, we also use highly correlated multivariate Gaussian distributions with different dimensions (128, 64, 32, 16) as the target distributions. Their covariance matrices are generated by the Wishhart distribution with identity scale matrix and degree of freedoms as 128, 64, 32, and 16, respectively, as suggested by the experiment setup in [3]. The inverse of the covariance matrices are pre-calculated and stored in the local BRAM of the NUTS accelerator.

The NUTS accelerator is implemented on Xilinx VCU-118 evaluation kit with single precision floating point representation at 200MHz. To evaluate our proposed architecture, 4 designs are implemented with the following setup: design 1 is the baseline accelerator design without the multi-chain sampling and the efficient memory architecture. Design 2 implements the multi-chain sampling but does not implement the efficient memory architecture. The maximum balanced tree depth of design 1 and design 2 is set to be 20. Design 3 implements both multi-chain sampling and the efficient memory architecture. All three designs are using a multiplier-adder tree with a depth of 8, which can process vector multiplication with a dimension of 64. Design 4 additionally explores the PE level parallelism by using multiple PEs with both multi-chain sampling and the efficient memory architecture. The depth of the multiplier adder trees in design 4 is 7 and the number of PEs is 2, to keep the computation resources the same as in design 3. All the designs limit the number of chains running to be 4. The samples of all designs are collected through a PCIE interface and the power is reported by Vivado 2019.2 including both static and dynamic power.

The R interface for highly optimized C++ NUTS sampling library: RSTAN, is used as software measurement which samples from the same multivariate Gaussian distributions. The number of chains is set to be 4 and the same number of samples (8000) are drawn as in the hardware implementation. RSTAN is running on an Intel I7-8750H CPU @ 2.2GHz, boost 3.7GHz and the power is reported with Intel power Gadget 3.6. The average power of the software routine for inputs with 128 dimension to 16 dimension are 33.46W, 24.09W, 17.99W, 14.83W respectively.

VI. EXPERIMENT RESULTS AND ANALYSIS

The resource utilization of all designs is reported in Table. I. Compared with design 1 and design 2, fitting 4 chains in a single pipeline results in the same logic and computational resource utilization but with additional 4X memory consumption. Compared with design 2 and design 3, our proposed incremental resampling technique provides additional saving in logic resources (6.8% saving in LUT, 7.3% saving in FF) and 13.9% in BRAM.

The execution time, normalized speedup and energy improvements compared to RSTAN are summarized in Table. II. Comparison between the speedup and the energy improvement between different designs are shown in Fig. 10 and Fig. 11. The speedup difference between design 2 and design 3 comes from the randomness during the simulation. In design 3, the number of chains for all inputs is limited to 4 while for lower dimension problems, the number of chains that can be accommodated can be larger. In this case, the pipeline is not functioning at maximum throughput and shows an decrease in speedup in lower dimension problems. In design 4, the input with 128 and 64 exceed the capacity of multiplier adder tree and vector processing array, thus, the computation of vector operations takes additional time cycles. This is the reason why we are not seeing expected 2X performance boost in these two cases.

The results also shows that implementing multi-chain sampling will bring up to additional 4X speedup (this may vary because of the randomness during NUTS simulation). We also demonstrate that leveraging the PE level parallelism by using 2 PEs as in design 4, while utilizing the roughly same amount of logic and computation resources as in design 3, can bring additional speedup of up to 50.6X and energy efficiency of up to 189.7X compared with RSTAN.

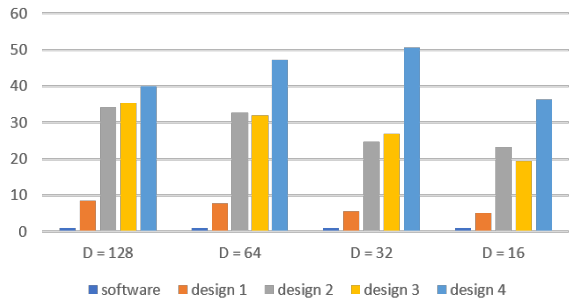

Fig. 10. Normalized speedup compared with RSTAN.

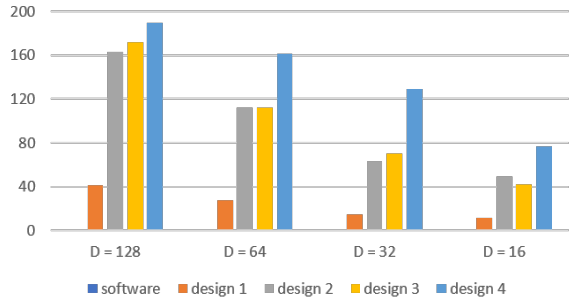

Fig. 11. Normalized energy improvement compared with RSTAN.

For sample evaluation, we compare 4000 samples from the 64 dimension target distribution which are collected from

TABLE I
RESOURCE UTILIZATION (%) OF FOUR DESIGNS.

Design	LUT	LUTRAM	FF	DSP	BRAM	Power (W)
Design 1	11.89	2.46	7.82	18.8	2.06	6.87
Design 2	12.92	2.86	8.81	18.8	8.21	7.01
Design 3	12.04	2.66	8.21	18.8	6.42	6.88
Design 4	13.23	2.89	9.33	18.9	12.84	7.04
Total	1182240	391840	2364480	6840	2160	\

TABLE II
EXECUTION TIME (S) OF NUTS ACCELERATOR, NORMALIZED ACCELERATION AND ENERGY IMPROVEMENT COMPARED WITH RSTAN FOR THE SAMPLING OF DIFFERENT MULTIVARIATE GAUSSIAN DISTRIBUTIONS.

	Execution time (s)				Normalized acceleration				Energy improvement over RSTAN			
	128-D	64-D	32-D	16-D	128-D	64-D	32-D	16-D	128-D	64-D	32-D	16-D
RSTAN	1164.1	67.9	32.1	4.4	1	1	1	1	1	1	1	1
Design 1	138.3	8.7	5.6	0.85	8.4	7.8	5.7	5.2	40.9	27.4	14.9	11.2
Design 2	34.0	2.07	1.3	0.19	34.2	32.8	24.7	23.2	163.3	112.7	63.4	49.1
Design 3	32.9	2.13	1.19	0.23	35.3	32.1	26.9	19.4	171.7	112.4	70.3	41.8
Design 4	29.2	1.43	0.63	0.12	39.9	47.3	50.6	36.4	189.7	161.9	129.3	76.7

RSTAN and the NUTS accelerator. Two dimensions are selected for illustration and the plot is shown in Fig. 12.

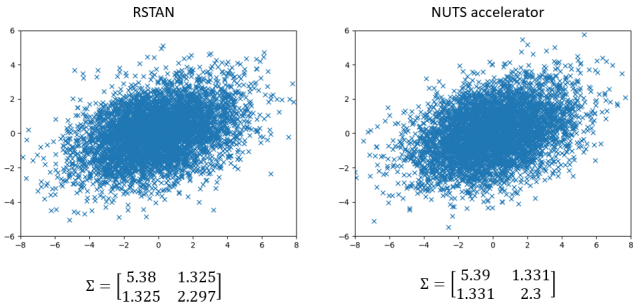

Fig. 12. Scatter plot of random two dimensions of the 4000 samples from the 64 dimension multivariate Gaussian distribution, the left plot is from RSTAN, the right plot is from NUTS accelerator

VII. CONCLUSION

In this paper, we present the first NUTS accelerator for the state of art algorithm on FPGA with hardware algorithm co-optimization on two aspects: first, we introduce multi-chain sampling in a single PE pipeline to maximum the throughput and hardware utilization. Second, we propose an incremental resampling technique to provide better memory efficiency and better memory management compared with standard NUTS algorithm. We also leverage three levels of parallelism in our hardware design to further improve the performance. Results show that our proposed ideas can lead to 4X additional speedup with multi-chain sampling and 13.9% saving in memory resources. Compared with optimized C++ library RSTAN, our design can achieve speedup up to 50.6X and energy improvement up to 189.7X.

VIII. ACKNOWLEDGEMENT

This paper is based upon work supported by the National Science Foundation (NSF) under Grant No.1741173. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953. [Online]. Available: http://link.aip.org/link/?JCP/21/1087/1

[2] R. Neal, "Mcmc using hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, 06 2012.

[3] M. D. Homan and A. Gelman, "The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1593–1623, Jan. 2014.

[4] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin, "Parallel gibbs sampling: From colored fields to thin junction trees," ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, 11–13 Apr 2011, pp. 324–332. [Online]. Available: http://proceedings.mlr.press/v15/gonzalez11a.html

[5] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Mathematical Programming*, vol. 120, pp. 221–259, 04 2009.

[6] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of Statistical Software, Articles*, vol. 76, no. 1, pp. 1–32, 2017. [Online]. Available: https://www.jstatsoft.org/v076/i01

[7] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *Journal of Machine Learning Research*, 2018.

[8] S. Liu, G. Mingas, and C. Bouganis, "An unbiased mcmc fpga-based accelerator in the land of custom precision arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 745–758, 2017.

[9] G. Mingas and C.-S. Bouganis, "Parallel tempering mcmc acceleration using reconfigurable hardware," in *Reconfigurable Computing: Architectures, Tools and Applications*, O. C. S. Choy, R. C. C. Cheung, P. Athanas, and K. Sano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 227–238.

[10] B. Rouhani, M. Ghasemzadeh, and F. Koushanfar, "Causalearn: Automated framework for scalable streaming-based causal bayesian learning using fpgas," 02 2018, pp. 1–10.

[11] T. Chen, E. B. Fox, and C. Guestrin, "Stochastic gradient hamiltonian monte carlo," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. II–1683–II–1691.

[12] R. M. Neal, *Bayesian Learning for Neural Networks*. Berlin, Heidelberg: Springer-Verlag, 1996.