Discrete Samplers for Approximate Inference in Probabilistic Machine Learning

Shirui Zhao*, Nimish Shah*, Wannes Meert†, Marian Verhelst*
*MICAS-ESAT, KU Leuven, †DTAI, KU Leuven
{shirui.zhao, nimish.shah, wannes.meert, marian.verhelst}@kuleuven.be

***Abstract*—Probabilistic reasoning models (PMs) and probabilistic inference bring advantages when dealing with small datasets or uncertainty on the observed data, and allow to integrate expert knowledge and create interpretable models. The main challenge of using these PMs in practice is that their inference is very compute-intensive. Therefore, custom hardware architectures for the exact and approximate inference of PMs have been proposed in the SotA. The throughput, energy and area efficiency of approximate PM inference accelerators are strongly dominated by the sampler blocks required to sample arbitrary discrete distributions.**
This paper proposes and studies novel discrete sampler architectures towards efficient and flexible hardware implementations for PM accelerators. Both cumulative distribution table (CDT) and Knuth-Yao (KY) based sampling algorithms are assessed, based on which different sampler hardware architectures were implemented. Innovation is brought in terms of a reconfigurable CDT sampling architecture with a flexible range and a reconfigurable Knuth-Yao sampling architecture that supports both flexible range and dynamic precision. All architectures are benchmarked on real-world Bayesian Networks, demonstrating up to $13\times$ energy efficiency benefits and $11\times$ area efficiency improvement of the optimized reconfigurable Knuth-Yao sampler over the traditional linear CDT-based samplers used in the PM SotA.

***Index Terms*—Probabilistic models, approximate inference, discrete sampling, CDT algorithm, Knuth-Yao algorithm**

I. Introduction

Many fields like robotics, biomedicine, finance, etc., have requirements unmet by the traditional "black-box" deep learning, such as ① the need for decision-making under uncertainty, ② for interpretable results, ③ a lack of human-annotated datasets, ④ imbalanced datasets, etc. Probabilistic models (PMs: e.g. Probabilistic Programs, Markov Networks, Bayesian Networks (BNs), etc.) and probabilistic inference have been shown as important techniques that can elegantly deal with these requirements, as they allow integrating experts' knowledge and quantify uncertainty. Many probabilistic programming languages (PPLs) [1] have been developed to ease the task of constructing models and deal with complicated inference tasks. PPLs represent PMs at a high level of abstraction and automatically perform the reasoning through exact or approximate inference. However, both the exact and approximate probabilistic inference has been shown to be computationally expensive [2]. Running probabilistic models on edge devices is hence extremely challenging in terms of real-time and power constraints. Sampling-based approximate inference has been shown to be a general and efficient solution, due to its potential to fulfill the real-time constraints for very large models [3]. The accuracy of sampling algorithms depends primarily on the sample size. As a result the random variables in the PM need to be sampled millions of times to achieve high accuracy results.

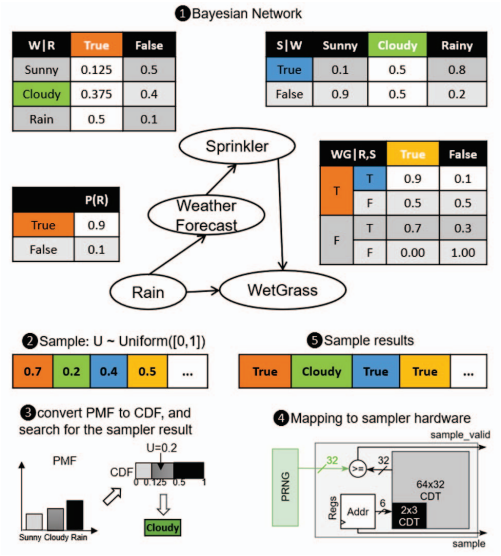


Fig. 1: Toy example of mapping the BNs on the existing SotA hardware

To illustrate how discrete samplers are used in approximate PM inference, we present an example of a forward sampling strategy for BNs in Fig. 1. The sampling algorithm follows the topological order of the network. Given the value of a node's parents, the corresponding discrete distribution $P(x|parents(x))$ is derived to do the sampling. The process is repeated million times until the result is converged. As such, the core operation in this sampling-based technique is to generate random samples from the discrete distributions, for which the cumulative distribution table (CDT) sampling algorithm is typically used [1]. More complicated inference strategies for other probabilistic programs and models similarly rely on the usage of discrete samplers [4], and can also exploit the results of this work.

To accelerate the time-intensive sampling process, many SotA hardware architectures have been developed, aiming at mapping the sampling task onto optimized fixed-point hardware. BIA [5], PMBA [6], implemented fixed-point discrete CDT samplers with linear search by making use of a Pseudo-Random Number Generator (PRNG). Yet, these architectures are based on sequential methods which result in low throughput performance. Moreover, these implementations waste resources when there is

no need to represent the probability data with high precision or big range.

Therefore, this paper presents and implements novel sampler hardware architectures based on the CDT and Knuth-Yao sampling algorithms [4], including area/power-efficient **reconfigurable samplers** for any discrete distributions with dynamic precision. The main contributions of the paper are:

- Different sampler algorithms and hardware implementations that demonstrate trade-offs between throughput, area, and power discussed in Section II.
- Proposed optimized reconfigurable sampler designs to enable flexible range and dynamic precision presented in Section III.
- Comparisons to SotA CDT sampler on real-world Bayesian Networks in Section IV.

Then, we conclude our work in section V.

II. Background and SotA on sampling

Discrete probability distributions describe the probability of a random variable to have one of N possible values. The discrete distribution with probability mass function (PMF) is noted as $\mathbf{p}[i] = p_i$, where $i \in \{0, ..., N-1\}$, in which the probabilities sum to 1. To avoid the cost of floating-point computation, fixed-point sampling is commonly used in low-power edge devices [3] and custom ASIC designs [5], [7]. The fixed-point precision of the distribution is denoted as k and it is commonly set to 32-bits in most PM designs mentioned. We can construct an $N \times k$ dimensional binary probability matrix P from the PMF $\mathbf{p}$, where $P[i, j]$ is equal to the jth bit of p_i. We denote N as the range of the random variable and k as its precision in following discussion.

A. CDT-based sampling

CDT sampling is the most commonly used sampling algorithm (see ❸ in Fig. 1), consisting of: ① generating the cumulative distribution function (CDF) table from the Probability Mass Function (PMF) with $F[j] = \sum_{i=0}^{j} p_i$, where $j \in \{0, ..., N-1\}$. ② draw a uniformly distributed random number from PRNG. ③ Finally, searching for the largest entry in the CDF table that is smaller than the random number. This entry is the output sample. Assume $Rain$ is $True$ in the toy example in Fig. 1, and we need to sample from the PMF of $P(W|R = True) = [0.125, 0.375, 0.5]$ for the node of $WeatherForecast$. Because U is between 0.125 and 0.5, thus the sample result should be $Cloudy$.

When sampling from a discrete distribution with range size of N, the bottleneck of CDT sampling is the search step to locate the interval corresponding to the random number in the CDT. Different methods can be used here: a linear search with time complexity of $O(N)$, a binary search with the complexity of $O(\log N)$, a parallel search with the complexity of $O(1)$.

Fixed-point CDT samplers are widely used in the SotA ASIC PM accelerators, for example, BIA [5] and PMBA [6]. But they employ the linear search method resulting in low throughput performance. Moreover, SotA samplers implement a fixed (worst case) range N and precision k to generate one sample, resulting in a huge area, power and latency waste. As the example in Fig. 1 shows, only a CDF table of size 2×3 needs to be stored for the $WeatherForecast$ node (the last value in the CDF table will always be 1 and does not need storage).

CDT samplers with binary search [8] and parallel search [9] are developed in the cryptographic community. To avoid timing attacks, the binary search method was designed to sample a discrete Gaussian distribution for lattice-based cryptography in constant time (8 cycles). It has a CDF table size of 180×64, and two binary search modules. MePLER [9] provided an in-memory CDT sampler with parallel search only support fixed range and precision. They show time/resource waste and a lack of flexibility for any discrete distribution, which is not suitable for PMs.

B. Knuth-Yao based sampling

Algorithm 1 Sampling using Knuth-Yao (KY) algorithm

INPUT: Given discrete distribution as matrix **P** with dimension of **N**×**k**
OUTPUT: Sample of R_c

```
d ← 0                                   ▷ init distance
row ← 0                                 ▷ init row
col ← 0                                 ▷ init column
for col = 0 to k do
    rb ∼ UniformInt(0, 1)               ▷ draw 1 random bit
    d ← 2d + (!rb)
    for row = 0 to N do
        d ← d − P[row][col]
        if d == −1 then
            return row
```

Knuth and Yao [10] presented an improved method to obtain random samples from any discrete distribution, which is shown in algorithm 1. Differ with the CDT-based sampling which traverses the probability matrix row-wise and required fixed PRNGs random bits, this algorithm searches the probability matrix in a column-first way, which leads to less random bit consumption in general. The random bits required per sample are between $H(\mathbf{p})$ and $H(\mathbf{p}) + 2$, where $H(\mathbf{p})$ is the Shannon entropy $\sum_{i=1}^{n} p_i \log(1/p_i)$ [4]. This reduction in random bit consumption turns out to be a significant advantage as shown in our experiments later, since it reduces the PRNG area overhead and runtime on low-precision distributions.

The Knuth-Yao (KY) sampling uses a Discrete Distribution Generating (DDG) tree data structure. Fig. 2 illustrates an example of such a tree, in which each node has two children, and the leaf nodes are the sampling results.

Construction of the DDG tree: The DDG tree can be directly constructed from the probability matrix [7]. As shown in the example, the corresponding probability matrix for the node $WeatherForecast$ is shown in Fig. 2(a). Since there is a 1 in row $Rain$ at level 1, the nodes of level 1 in the DDG tree will be denoted as [$Rain$, $Internal$] from the right to left. Next, the internal node is split into two child nodes corresponding to level 2. Thus [$Cloudy$, $Internal$]. Finally, for level 3, there

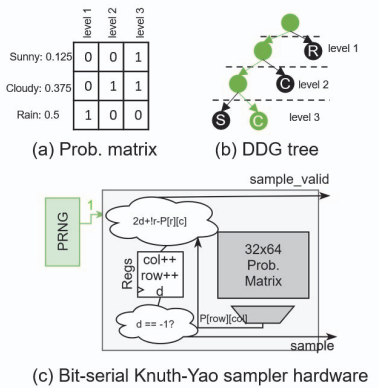


Fig. 2: The Knuth-Yao based discrete sampling algorithm and diagram of sampler hardware implementation from work [7]

are two rows with a 1, thus both child nodes receive a label, [$Cloudy$, $Sunny$].

DDG sampling: After we have obtained the DDG tree, the sampling procedure is as follows: ① starting at the root node, if a uniformly drawn random bit is 0, go to the left, otherwise, go to the right; ② When a leaf node is reached, return its label as the sample result. For example, assume the same random number U=0.2 as the CDT sampling shown in Fig. 1, the same outcome is achieved by Knuth-Yao sampling. As the first three random bits are $[0, 0, 1]$ in U, following the green path, the DDG tree in Fig. 2 will return the same sample $Cloudy$ as the CDT sampling. Compared to CDT sampler with 32 random bits, the Knuth-Yao sampler only consumes a maximum of 3 random bits for this DDG tree.

A hardware implementation of the Knuth-Yao sampler [7] from the cryptographic community is shown in Fig. 2 (c). This implementation will scan the probability matrix bit sequentially which shows low throughput performance. We will refer to this implementation as the bit-serial Knuth-Yao sampler. The cryptography community focuses on unpredictable randomness and constant time execution, and they only need support for discrete Gaussian distributions. The constant time Knuth-Yao based hardware samplers [11] [12] were proposed to gain security only for discrete Gaussian distributions. This results in different challenges and opportunities compared to those seen in PM sampler design.

C. Sampler specification for PMs

For PMs, we need to support any discrete distribution. Moreover, as we care more about sampling efficiency rather than security, non-constant time execution and dynamic random bits consumption can be utilized to gain performance. Therefore, we aim towards a sampler design with flexible range and dynamic precision, without sacrificing throughput performance.

III. Sampling Implementations

We present two novel parallel sampling hardware designs, based respectively on CDT sampling and Knuth-Yao sampling. These novel hardware samplers could be configured as multi-parallel samplers for distributions that have a smaller range than the maximally supported range. Our implementations optimize the throughput performance and reduces unnecessary resource usage.

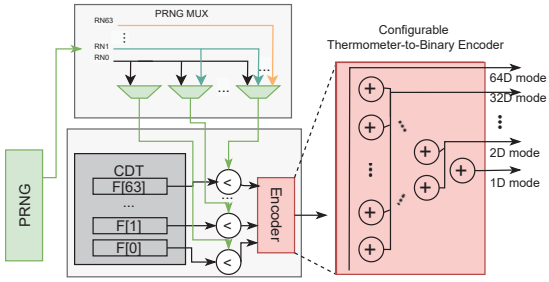


Fig. 3: Diagram of the reconfigurable CDT sampler with parallel search

A. CDT sampler

We propose a range reconfigurable CDT sampler architecture with the parallel search method, as shown in Fig. 3. The sampler contains 64 32-bits comparators. We can configure the sampler at different work modes for the distribution with different range. The mode indicates the maximal number of distributions that is supported to run at same time, which is denoted as $64D$, $32D$, $16D$, $8D$, $4D$, $2D$ and $1D$ for random variables with maximum range size of 2, 3, 5, 9, 17, 33 and 65, respectively.

For a random variable with the range size between 33 and 65, we configure the sampler in the $1D$ mode. First, we need to store the CDF table of the distribution into the register file and all the registers not used are set as all **1**. The sampler need to access the whole CDT table simultaneously but only one 32-bit uniform random number from PRNG. Then, the sample result is generated by the thermometer-to-binary encoder with the input of the binary results from comparators. To sample from 2-dimensional probability matrix, we can store 64 CDF tables (only $F[0]$ stored per table) inside the register file at once and compare each of them with different uniform random numbers. The comparison results are exactly the samples, which is 64 1-bit samples in parallel.

Thus, we implement the encoder as a reconfigurable adder tree. Each level of the adder tree will be the output of the sample results for different modes.

Therefore, for range smaller than 65, samples can be drawn from one or multiple distributions simultaneously, resulting in high hardware utilization and throughput, irrespective of the range. However, this flexibility needs large PRNG multiplexers because of the variable bandwidth of PRNG, which have an area overhead of $2.5\times$ compared to a fixed-range sampler. Furthermore, the sampler consumes exponentially more random bits from the PRNG as the range size decreases, which requires more PRNG modules in the sampler, causing additional area overhead. The Knuth-Yao based design addresses this with a significantly lower demand of PRNG, as shown in the subsequent subsections.

B. Knuth-Yao(KY) sampler

In this section, we discuss the hardware implementation detail of the novel, reconfigurable, Knuth-Yao based sampler.

Algorithm 2 Column-wise KY algorithm for hardware

INPUT: Given discrete distribution as matrix **P** with dimension of **N**×**k**
OUTPUT: Sample of R_c

```
d ← 0
col ← 0
for col = 0 to k do                          ▷ distance update
    H[col] ← Σ_{i=0}^{N} P[i][col]           ▷ parallel computing
    rb ∼ UniformInt(0, 1)
    d ← 2d + (!rb) − H[col]
    if d < 0 then                            ▷ sample search
        for row = 0 to N do                  ▷ loop unrolling
            if d + H[col] − P[col][row] == −1 then
                return row
```

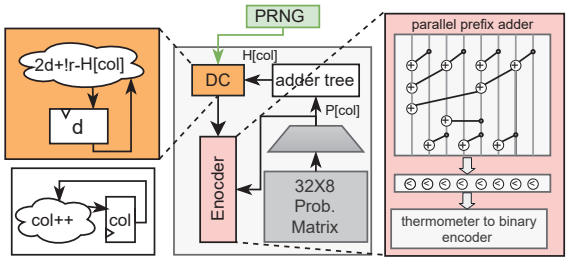


Fig. 4: Column-wise KY sampler hardware architecture

First, we will introduce the techniques used to improve the throughput performance of the bit-serial Knuth-Yao sampler, parallel computing for Hamming weight and loop unrolling for the sample search. To further reduce resource waste when running distributions with a small size, we then propose the reconfigurable Knuth-Yao sampler with flexible range and dynamic precision to improve resource utilization. It should be noted that in this section we use illustrations of a Knuth-Yao sampler with a maximum supported range size of 8 to simplify the drawings. In section IV, we expand the actual samplers with the maximum supported range of 64 for benchmarking.

1) Column-wise Knuth-Yao sampler: Due to the low throughput performance of the bit-serial Knuth-Yao sampler, we need to modify the original algorithm to support parallel computing. The key idea is that if the Hamming weight is used instead of scanning the probability matrix column $P[col]$ in a sequential way, then the throughput will be improved N times (N is the range of the random variable).

As shown in algorithm 2, compared with the original algorithm 1, we can use parallel computing for the Hamming weight and adapt the loop unrolling method for the sample encoder to improve the throughput performance. Thus, the algorithm is modified to be a three-stage process. The first stage is to compute the Hamming weight $H[col]$ of the current column. Then, update distance $d \leftarrow 2d + (!rb) - H[col]$, where rb indicates a random bit from PRNG. And finally, apply loop unrolling on the row loop for the sample search.

The column-wise Knuth-Yao hardware is developed based on the modified KY algorithm. The whole hardware architecture is shown in Fig. 4, which consists of three modules, the adder tree for Hamming weight, the distance computing module, and the encoder for sample search. Because we access the probability matrix in a column-wise way, we transpose the matrix before loading it into memory. The column index adder is used to access the memory.

The **adder tree for Hamming weight** is used to compute the Hamming weight with the N-bit adder tree. The output will have a bit size of $\log(N)$.

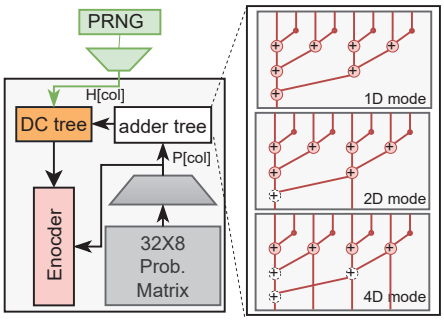
The **Distance Computing module** is developed to update value of distance d register with the input of a random bit rb from PRNG and the Hamming weight $H[col]$. For the hardware implementation, we use the two's complement representation such that we can compute the distance d with an adder, and the carry bit of the addition will be discarded. When the most significant bit (MSB) is equal to 1, this means the updated distance d is the negative value.

The **sample encoder module** is activated when the distance d is smaller than 0. Given the column number col where $d < 0$, the input of the encoder is the N-bits vector from the current column of the probability matrix and the index of the sample result from the distance computing module. The purpose of this stage is to find the location of n-th 1 in the column vector, where $n = d + H[col] + 1$. As shown in the sample encoder fragment of algorithm 2, we can sequentially search the column but with low throughput performance. To further reduce the latency, we apply loop unrolling. The sample encoder contains three parts shown in Fig. 4. The first part is the parallel prefix adders for $P[col]$ which is work-efficient and has a depth of $O(\log N)$, We can reuse the adder tree in the parallel prefix adder for Hamming weight to save resource. The second part is N comparators which compare the index n with the prefix sum results. The last part is the thermometer-to-binary encoder which is implemented as an adder tree. The output of the encoder is the sample result.

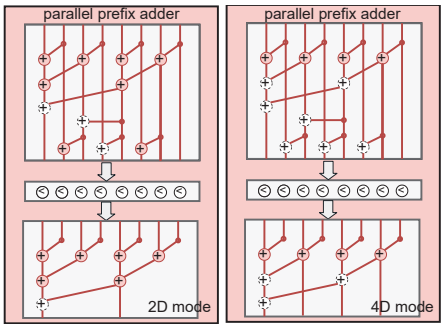
2) Reconfigurable KY sampler with flexible range: We improved the throughput performance of the KY sampler by using parallel computing methods but lost the range flexibility. Because when using the column-wise KY sampler to sample from the distribution with a small probability matrix, many elements in the sampler are wasted. In this section we optimize the design further to support flexible range. The architecture of a reconfigurable sampler with maximum support for range size of 8 is shown in Fig. 5. It can be configured as $1D$, $2D$ or $4D$ sampler for random variable with maximum range size of 8, 4 and 2, respectively.

The **reconfigurable adder tree for Hamming weight** is similar to the column-wise KY sampler, it contains an 8-bit adder tree for the Hamming weight computation. As shown on the right side of Fig. 5, The results of different levels of the adders in the tree are exactly the Hamming weights for different sampling modes.

The **distance computing tree** requires more distance computing blocks for different sampling modes than the cloumn-wise KY sampler. The column-wise KY sampler only contains one distance computing block. To support different modes in our



(a) Overview and adder tree



(b) Configurable encoder

Fig. 5: Diagram of the reconfigurable KY sampler

design, we need the same amount of distance computing blocks that will form as a tree. Here we need 4 extra blocks to support distribution with range size of 2 and 2 blocks for the range size of 4. At different mode, distance computing modules in different levels of the tree will be activated, and they are independent of each other. Accordingly, the bandwidth of the PRNG varies with the mode of the sampler. It requires 4, 2 and 1 bit/cycle for the $4D$, $2D$ and $1D$ mode in the example.

The **reconfigurable sample encoder** contains a configurable parallel prefix adder, comparators and a configurable thermometer-to-binary encoder. The encoder will be activated when the corresponding distance computing module triggers the sample valid signal and the sampler results in the register need to be buffered until all the distributions in the sampler have finished their sampling. When setting the sampler in Fig. 5 to $8D$ mode, all the computing elements in the parallel prefix sum adder, comparators, and the thermometer-to-binary encoder are activated. When it works in the $2D$ or $4D$ mode, some of the adders in the parallel prefix adder and thermometer-to-binary encoder are skipped.

Performance analysis of reconfigurable KY sampler When we configure the sampler as a low-range mode, we scan the column of the mixed probability matrix until all the samples from different distributions are valid. Thus, the latency of the sampler depends on the worst case of all the discrete distributions mapped on the sampler. As we discussed in section II, the random bit per sample is related to the entropy of the distribution. In our reconfigurable KY sampler design, we consume one random bit per cycle, so the latency is also between

Sampler	Peak PRNG bandwidth	Area	Power @100MHz
	bits/cycle	mm^2	uW
linear CDT	32	0.026	1.26
binary CDT	32	0.026	1.27
parallel CDT	32	0.030	1.33
Reconf. CDT	32×64	0.182	7.01
Bit-serial KY	1	0.025	1.41
Col-wise KY	1	0.028	1.38
Reconf. KY	32	0.038	1.62

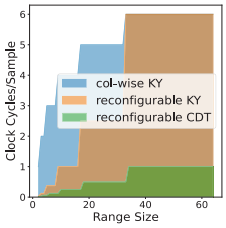


Fig. 6: The samplers hardware information and clock cycles per sample for discrete uniform distributions.

$H(\mathbf{p})$ and $H(\mathbf{p})+2$. Therefore, combining the distributions with similar entropy in one sampler can reduce the overhead. For the forward sampling of the Bayesian Networks, we can sample the same distribution multiple times and then switch to another node, which means the sampler will contain multiple version of the same probability matrix, which will cause less overhead on this issue.

IV. Benchmarking Results

We implemented the architectures of the two newly proposed samplers, together with existing SotA serial and parallel CDT and KY samplers in Verilog and evaluated them with Synopsys 32nm Generic Library (SAED32). The RTL was synthesized with the time constraint of 100MHz under 125°C and $0.7V$ corner by Synopsys Design Compiler 2019.03.

For a fair comparison with our sampler design, we implemented and modified the linear CDT sampler [5] removing its computing logic for the CDF table with the assumption that the CDF table is given. For the binary CDT sampler, we modified the design from [8] with only one binary search module and reduce the table size to 64×32. We implemented the digital counterpart of MePLER [9] as the parallel CDT sampler.

The implemented architectures are analyzed on the discrete uniform distribution and benchmarked a set of real-world Bayesian Networks from an open-source repository [13].

We use the LFSR-based PRNGs in all compared architectures. The area and power results are shown in Fig. 6. To support any distribution in our reconfigurable samplers, the PRNG bandwidth should support the worst case requirement, i.e. 64 parallel variables in CDT sampler and 32 in KY sampler with range size of 2. Compared with the CDT sampler, the KY sampler requires less PRNG bandwidth resulting in reduction of area and power.

In the CDT samplers, the time complexity of the linear, binary, parallel, and reconfigurable sampler is N, $\log N$, 1 and $2^{\lceil \log_2(N-1) \rceil - 6}$, respectively. In KY samplers, the complexity not only depends on the range of the random variables but also the entropy of the distributions, which is $N \times H(\mathbf{p})$, $H(\mathbf{p})$ and $H(\mathbf{p}) \times 2^{\lceil \log_2(N) \rceil - 6}$, for the bit-serial, column-wise and reconfigurable KY sampler, respectively. Thus, the worst case run-time of the KY sampler occurs for discrete uniform distributions. The throughput results are shown in Fig. 6. The reconfigurable KY sampler shows a maximum of $6\times$ less throughput performance than the reconfigurable CDT sampler for uniform distributions and when the range is larger than 32.

TABLE 1: Comparison of mixed-range Knuth-sampler architecture with the SotA. *Note that the throughput and power efficiency in this work are average results on BNs with fixed 32-bit precision. VLSI 20 [5] reported on MRF with extra logic.

	This Work		Swift [1]	VLSI 20 [5]	VLSI 21 [9]	ISSCC 19 [14]	CICC 18 [11]	ToC 21 [12]
Application	Probabilistic machine learning				Cryptography			
Method	**Reconfig. CDT**	**Reconfig. KY**	Linear CDT	Linear CDT	CAM parallel CDT	Linear CDT	Bit-serial KY	Boolean KY
Technology	**32nm**		CPU i7-7800x	16nm	65nm	40nm	40nm	FPGA Virtex-6
Frequency (MHz)	**100**	**100**	3500	451.2	85.9	72	300	291
Memory	**0.25KB**	**0.25KB**	32GB	0.25KB	0.5KB	0.25KB	65KB	-
Area (mm^2)	**0.182**	**0.038**	-	2.99	0.034	0.1	0.4	-
Throughput (Samples/cycle)	**31.35**	**9.14**	0.21	0.2-0.4	1	0.02	0.07	1
Power Eff. (Samples/pJ)	**5891**	**8083**	1×10^{-7}	200-1000	0.03	4×10^{-4}	2×10^{-4}	-
Precision	**32**	**1-32**	FP	32	64	32	256	128
Range	**2,3,5,9,17,33,65**	**2,4,8,16,32,64**	any	64	64	64	12289	82

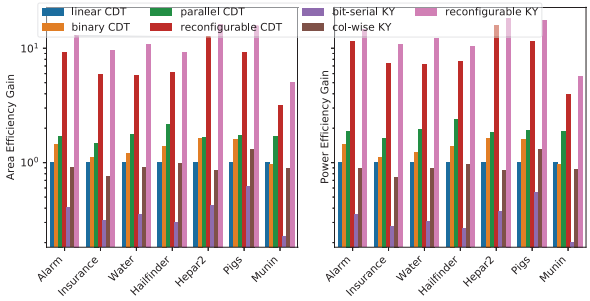


Fig. 7: Area and power efficiency results on BNs, normalized relative to the linear CDT sampler

However, for the real-world Bayesian Networks with dynamic entropy, as shown in Fig. 7, the reconfigurable KY sampler shows the best area/power efficiency, achieving an average improvement of $11\times$ and $13\times$, respectively. The gains are normalized relative to the linear CDT sampler.

Comparisons of SotA implementation are presented in Table 1. Throughput for the CPU is measured by average runtime on different BNs. The floating-point CDT sampling algorithm results in low throughput and extremely low power efficiency (the CPU TDP power of 140 W). Compared with BIA [5] with the 32-bits fixed-point linear CDT sampler, we show a throughput improvement of $45\times$. Compared with the samplers from the cryptographic community, the MePLER [9] presented an in-memory CAM-based sampler that shows low power efficiency and throughput because of a lack of configurability. The bit-serial Knuth-Yao shows low throughput performance and the Boolean function based Knuth-Yao sampler will have a constant sampling time only suitable for FPGA.

The reconfigurable Knuth-Yao sampler shows a better sampler design for the PMs with dynamic range and flexible precision supported. For the distributions with high entropy, multi-column-wise methods could be used to improve the performance further.

V. Conclusion

In this paper, we presented new sampler hardware implementations for approximate inference in probabilistic machine learning. We further compared seven different sampler architectures and proposed the reconfigurable CDT and Knuth-Yao sampler with range flexibility to support any discrete distribution. The Knuth-Yao sampler further supports dynamic precision, which minimizes the requirements for the bandwidth of PRNGs and improves the throughput, area-/power-efficiency performance for probabilistic models. As a result, it can form the core module for future high-performance and low-power PM accelerators.

VI. Acknowledgement

This work has been supported by the EU ERC project Re-SENSE under grant agreement ERC-2016-STG-715037, and we acknowledge support from Huawei Kirin Solution.

References

[1] Y. Wu *et al.*, "Swift: Compiled inference for probabilistic programming languages," *arXiv preprint arXiv:1606.09242*, 2016.
[2] H. Guo and W. Hsu, "A survey of algorithms for real-time Bayesian network inference," in *Joint Workshop on Real Time Decision Support and Diagnosis Systems*, 2002.
[3] J. Laurel *et al.*, "Statheros: A compiler for efficient low-precision probabilistic programming," *Design Automation Conference*, 2021.
[4] F. Saad *et al.*, "The fast loaded dice roller: A near-optimal exact sampler for discrete probability distributions," in *AISTATS 2020*, 2020, pp. 1036–1046.
[5] G. G. Ko *et al.*, "A 3mm2 programmable bayesian inference accelerator for unsupervised machine perception using parallel gibbs sampling in 16nm," in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.
[6] Y. Ni *et al.*, "PMBA: A parallel MCMC Bayesian computing accelerator," *IEEE Access*, vol. 9, pp. 65 536–65 546, 2021.
[7] S. S. Roy *et al.*, "High precision discrete Gaussian sampling on FPGAs," in *International Conference on Selected Areas in Cryptography*. Springer, 2013, pp. 383–401.
[8] J. Howe *et al.*, "On practical discrete gaussian samplers for lattice-based cryptography," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 322–334, 2018.
[9] D. Li *et al.*, "Mepler: A 20.6-pJ side-channel-aware in-memory CDT sampler," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2.
[10] D. Knuth and A. Yao, "The complexity of nonuniform random number generation," 1976.
[11] S. Song *et al.*, "Leia: A 2.05 mm 2 140mW lattice encryption instruction accelerator in 40nm CMOS," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2018, pp. 1–4.
[12] L. Kong *et al.*, "High-performance constant-time discrete Gaussian sampling," *IEEE Transactions on Computers*, vol. 70, no. 7, pp. 1019–1033, 2020.
[13] "Bayesian network repository," https://www.bnlearn.com/bnrepository.
[14] U. Banerjee *et al.*, "2.3 an energy-efficient configurable lattice cryptography processor for the quantum-secure internet of things," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 46–48.