AFRL-RI-RS-TR-2018-188

STORAGE, REPRESENTATION, AND MANIPULATION OF DISTRIBUTION FUNCTIONS

YALE UNIVERSITY

AUGUST 2018

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2018-188 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
UTTAM K. MAJUMDER
Work Unit Manager

/ S /
JOHN D. MATYJAS
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
AUG 2018	FINAL TECHNICAL REPORT	MAY 2017 – FEB 2018

4. TITLE AND SUBTITLE

STORAGE, REPRESENTATION, AND MANIPULATION OF DISTRIBUTION FUNCTIONS

5a. CONTRACT NUMBER
FA8750-17-1-0113

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)

Rajit Manohar

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
(R28K)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Yale University
Computer Systems Lab
New Haven, CT 06520

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2018-188

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Probabilistic reasoning techniques are emerging as one of the most powerful ways to extract information from complex spatio-temporal data streams generated by modern sensors. Representing features and estimating their probability requires manipulating high-dimensional functions that have a priori unknown structure. This study examines alternate approaches to representing such distribution functions using implicit rather than explicit representations

15. SUBJECT TERMS
Static random access memory, Dynamic random access memory, Cumulative distribution function, Probability distribution function

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			UTTAM K. MAJUMDER
U	U	U	UU	15	19b. TELEPHONE NUMBER (*Include area code*)

Table of Contents

List of Figures

1. Summary

The goal of this effort was to provide an alternate way to represent complex high-dimensional distribution functions with an implicit representation. Such distribution functions are important for estimating the probability of rare events. When the distributions are unknown, as is commonly the case, the traditional approach is to represent the distribution by a large dataset that corresponds to the sampled value of the distribution. This representation is then updated from measurements/observations to improve the estimate—essentially to "learn" the distribution from samples.

2. Introduction

An unknown high-dimensional probability distribution (PDF) can be viewed as a map from a high-dimensional space of features to [0,1]. For simplicity, suppose each feature can be viewed as a real number from 0 to 1. The problem with rare events is that the event of interest might occur in a very "small" part of the input feature space. Capturing it requires having extremely high resolution in the input feature space, and this means that high precision representations are needed for the features as well as the output. If the input feature space is n-dimensional, and we have p-bit precision for the input features and output, then the PDF has 2^{pn} possible input combinations, and hence a total storage of $(p \times 2^{pn})$ bits—which can be extremely large.

Updating such a large storage array is expensive not only from a time complexity stand-point, but also from an energy-consumption standpoint. Memories are organized to maximize the storage per unit of area/volume. Reading or writing the entire contents of memory is extremely slow due to resource sharing constraints imposed to maximize density. Hence, an efficient representation of PDFs is essential to minimize the cost of accessing and manipulating complex density functions. One approach is to compress the raw bits, but good compression algorithms do not permit easily manipulation of the components of the compressed data. This study takes a non-standard approach to this problem, but examining alternative strategies for representing PDFs.

3. Methods, Assumptions, and Procedures

3.1 Stochastic Expansions

We studied the possibility of representing complex PDFs using stochastic expansions [1][2]. In a stochastic expansion, we attempt to model a complex PDF by combining a collection of known, simple PDFs. To illustrate the one-dimensional case, consider the problem of sampling a one-dimensional distribution with a cumulative density function CDF(x) where x is in some range [0,N]. It is well-known that, given a random variable y that is uniformly distributed over the interval [0,1], we can produce a random variable with a distribution that matches the given distribution by simply using $x = \text{CDF}^{-1}(y)$. The question studied was to determine if this can be generalized. It is important to note that "$x = \text{CDF}^{-1}(y)$" can be viewed as an implementation of the specified CDF; however, the implementation is not unique. In other words, while "$x = \text{CDF}^{-1}(y)$" describes *how* we can generate a random variable with a distribution that matches the given CDF, it is not the only way to do so. For example, the variable (1-y) is also a uniformly distributed random variable over the interval [0,1]—but it is a *different* random variable from y. So, from a statistical perspective, it might be better to say "$x \sim \text{CDF}^{-1}(y)$" to refer to the fact that the distributions match.

We examine the possibility of polynomial expansions as a way to generate arbitrary distributions from random variables with known distributions. If we can do this accurately, then it would be possible to build hardware that, given a *high-resolution* sample of a known distribution, can compute a sampled value of the distribution of interest with multiply-accumulate hardware. In other words, a polynomial $g_n(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ can be efficiently evaluated using n multiply-add operations by re-parenthesizing the expression.

One of the typical challenges with a digital representation of information is that the hardware cost in terms of area/footprint to store information grows with the precision requirement. This can be seen with the PDF above, where increasing the precision by one bit increases the storage requirement by at least a factor of 2^n. Furthermore, it is not easy to perform some computation with low precision, and then determine that a higher precision is necessary without losing accuracy. If, instead, we could represent the PDF using coefficients of a polynomial expansion,

we will have de-coupled the precision of the input features from the number of bits required to store the PDF. The input feature precision determines *x*, the argument of the polynomial, and this in turn impacts the computation cost needed to compute the polynomial. However, we do not need to a-priori determine what this precision is, thereby eliminating the problem encountered by the traditional approach where we must know this precision up-front.

To increase the input precision, we need to *generate* the random variable used to compute the unknown distribution with high precision—precision that can be easily adapted at run-time. We do this by representing a random variable using an *implicit* representation rather than an *explicit* representation. In particular, we examine Poisson random variables as a way to efficiently implement randomness.

3.2 Implementing exponentials and randomness

There are methods for measuring and amplifying physical sources of randomness such as thermal noise. However, most of these techniques are quite costly in terms of energy and area. Most common hardware random number generators are built using linear feedback shift registers (LFSRs), which generate *m*-sequences using polynomials over the field GF(2) [3]. The hardware implementation is extremely cheap, and consists of a combination of shift registers and exclusive-OR (XOR) gates. The hardware is organized based on the generating polynomial used for the *m*-sequence, and each iteration generates a new pseudo-random number value. If there are *k* shift registers, then *k* bits are generated per clock cycle and the maximum period of the pseudo-random number sequence is 2^k. If more random bits are desired, either a different LFSR can be created or the number of shift register bits can be extended along with the generating polynomial. It is well-known that for good choices of generating polynomials, the pseudo-random numbers behave as uniformly distributed random variables. The largest disadvantage with this approach is the incremental cost for randomness.

To reduce the cost of randomness, we designed a new memory architecture where the core of the memory array holds the random bits of state. The rationale is that a memory is the densest logic structure that can be reliably built, so it is the cheapest source of bits. Also, since dense memory

is so important for a wide range of applications, there is enormous investment by industry in improving the density, reliability, performance, and power consumption of memory. For example, many novel material systems are being considered for integrated two-terminal devices for non-volatile memory applications. While standard static random access memories (SRAMs) are roughly 120-150F^2 per bit (where F is the feature size), new memory technologies that can be integrated with a standard CMOS process a projecting densities of 4-6F^2 per bit—the same range as main stream dynamic random access memory (DRAM). Hence, these technological improvements would immediately be usable if we use memory bits for randomness.

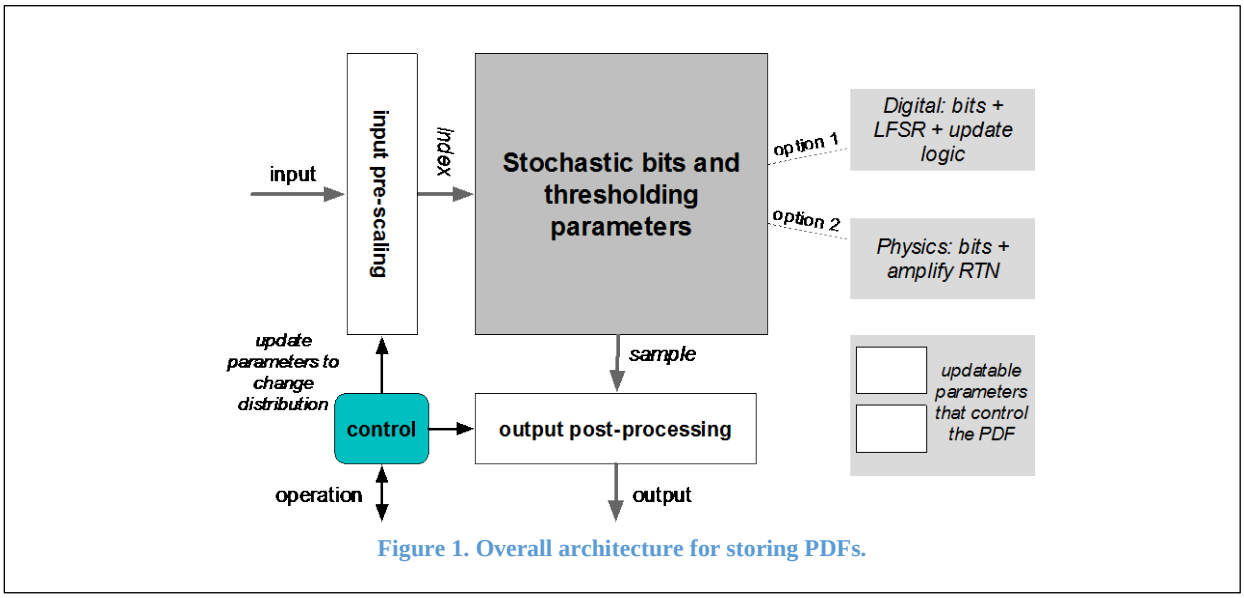

Figure 1. Overall architecture for storing PDFs.

The overall architecture for storing PDFs is illustrated in Figure 1. The core of the architecture consists of a dense array of memory bits, that are accessed and updated in a non-traditional fashion that requires modifying the memory control circuitry. Each bit of memory is stochastically updated, and its 0/1 state follows a Poisson distribution. To illustrate the concept, consider a variable that is initially 1. At some time interval *t*, we toss a biased coin with bias λ. If the coin is heads, we reset the bit to 0; otherwise it remains 1. If we had *k* such bits, then the distribution of the sum of those bits would be the binomial distribution. Taking the limiting case of the distribution results in a Poisson distribution.

To implement this idea, we must have a cheap mechanism to update the bits of a memory. Initializing the memory is straightforward, since all the bits are set to 1. To do so, we assert *all* the

word lines *in parallel*, and then simply drive the bit lines so that all the bits are set to a 1. This is significantly less expensive than writing each line of the memory, since the bit-line cost is incurred once. The savings depends on the relative cost of switching word lines versus bit-lines, which is a function of the aspect ratio of the memory. For memories where the bit-line and word-lines are optimized to have roughly the same load, this operation costs less than ½ the energy in the memory core itself. However, the addressing costs are much lower since this operation requires one global address request rather than using the address decoder once per row.

Instead of the coin-tossing operation we described earlier being performed per bit, we instead toss a large number of coins in parallel. If we assume the memory has M bits organized as $\alpha\sqrt{M}$ bits per row and $1/\alpha\sqrt{M}$ total rows, then we toss $\alpha\sqrt{M}$ coins in parallel and update all of those bits with a single memory write operation. Note that we have to modify the write circuitry to *skip* the write operation when the coin is tails. For the bits whose states do not change, we keep the bit-lines pre-charged; this automatically preserves the old state since read stability of a memory bit-cell requires that the cell state be undisturbed when the bit-lines are in the (1,1) state. Hence, we still need a source of random numbers, but the state of the random number generator is now slightly decoupled from the random state being used to sample the PDF.

A second approach that we also studied was the possibility of using the pseudo random number generator as a way to select a random address of the memory for an update. For example, consider a case where the coin bias λ is small. For situations like this, we might need *many* bits of randomness. This is because the way one would implement such a bias would be to take a uniformly distributed random number in the [0,1] range and compare it with λ. The *resolution* of the comparison has to match the resolution of λ—and so does the resolution of the [0,1] random number that is generated. If the memory has $1/\alpha\sqrt{M}$ rows and $\lambda/\alpha\sqrt{M} < 1$, then we can instead pick a row at random (using one LFSR), and then toss coins at a scaled value of λ with a second, lower resolution LFSR.

Both these approaches allow us to create a Poisson distribution that is quite accurate. However, what is more important, we can *control* the accuracy of the distribution *after the fact* by changing

the number of bits that are averaged. Also, the bits in the memory can be allocated in a manner that depends on the accuracy/resolution needed for each feature.

The third interesting observation is that there is a temporal component to this operation. The state change operation is performed with some bias λ at some time interval *t*. While the bias has some finite resolution because it depends on the number of pseudo-random bits generated as described above, the *time* component is continuous and hence, can be adjusted at any desired resolution. Both of these components determine the state of each random bit at a given point in time, providing an externally tunable resolution for at least the temporal dimension.

The motivation for this approach originated when studying the design of neuromorphic hardware architectures for modeling synaptic dynamics. Synapse dynamics are often modeled as decaying exponentials, and these are in fact quite expensive to implement in standard digital hardware. Hence, instead of computing an exponential function, we used a collection of independent coins whose normalized sum was an approximation of the synaptic output. In some sense, the exponential synapse is in fact the continuous approximation of an aggregate 0/1 coin toss; so the coin-tossing approximation in some ways is more physically accurate than the exponential synapse model!

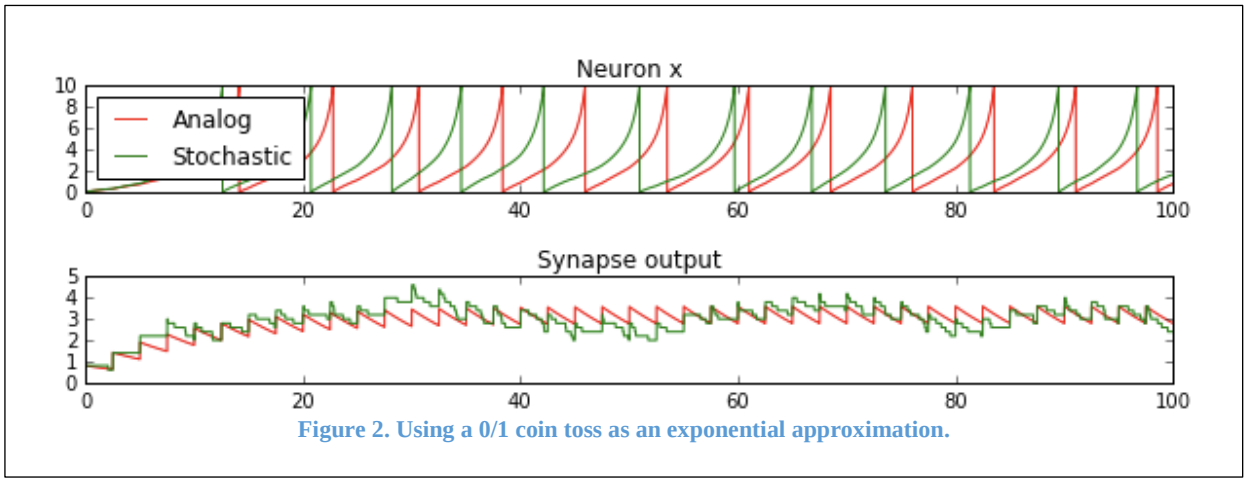

Figure 2. Using a 0/1 coin toss as an exponential approximation.

Figure 2 shows the result of this approximation. The "Analog" output (computed using double-precision floating-point arithmetic) is shown alongside the "Stochastic" output, which corresponds to the 0/1 coin-tossing approximation to the exponential. What is interesting is that even after a

complex non-linear differential equation transform that corresponds to the neuron dynamics, the important feature of the neuron output (the average firing rate) is preserved by this approximation. The neuron dynamics in question correspond to a quadratic integrate and fire neuron, so this includes non-linear feedback (as can be seen in the peaking effect of the neuron output). The 0/1 approximation is visible in the synaptic output in the form of a "staircase" curve, rather than a smooth exponential decay.

4. Results and Discussion

We studied the accuracy of representing two common distributions—Poisson and Gaussian—using this approach. Figure 3 shows the result of approximating a standard Poisson distribution.

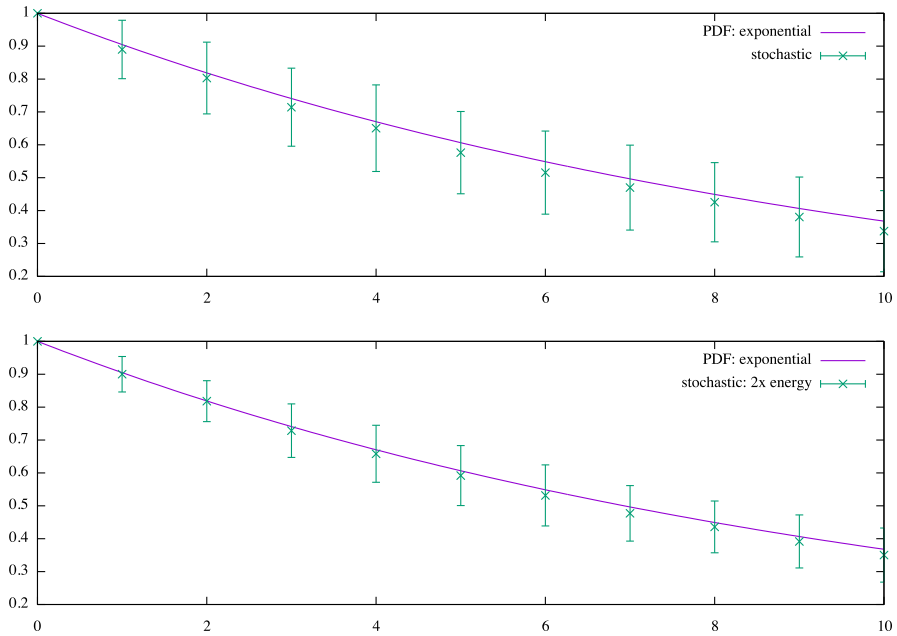

Figure 3. Accuracy of ideal Poisson PDF versus 0/1 approximation, with different number of bits averaged.

The distribution being approximated is shown as a continuous line, with the discrete points corresponding to sampled values of the distribution. As should be evident, the mean of the samples is very close to the actual PDF. By increasing the number of bits used for the approximation, the variance can be significantly reduced. The cost of using more bits is simply more energy to read the bits—a larger number of the memory bits are allocated for one distribution function.

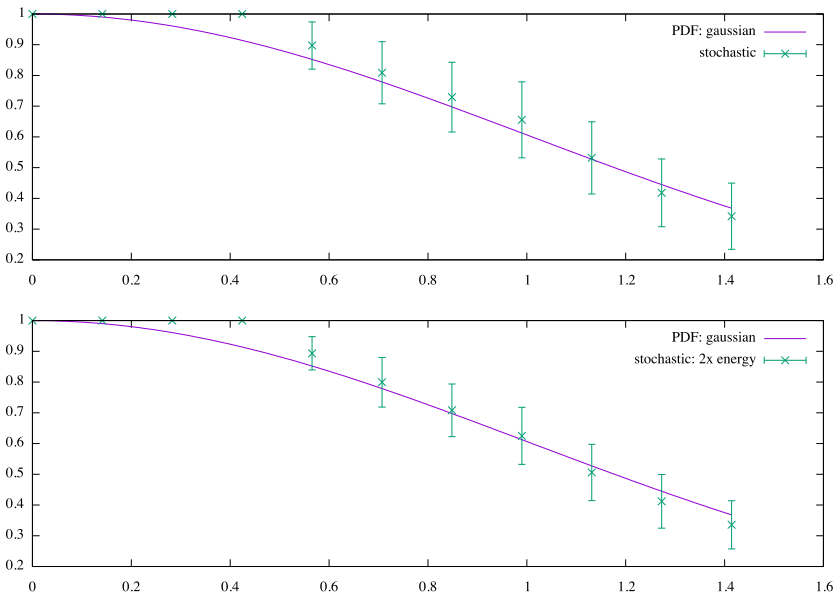

Figure 4. Accuracy of ideal Gaussian PDF versus 0/1 approximation, with different number of bits averaged.

Poisson distributions are "native" to this type of 0/1 stochastic memory architecture. To examine the impact of the coin-tossing scheme on different types of PDFs, we also examined the accuracy of a Gaussian approximation using the same approach. While the core of the memory architecture remains unchanged, the way the bits are *interpreted* changes. We use the stochastic expansion idea to generate a Gaussian random variable from a Poisson one. As before, we can change the accuracy of the representation by using a larger number of bits. The result of this are shown in Figure 4. The results are roughly similar to those for the Poisson distribution, which validates the basic idea of using 0/1 stochastic memory as a source of randomness, and then post-processing the result to generate a different distribution function.

For this study, we used two different 28nm process technologies to evaluate the performance, power, area, and efficacy of the approach. The total area required for modeling a PDF is dominated by the SRAM bit-cell size, which is roughly $0.15\mu m^2$. Hence, the area of this design is extremely compact (more than an order of magnitude smaller than a pseudo-random number generator bit). There is some additional overhead compared to a standard memory architecture. The sources of overhead are:

- The pseudo-random number generators used for row and column addressing. These scale as $\sqrt{M}$, where M is the size of the memory compared to a traditional approach where this overhead would scale as M.
- The additional circuitry needed for initializing the memory, and for not driving the bitlines.

The area overhead of this compared to a standard memory, with 8Kb of random bits, is roughly 12%.

We designed the memory architecture to support a throughput of 1 GHz. While the design is able to operate at this throughput, it means that the *resolution* of fine-grained timing updates would be limited by 10^{-9}. However, each request can be provided at an arbitrary time point, so while the gap between two updates has to be at least 10^{-9}, the gap can be arbitrary and does not have to be discretized to this resolution.

An evaluation of a random number is estimated from circuit simulations to be 83 fJ/bit, and updating the state of the memory is approximately 102 fJ/bit. Compared to a low power embedded CPU, our estimate is that this reflects a throughput increase of a factor of 40, with an energy consumption that is 15,000 times lower.

5. Conclusions

One of the limitations is that studying the precise way such an approximation can be applied to higher dimensional distribution functions is not well-understood in the mathematics community. There are cases where this is straightforward, for example, when the distribution function can be factored. However, the goal of this study was to examine cases when the function was in fact unknown. From our investigation, we determined that higher dimensional stochastic expansions are an area with limited mathematical/statistical results. Hence there is no systematic method or accepted method to construct such expansions for high dimensional inputs.

Evaluating the impact of this approximation on Bayesian inference algorithms would be also valuable. The approximation from this approach is stochastic, and hence it should not negatively

impact the overall accuracy of inference algorithms (there should not be a systematic bias, for example).

6. References

[1] Michael A Magdalinos. Stochastic expansions and asymptotic approximations. Econometric Theory, 8(03):343–367, 1992.

[2] Xiaoliang Wan and George Em Karniadakis. Multi-element generalized polynomial chaos for arbitrary probability measures. SIAM Journal on Scientific Computing, 28(3):901–928, 2006.

[3] Neal Zierler. Linear recurring sequences. SIAM Journal, 7(1):31—48, 1959.

7. List of Acronyms

Acronym	Expanded Form
CDF	Cumulative distribution function
DRAM	Dynamic random access memory
LFSR	Linear feedback shift register
PDF	Probability distribution function Probability density function
RTN	Random telegraph noise
SRAM	Static random access memory
XOR	Exclusive OR Boolean operation