

Electron-Tunnelling-Noise Programmable Random Variate Accelerator for Monte Carlo Sampling

James T. Meech¹, Vasileios Tsoutsouras², and Phillip Stanley-Marbell^{1,2}

¹Department of Engineering, University of Cambridge; ²Signaloid

This manuscript was compiled on April 9, 2024

This article presents an electron tunneling noise programmable random variate accelerator for accelerating the sampling stage of Monte Carlo simulations. We used the LiteX framework to generate a FemtoRV imfc RISC-V instruction set soft processor and deploy it on a Digilent Arty-100T FPGA development board. The RISC-V soft processor augmented with our programmable random variate accelerator achieves an average speedup of $8.70\times$ and a median speedup of $8.68\times$ for a suite of twelve different benchmark applications when compared to GNU Scientific Library software random number generation. These speedups are achievable because the benchmarks spend an average of 90.0% of their execution time generating random samples. The results of the Monte Carlo benchmark programs run over the programmable random variate accelerator have an average Wasserstein distance of $1.48\times$ and a median Wasserstein distance of $1.41\times$ that of the results produced by the GNU Scientific Library random number generators. The soft processor samples the electron tunneling noise source using the hardened XADC block in the FPGA. The flexibility of the LiteX framework allows for the deployment of any LiteX-supported soft processor with an electron tunneling noise programmable random variate accelerator on any LiteX-supported development board that contains an FPGA with an XADC.

Programmable Random Variate Accelerator | Monte Carlo Simulation | FPGA | RISC-V

1. Introduction

Monte Carlo simulations for quantifying uncertainty in computations are becoming increasingly more important (1, 2). Computer systems are expected to make increasingly complicated real-time decisions in life-critical applications using uncertain data (3). At the same time, Moore's law has ceased to provide increased computer hardware performance through transistor scaling laws (4). These two events call for domain-specific architectures to accelerate applications such as Monte Carlo simulations (5, 6). This article introduces a domain-specific hardware accelerator for the sampling stage of Monte Carlo simulations. Our accelerator will complement other domain-specific architectures designed for propagating uncertainty through calculations (1, 2). The programmable random variate accelerator is a faster and more efficient replacement for digital electronic hardware for Monte Carlo sampling. The programmable random variate accelerator replaces function calls to random number generator libraries with code to sample from the programmable random variate accelerator. The accelerator uses an analog-to-digital converter with direct memory access to allow a RISC-V instruction set processor to sample from any univariate probability distribution using a physics-based programmable non-uniform random variate generator.

A. Contributions. This article presents the following contributions:

1. A programmable random variate accelerator design that can sample from arbitrary univariate Gaussian distributions and arbitrary univariate distributions as Gaussian mixtures.

2. A prototype interfacing the programmable random variate accelerator with a FemtoRV Petitbateau imfc RISC-V instruction set soft processor on a field programmable gate array.
3. Evaluation of the speed and efficiency improvements gained by using the programmable random variate accelerator to accelerate a suite of twelve benchmark applications.
4. Comparison of the programmable random variate accelerator Monte Carlo simulation accuracy for a suite of twelve benchmark applications using Wasserstein distance from a reference result.

B. Motivation for Monte Carlo Sampling. Figure 1 panel ① shows an application where the user wants to generate random samples to perform a Black Scholes Monte Carlo simulation. The application code will call a random number generation function that will perform the Box-Muller transform on the output of a uniform pseudorandom number generator to obtain the samples from a Gaussian distribution required for the Black-Scholes method. Traditional digital electronic hardware will then execute assembly instructions to run the program for the user. For named probability distributions with a closed-form inverse cumulative distribution function other than the Gaussian the random number generation function will use the inversion method. If there is no closed form for the inverse cumulative distribution function the random number generation function will resort to using an even more inefficient rejection sampling method (7). All of the math operations required for the application will be converted to assembly instructions to be run on an in-order digital electronic processor. In this architecture, each instruction and the corresponding data will have to be read from the main memory over the data bus incurring long latency. Finally, the digital electronic transistors that implement the hardware will switch the flow of electrons to perform the computation.

A more efficient alternative to the traditional hardware shown in the top panel of Figure 1 would be to offload all of the random number generation to a dedicated programmable random variate accelerator. The user can simply replace all random number generator calls with calls to the programmable random variate accelerator application programming interface. Then the compiler generates assembly instructions to program and sample from the programmable random variate accelerator. These assembly instructions will allow the programmable random variate accelerator to use an analog-to-digital converter to sample from a Gaussian fast and efficiently using an analog noise source. The programmable random variate accelerator will then quickly and efficiently transform the raw Gaussian samples to samples from the required probability distribution and store them in a register in the processor. This replaces the multiple assembly instructions required to perform the Box-Muller transform, inversion, or rejection sampling by a single instruction to sample from the programmable random variate accelerator.

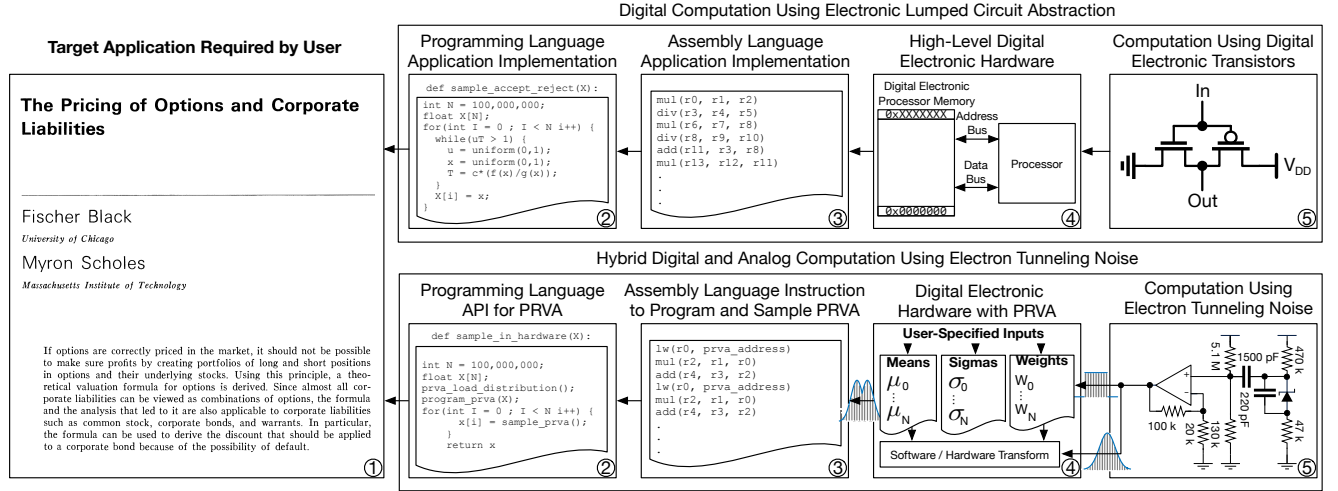


Fig. 1. The steps required to perform a Black Scholes Monte Carlo simulation using a programmable random variate accelerator instead of a digital electronic processor diverge at the first abstraction level below the abstract Black Scholes application required by the user. The programmable random variate accelerator requires small changes at every level of the software and hardware stack to use electron tunneling noise to generate samples from parameterized probability distributions to run the Monte Carlo simulation.

2. Programmable Random Variate Acceleration

Sampling from empirical application-specific non-uniform probability distributions is slow and inefficient. Digital electronics processors use the fast and efficient inversion method (7) shown in Algorithm 1 to transform uniformly-distributed random samples to non-uniformly-distributed random samples. Digital electronic processors can only use the inversion method to obtain random samples for an application when there exists an analytical closed-form solution for the inverse cumulative distribution function of the probability density function that the application requires samples from. For probability density functions for which there is no analytic form for the inverse cumulative distribution function, the digital electronic processor must instead use the accept-reject method (7) to transform random samples with a uniform distribution to samples from the target non-uniform distribution. Algorithm 2 shows the accept-reject method. The accept-reject method repeatedly generates samples until a sample satisfies the condition on line four of Algorithm 2. The algorithm discards any samples that do not meet the condition. As a result, the accept-reject method is slow and inefficient.

Algorithm 1: Inversion method (7). Let X be a non-uniform random variable that the digital electronic processor needs to sample, U be the uniform random variable that the digital electronic processor can sample from, and F^{-1} the inverse cumulative distribution function that the digital electronic processor requires samples from.

Result: Sample from non-uniform random variable X

- 1 Generate uniform $[0, 1]$ random variate u
- 2 RETURN $x \leftarrow F^{-1}(u)$

3. Arbitrary Distribution From Gaussian Mixture

Many Monte Carlo simulations require samples from bespoke empirical non-uniform probability distributions. Our analog noise source can only generate samples from a univariate Gaussian. We therefore present the theory required to generate samples from any univariate probability distribution using samples from a univariate Gaussian

Algorithm 2: Accept-reject method (7). Let g be the density of U and f be the density of X . Let $c \geq 1$ be a constant such that the condition $f(x) \leq cg(x)$ holds for all x . The accept-reject method probabilistically rejects random variates from a probability distribution that is easy to sample to produce samples from a probability distribution that is hard to sample.

Result: Sample from non-uniform random variable X

- 1 repeat
- 2 Generate uniform $[0, 1]$ random variate u
- 3 Generate uniform $[0, 1]$ random variate x
- 4 Set $T \leftarrow c \frac{f(x)}{g(x)}$
- 5 until $uT \leq 1$
- 6 RETURN x

random variable. Starting from a univariate distribution described in terms of discrete samples, it is possible to reconstruct the probability density function using a kernel density. In the case where we select the Gaussian as our kernel function the kernel density is a mixture of Gaussian distributions.

A programmable random variate accelerator can generate samples from a Gaussian distribution using Gaussian electronic noise and then transform those samples to change the mean and standard deviation of the distribution. The programmable random variate accelerator can generate samples from any non-uniform distribution by decomposing it into a mixture of Gaussians and then using the Gaussian-to-Gaussian transform (Section B) to sample from each Gaussian component of the kernel density. The accelerator can then use a uniform-random-number generator to sample from each Gaussian distribution with likelihood proportional to the relative weight of each Gaussian (or use equal weights).

A. Kernel Densities. If we take a set of N data points that represent a distribution, we can construct a kernel density estimate of the probability density function. We do this by fitting a Gaussian mixture to the points. Let N be the number of samples, M be the number of component functions, K be the kernel density function that we use (a Gaussian distribution) and h be the bandwidth which is a smoothing

parameter. Wand et al. (8) show that we can write the normalized version of our kernel density estimate as

$$\hat{f}_X(x) = \frac{1}{Mh} \sum_{i=1}^M K\left(\frac{x - x_i}{h}\right). \quad [1]$$

Assuming that the data has a Gaussian distribution we can use Silverman's rule to calculate h . Let σ_N be the standard deviation of all N data points, Silverman (9) shows that we can estimate h as

$$h = \left(\frac{4\sigma_N^5}{3N} \right)^{\frac{1}{5}}. \quad [2]$$

More sophisticated kernel density estimation methods exist if the application demands a better approximation (10–12).

B. Gaussian-to-Gaussian Transform. The programmable random variate accelerator can obtain a Gaussian-distributed random variable X' with any desired mean μ' and standard deviation σ' from a Gaussian-distributed random variable X with a mean μ and standard deviation σ (13) by applying the transformation

$$X' = (Xa) + b \quad [3]$$

where

$$a = \frac{\sigma'}{\sigma} \quad [4]$$

and

$$b = \mu' - \mu a. \quad [5]$$

Sampling from a random variable by transforming samples from an existing Gaussian or mixture of Gaussian random variables always produces a sample, unlike the accept-reject method which does not.

4. Electron Tunelling Noise Source Implementation

The implementation of the Gaussian random number generator consists of a constant current reverse bias generating circuit, a reverse-biased Zener diode noise source, a direct current blocking capacitor, a bias setting potential divider, and an operational amplifier. Both the reverse bias generating circuit and the op-amp can be turned off to reduce power consumption by connecting the “On” pin to 0 V.

A. Circuit Design. Figure 2 shows the circuit diagram of the programmable noise source design slightly modified from prior work by Huang (14). The design leverages a high voltage constant current light emitting diode driver to provide a Zener diode with a 15 V reverse bias. The light-emitting diode driver enforces a constant current to minimize the overall power consumption. The reverse-biased Zener diode generates a noise voltage which is amplified by an operational amplifier with a gain of $5\times$. Prior to this, a direct current blocking capacitor removes the large 15 V bias voltage from the noise signal and a potential divider sets the mean of the noise signal. Figure 3 shows a prototype printed circuit board that implements the circuit shown in the shaded gray box on the left side of Figure 2 (i.e., excluding the mux, ADC, and CPU).

The circuit uses a constant current light emitting diode driver powered by the supply rail to apply a 15 V reverse bias to the Zener diode to cause electrons to randomly tunnel through the PN junction. This random tunneling activity manifests itself as a random voltage signal that the circuit amplifies using a non-inverting operational amplifier circuit. The direct current blocking capacitor removes the 15 V direct current bias, and the circuit adds a new bias to the signal

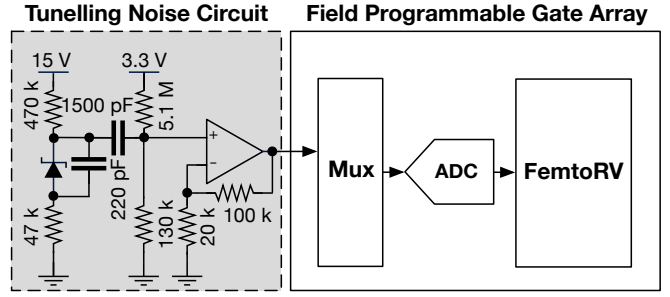


Fig. 2. The physics-based Gaussian random number generator circuit. Details of circuits such as the constant current light emitting diode driver circuit that produces the 15 V bias voltage are omitted here but available in (14).

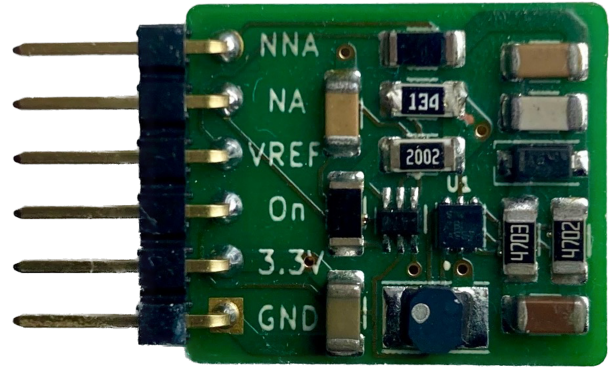


Fig. 3. The low power Gaussian noise source printed circuit board. The printed circuit board consumes 1.62 mW of power in the on state and 32.4 nW in the off state (14).

using a programmable resistor divider. The resistor divider controls the mean of the Gaussian noise distribution. The gain of the non-inverting operational amplifier controls the standard deviation of the distribution. Finally, the analog-to-digital converter quantizes the output of the amplifier to 12-bit unsigned integers. We set the bias and gain to ensure that the analog-to-digital converter sees a large range of unique signal values and captures the Gaussian shape of the distribution.

B. Kernel Density Programmable Random Variate Accelerator for Programmable Univariate Distributions.

Figure 5 shows how the programmable random variate accelerator transforms the random samples in software to produce the target distribution. The user programs the processor with three arrays containing the mean, standard deviation, and weight of each Gaussian in the kernel density. The processor uses a software-uniform-pseudorandom number generator (16) to select a Gaussian to generate samples from. The transform code transforms a sample from the analog-to-digital converter to a sample from the required Gaussian. The processor uniformly interpolates the analog-to-digital converter samples using the same uniform-pseudorandom number generator to increase their resolution from 12 to 64 bits. Algorithm 3 shows the process of transforming the generated Gaussian to the required Gaussian and then linearly interpolating with the uniform-pseudorandom-number generator. This process repeats each time the processor samples from the distribution.

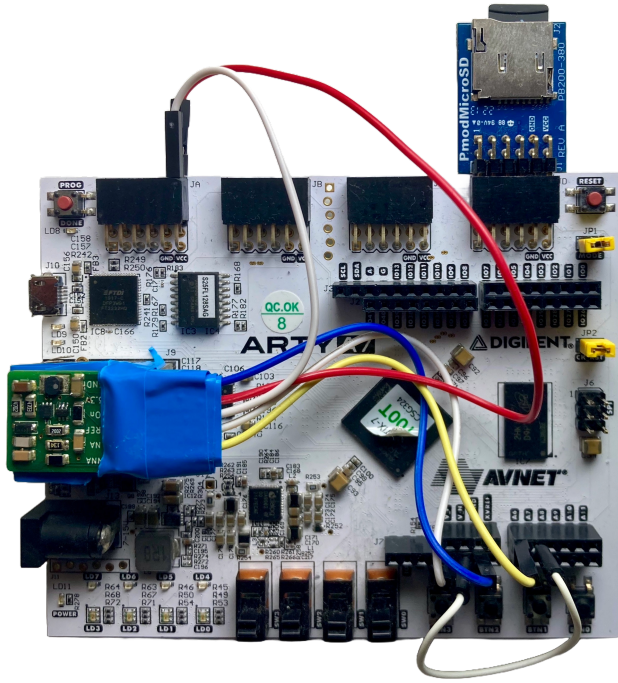


Fig. 4. The low power Gaussian noise source printed circuit board connected to the XADC of an Artix-7 XC7A100T FPGA on a Digilent Arty development board. The microSD card PMOD stores .elf program files to be executed on the soft processor on the FPGA. LiteOS provides the functionality to compile C language programs to .elf files for the RISC-V Petibateau soft processor (15). The FPGA development board and noise source board combined consume approximately 1.983 W of power when sampling from a univariate Gaussian.

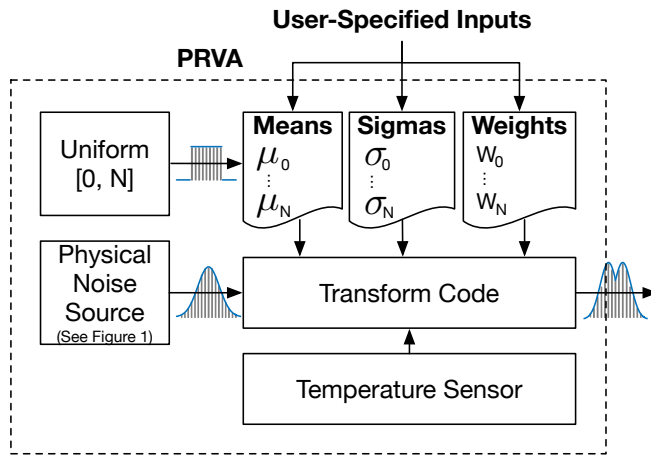


Fig. 5. The programmable random variate accelerator can use a kernel density encoded as a list of means, standard deviations, and weights to randomly sample from the mixture of Gaussian distributions that makes up the kernel density.

Algorithm 3: Method to transform to transform N samples from a distribution with a mean μ and standard deviation σ to N samples with mean μ' and standard deviation σ' .

input : $\mu, \mu', \sigma, \sigma', U, X$

Result: Sample N values from Gaussian random variable X

```

1  $a = \frac{\sigma'}{\sigma}$ 
2  $b = \mu' - \mu a$ 
3 for  $i = 0$  to  $N$  do
4   Read in random integer  $x$  from the ADC
5   Sample =  $\frac{x+u}{2^{64}}$ 
6   Samples[ $i$ ] =  $a * \text{Sample} + b$ 
7 end
8 RETURN Samples

```

5. Noise Source Temperature Dependence

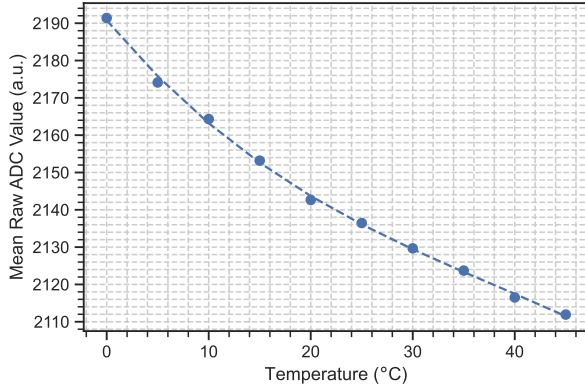
We placed the field programmable gate array development board and the Gaussian noise source inside a Binder MK56 thermal chamber. We set the thermal chamber to 0°C and allowed the temperature to reach equilibrium for 30 minutes. While waiting for the temperature to reach equilibrium we had the soft processor on the field programmable gate array consistently running a program to print raw analog-to-digital converter values. When equilibrium was reached we ran a program to have the processor sample and print 10^6 raw analog-to-digital converter values. We repeated this for ten different temperatures between 0°C and 45°C with a 5°C temperature step.

Figure 6a shows the dependence of the mean of the raw analog-to-digital converter values on temperature. We can eliminate this dependence by having the program that samples the analog-to-digital converter randomly subtract half of the samples from the maximum analog-to-digital converter value. Figure 6b shows the dependence of the standard deviation of the raw analog-to-digital converter values upon temperature. The curve with the legend “Flipped” shows that the subtraction method that removes the temperature dependence of the mean of the raw analog-to-digital converter values shifts the standard deviation upwards by a constant factor but does not remove the dependence of the standard deviation upon temperature.

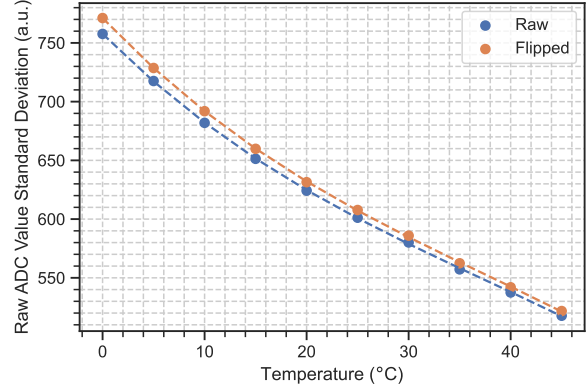
Figure 7a shows a violin plot of the kernel density of the distribution of raw analog-to-digital converter values at each temperature. Figure 7b shows how randomly subtracting the raw analog-to-digital converter value from the maximum possible value creates symmetric distributions from the skewed raw samples. This processing does not remove the temperature dependence of the standard deviation and higher-order statistics. We should therefore measure the temperature of the Gaussian noise source and compensate for it or keep the noise source at a constant temperature when sampling it.

6. Speed and Power Consumption Measurements

We measured that the FemtoRV RISC-V soft processor augmented with a programmable random variate accelerator took approximately 130 s to generate 10^9 64-bit random samples. This is a sampling speed of 492 Mb/s. We used a Microchip ADM00921 Power Meter to measure the combined average power consumption of the FPGA development board and the low-power Gaussian noise source printed circuit board to be 1.983 W while running the program that we used to measure the univariate Gaussian sampling speed. We used a Keithly 2450 SourceMeter to measure the power consumption of the low-power Gaussian noise source printed circuit board to be 1.62 mW in the on state and 32.4 nW in the off state.

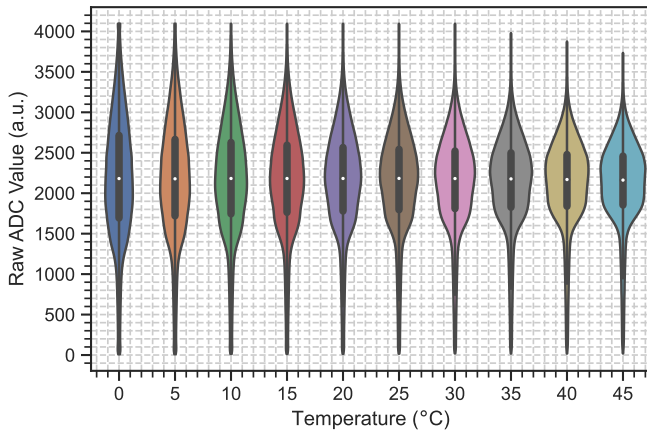


(a) Mean of raw ADC samples from the Gaussian noise source.

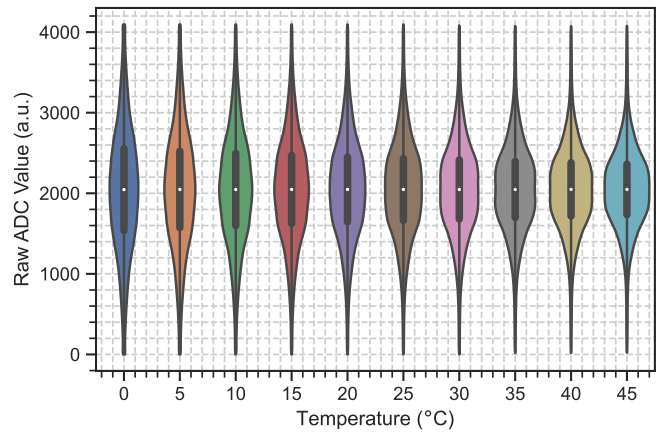


(b) Standard deviation of raw ADC samples from the Gaussian noise source.

Fig. 6. Mean and standard deviation of ten batches 10^6 raw analog-to-digital converter samples over a temperature range of 0°C to 45°C with a 5°C temperature step. The blue data points and curves show the mean and standard deviation of the raw ADC values. The orange curve shows the standard deviation of the raw ADC values after 50 % of them have been randomly subtracted from the maximum ADC output value.



(a) Raw unprocessed ADC samples from the Gaussian noise source.



(b) Raw ADC samples randomly subtracted from 4095 with 0.5 probability.

Fig. 7. Violin plot of kernel density estimates of 10^6 ADC samples over a temperature range of 0°C to 45°C with a 5°C temperature step.

Table 1. The end-to-end speedup achieved by a programmable random variate accelerator for a range of C language benchmark applications. The average speedup was $8.70\times$ and the median speedup was $8.69\times$. The average Wasserstein distance from the reference result was $1.48\times$ and the median was $1.41\times$ that of the results produced by the GNU Scientific Library random number generators. The lines column is the number of lines in the GNU scientific library implementation of the benchmark. We calculate the Wasserstein distance from a 10^8 sample reference Monte Carlo run on a workstation computer. We measured the Wasserstein distance and speedup results by running 100 repeats with 10^4 samples for each benchmark. We did this using the programmable random variate accelerator for sampling and also separately, the GNU scientific library random number generator. We calculate the random sampling fraction by running each benchmark 10^4 times and calculating the average fraction of clock cycles spent generating random samples. We used the `clock()` function from the `time` C library.

Application	Wasserstein Distance Ratio PRVA / GSL	Random Sampling Fraction (%)	End-to-End Speed Up (\times)	Number of Lines	Sampling Distribution	Source
Gaussian Sampling	1.98	98.8	9.36	80	Gaussian	This Work
Gaussian Mixture	1.17	97.5	6.89	80	Mixture	This Work
Addition	1.12	92.1	9.31	87	Gaussian	(1, 17)
Divide	1.51	92.1	8.59	87	Gaussian	(1, 18)
Multiply	1.61	92.4	8.78	87	Gaussian	(1, 19)
Subtract	1.21	92.2	10.24	87	Gaussian	(1, 20)
Schlieren	1.26	91.5	8.83	87	Gaussian	(1, 21)
NIST-UM Dynamic Viscosity	1.84	96.0	6.88	90	Gaussian	(1, 22)
NIST-UM Thermal Expansion Coefficient	1.30	98.3	25.24	84	Student-T	(1, 23)
Medical Covid-19 R0	1.09	82.5	5.40	143	Mixture	(24)
Geometric Brownian Motion	1.72	69.3	2.35	102	Gaussian	(25)
Black Scholes Monte Carlo Pricing	1.93	71.9	2.57	98	Gaussian	(26)

7. Monte Carlo Program Benchmarking Study

We benchmarked twelve C programming language applications on the LiteX-generated FemtoRV Petitbateur RISC-V processor. The processor used a built-in 64-bit timer to measure the number of clock cycles required to run each application. Table 1 shows the speedup gained by using Gaussian random numbers generated by the electron tunneling noise programmable random variate instead of the GNU scientific library Gaussian and Student T random number generators. We ran each benchmark one hundred times and found that the average speedup was $8.70\times$ and the median speedup was $8.69\times$. Gaussian sampling and Gaussian mixture sampling respectively have an end-to-end speed up of $2815\times$ and $2899\times$ respectively if the program does not store the samples in an array. All benchmark programs shown in Table 1 store the results of the Monte Carlo simulation in an array.

8. Related Research

A. Programmable Random Variate Accelerators. Table 2 shows a summary of state-of-the-art programmable random variate accelerators. As far as we are aware this article presents the fastest and most efficient electronic noise programmable random variate accelerator that can generate samples from any univariate non-uniform probability distribution. Three approaches from Table 2 record a lower power consumption than our approach but are inferior in other aspects. The power measurement for the memristor-based approach (27) does not include the power consumption of any of the circuitry required to sample the device or bias it to the correct voltage. The power quoted for the resonance energy transfer approach (28) is a back-of-the-envelope estimate and not experimentally measured. The implementation we present in this paper is approximately $35,652\times$ and $72\times$ faster than programmable random variate accelerators we presented in previous publications (29, 30). As the sampling speed is mainly limited by the analog-to-digital converter sampling rate, using LiteX to deploy the design on FPGAs with a higher analog-to-digital converter sampling rate will lead to further speed increases.

B. Thermodynamic Computing. Thermodynamic computing (39, 40) uses hardware capable of sampling from an arbitrary two-

dimensional Gaussian for solving a linear system of equations, estimating the inverse of a matrix, solving the Lyapunov equation, and estimating the determinant of a matrix (41–44). A blog post describing thermodynamic computing hardware for Gaussian sampling reports a speedup of approximately $13\times$ and an energy saving of approximately $25\times$. Detailed systems evaluations explaining their hardware architecture and how it achieves this are not available (45).

C. Tuned Stochastic Probability Trees. Prior work to design and build stochastic magnetic tunnel junction devices to implement a computing device that can perform single random bit sampling with a programmable bias exists (46–55). This approach is very different from our approach as it builds up probability distributions using many single programmable random bits.

Conclusion

This article presented an electron tunneling noise programmable random variate accelerator for accelerating the sampling stage of Monte Carlo simulations. We used the LiteX framework to generate a Petitbateur FemtoRV RISC-V instruction set soft processor and deploy it on a Digilent Arty-100T FPGA development board. The RISC-V soft processor augmented with our programmable random variate accelerator achieves an average speedup of $8.70\times$ and a median speedup of $8.69\times$ for a suite of twelve different benchmark applications when compared to GNU Scientific Library software random number generation. These speedups are achievable because the benchmarks spend an average of 90.0 % of their execution time generating random samples. The results of the Monte Carlo benchmark programs run over the programmable random variate accelerator have an average Wasserstein distance of $1.48\times$ and a median Wasserstein distance of $1.41\times$ that of the results produced by the GNU Scientific Library random number generators. The soft processor samples the electron tunneling noise source using the hardened XADC block in the FPGA. The flexibility of the LiteX framework allows for the deployment of any LiteX-supported soft processor with an electron tunneling noise programmable random variate accelerator on any LiteX-supported development board that contains an FPGA with an XADC.

Table 2. Comparison of state-of-the-art programmable random variate accelerator (PRVA) methods (29).

Source	Speed	Efficiency	Power	Distribution(s)	Programmable	Publication
Central Processing Unit	890 Mb/s	3.17 Mb/J	281 W	Gaussian	Yes	(31), 2009
Graphics Processing Unit	12.9 Gb/s	108 Mb/J	119 W	Gaussian	Yes	(31), 2009
Massively Parallel Processor Array	860 Mb/s	403 Mb/J	2.13 W	Gaussian	Yes	(31), 2009
Field Programmable Gate Array	12.1 Gb/s	645 Mb/J	18.8 W	Gaussian	Yes	(31), 2009
Memristor	6000 b/s	120 Gb/J	50.0 nW	Unnamed	No	(27), 2017
Photo Detector	1.77 Gb/s	-	-	Gaussian	No	(32), 2017
Resonance Energy Transfer	2.89 Gb/s	578 Gb/J	5.00 mW	Exponential	Yes	(28), 2018
Photo Diode	17.4 Gb/s	-	-	Husumi	No	(33), 2018
Photo Diode	66.0 Mb/s	-	-	Programmable	Yes	(34), 2018
Photo Diode	320 Mb/s	-	-	Exponential	No	(35), 2018
Field Programmable Gate Array	6.40 Gb/s	-	-	Gaussian	Yes	(36), 2019
Photo Diode	8.25 Gb/s	-	-	Gaussian	No	(37), 2019
Electronic Noise	13.8 kb/s	209 kb/J	66.0 mW	Gaussian	Yes	(29), 2020
Pseudorandom	-	-	78.9 mW	Programmable	Yes	(38), 2021
Electronic Noise	6.82 Mb/s	13.4 Mb/J	484 mW	Programmable	Yes	(30), 2022
Electronic Noise	492 Mb/s	248 Mb/J	1.98 W	Programmable	Yes	This work

ACKNOWLEDGMENTS. We thank Florent Kermarrec and all the contributors to LiteX for creating the open-source tool we leveraged to create a portable implementation of our architecture (56). We thank Bruno Levy for creating LiteOS and for extensive help getting LiteOS working to load elf program files into the LiteX-generated soft processor on an FPGA (15). In addition, we thank Andrew "Bunnie" Huang for extensive help and advice in getting the XADC working with the LiteX framework.

References

- Tsoutsouras V, et al. (2021) The laplace microarchitecture for tracking data uncertainty and its implementation in a risc-v processor in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21. (Association for Computing Machinery, New York, NY, USA), p. 1254–1269.
- Tsoutsouras V, et al. (2022) The laplace microarchitecture for tracking data uncertainty. *IEEE Micro* 42(4):78–86.
- Choi J, Chun D, Kim H, Lee HJ (2019) Gaussian yolov3: An accurate and fast object detector using localization uncertainty for autonomous driving in *Proceedings of the IEEE/CVF International conference on computer vision*. pp. 502–511.
- Tye NJ, Hofmann S, Stanley-Marbell P (2023) Materials and devices as solutions to computational problems in machine learning. *Nature Electronics* 6(7):479–490.
- Dally WJ, Turakhia Y, Han S (2020) Domain-specific hardware accelerators. *Commun. ACM* 63(7):48–57.
- Hennessy JL, Patterson DA (2019) A new golden age for computer architecture. *Communications of the ACM* 62(2):48–60.
- Devroye L (1986) *Non-Uniform Random Variate Generation*. (Springer-Verlag), p. 42. ISBN: 1461386454.
- Wand MP, Jones MC (1994) *Kernel smoothing*. (Crc Press).
- Silverman BW (1986) *Density estimation for statistics and data analysis*. (CRC press) Vol. 26.
- Heidenreich NB, Schindler A, Sperlich S (2013) Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *ASIA Advances in Statistical Analysis* 97:403–433.
- Chiu ST (1996) A comparative review of bandwidth selection for kernel density estimation. *Statistica Sinica* pp. 129–145.
- Sheather SJ, Jones MC (1991) A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society: Series B (Methodological)* 53(3):683–690.
- Leemis LM, McQueston JT (2008) Univariate distribution relationships. *The American Statistician* 62(1):45–53.
- Andrew "bunnie" Huang (2024) Avalanche noise source design. [online] <https://betrusted.io/avalanche-noise>.
- Bruno Levy (2023) Liteos. [online] <https://github.com/BrunoLevy/learn-fpga/tree/master/LiteX/software/LiteOS>.
- O'Neill ME (2014) Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. (Harvey Mudd College, Claremont, CA), Technical Report HMC-CS-2014-0905.
- Meech JT, Kaparounakis O (2022) Micro benchmark: double-add. [online] <https://github.com/signaloid/Signaloid-Demo-Basic-Addition>.
- James Timothy Meech PSM, Kaparounakis O (2022) Micro benchmark: double-divide. [online] <https://github.com/signaloid/Signaloid-Demo-Basic-Division>.
- Meech JT, Stanley-Marbell P (2022) Micro benchmark: double-multiply. [online] <https://github.com/signaloid/Signaloid-Demo-Basic-Multiplication>.
- Meech JT, Kaparounakis O (2022) Micro benchmark: double-subtract. [online] <https://github.com/signaloid/Signaloid-Demo-Basic-Subtraction>.
- James Timothy Meech, Vasileios Tsoutsouras PSM, Kaparounakis O (2022) Micro benchmark: schlieren. [online] <https://github.com/signaloid/Signaloid-Demo-Basic-Schlieren>.
- Kaparounakis O (2022) Micro benchmark: Dynamic viscosity. [online] <https://github.com/signaloid/Signaloid-Demo-Engineering-NISTUMDynamicViscosity>.
- Kaparounakis O (2022) Micro benchmark: Thermal expansion coefficient. [online] <https://github.com/signaloid/Signaloid-Demo-Basic-NISTUMThermalExpansionCoefficient>.
- Plevris A (2024) Calculating r0 for covid-19. [online] <https://github.com/signaloid/Signaloid-Demo-Medical-CovidR0>.
- Oosterlee CW, Grzelak LA (2019) *Mathematical modeling and computation in finance: with exercises and Python and MATLAB computer codes*. (World Scientific).
- Armstrong J (2017) *C++ for financial mathematics*. (Chapman and Hall/CRC).
- Jiang H, et al. (2017) A novel true random number generator based on a stochastic diffusive memristor. *Nat. Commun.* 8(1):1–9.
- Zhang X, et al. (2018) Architecting a stochastic computing unit with molecular optical devices XXmissing booktitle & seriesXXISCA'18. (ACM/IEEE), pp. 301–314.
- Meech JT, Stanley-Marbell P (2021) Efficient programmable random variate generation accelerator from sensor noise. *IEEE Embedded Systems Letters* 13(3):73–76.
- Meech JT, Stanley-Marbell P (GB Patent GB2620734, Filed July 2022) A physical random variate generator.
- Thomas DB, Howes L, Luk W (2009) A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation XXmissing booktitle & seriesXXFPGA'09. (ACM), pp. 63–72.
- Marangon DG, Villoresi P (2017) Source-device-independent ultrafast quantum random number generation. *Phys. Rev. Lett.* 118(6):060503–1–060503–5.
- Avesani M, et al. (2018) Source-device-independent heterodyne-based quantum random number generator at 17 gbps. *Nat. Commun.* 9(1):1–7.
- Nguyen L, et al. (2018) Programmable quantum random number generator without postprocessing. *Opt. Lett.* 43(4):631–634.
- Tomasi A, et al. (2018) Model, validation, and characterization of a robust quantum random number generator based on photon arrival time comparison. *JLT* 36(18):3843–3854.
- Hu Y, et al. (2019) Gaussian random number generator: Implemented in fpga for quantum key distribution. *Int. J. Numer. Model. El.* 32(3):2554.
- Guo X, et al. (2019) Parallel real-time quantum random number generator. *Opt. Lett.* 44(22):5566–5569.
- Bashizade R, Zhang X, Mukherjee S, Lebeck AR (2021) Accelerating markov random field inference with uncertainty quantification. *arXiv preprint arXiv:2108.00570*.
- Conte T, et al. (2019) Thermodynamic computing.
- Hylton T, Conte TM, Hill MD (2021) A vision to compute like nature: thermodynamically. *Commun. ACM* 64(6):35–38.
- Coles PJ, et al. (2023) Thermodynamic ai and the fluctuation frontier.
- Aifer M, et al. (2023) Thermodynamic linear algebra.
- Duffield S, Aifer M, Crooks G, Ahle T, Coles PJ (2023) Thermodynamic matrix exponentials and thermodynamic parallelism.
- Aifer M, et al. (2024) Error mitigation for thermodynamic computing.
- Gordon MH, et al. (2023) Exploring thermodynamic ai. [online] <https://normalcomputing.substack.com/p/exploring-thermodynamic-ai>.
- Cardwell SG, et al. (2022) Probabilistic neural circuits leveraging ai-enhanced codesign for random number generation.
- Liu S, et al. (2022) Random bitstream generation using voltage-controlled magnetic anisotropy and spin orbit torque magnetic tunnel junctions. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 8(2):194–202.
- Misra S, et al. (2023) Probabilistic neural computing with stochastic devices. *Advanced Materials* 35(37):2204569.
- Theilman BH, et al. (2022) Stochastic neuromorphic circuits for solving maxcut.
- Rehm L, et al. (2023) Stochastic magnetic actuated random transducer devices based on perpendicular magnetic tunnel junctions. *Phys. Rev. Appl.* 19(2):024035.

51. Aimone JB, Misra S (2023) Will stochastic devices play nice with others in neuromorphic hardware?: There's more to a probabilistic system than noisy devices. *IEEE Electron Devices Magazine* 1(2):50–56.
52. Rehm L, et al. (2023) Temperature-resilient true random number generation with stochastic actuated magnetic tunnel junction devices.
53. Aimone JB, Severa W, Smith JD (2023) Synaptic sampling of neural networks.
54. Wolpert D, et al. (2023) Is stochastic thermodynamics the key to understanding the energy costs of computation?
55. Maicke A, et al. (2023) Magnetic tunnel junction random number generators applied to dynamically tuned probability trees driven by spin orbit torque.
56. Kermarrec F, Bourdeauducq S, Lann JCL, Badier H (2020) Litex: an open-source soc builder and library based on migen python dsl. *arXiv preprint arXiv:2005.02506*.