Agile Simulation of Stochastic Computin

Sercan Aygun , M. Hassan Najafi

***Abstract*—The rapid computerized simulation of stochastic computin (SC) systems is a challenging problem. A method for agile simulatio of SC image processing is proposed in this work. The input operand are processed with the aid of a correlation-controlled contingency tabl (CT) construct without using actual stochastic bit-streams. The propose approach underlines the validity of CT simulation with (i) imag compositing, (ii) pattern detection, and (iii) bilinear interpolation cas studies. Using the corresponding error models, we emulate the state of-the-art pseudo-random and quasi-random bit-streams. Experimenta results show that the proposed approach achieves similar computatio accuracy to the traditional SC simulation while performing runtime- an memory-efficient computations. The execution time reduces more tha 200× for the image compositing task when emulating random bit-stream with CT. Pattern detection and bilinear interpolation further showed 76 and 22× lower memory usage, respectively, when employing CT.**

***Index Terms*—Bit-stream processing, computer-aided simulation, con tingency table, image processing, stochastic computing.**

I. Introduction

THE simulation of stochastic computing (SC) systems [1] face time and memory complexity challenges due to conducting ver long bit-by-bit processing. Especially, simulating SC-based process ing of high-density data like image matrices in row-column forma is very time-consuming. In SC, the precision of data and the qualit of the results are affected by the length of stochastic bit-streams [1] The larger the bit-stream size (N), the higher the accuracy of th computations [2]. Often very long bit-streams, in orders of 10^3 to 10 bits, must be processed for high-quality results. To have an outpu bit-stream with a resolution of 8 bits, bit-streams of at least 2^{16} bit need to be processed [3].

SC has been popular for simple execution of complex imag processing tasks. Fig. 1 summarizes some important prior SC imag processing case studies and the bit-stream size used in each case Considering the large number of image data that need to be processed fast simulation of these SC-based systems is challenging even wit short bit-stream sizes of 100 bits. Sometimes large design spac explorations are done by varying the bit-stream size from hundred to tens of thousands bits to study the performance of an SC system

Instead of explicitly generating and processing stochastic bit streams, this work performs SC in a radically different way. Sim ple arithmetic operations on scalar values replace traditional bit-wise operations on stochastic bit-streams. The input operands are processed with the aid of a contingency table (CT) construct [4] without explicitly processing bit-streams. This allows latency-free and memory-aware emulation of SC systems without impacting the results. We show that similar accuracy to traditional stochastic bit-stream processing is achieved with this approach. CT's usage has not been explored before at the application level. This work studies the effect of CT in the simulation of SC image processing case studies. We utilize some simple SC circuits (2-to-1 MUX, XOR, and 4-to-1 MUX) for *image compositing*, *pattern detection*, and *bilinear interpolation* tasks for the first time. In summary, the main contributions of this work are as follows:

- Developing simple SC designs for three image processing tasks: 2-to-1 MUX (image compositing), XOR (pattern detection), and 4-to-1 MUX (bilinear interpolation).

Manuscript received XXXXXXXX XX, 2022; accepted XXXXXXXX XX, 2023. Date of publication XXXXXXXX XX, 2023; date of current version XXXXXXXX XX, 2023. This article was recommended by Associate Editor XXXXXXX XXXXXXX. *(Corresponding author: Sercan Aygun.)*

Sercan Aygun is with the ECE Dept., Istanbul Technical Univ., 34469 Istanbul, Turkey, and also with the Sch. Comp. and Info., University of Louisiana at Lafayette, Lafayette, LA, 70503, USA (e-mail: ayguns@itu.edu.tr).

M. Hassan Najafi is with the Sch. Comp. and Info., University of Louisiana at Lafayette, Lafayette, LA, 70503, USA (e-mail: najafi@louisiana.edu).

Mohsen Imani is with the Department of Computer Science, University of California Irvine, Irvine, CA, 92697, USA, (e-mail: m.imani@uci.edu).

Ece Olcay Gunes is with the ECE Dept., Istanbul Technical Univ., 34469 Istanbul, Turkey (e-mail: gunesec@itu.edu.tr).

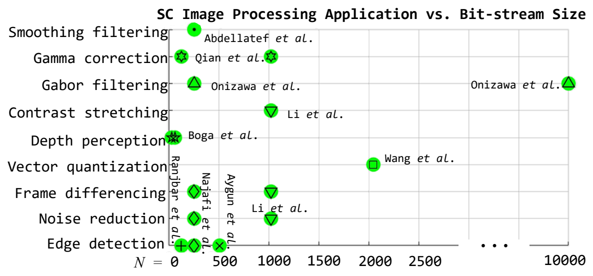


Fig. 1. Examples from the literature for SC image processing applications and applied bit-stream size. (◆ → [5], ✡ → [6], ⊕ → [7], ■ → [8], ⊗ → [9], ▼ → [10], ▲ → [11], [12], ✪ → [13], ⊙ → [14].)

- Evaluating CT-based SC simulation for image processing case studies.
- Emulating state-of-the-art Sobol-based low-discrepancy (LD), binomial distributed, and linear-feedback shift register-based (LFSR) bit-streams with CT.
- Runtime comparison of SC bit-stream processing and CT-based simulation on CPU and GPU.

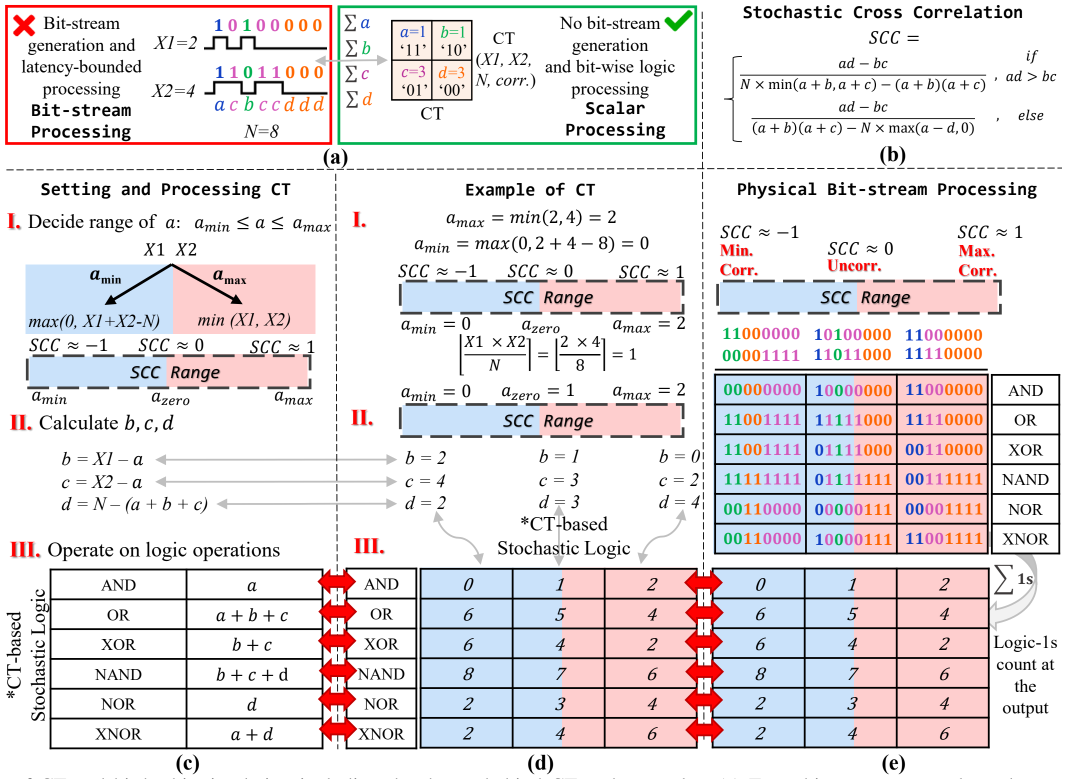
This article is organized as follows: Section II presents the motivation of the study and gives the basic concepts of SC and CT. Section III is devoted to the methodology. Experiments results are given in Section IV. Section V concludes the paper.

II. Stochastic Computing and Contingency Tables

SC is a re-emerging paradigm for the cost-efficient and fault-tolerant design of digital systems. Data values are represented with bit-streams with equal bit significance. Arithmetic operations are implemented by performing simple bit-wise operations on the bit-streams. For example, multiplication is realized by bit-wise AND (XNOR) on unipolar (bipolar) bit-streams [6]. To accurately multiply two n-bit precision data, bit-streams of at least 2^{2n} bits must be processed [3]. Generating and processing such long bit-streams is very time and memory consuming, especially when n increases.

The power of SC stems from the probabilistic behavior of random bit-streams in which 1s and 0s occur randomly in no specific order. The state-of-the-art distributions for stochastic bit-streams are: Binomial distribution, Sobol-based LD, and LFSR-based pseudo-random. During bit-stream generation, a *stochastic bit-stream generator* is coupled with a *comparator* that compares a randomly generated binary number with the to-be-encoded input scalar value. If the scalar value is greater than the random number, the corresponding bit of the bit-stream is 1; otherwise, it is 0. LFSR-based randomization is the most popular bit-stream generation method in the literature [15]. LFSR provides pseudo-randomness with binary numbers that are generated circularly. Some recent works use Sobol sequences to generate quasi-random numbers. Sobol sequences are used to generate LD bit-streams for highly-accurate SC arithmetic [16], [17]. The third approach uses binomial distribution, and has been used in the literature in quantifying the random fluctuation error of SC operations. The probability P, represented by a stochastic bit-stream, can be considered based on a set of samples from a random variable (RV) having a Bernoulli distribution with a success probability of P [18]. Bernoulli distribution is employed to produce bit-streams with uniformly distributed bits. This distribution is obtained by using N different Bernoulli trials.

As a bit-stream emulation construct, CT, is a 2×2 table of four scalar values. As if i-bit bit-streams are generated, the binary interaction between input operands (i.e., bit-streams) are recorded for 11, 10, 01, and 00 logic pair overlaps in any i position of the bit-streams. Fig. 2 (a) shows an example of generating a CT. Logic pairs at any bit position i are cumulatively denoted using


Bit-stream generation and latency-bounded processing
Bit-stream Processing
X1=2 1 0 1 0 0 0 0 0
X2=4 1 1 0 1 1 0 0 0
a c b c c d d d
N=8
Σa Σb Σc Σd
a=1 '11' b=1 '10' c=3 '01' d=3 '00'
CT (X1, X2, N, corr.)
No bit-stream generation and bit-wise logic processing
Scalar Processing
(a)
Stochastic Cross Correlation
SCC = (ad − bc) / (N × min(a+b, a+c) − (a+b)(a+c)), if ad > bc; (ad − bc) / ((a+b)(a+c) − N × max(a−d, 0)), else
(b)
Setting and Processing CT
I. Decide range of a: a_min ≤ a ≤ a_max
max(0, X1+X2−N)
min (X1, X2)
SCC ≈ −1 SCC ≈ 0 SCC ≈ 1
SCC Range
II. Calculate b, c, d
b = X1 − a
c = X2 − a
d = N − (a + b + c)
III. Operate on logic operations
Example of CT
a_max = min(2,4) = 2
a_min = max(0, 2+4−8) = 0
a_zero = ⌊X1 × X2 / N⌋ = ⌊2 × 4 / 8⌋ = 1
b = 2 c = 4 d = 2
b = 1 c = 3 d = 3
b = 0 c = 2 d = 4
*CT-based Stochastic Logic
Physical Bit-stream Processing
SCC ≈ −1 Min. Corr.
SCC ≈ 0 Uncorr.
SCC ≈ 1 Max. Corr.
Σ 1s
Logic-1s count at the output
(c) (d) (e)

*CT-based Stochastic Logic		
AND	a	
OR	$a+b+c$	
XOR	$b+c$	
NAND	$b+c+d$	
NOR	d	
XNOR	$a+d$	

	(d) a_{min}	(d) a_{zero}	(d) a_{max}	(e) 11000000 / 00001111	(e) 10100000 / 11011000	(e) 11000000 / 11110000
AND	0	1	2	00000000 → 0	10000000 → 1	11000000 → 2
OR	6	5	4	11001111 → 6	11111000 → 5	11110000 → 4
XOR	6	4	2	11001111 → 6	01111000 → 4	00110000 → 2
NAND	8	7	6	11111111 → 8	01111111 → 7	00111111 → 6
NOR	2	3	4	00110000 → 2	00000111 → 3	00001111 → 4
XNOR	2	4	6	00110000 → 2	10000111 → 4	11001111 → 6

Fig. 2. Summary of CT and bit-by-bit simulation including the theory behind CT and examples. (a) From bit-streams to scalar-only processing. (b) SCC formula for quantifying correlation. (c) The steps for setting up CT, and the relation between CT primitives (a, b, c, d) and logic operations. (d) A numerical example with $X1 = 2$, $X2 = 4$ from bit-streams to scalar-only processing. (e) The same example with actual bit-streams.

a, b, c, and d. These are called *CT primitives*. Logic operations are defined using these primitives. For example, AND operation corresponds to 11 (i.e., a) bit pairs between the two bit-streams, $\boldsymbol{X1}$ and $\boldsymbol{X2}$. Without explicitly generating bit-streams, the CT is directly generated from the scalar (binary equivalent) values of $X1$ and $X2$, where $0 \leq X1, X2 \leq N$, and N is the bit-stream length. Now the question is how to find a, b, c, and d values directly from scalar values. The answer depends on "correlation". For correct functionality, some SC designs need input bit-streams with maximum correlation. Some others need inputs with minimum correlation, and some need uncorrelated inputs [1].

To emulate maximally correlated bit-streams, a is set maximally, while for minimum correlation, a is minimum. a is the first CT primitive we find. The formulas for the maximally (a_{max}) and minimally (a_{min}) "11" occurrences are presented in Fig. 2 (c)-I. After finding a, by using the scalar values of $X1$, $X2$, and N, the other primitives (b, c, d) are found. The formulas are shown in Fig. 2 (c)-II. Finally, logic operations are converted to basic arithmetic on the primitives. Fig. 2 (c)-III shows how different logic operations can be obtained from CT primitives.

Besides the cases with maximum and minimum corr
ulating *uncorrelated* or independent bit-streams is in
modeling SC systems. Stochastic Cross Correlation (SC
(b)) [19] with range of $[-1, 1]$ is suggested as a metric
the correlation level between two bit-streams. Two bit-
uncorrelated if $SCC \approx 0$. When the SCC formula is
$SCC = 0$, the following equation is obtained to set up
correlation: $a_{zero} = \lfloor \frac{X1 \times X2}{N} \rfloor$. Fig. 2 (d) provides an
setting up CT for $X1 = 2$ and $X2 = 4$. Steps I, II, an
the CT formation for three correlation points: minimum
maximum. The numeric table in step III shows the num
the output bit-stream resulting from different logic operat
(e) shows example bit-streams for $X1$ and $X2$ scalars
correlation points and the corresponding logic results usin
bit-by-bit processing. As it can be seen, the logic resul
Fig. 2 (d) and Fig. 2 (e) are the same, proving the correc
CT method.

III. Proposed Approach

A. A Generic CT Set-Up

We first define a generic CT set-up for simulating multi-level cascaded SC circuits. Fig. 3 depicts the encapsulated CT setting followed from the circuits' inputs to the output. Considering $Y1$ and $Y2$ connected to any gate type at the mid-level, the AND reference gate is temporarily evaluated using near-zero correlation, a_{zero} (Step I). The deviation from the *expected* value (i.e., error-free multiplication value) is determined by random fluctuation error, ϵ, considering the model of random sources that generate input bit-streams (Step II). The *actual* value is estimated through ϵ. Finally, other CT primitives are calculated based on actual a (Step III). For more complex circuits with reconvergent paths (especially when processing longer bit-streams, e.g., 512, 1024), the designer may benefit from the reconvergence involving correlation. At the output of the reconvergent paths, signal correlation can be used to define the expected value via a_{min} or a_{max}. We observe that reconvergence awareness in CT simulations may be beneficial to reduce error especially for complex circuits (*levels* > 3) when processing long bit-streams (please see the GitHub repository [20] for some examples.) Nevertheless, we recommend the expected value assignment in Fig. 3 for the image processing circuits (those having $\epsilon \neq 0$) in Section III-B.

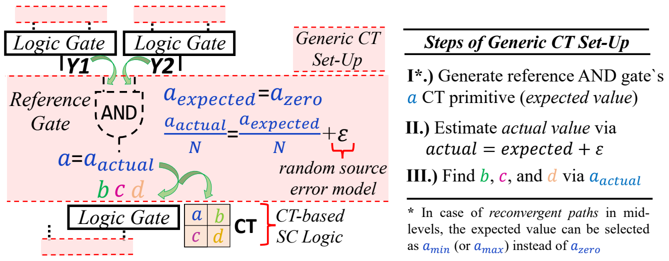

Logic Gate
Logic Gate
Y1
Y2
Generic CT Set-Up
Reference Gate
AND
a_expected = a_zero
a_actual / N = a_expected / N + ε
random source error model
a=a_actual
b c d
Logic Gate
a b c d
CT
CT-based SC Logic
Steps of Generic CT Set-Up
I*.) Generate reference AND gate`s a CT primitive (expected value)
II.) Estimate actual value via actual = expected + ε
III.) Find b, c, and d via a_actual
* In case of reconvergent paths in mid-levels, the expected value can be selected as a_min (or a_max) instead of a_zero

Fig. 3. Generic CT set-up.

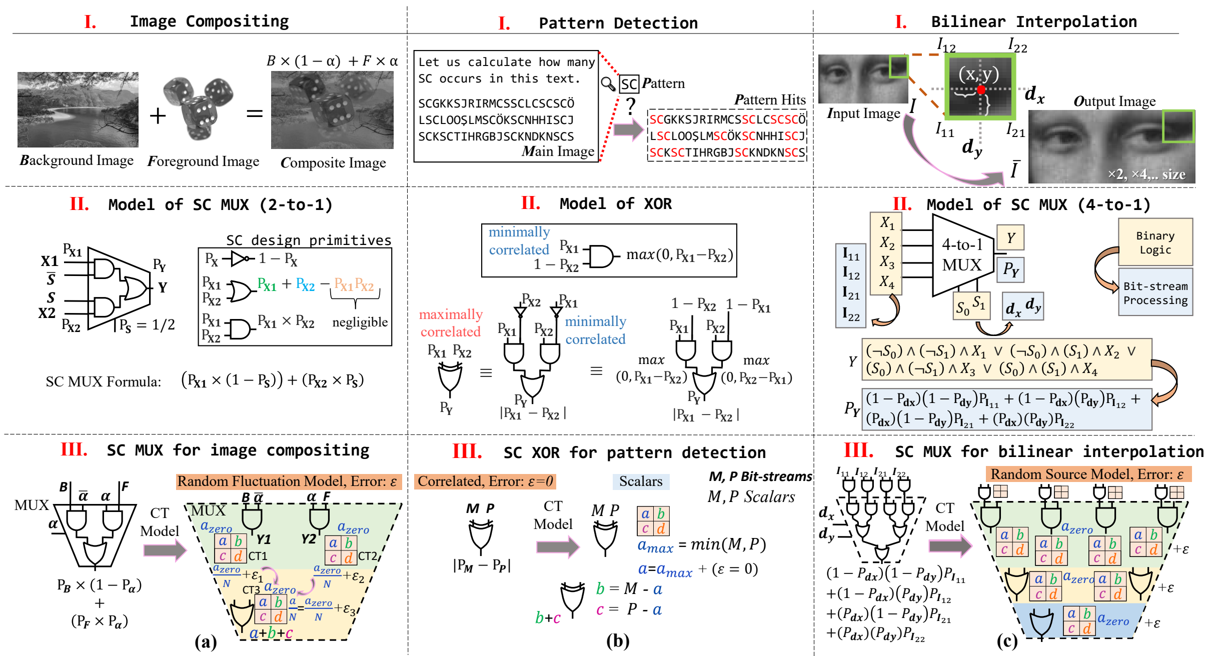


Fig. 4. Proposed CT-based (a) image compositing, (b) pattern detection, and (c) bilinear interpolation.

detection, and (iii) bilinear interpolation. The first case study is image compositing. Szeliski [21] elaborates image compositing by giving the compositing (C) formula $C = B(1-\alpha) + F\alpha$, where B is the original background image, α is the foreground image opacity, and F is the foreground image. Fig. 4 (a)-I shows an example of image compositing. The image compositing formula reminds the equation of an SC scaled adder implemented using a multiplexer (MUX): $MUX = P_{\boldsymbol{X1}}(1-P_{\boldsymbol{S}})+P_{\boldsymbol{X2}}P_{\boldsymbol{S}}$, where $P_{\boldsymbol{S}}$ is the probability of the select input [6]. The SC MUX formula is elaborated in Fig. 4 (a)-II. As the image compositing and MUX formulas coincide, the image compositing can be realized by simply using a MUX unit. The main inputs are the background and foreground image pixel bit-streams, and the select input is the foreground image opacity, α. The CT model for a MUX (built from two AND and one OR gates) is obtained using multiple tables as depicted in Fig. 4 (a)-III. Using the scalar values of B, F, and α, first the near-zero-correlation CTs are created. Since the AND operation result is the same as the "a" primitive, CT1 is set by considering near-zero a (a_{zero}): $CT1_{a_{zero}} = \lfloor \frac{B\times(N-\alpha)}{N} \rceil$. Likewise, CT2 is set by $CT2_{a_{zero}} = \lfloor \frac{F\times\alpha}{N} \rceil$. After calculating the zero-correlated a value, the random fluctuation error (ϵ) from the input bit-stream generators should also be taken into account by carrying out the steps of the generic CT set-up shown in Fig. 3. In this study, we emulate three different bit-stream generation models: binomial distribution, Sobol LD, and LFSR.

For the binomial distribution case, a stochastic bit-stream having probability P is considered as a set of samples from RV with a Bernoulli distribution having a success probability, P. Considering the *Independent and Identically Distributed* RV [22], a stochastic number is represented yielding binomial distribution with a variance $\sigma^2 = P(1-P)/N$. As a reference, the AND operation (corresponding to CT primitive "a") has an expected output probability $P_Y = \mathbb{E}[\boldsymbol{Y}]$, yielding $P_{X1} \times P_{X2}$ for independent variables. Nevertheless, the obtained probability, $\hat{P}_Y$, may differ due to *random fluctuations*. Using squared error, the random fluctuations error for the case of Bernoulli RVs is obtained as $err_Y = \mathbb{E}[(P_Y - \hat{P}_Y)^2] = P_Y(1-P_Y)/N$ [1], [23]. From the squared error to the square-rooted format, the error is ϵ=$\sqrt{err_Y}$. Considering AND as the reference and initial operation, we include $\pm\epsilon$ in the obtained CT output probability. Fig. 4 (a)-III has AND gates in the first level of MUX. After obtaining a_{zero}, the obtained output probabilities of CT1 and CT2, $\frac{a}{N}$, are determined via $\frac{a_{zero}}{N} \pm \epsilon$ (our error calculations show that $\frac{a}{N} + \epsilon$ has better error performance than $\frac{a}{N} - \epsilon$; therefore, we use $+\epsilon$ for the model.) For the LFSR case, we model error by assuming hypergeometric distribution as recommended by Baker and Hayes [24]. They define the output deviation of performing bit-wise AND operation on LFSR-based bit-streams as $\epsilon = \sqrt{\frac{P_{X1}\times P_{X2}\times(1-P_{X1})\times(1-P_{X2})}{N-1}}$, where $X1$ and $X2$ are the inputs. Finally, unlike the binomial and LFSR cases, the Sobol-based LD model almost equals the near-zero correlation CT, i.e., a_{zero}, where ϵ=0.

In Fig. 4 (a)-III, after including the relevant error in CT1 and CT2, a new CT is set for the next logic operation, which is an OR operation. Similar to the first level of the circuit, the generic CT set-up is followed for this level of logic. The expected output probability and the input probabilities are important for calculating ϵ in the binomial and LFSR-based random source error, respectively. The AND gate model as a reference is initially calculated for the random source error; thereby, the CT prior primitive a is first fine-tuned. Returning to Fig. 4 (a)-III, for modeling MUX, CT3's near-zero correlated output probability, $\frac{a_{zero}}{N}$, is updated with ϵ. After updating the CT prior primitive, the other primitives (b, c, d) are calculated for the final logic operation (e.g., OR gate: $a+b+c$, recall Fig. 2).

The second case study is pattern detection, shown in Fig. 4 (b)-I. This task is based on *maximum correlation*. The two inputs from the main image and the pattern image are maximally correlated to utilize XOR gate as an SC subtractor. Fig. 4 (b)-II elaborates on the formula and functionality of the XOR gate as an SC primitive. As shown, when input bit-streams are positively correlated (i.e., have maximum overlap between the position of 1's), bit-wise XOR acts as an absolute subtractor, $|P_{\boldsymbol{X1}} - P_{\boldsymbol{X2}}|$ [25]. This can be used for pattern search in an image, where $\boldsymbol{X1}$ relates to the main image (M), and $\boldsymbol{X2}$ holds the pattern (P) to be searched in $X1$. The difference gives zero value for the exact pattern match. The CT for the pattern detection task is elaborated in Fig. 4 (b)-III. The model of positively correlated input operands is established in CT using a_{max}. For this case study, the generic CT setting assigns a_{max} to the expected value, which is also equal to the actual value since ϵ=0. First a_{max} is found using $min(X1, X2)$ formula, and then the b and c primitives are calculated. The XOR result is obtained from CT via $b+c$, without explicit bit-stream computing. Thus, the difference between M and P images is obtained. This operation can be considered a convolution-like operation; P is shifted as a sliding window over M.

The third case study is bilinear interpolation used for image resizing. This task is based on linear interpolations in both x (width) and y (height) directions. By repeating linear interpolations for x and y, bilinear interpolation is performed. Assume an image I defined by a rectangular region with four points: $(\mathrm{x}_1, \mathrm{y}_1)$, $(\mathrm{x}_1, \mathrm{y}_2)$, $(\mathrm{x}_2, \mathrm{y}_1)$, and $(\mathrm{x}_2, \mathrm{y}_2)$. A new pixel point in this region is denoted as (x, y), and is to be estimated. Considering the unit square for the rectangular

TABLE I
COMPARING BIT-WISE BIT-STREAM PROCESSING AND CT APPROACH IN DIFFERENT COMPUTING PLATFORMS*: CPU AND GPU

Conventional Bit-stream Processing		Contingency Tables					
		2^5	2^6	2^7	2^8	2^9	2^{10}
		NO DATA DEPENDENCY					
		1.504	1.494	1.499	1.661	1.518	1.560
		0.470	0.445	0.438	0.442	0.432	0.428
		DATA DEPENDENCY					
		2.626	2.891	2.885	2.777	2.823	2.811
		0.644	0.648	0.638	0.640	0.637	0.641

ınd with NVIDIA GTX 1070 GPU, using Python.

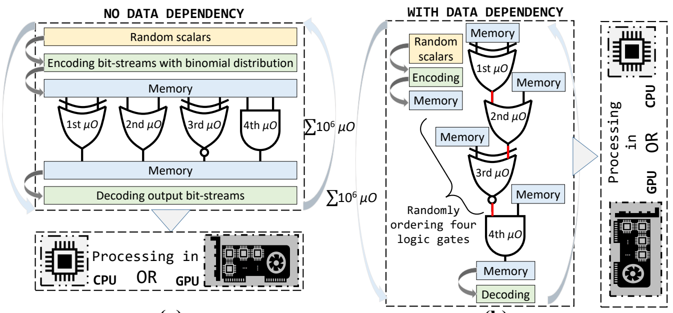


Fig. 5. Test procedure for simulating SC operations (a) with no data dependency and (b) with data dependency.

region, the expression for $I(\mathrm{x,y})$ is $(1-dx)(1-dy)I_{11} + (1-dx)(dy)I_{12} + (dx)(1-dy)I_{21} + (dx)(dy)I_{22}$ [26], where I_{11}, I_{12}, I_{21}, I_{22} are neighbouring pixel values, and dx, dy are new pixel's relative positions that define resizing coefficient, ρ. An example of bilinear interpolation is shown in Fig. 4 (c)-I. We use a 4-to-1 MUX to realize this task in stochastic domain. Fig. 4 (c)-II shows the binary logic of a 4-to-1 MUX and the corresponding transformation to bit-stream processing using probabilities. dx, dy are connected to the select ports of the MUX. Fig. 4 (c)-III depicts the CT model that follows the same procedures in Fig. 3 and Fig. 4 (a)-III explained above.

IV. TESTS AND RESULTS

In this section, we evaluate the performance of the proposed technique compared to the conventional approach of simulating SC. Before evaluating the performance of the SC image processing designs, we conduct a preliminary analysis of computer-aided simulation of SC operations on CPU and GPU. We use Python for the tests and the `Cuda` library for GPU simulations. The operations are guaranteed to run on a single core for the CPU and in parallel for the GPU tests. For simulating the conventional bit-stream processing, the bit-streams are first generated and stored in memory and then used for logic operations. We consider two cases of "no data dependency" (Fig. 5 (a)) and "with data dependency" (Fig. 5 (b)) in the SC operations. For the "no data dependency" case, simulations of single and independent logic operations (i.e., 2-input `AND`, `OR`, `XOR`, and `XNOR`) are evaluated. In this case, the logic operation does not depend on the output of other operations. We denote the total number of logic operations in each test by micro-operation (μO). N is selected from $\{2^3, 2^4, ..., 2^{10}\}$ during the tests. The cases "with data dependency" include cascaded logic operations. In each test, four-level cascaded logic operations are performed. Each operation waits for the result of another operation (except the first operation), thereby having data dependency. The processing times are reported in Table I for an average of 1000 independent tests. As it can be seen, the conventional bit-stream simulations are significantly slower than the CT-based simulations. In all cases, the GPU-based simulation is faster than CPU-based run. In particular, running on GPU reduces the simulation time of the conventional bit-stream processing up to 189$\times$ compared to running on CPU. We observe that the CT-based simulation is 142$\times$ faster than the conventional approach for simple logic operations with no data dependency, and delivers 229$\times$ better runtime for the case with data dependency when running on GPU. We also conducted an accuracy evaluation for the circuit structure shown in Fig. 5 (b). We first fed this four-level circuit by binomially distributed bit-streams, and then constructed the corresponding CT-based model by considering ϵ at each level. We compared the output probabilities from the two models and measured the mean absolute error (MAE) for 1000 times simulating the circuit with different random input values. For N=1024, the MAE between two simulation models was 0.024.

Next, we evaluate the runtime, memory usage, and accuracy of the three discussed image processing tasks, as shown in Fig. 6. Tests are developed in the MATLAB tool. For the conventional simulation method, the `binornd()` function is utilized for generating the binomially distributed bit-streams. The MATLAB built-in Sobol generator is used for generating LD sequences. For the LFSR-based approach, random numbers are generated using the maximal-period LFSRs described in [27].

First, different foreground images are embedded in different-sized background images for the image compositing design with N=256 which can accurately represent 8-bit pixel values of grayscale images. We evaluate five different cases: (i) SC using binomial random bit-streams (SCRandom), (ii) SC using state-of-the-art Sobol bit-streams (SCSobol) [16], (iii) CT-based SC with near-zero correlation (CT0), and (iv) Conventional binary image processing (CONVN). We also evaluate a case where the random fluctuations error of random bit-streams is included in the CT method. Built-in fluctuations based on Bernoulli distribution are considered during a primitive calculation. This is given as (v) CT-based SC with random fluctuations (CTRAND).

We note that the emulation of random fluctuations is fairly controlled as both SCRandom and CTRAND methods show similar PSNR (peak signal-to-noise ratio) values in the image compositing task. Besides, CT0 competently emulates Sobol-based bit-stream processing (SCSobol) as their PSNR values also match. Therefore, this study also provides a fast and efficient way to simulate LD Sobol-based SC [17]. As it can be seen in Table II, for both "People" and "Plane" test images of the image compositing task, the CT-based approaches (CT0 and CTRAND) are far better in runtime compared to SCSobol and SCRandom methods. Memory performance of the image compositing task is enhanced by $\sim$1048.58 Bytes per MUX-based operation compared to the conventional bit-stream processing. Since PSNR is an aggregate measure over all image pixels, we also evaluate the per-pixel performance between the composite images produced by the SCRandom and CTRAND methods using MAE. First, excluding pixels that bring an absolute error of zero ($|CTRAND - SCRandom| = 0$), we only considered the pixels that have an error ($|CTRAND - SCRandom| \geq 1$). The MAEs were 5.897 and 5.352 for the people and plane test images, respectively. MAEs, when including all pixels, were 2.762 and 2.496, respectively. We also measured the PSNR values for the case of having all pixels. The PSNR for the people example was 34.416 dB, and that for the plane example was 34.950 dB.

We evaluate the pattern detection case study in the following scenarios: (i) SC with maximally correlated random bit-streams (SCMax), (ii) CT with maximum correlation (CTMAX), and (iii) Conventional binary image processing (CONVN). The simulation results are shown in Table II.

For pattern detection, the proposed CTMAX method is as fast as the conventional binary execution (CONVN). While the SCMax's runtime increases by increasing the bit-stream length (N), CTMAX's runtime remains constant. The CTMAX's runtime was the same

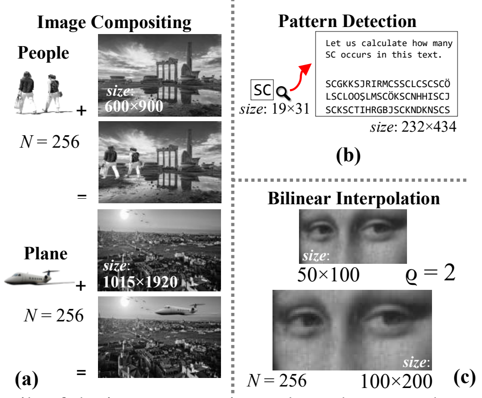


Fig. 6. Details of the image processing tasks under test and sample inputs.

TABLE II
SIMULATION RESULTS OF THE IMAGE PROCESSING TASKS

Image Compositing				
	People		**Plane**	
Method	**RT (sec)**	**PSNR (dB)**	**RT (sec)**	**PSNR (dB)**
SCRandom	46.34	31.17	166.5	31.80
CTRAND	0.194	30.66	1.101	31.25
SCSobol	604.2	50.42	2662.2	52.88
CT0	0.153	50.38	0.961	52.86
CONVN	0.072	ref.	0.618	ref.

Pattern Detection*				
N	**CONVN**	**SCMax (sec)**	**CTMAX (sec)**	**Memory**
8	7.653 sec	91.43	7.59	× 5.075
16		128.69	7.62	× 9.819
32		211.60	7.60	× 19.305
64		384.88	7.61	× 38.277
128		755.64	7.62	× 76.223

*Accuracy: 100% Pattern hits are obtained in all tests.

Bilinear Interpolation			
Method	**RT* (sec)**	**PSNR (dB)**	**Memory**
SCRandom	8.186	30.496	× 22.792
CTRAND	0.024	29.857	
SCSobol	9.325	45.778	× 22.803
CT0	0.015	44.661	
SCLFSR	7.707	35.927	× 22.779
CTLFSR	0.025	34.665	
CONVN	0.009	ref.	

*RT: RunTime. For memory monitoring, MATLAB *whos* command is utilized. Tests are performed on a computer with Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz, 16GB RAM.

until $N = 2^{32}$, after which MATLAB was unresponsive due to the array size limit. A constant runtime independent of N is another important advantage of the proposed technique. As Table II shows, the memory efficiency of CT simulation improves compared to the bit-stream processing as N increases. The MATLAB built-in absolute value function (`abs()`) makes CONVN slightly slower in pattern detection application. However, we observed that the MATLAB built-in functions speed up for large number of iterations ($> 10^5$) when the size of testing images increases. Our simulation results show that the CT-based approach can address the long latency issue of simulating SC image processing designs, completing the computations as fast as conventional binary processing.

Finally, we evaluate the simulation of the bilinear interpolation design. In addition to the previous computing scenarios, we include conventional SC with LFSR-based bit-stream processing (SCLFSR) and CT with LFSR-model fluctuations (CTLFSR) in our simulations. We validated the emulation of the three random source methods. Particularly, we considered hypergeometric distribution for emulating LFSR-based bit-streams in the CT model. As expected, the run-times were much shorter with CT, and the PSNR values were closely followed. Memory efficiency was also 22× better with CT-based simulations compared to conventional SC simulation.

V. CONCLUSION

This work proposes a fast method for simulation of SC image processing systems. The data are processed with the aid of CTs without actual bit-stream processing. The methodology is evaluated on three SC image processing case studies: image compositing, pattern detection, and bilinear interpolation. Experimental results show that the proposed approach can simulate SC nearly as fast as conventional binary processing with a substantial reduction in memory usage. In future work, we will employ the CT-based SC simulation technique for fast and efficient simulation of SC-based deep learning systems.

ACKNOWLEDGMENT

This work was supported in part by National Science Foundation (NSF) grants #2127780 and #2019511, SRC Global Research Collaboration, AIHW and HW Security, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225 and #N00014-22-1-2067, Air Force Office of Scientific Research, grant #22RT0060, the Louisiana Board of Regents Support Fund #LEQSF(2020-23)-RD-A-26, and generous gifts from Cisco, Xilinx, and Nvidia.

REFERENCES

[1] A. Alaghi *et al.* The promise and challenge of stochastic computing. *IEEE Trans. on Comp.-Aided Des. of Integ. Circ. and Syst.*, 37(8), 2018.
[2] A. Alaghi *et al.* Stochastic circuits for real-time image-processing applications. In *DAC*, Austin, Texas, USA, 2013.
[3] M. H. Najafi *et al.* Performing stochastic computation deterministically. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 27(12):2925–2938, 2019.
[4] S. Aygun and E. O. Gunes. Utilization of contingency tables in stochastic computing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6):2942–2946, 2022.
[5] M. H. Najafi and D. J. Lilja. High-speed stochastic circuits using synchronous analog pulses. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 481–487, 2017.
[6] W. Qian *et al.* An Architecture for Fault-Tolerant Computation with Stochastic Logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.
[7] M. Ranjbar *et al.* Using stochastic architectures for edge detection algorithms. In *2015 23rd Iranian Conference on Electrical Engineering*, pp. 723–728, 2015.
[8] R. Wang *et al.* Stochastic circuit design and performance evaluation of vector quantization for different error measures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10):3169–3183, 2016.
[9] S. Aygün *et al.* Sobel filter operation in image processing via stochastic arithmetic-logic unit design. In *2017 IEEE SIU*, 2017.
[10] P. Li *et al.* Computation on stochastic bit streams digital image processing case studies. *IEEE Trans. on Very Large Scale Integ. (VLSI) Systems*, 22(3), 2014.
[11] N. Onizawa et al. Gabor filter based on stochastic computation. *IEEE Signal Process. Lett.*, 22(9):1224–1228, 2015.
[12] N. Onizawa *et al.* An accuracy/energy-flexible configurable gabor-filter chip based on stochastic computation with dynamic voltage–frequency–length scaling. *IEEE J. Emerg. Sel. Top. Circuits Syst.*, 8(3):444–453, 2018.
[13] K. Boga *et al.* Stochastic implementation of the disparity energy model for depth perception. In *SiPS*, 2015.
[14] H. Abdellatef *et al.* Accurate and compact stochastic computations by exploiting correlation. *T. J. of Elect. Eng. and Comp. Sci.*, 27, 2019.
[15] J. H. Anderson *et al.* Effect of lfsr seeding, scrambling and feedback polynomial on stochastic computing accuracy. In *2016 DATE*, pp. 1550–1555, Dresden, Germany, 2016.
[16] S. Liu and J. Han. Toward energy-efficient stochastic circuits using parallel sobol sequences. *IEEE Trans. VLSI Sys.*, 26(7), 2018.
[17] M. H. Najafi *et al.* Deterministic methods for stochastic computing using low-discrepancy sequences. In *2018 ICCAD*, 2018.
[18] A. Alaghi. *The logic of random pulses: Stochastic computing*. PhD thesis, University of Michigan, Ann Arbor, USA, 2015.
[19] A. Alaghi and J. P. Hayes. Exploiting correlation in stochastic circuit design. In *IEEE 31st ICCD*, pp. 39–46, 2013.
[20] S. Aygun *et al.* Github. https://github.com/serco425/Agile-SC-Simulation, 2022.
[21] R. Szeliski. Computer vision algorithms and applications, 2011.
[22] R. Manohar. Comparing stochastic and deterministic computing. *IEEE Computer Architecture Letters*, 14(2):119–122, 2015.
[23] B. Moons and M. Verhelst. Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies. *IEEE J. Emerg. Sel. Topics in Cir. and Sys.*, 4(4):475–486, 2014.
[24] T. Baker and J. Hayes. The hypergeometric distribution as a more accurate model for stochastic computing. In *DATE*, France, 2020.
[25] V. T. Lee *et al.* Correlation manipulating circuits for stochastic computing. In *DATE'18*, pp. 1417–1422, 2018.
[26] K. Kim *et al.* An alternative bilinear interpolation method between spherical grids. *Atmosphere*, 10(3), 2019.
[27] P. Koopman. Maximal length lfsr feedback terms. https://users.ece.cmu.edu/~koopman/lfsr/.