Joint Optimization of Randomizer and Computing Core for Low-Cost Stochastic Circuits

Kuncai Zhong, Xuan Wang, Chen Wang, Weikang Qian*
University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China
{kczhong,xuan.wang,wangchen_2011,qianwk}@sjtu.edu.cn

ABSTRACT

Stochastic computing (SC) is an unconventional computing paradigm that computes on stochastic bit streams. It is promising to implement complex functions with low-cost circuitry. A stochastic circuit typically consists of a randomizer to generate the stochastic bit streams and an SC core computing on the bit streams. To design a low-cost stochastic circuit, many works have been proposed to optimize these two parts. However, the works optimize them insufficiently due to the overlook of some optimization space and separately without considering their mutual influence, thus causing the final stochastic circuit sub-optimal. In this work, to address this issue, we first introduce a low-cost randomizer architecture and a method for optimizing the SC core. Then, by combining these two techniques together, we further propose a method to jointly optimize the randomizer and the SC core. Our experimental results show that compared to the conventional method, the proposed joint optimization method can reduce 39.70% area and 42.74% power for the stochastic circuit.

CCS CONCEPTS

• **Hardware → Arithmetic and datapath circuits**.

KEYWORDS

joint optimization, randomizer, stochastic computing core

ACM Reference Format:
Kuncai Zhong, Xuan Wang, Chen Wang, Weikang Qian. 2022. Joint Optimization of Randomizer and Computing Core for Low-Cost Stochastic Circuits. In *17th IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH '22), December 7–9, 2022, Virtual, OR, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3565478.3572540

*This work is supported by the National Key R&D Program of China under Grant 2020YFB2205501. Corresponding author: Weikang Qian. Weikang Qian is also with the MoE Key Lab of AI, Shanghai Jiao Tong University, China.

NANOARCH '22, December 7–9, 2022, Virtual, OR, USA

ACM ISBN 978-1-4503-9938-8/22/12…$15.00
https://doi.org/10.1145/3565478.3572540

1 INTRODUCTION

Stochastic computing (SC) is an unconventional computing paradigm proposed in 1960s [1]. It computes on stochastic bit streams (SBSs), which consists of zeros and ones and encodes the value by the ratio of ones. Compared to the binary computing, it can implement complex functions with simple circuitry and has strong fault tolerance. For example, it only needs an AND gate to implement the multiplication.

A stochastic circuit typically consists of a randomizer and an SC core as shown in Fig. 1. The randomizer generates d_i SBSs encoding the input binary number X_i for $1 \le i \le k$ and m SBSs of constant values. The SC core computes on these SBSs to implement the target function. In this paper, we consider optimizing the stochastic circuits that implement *univariate functions*. In this case, we have $k = 1$. For simplicity, in the following, we denote X_1, d_1, and $x_{1,i}$ as X, d, and x_i, respectively.

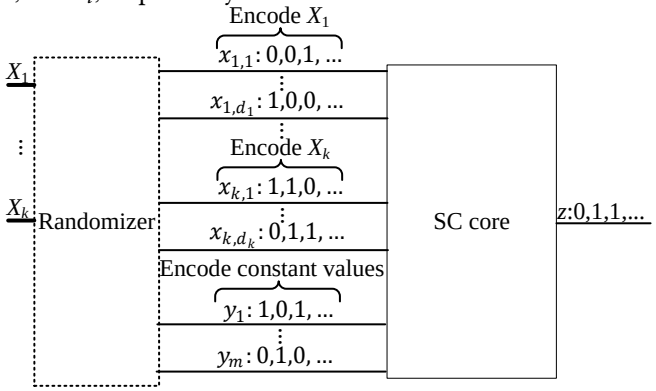

Figure 1: Illustration of a stochastic circuit.

There are many works proposed for stochastic circuit optimization by optimizing either the randomizer [2–5] or the SC core [6–8]. However, these methods generally optimize the two parts insufficiently due to the overlook of some optimization space and separately without considering their mutual influence, thus causing the final stochastic circuit sub-optimal. For example, for the optimization of the randomizer, Ting *et al.* [3] propose a method to efficiently generate multiple SBSs of variable values by inserting D flip-flops (DFFs).

The method can signi cantly reduce the hardware cost. Nevertheless, it does not give an e cient way to generate SBSs of constant values and does not consider further optimizing the SC core.

In this special session paper, we rst review two of our previously proposed techniques for randomizer optimization and SC core optimization, respectively. To optimize the randomizer, we proposed a low-cost randomizer architecture and its associated con guration method in [9]. The architecture only needs a single random number source (RNS), a single comparator, and the minimum number of DFFs to generate all the SBSs of variable and constant values, and the con guration method can lead to a high accuracy. To optimize the SC core, we proposed a dynamical approximation method in [10]. It can e ciently obtain a low-cost SC core satisfying the accuracy requirement.

Then, based on the above two techniques, we propose a method to jointly optimize the randomizer and the SC core in the work. It uses an early-termination strategy for acceleration. The experimental results show that compared to the conventional method, the proposed joint optimization method can reduce 39.70% area and 42.74% power for the stochastic circuit.

We organize the rest of the paper as follows. Section 2 introduces the background and the related works. Section 3 introduces the low-cost randomizer architecture and the method to optimize the SC core, and presents the proposed joint optimization method. Section 4 shows the experimental results, and Section 5 concludes the paper.

2 BACKGROUND AND RELATED WORKS

We introduce some background and related works about the optimization of the stochastic circuit.

2.1 SC core optimization

In general, the SC core approximately implements the target function $5(G)$ by a Bernstein polynomial $5(G)$ as

$$5(G) = \sum_{8=0}^{3} \frac{\binom{3}{8}}{2^<} G^8 (1 - G)^{3-8} \quad (1)$$

where 3 and $<$ are the degree and the precision, respectively, and 8's are the integers in the range of $[0, 2^< - 1]$ [6, 7]. The vector $(0, 1, \ldots, 3)$ is called the feature vector. As shown in Eq.(1), the degree, the precision, and the feature vector determine the approximation error between the Bernstein polynomial $5(G)$ and the target function $5(G)$. To obtain a high-accuracy SC core, we need to rst obtain a Bernstein polynomial with a proper choice of degree, precision, and feature vector, and then design a core to implement this polynomial.

To achieve this, for a speci ed pair of 3 and $<$, Qian et al. rst propose a method to obtain a feature vector giving the minimum approximation error [11]. Then, for this polynomial, Zhao et al. propose a core to implement it as shown in Fig. 2 [6]. The core is based on the combinational logic and takes as inputs 3 SBSs of the variable value G and $<$ SBSs of the constant value 0.5. To e ciently synthesize the core with low hardware cost, Peng et al. propose a method based on the cube assignment [7]. It tries many possible ways to iteratively split the feature vector into some cubes, where a cube can be directly implemented by an AND gate, and the cubes of a split are ORed together to form a sum-of-product (SOP) expression. Then, it further simpli es each SOP expression by applying the two-level logic optimization tool ESPRESSO and obtains the corresponding SC core [12]. Finally, after obtaining all the SC cores for all the feature vector splittings, it chooses the one with the minimum hardware cost.

By the above methods, we can obtain a low-cost SC core approximately implementing the target function with a small approximation error. However, these methods only consider and synthesize a single feature vector for a pair of 3 and $<$. They ignore many other possible feature vectors, which also have small approximation errors and may lead to lower hardware costs.

2.2 Randomizer optimization

SC generally applies a stochastic number generator (SNG) to generate an SBS. As shown in Fig. 2, an SNG typically consists of an RNS and a comparator, where the RNS generates a random binary number $-$ in a clock cycle, and the comparator outputs a 1 if $-$ is less than the input G. In general, to ensure a high accuracy for the stochastic circuit, we apply 3 di erent SNGs to generate 3 independent SBSs. However, it leads to a considerable hardware cost.

Figure 2: Conventional architecture of the stochastic circuit.

To optimize it, several works are proposed [2, 5]. Among them, the one proposed in [3] has a low hardware cost for generating the SBSs of the same value. It can generate 3 SBSs of the same value by applying $3 - 1$ DFFs after an SNG as shown in Fig. 2, where a DFF delays an SBS for a clock cycle and generates another independent one of the same value. However, it does not propose an e cient way to generate SBSs of constant values. Note that in an RNS

generates n bits with the probability of 0.5 to be a one in a clock cycle, and we can use each of them to constitute an SBS of the value 0.5, [13]. Therefore, to generate k SBSs of the value 0.5 and the length L, we can apply an n-bit RNS and select k outputs from it. The conventional architecture for the stochastic circuit is shown in Fig. 2, where BS is a bit selection module to select k outputs from the n-bit RNS. Note that BS costs no hardware overhead. The design only needs 2 RNSs, a comparator, and $\lceil \log_2 k \rceil$ DFFs to generate $\lceil \log_2 k \rceil$ SBSs. However, it still needs k RNSs. In this paper, we will introduce a low-cost architecture that can reduce the number of used RNSs to 1.

2.3 Conventional optimization method

A conventional method optimizes the stochastic circuit based on the above techniques. For a target function $f(x)$ with an error bound given, it first applies the method in [11] to obtain the feature vectors for many different pairs of m and n. Then, for these feature vectors, it applies the method in [7] to synthesize and obtain many SC cores. Afterward, it applies the conventional architecture shown in Fig. 2 to optimize the randomizer for each of these SC cores and obtain many stochastic circuits. Finally, it outputs the stochastic circuit, which has an error less than the error bound and has the minimum hardware cost. The optimized stochastic circuit generally has low hardware cost. However, the method separately optimizes the SC core and the randomizer without considering their mutual influence and needs to synthesize many different SC cores. This generally leads to a sub-optimal stochastic circuit and a long runtime.

3 METHODOLOGY

In this section, we first introduce a low-cost randomizer architecture and its associated configuration method. Then, we introduce a method to optimize the SC core. Finally, based on them, we propose a method to jointly optimize the randomizer and the SC core.

3.1 Method to optimize randomizer

For the optimization of the randomizer, we introduce a low-cost architecture and a configuration method proposed in our previous work [9]. The low-cost architecture is shown as Fig. 3, where BS, SB_i's, and NB_i's are the bit selection module, the scrambling modules, and the negation modules, respectively. Compared to the conventional architecture shown in Fig. 2, it shares an RNS for generating the SBSs of both the value G and the value 0.5, and applies several scrambling and negation modules to permute and negate the output bits, respectively, to improve the accuracy, [9]. Note that the bit selection module, the scrambling modules, and the negation modules cost no hardware overhead, and the number of used DFFs is the minimum to generate k SBSs of the same value G. Thus, the randomizer architecture has a low hardware cost with only a single RNS, a single comparator, and the minimum number of DFFs.

Figure 3: Low-cost architecture of the stochastic circuit [9].

To optimize the accuracy of the randomizer, we also proposed a method to configure the modules' $BS(\ ,\ ,\ NB_1, NB_2, (SB_1, (SB_2$, and (SB_3) as shown in Fig. 4 [9]. It first initializes the modules. Then, it iteratively searches the configurations for some of them while fixing the others in a loop, which consists of 3 steps. In the first step, it exhaustively tries all possible configurations for $(\ , NB_1$, and NB_2, and randomly tries the configurations for $NB(\ $. In the second step, it first exhaustively tries all possible configurations for $(SB_1$ and randomly tries the configurations for $(SB_2$ and $NB(\ ;$ then, it exhaustively tries all possible configurations for $(SB_2$ and randomly tries the configurations for $(SB_1$ and $NB(\ $. In the third step, it tries all possible configurations for $(SB_3$. The method updates the best configuration for the architecture once obtaining a higher accuracy, and terminates when no update is obtained after traversing these 3 steps. As shown in [9], the method can generally lead to a high accuracy.

Figure 4: Flow chart of the configuration method [9].

3.2 Method to optimize the SC core

As discussed in Section 2.1, the previous method [7] only considers a single feature vector for a specified pair of m and n. It ignores other feature vectors, which may lead to lower hardware costs. To address this issue, we propose a dynamic approximation method in our previous work as shown in Fig. 5 [10]. It synthesizes an SC core for a target function $f(x)$ satisfying a given error bound. It additionally takes a

feature vector as input, which we call input feature vector It synthesizes an SC core iteratively with 2 steps in each iteration.

Figure 5: Flow chart of the dynamic approximation method [10].

The first step is to try many possible ways to split the feature vector into a cube and a remaining feature vector The cube is directly implemented by an AND gate, and the remaining feature vector will be left for the next split step. Different than the method proposed in [7], this step does not require that their sum exactly equals the original feature vector. For example, when $n = 3$ and $< = 2$, the feature vector $\{1, 1, 6, 2\}$ can be split into the cube $\{0, 0, 2, 0\}$ and the remaining feature vector $\{1, 1, 4, 2\}$, or the cube $\{0, 2, 4, 2\}$ and the remaining feature vector $\{1, 0, 2, 0\}$, where the first split is exact, while the second is inexact. The cube $\{0, 0, 2, 0\}$ can be implemented by a 4-input AND gate with 3 SBSs of the value 6 and an SBS of the value 0.5 as inputs. The cube $\{0, 2, 4, 2\}$ can be implemented by a 2-input AND gate with an SBS of the value 6 and an SBS of the value 0.5 as inputs. The remaining feature vectors $\{1, 1, 4, 2\}$ and $\{1, 0, 2, 0\}$ will be left for the next split step. For the second split, as the sum of the cube and the remaining feature vector does not equal the original feature vector, the sum leads to a new feature vector for the target function $5(G)$ with a new approximation error, which further leads to some new SC cores. To satisfy the accuracy requirement, we abandon any split with an approximation error larger than the error bound.

By applying the first step, we obtain many different splits for the input feature vector and have more possibility to find an SC core with a lower cost. However, as the process goes on, the number of splits grows exponentially, leading to a very large design space that needs a very long runtime to explore. To speed up the process, the second step prunes some unpromising splits, which are more likely to have high hardware costs than the others. For example, for the splits $\{0, 0, 2, 0\}, \{1, 1, 4, 2\}$ and $\{0, 2, 4, 2\}, \{1, 0, 2, 0\}$, their cubes need a 4-input and a 2-input AND gates to implement, respectively. The former split is more likely to have a high hardware cost than the latter. Therefore, we can prune the former.

The loop terminates when no feature vector is left to split. Then, similar to [7], we construct many Boolean functions based on the cubes and further simplify them to the SC cores by applying ESPRESSO [24]. Among the cores, we choose the one with the lowest hardware cost. Compared to the method proposed in [7], this method dynamically considers many possible feature vectors to approximate the target functions in the synthesis process and generally obtains an SC core with a lower cost.

3.3 Joint optimization method

To optimize a stochastic circuit, a simple method is to separately apply the above randomizer and SC core optimization methods. Specifically, for a target function with an error bound given, we can first apply the dynamic approximation method to obtain an optimized SC core based on the initial input feature vector obtained by the method from [11], and then apply the low-cost architecture and the configuration method to optimize the randomizer. The method is very efficient to optimize a stochastic circuit. However, it still optimizes the randomizer and the SC core separately Such a separate method will sometimes lead to an optimized stochastic circuit with an error either much less or larger than the error bound. For the former case, it means that we can further relax the accuracy constraint for the dynamic approximation method to obtain an SC core with a lower cost. For the latter case, it means that the stochastic circuit does not satisfy the accuracy requirement, and we need to tighten the accuracy constraint for the dynamic approximation method. In the following, we call this method the separate optimization method

To address the issue of the separate method, we propose a method for joint optimization of the randomizer and the SC core as shown in Algorithm 1. Its basic idea is to dynamically update the accuracy constraint for the dynamic approximation method by taking the influence of the randomizer configuration method into consideration. A proper accuracy constraint will lead to a properly optimized SC core and finally a low-cost stochastic circuit.

For a target function $5(G)$, based on the error bound for the entire stochastic circuit, $_{\text{bound}}$, we first initialize the error bound for the dynamic approximation method, which optimizes the SC core, as $_{\text{core-bound}} = C \cdot _{\text{bound}}$, where C is a parameter. In general, to do optimization in a large design space and avoid missing the optimal design, we set C 1 so that the error bound for the SC core optimization is larger than that for the entire stochastic circuit. Then, we optimize the stochastic circuit in a loop, which consists of 3 steps. First, in Line 5, we consider many different pairs of n and $<$, and apply the method in [11] to obtain the corresponding feature vectors. Note that a pair of n and $<$ with a smaller sum is more likely to lead to a low-cost SC core [14]. For these feature vectors, we choose the one as the input feature vector with its approximation error $_{\text{input}}$ less than the error bound

$_{\text{core-bound}}$ and having the minimum sum of 3 and $<$. Then, in Line 6, based on the input feature vector, we apply the dynamic approximation method to obtain an optimized SC core under the error bound $_{\text{core-bound}}$. Finally, in Line 7, based on the obtained SC core, we apply the low-cost architecture and the con guration method to optimize the randomizer and obtain the error $_{\text{circuit}}$ for the optimized stochastic circuit. If $_{\text{circuit}}$ is larger than $_{\text{bound}}$, we start the next iteration by setting $_{\text{core-bound}}$ as $_{\text{input}}$ in Line 8. This will ensure an improvement in accuracy with a new and more accurate input feature vector obtained. Otherwise, Line 9 terminates the loop, and Line 10 outputs the optimized stochastic circuit.

Algorithm 1: Joint optimization method.

1 input: Target function $5^1(G^9$ and error bound $_{\text{bound}}$;
2 output: Optimized stochastic circuit;
3 $_{\text{core-bound}} \leftarrow _{\text{bound}}$;
4 while true do
5 Apply the method in [11] to obtain an input feature vector $+$ with the error $_{\text{input}} \ddot{Y} _{\text{core-bound}}$;
6 Based on the input feature vector $+$, apply the dynamic approximation method to obtain an optimized SC core under the error bound $_{\text{core-bound}}$;
7 Based on the obtained SC core, apply the low-cost architecture and the con guration method to optimize the randomizer and obtain the error $_{\text{circuit}}$;
8 if $_{\text{circuit}} \mathrel{i} _{\text{bound}}$ then $_{\text{core-bound}} \leftarrow _{\text{input}}$;
9 else break;
10 return Optimized stochastic circuit;

Note that we also slightly modify the randomizer con guration method for acceleration. The previous randomizer con guration method targets at minimizing the error $_{\text{circuit}}$ through an iterative improvement loop. We observe that we only require the optimized stochastic circuit to have its error $_{\text{circuit}} \leq _{\text{bound}}$. Thus, when applying the con guration method, we terminate its loop early once it nds a con guration with $_{\text{circuit}} \leq _{\text{bound}}$.

Compared to the separate method, the joint method can obtain a stochastic circuit that always satis es the accuracy requirement and is more likely to have a lower hardware cost.

4 EXPERIMENTAL RESULTS

In this section, we show the experimental results.

4.1 Experimental setup

In this work, we consider linear feedback shift registers (LFSRs) as the RNSs. We choose 6 functions as the test cases, which are listed in Table 1 together with their IDs. We compare the proposed joint method with the conventional and the separate optimization methods. For the joint optimization method shown in Algorithm 1, we set $1 = 3$ and consider all possible pairs of positive 3 and positive $<$ with their sums larger than 2 and less than 8. For a fair comparison, for the conventional optimization method, we consider the same pairs of 3 and $<$ and optimize the randomizer by randomly con guring the feedback polynomials and the seeds of LFSRs for a half hour. For the separate optimization method, we directly apply the error bound for the stochastic circuit as the accuracy constraint for the SC core optimization and obtain the input feature vector in the same way as the joint optimization method. Note that these 3 methods need to measure the hardware costs of some possible SC cores in the SC core optimization step. In this work, we apply area-delay product (ADP) as the hardware cost measurement, and obtain the ADP by applying ABC [15] based on the MCNC standard cell library [16]. For simplicity, in the following, we denote the conventional, the separate, and the joint optimization methods as *Conventional*, *Separate*, and *Joint*, respectively. We test them with the bit-width as 8, and correspondingly, the length of an SBS is 256.

Table 1: The target functions.

Function	$\sin(G)$	$\cos(G)$	$\tanh(G)$	$4^{\,G}$	$\log(1 , G)$	$\frac{1}{1, 4^{\,G}}$
ID	1	2	3	4	5	6

4.2 Accuracy comparison

We rst compare the accuracy of the 3 methods. We apply root mean square error (RMSE) as the accuracy measure, and compute it over 1000 di erent inputs. We set the error bound $_{\text{bound}}$ for the stochastic circuit as 0.02. We apply the 3 methods to optimize the stochastic circuits. The RMSEs of the optimized stochastic circuits to implement di erent functions are shown in Table 2.

Table 2: RMSE comparison for 3 methods.

function	1	2	3	4	5	6	Average
Conventional	0.007	0.012	0.006	0.009	0.012	0.003	0.008
Separate	0.006	0.005	0.007	0.005	0.007	0.004	0.006
Joint	0.011	0.014	0.013	0.019	0.019	0.007	0.014

As shown in Table 2, the RMSEs of the optimized stochastic circuits for the 3 methods are all less than the error bound. Thus, the optimized stochastic circuits satisfy the accuracy requirement. Note that the RMSEs of *Joint* are higher than those of the other 2 methods. This is because we terminate the randomizer con guration process early once nding a con guration satisfying the accuracy requirement as introduced in Section 3.3. Actually, the accuracy of *Joint* can be further improved. However, this will lead to a longer runtime.

4.3 Hardware cost comparison

For the hardware cost comparison, we rst compare the 3 methods in terms of area. We synthesize the optimized stochastic circuits by Synopsys Design Compiler [17] and