Normalized Stability: A Cross-Level Design Metric for Early Termination in Stochastic Computing

Di Wu
University of Wisconsin-Madison
Madison, WI, USA
di.wu@ece.wisc.edu

Ruokai Yin
University of Wisconsin-Madison
Madison, WI, USA
ryin25@wisc.edu

Joshua San Miguel
University of Wisconsin-Madison
Madison, WI, USA
jsanmiguel@wisc.edu

ABSTRACT

Stochastic computing is a statistical computing scheme that represents data as serial bit streams to greatly reduce hardware complexity. The key trade-off is that processing more bits in the streams yields higher computation accuracy at the cost of more latency and energy consumption. To maximize efficiency, it is desirable to account for the error tolerance of applications and terminate stochastic computations early when the result is acceptably accurate. Currently, the stochastic computing community lacks a standard means of measuring a circuit's potential for *early termination* and predicting at what cycle it would be safe to terminate. To fill this gap, we propose *normalized stability*, a metric that measures how fast a bit stream converges under a given accuracy budget. Our unit-level experiments show that normalized stability accurately reflects and contrasts the early-termination capabilities of varying stochastic computing units. Furthermore, our application-level experiments on low-density parity-check decoding, machine learning and image processing show that normalized stability can reduce the design space and predict the timing to terminate early.

1 INTRODUCTION

Stochastic computing (SC) [7] has regained research interest in error-tolerant applications, like low-density parity-check decoding [17, 19], machine learning [12, 16] and image processing [1, 11], as it achieves high energy efficiency by trading off latency for computing complexity. SC is a statistical computing paradigm over serially streaming (unary) bits as SC data, with the latency identical to the bit stream length. Given the ratio of ones in the bit stream as p_1, unipolar SC data is unsigned with the value $V_{unipolar} = p_1$ and data range $[0, 1]$. To generate an SC bit stream, the N-bit binary source value is compared to an N-bit random number from a random number generator (RNG) at each cycle [7]. If the random number is smaller, a bit one is generated at this cycle; otherwise, a bit zero is generated. Statistically, the ratio of ones in the resultant bit stream is proportional to the binary value. Examples to illustrate unipolar SC data representation are shown in Fig. 1a. All three bit streams have in total 16 bits, among which A, B, and C have 12, 8, and 8 bit ones, respectively, leading to the unipolar values of 0.75, 0.50, and 0.50. Bit ones in both A and B are almost uniformly distributed, abiding by the rule that each bit has an equal probability to be one [7]. Data B has the same value as C due to the identical one counts, though the distribution of ones varies.

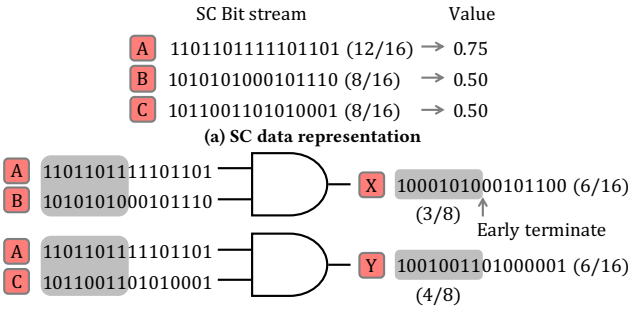

Figure 1: An example of SC early termination

The Benefit of Early Termination. With bit streams as SC data, SC operations can be implemented with extremely simple logic. As in Fig. 1b, a unipolar SC multiplication can be implemented with an AND gate [7]. Given input bit streams A and B, the same as in Fig. 1a, the final value of the output bit stream X is precisely 6/16 as expected. In this example, both input and output bit streams are of length 16, and the computation takes 16 cycles to fully complete, corresponding to 4-bit binary data. In general, to achieve the same level of resolution as in N-bit binary computing, SC requires 2^N cycles, introducing exponential latency overhead. To mitigate this overhead, *early termination is a desired property for SC operations, with which the computing latency and energy consumption are reduced while the accuracy is maintained at an acceptable level.* Back to the multiplication of A and B in Fig. 1b, by further examining the cycle-level value of bit stream X, it is observed that the first 8 cycles of X give an identical value to the final result with 16 cycles, and early terminating the calculation at cycle 8 has no influence to the result yet reduces the latency and energy consumption by 50%. On the other hand, also in Fig. 1b, bit stream Y is another accurate product of unipolar data A and C, but now early termination cannot be enabled without incurring accuracy loss. If some errors can be tolerated, early termination might still be applicable to Y. Given real-world SC applications that are error-tolerant (i.e., robust to minor errors), early termination can greatly benefit both latency and energy efficiency [3, 19, 20].

A Metric for Early Termination. Enabling early termination for an arbitrary SC system is a challenging task. While some prior work [18] has proposed hardware mechanisms for terminating early, they are specific to multiplication and do not have broad applicability to other SC circuits. Furthermore, prior metrics do

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431549

not sufficiently capture the potential for early termination in SC systems. The most relevant method, progressive precision [3], can inform whether or not early termination may be applicable for an SC operation but provides no indication of when to actually terminate. We propose a new metric, *normalized stability*, that measures *how long a bit stream (for a static data value) has been stable within a pre-defined accuracy budget, normalized to the maximum achievable stable duration.*Our metric offers two key benefits for designing SC systems with early termination. First, we can derive the ratio of output-to-input normalized stability of an SC unit, which is an intrinsic property of the unit that we coin as *flux normalized stability*. This informs how well the SC unit is able to propagate the stability of the input bit streams to the resulting bit stream and thus informs how well the unit supports early termination, shrinking the design space of SC systems. Second, we can profile the normalized stability of an application with representative training data and use our metric to predict when to early-terminate the application in practice while maintaining acceptable accuracy.

The contributions in this work are as follows:

- We introduce the metrics *normalized stability* and *flux normalized stability* and present their derivations.
- We simulate a wide range of SC units with varying implementation schemes to show how flux normalized stability accurately measures the SC units' intrinsic ability to support early termination.
- We implement popular SC applications to demonstrate how profiled normalized stability can accurately predict when it is acceptable to early-terminate given an accuracy budget.

2 BACKGROUND

This section reviews two existing approaches for SC early termination, including one hardware and one metric approach. Then different implementation schemes of SC units are presented, including combinational logic, counter based and shift register based sequential logic (the latter two are FSM based approaches).

2.1 Early Termination

Hardware Approach. A prior hardware mechanism for early termination [18] is shown in Fig. 2. One multiplicand A, valued 12/16, is generated as a bit stream that needs to be as uniform as possible, while the multiplicand B, valued 8/16, is constructed to be deterministic with ones ahead of zeros, equivalent to the temporal multiplier in [20]. With such construction, if the computation moves to the all-zero part, it can be skipped, by when six ones are accumulated in the accumulator (ACC), producing a binary value of 6/16. This method is able to eliminate ineffectual computations when bits in multiplicand B are zero. However, there are three fundamental limitations in this technique. First, for SC operations that require information from all ones (e.g., the non-scaled addition of two bit streams [20]) , this method is not applicable, as some ones in A are never processed. Second, when multiple multiplications are performed in parallel, the latency is bottlenecked by the largest multiplicand B. Though early termination saves energy for each individual multiplication, the total energy saving is not comparable when early terminating all computations simultaneously. Last, this approach requires unary-binary inter-conversion for every multiplication operation, both when formatting the input

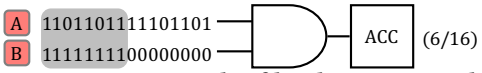

Figure 2: An example of hardware approach

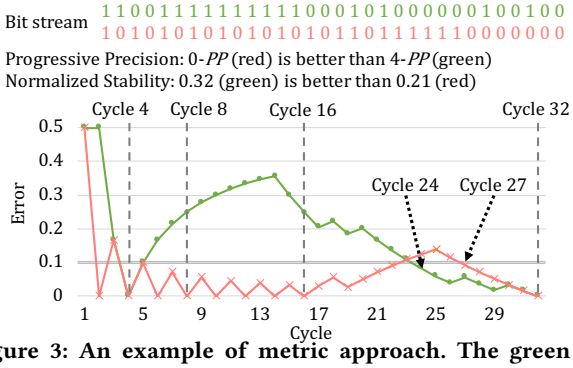

Figure 3: An example of metric approach. The green bit stream is better for early termination than the red one.

bit streams and accumulating the output bit stream, which inhibits the multiplication from being fully streaming, i.e., no bit stream stalling. This imposes extra latency and hardware, nullifying the latency and energy savings from early termination. Due to these limitations, this hardware approach cannot be generalized to other fully streaming SC systems.

Metric Approach. Progressive precision (*PP*) [3, 4] is a metric for measuring accuracy at different lengths of a given bit stream. With PP, accuracy is measured in terms of binary resolution; i.e., SC bit streams are represented as binary values. For a length-N stochastic bit stream, it is defined as k-*PP* if the bit error of its initial subsequence of length 2^i is at most k for all i. Here, the bit error is defined as the absolute value of the difference between the expected one count and the actual one count in the bit stream. Thus, *PP* reflects the max bit error it can reach throughout all sub-sequences with the length of a power of 2. The benefit of *PP* is twofold. First, it can be informative for early termination. Lower *PP* implies more accurate intermediate results, and the calculation can be stopped early [3]. Second, the complexity of measuring *PP* scales linearly with the application size, as it is designed for individual bit streams.

Fig. 3 shows two curves representing the cycle-level errors of 4-*PP* and 0-*PP* bit streams of length 32 and unipolar value 0.5. Cycles with a value of the power of 2, are labeled with vertical dash lines, including 4, 8, 16, and 32. The maximum error count at those cycles denotes the k value for *PP*. As such, *PP* is able to measure accuracy. Given the tolerable error as 10% in the gray horizontal line, at cycle 24 and 27, the 4-*PP* and 0-*PP* bit streams can be early terminated safely, i.e., the error will never exceed the tolerance level later. From Fig. 3, two defects of *PP* can be observed. First, it lacks the capability to identify when to perform early termination. For the 4-*PP* bit stream, the maximum error count comes from cycle 16. However, early termination at cycle 24 is not indicated by $k = 4$. Similarly, cycle 27 to early terminate the 0-*PP* bit stream is not implied by $k = 0$. As a result, *PP* fails to measure latency. Second, *PP* is incapable to tell the quality of a bit stream in terms of early termination. Though the 4-*PP* bit stream is technically worse than 0-*PP*, i.e., having a larger maximum error count, the 4-*PP* bit stream can be terminated earlier than the 0-*PP* bit stream, given the error tolerance. Furthermore, based on such incapability, we consider that *PP* can not propagate in SC applications, though it is quantitative and

scales well at the unit and application level. In this work, normalized stability is proposed to circumvent the above defects by providing a numerical measurement of early termination, while maintaining the scalability as in *PP*. For example, the 4-*PP* bit stream has 0.32 normalized stability, higher than 0.21 for the 0-*PP* one, and can be classified as "more stable", i.e., the ability for earlier termination.

2.2 Stochastic Computing Units

Stochastic computing supports varieties of general mathematical operations, including (though not limited to) addition [7, 8, 20], multiplication [7, 18, 20], division [6, 7, 21], square root [7, 21], exponentiation [10, 15], as well as some deep learning activation functions, like hyper tangent [10, 15] and ReLU [9, 23]. Their SC implementation schemes can be categorized into combinational logic, counter based and shift register based sequential logic, as listed in Table 1. Combinational logic exclusively measures the influence of the *current* input to the output. Then counter based sequential logic refers to using counting logic to record the *entire* history of bit streams, while shift register based sequential logic applies shift register to record the *recent* history of bit streams. Both counter based and shift register based implementations are popular for nonlinear SC operation design.

Table 1: Implementation schemes for SC operations

Op.	Combinational	Counter	Shift Register
add	[7]	[20]	
mul	[7]	[20]	
div		[7]	[21]
sqrt		[7]	[21]
exp	[15]	[10]	
tanh	[15]	[10]	
ReLU		[23]	[9]

In this paper, we focus on the evaluation of SC units that take bit streams as input and output in a fully streaming manner [21], not those using binary input/output as in [8, 18], as fully streaming SC units can take more advantage of early termination. Examples of three implementation schemes are shown in Fig. 4. In Fig. 4a, combinational *tanh* is the Maclaurin series expansion of $\tanh(x)$ [15], and the numbers represent the coefficients of different terms. This unit involves only AND gates and NAND gates, as well as D-Flip-Flops (DFFs), which act as the isolator for multiplication accuracy [5]. In Fig. 4b, counter based *tanh* is an FSM [10], where the output is directly related to the entire history of the input bit, x. The complete history is recorded in the N-bit saturating counter, denoted as CNT, and initialized with 2^{N-1}. If input bit x is one, CNT will increase by one; otherwise, it will decrease by one. The output of CNT is compared with 2^{N-1}, and output a bit one if CNT value is larger. In Fig. 4c, *ReLU* function, $\max(0, x)$, is presented based on N-bit shift register (SR). The most recent N-bit input x is stored sequentially in the shift register, and those bits are summed up in the parallel counter (PC). The sum is compared with $\frac{N}{2}$ to select the output. If the sum is less than $\frac{N}{2}$, the bit stream is supposed to be negative and a bit one is output for compensation; otherwise, output the current input bit x. Similar to counter based logic, SR based designs also leverage FSM. As those SC implementation schemes leverage different input bits, they exhibit varying accuracy and speed of convergence, i.e., the proposed normalized stability, and will all be evaluated in this work.

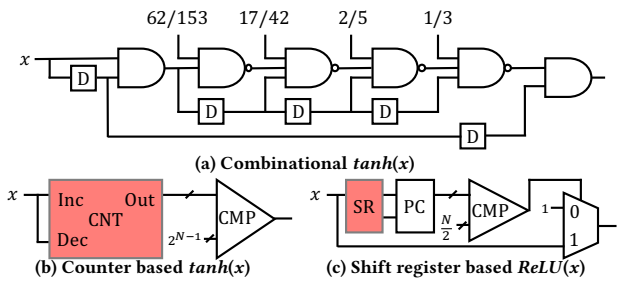

Figure 4: Examples of SC implementation schemes

3 NORMALIZED STABILITY

Normalized stability measures how long a bit stream (for a static value) has been stable within a pre-defined accuracy budget, normalized to the best case. Its derivation is separated into three phases, including 1) retrieving the actual stability of the bit stream, 2) constructing a bit stream of the best stability and 3) normalizing the actual stability to the best stability. In the following, we assume unipolar SC data representation for clarity.

3.1 Actual Stability of Bit Stream

In the first phase, provided a target accuracy budget, what is the actual stability of a bit stream? To answer this, a criterion is needed to judge whether a bit stream stabilizes. In this work, we set the criterion to be that the value of a bit stream always fluctuates under the target absolute error. Then the ratio of the cycle count that the bit stream has been in the stable state is defined as the stability. Note that in this work we choose the absolute error instead of the relative error due to two reasons. First, the absolute error directly corresponds to the bit error in *PP* [3]. Second, using the relative error is a stronger constraint on accuracy and decreases the propagability of normalized stability in the application.

$$Stability_T = \frac{N - \max\{n | \Delta P_n > T\}}{N} \tag{1}$$

The stability of a length-N bit stream with a user-defined error threshold T is formulated in Eq. (1), where P_n represents the actual value of the partial bit stream based on the first n bits ($n \leq N$), with P_E denoting the expected value, and ΔP_n is the absolute error $|P_n - P_E|$ of the first n bits in the stream. The max operation finds the bit after which the absolute error of the bit stream is always bounded to the target threshold T. The selection of T empirically depends on user's accuracy requirement for the error-tolerant application [13], and T values of 5% and 10% for unipoalr SC are evaluated in this paper. Stability varies within $[0, 1)$, with a higher value indicating an earlier convergence of the bit stream and better capability towards early termination. Note that the accuracy used here is of cycle-level granularity, differing from power of 2 in *PP*.

3.2 Best Stability of Bit Stream

At the second phase, we need to construct a length-N bit stream with the best stability for normalization under the user-defined error threshold T. The final value of the best-stability bit stream also fluctuates within the error threshold, implying a lower and upper bound of the final value, denoted as $P_{low} = \max(0, P_E - T)$ and $P_{high} = \min(P_E + T, 1)$. For an arbitrary bit stream, traversing all different orders of bits can locate the bit stream with the best stability, however, at the cost of exponentially increasing complexity

with the bit stream length N. Thus, we introduce an approximate approach to find the bit stream of best stability in this work, with bit stream A in Fig. 5 as an example, where $T = 5\%$ and $N = 32$. For values $p \in [P_{low}, P_{high}]$ with the desired precision, it is possible to find the shortest bit stream that precisely represents p. The shortest length for each precise p value, l_p, is formulated based on the greatest common divisor in Eq. (2), where $L = 2^{\lceil \log_2 N \rceil}$. For bit stream A in Fig. 5, $l_p = 2$ when $p = 0.5$.

$$l_p = \frac{L}{\gcd(\lceil p * L \rceil, L)} \tag{2}$$

Though each value p precisely corresponds to a length-l_p bit stream, improperly manipulating such bit stream might not lead to accurate convergence at a maximized speed for the entire bit stream. Our solution is to repeat this shortest bit stream multiple times until the bit stream converges. Assuming R repetitions, this implies that starting from the $(R \cdot l_p + 1)$-th bit, the output accuracy is always bounded in $[P_{low}, P_{high}]$, no matter this bit is zero or one. Now the problem translates to determine the minimal repetition count, R, formulated as in Eq. (3). The two inequalities denote appending an extra bit ($\mathbf{B}$), either zero or one, to R repeated shortest bit streams.

$$P_{low} \le \frac{p \cdot R \cdot l_p + \mathbf{B}}{R \cdot l_p + 1} \le P_{high} \tag{3}$$

The solution is given by Eq. (7), with the derivation shown in Eq. (4, 5, 6). This solution satisfies that with R repeated length-l_p bit streams at the front, the bit stream is always stable with more repeated length-l_p bit streams. And among all R values for all $p \in [P_{low}, P_{high}]$, $R = \underset{R}{\text{argmin}}(R \cdot l_p)$ is the final value we want. For bit stream A in Fig. 5, $R = 5$ when $l_p = 2$, and $R \cdot l_p = 10$ is the shortest length for stabilization.

$$\begin{aligned} R_{low} \cdot l_p \cdot (p - P_{low}) &\ge P_{low} - \mathbf{B} \\ R_{high} \cdot l_p \cdot (P_{high} - p) &\ge \mathbf{B} - P_{high} \end{aligned} \tag{4}$$

$$R_{low,\mathbf{B}} = \begin{cases} \dfrac{P_{low} - \mathbf{B}}{l_p \cdot (p - P_{low})}, & \text{if } p \ne P_{low} \\ 0, & \text{if } p = P_{low} \text{ and } P_{low} \le \mathbf{B} \\ L, & \text{otherwise} \end{cases} \tag{5}$$

$$R_{high,\mathbf{B}} = \begin{cases} \dfrac{\mathbf{B} - P_{high}}{l_p \cdot (P_{high} - p)}, & \text{if } P_{high} \ne p \\ 0, & \text{if } P_{high} = p \text{ and } \mathbf{B} \le P_{high} \\ L, & \text{otherwise} \end{cases}$$

$$\begin{aligned} R_{\mathbf{B}=0} &= \max(R_{low,\mathbf{B}=0}, R_{high,\mathbf{B}=0}) \\ R_{\mathbf{B}=1} &= \max(R_{low,\mathbf{B}=1}, R_{high,\mathbf{B}=1}) \end{aligned} \tag{6}$$

$$R = \lceil \min(R_{\mathbf{B}=0}, R_{\mathbf{B}=1}) \rceil \tag{7}$$

3.3 Normalization of Stability

The last step is to normalize the actual stability of a bit stream with the best stability, shown in Eq. (8). The best stability is calculated based on R repeated length-l_p bit streams. The max operation is to cover the corner case when the bit stream value is close to either 1 or 0, indicating that only one bit is already stable.

$$\textit{Normalized Stability}_T = \frac{\textit{Stability}_T}{1 - \max(R \cdot l_p, 1)/N} \tag{8}$$

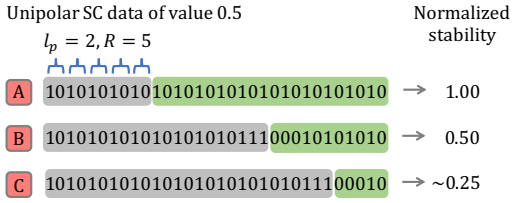

Figure 5: An example of normalized stability with $T = 5\%$

An example of normalized stability for unipolar SC data with $T = 5\%$, $N = 32$, $l_p = 2$ and $R = 5$ is depicted as in Fig. 5. All three length-32 SC data, A, B, and C, have the same value of 0.5. For each of them, bits in gray blocks are in the non-stable state, after which bits in green blocks are in the stable state and the values are constantly within the error budget. Therefore, the computation can early terminate anytime during the stable state. SC data A has a normalized stability of 1.00, as explained in Section 3.2. It is one of the bit streams that have the most bits in stable states among all possible bit streams. Then SC data B and C have the normalized stability of 0.50 and around 0.25, which represent the ratio of bit counts in the stable state of B and C and that of A. Therefore, given a constant bit stream length, more bits in the green block means shorter computing time, and the early termination can be activated earlier linearly to the normalized stability. Normalized stability explicitly involves time information in the definition, and acts as an attribute for a single bit stream. To extend normalized stability to SC units and applications with multiple bit streams, the average of multiple individual normalized stability values can represent the normalized stability of multiple bit streams.

4 UNIT-LEVEL NORMALIZED STABILITY

In this section, we analytically present, for various SC units in Table 1 in Section 2.2, how the *flux normalized stability* varies, given different input normalized stability and error thresholds.

4.1 Experimental Setup

We set the normalized stability of input bit streams approximately to 0.25, 0.50, and 0.75 with T values of 5% and 10%, instead of examining all possibilities. The input covers all 8-bit binary decimals in the legal data range, and we ensure the final result will always converge to the target budget by forcing near-zero input correlation [2]. All simulations, leveraging hundreds of RNGs, are publicly available on our open-source simulator, UnarySim [20, 22], which supports stochastic computing and integrates stability metrics.

4.2 Flux Normalized Stability

Flux normalized stability is defined as the quotient of the output and input normalized stability. This parameter can accurately reflect the variance of normalized stability before and after an SC unit, i.e., the capability of the SC unit to maintain the normalized stability. Therefore, adding an SC unit with high flux normalized stability, the timing for early termination will not vary significantly.

The experimental result is presented in Table. 2, with each block containing three flux normalized stability (saturated to 1.00) *w.r.t.* different input normalized stability, 0.25, 0.5 and 0.75, respectively, for a specific T. Values within the same block are consistently close to each other, with the largest difference, 0.17, from counter based *exp* with 10% T. On the other hand, values from different blocks vary. Thus, we have four key takeaways. First, the SC operation type has

varying flux normalized stability, regardless of the implementation scheme. Second, the implementation scheme influences the flux normalized stability for a target SC operation. Third, a certain SC unit has almost constant flux normalized stability, given varying input normalized stability and a specific T. Last, larger T slightly increases the flux normalized stability. Those takeaways conclude that the attenuation of normalized stability is almost linear for a certain SC unit, given arbitrary input normalized stability and specific T. *Therefore, flux normalized stability reflects the quality of an SC unit to maintain normalized stability, with a higher value benefiting early termination more.* Moreover, it is intuitive to consider that, at the application level, given multiple candidate SC units, selecting the one with the best flux normalized stability will lead to the best normalized stability for the application output. In [16], the authors extensively profile multiple SC units for neural networks, their final choice of adders matches what flux normalized stability implies, i.e., counter-based adders are better than combinational adders. Therefore, flux normalized stability reduces the design space complexity from exponential as in [16] to linear.

Table 2: Unit-level flux normalized stability under varying input normalized stability (0.25, 0.5, 0.75) and T (5%, 10%).

Op.	T	Comb.	Counter	Shift Register
add	5%	0.94, 0.92, 0.92	0.98, 0.98, 0.99	
	10%	0.97, 0.97, 0.97	0.98, 0.99, 0.99	
mul	5%	0.87, 0.89, 0.91	1.00, 1.00, 1.00	
	10%	0.86, 0.91, 0.94	1.00, 1.00, 1.00	
div	5%		0.23, 0.23, 0.20	0.68, 0.71, 0.71
	10%		0.58, 0.63, 0.59	0.61, 0.73, 0.77
sqrt	5%		0.68, 0.63, 0.57	0.39, 0.40, 0.39
	10%		0.85, 0.82, 0.81	0.49, 0.49, 0.50
exp	5%	0.93, 0.89, 0.88	0.70, 0.82, 0.79	
	10%	0.98, 0.99, 0.98	0.79, 0.91, 0.96	
tanh	5%	0.88, 0.90, 0.85	0.27, 0.27, 0.27	
	10%	0.93, 0.95, 0.95	0.29, 0.32, 0.32	
ReLU	5%		1.00, 1.00, 1.00	0.61, 0.61, 0.61
	10%		1.00, 1.00, 1.00	0.82, 0.88, 0.90

5 APPLICATION-LEVEL STUDY

Based on the unit-level discussion above, to better understand the application-level early termination, we further test popular SC applications, including low-density parity-check (LDPC) decoding [17, 19], machine learning [12, 16] and image processing [1, 11]. For LDPC, we examine the propagation of normalized stability. For machine learning, we additionally present how profiled normalized stability predicts early termination. For image processing, we extensively demonstrate that higher flux normalized stability leads to earlier termination. For LDPC and machine learning, the normalized stability equals the stability, as the output is one-hot encoded with a best-stability of 1, while image processing outputs pixel intensity, leading to unequal normalized stability and stability.

5.1 Low-Density Parity-Check Decoding

LDPC code is an error correction code to recover the correct message after a message is generated but transmitted through a noisy channel [17, 19]. The message is encoded with a size-(k, n) sparse matrix G, and decoded with a size-(m, n) sparse matrix H, where $G \cdot H^{\mathrm{T}} = 0$. With the length-$k$ binary source message s encoded

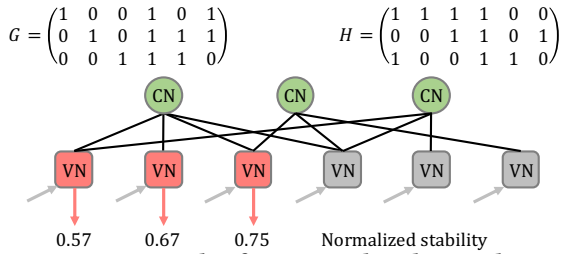

Figure 6: An example of SC LDPC decoding with $T = 5\%$

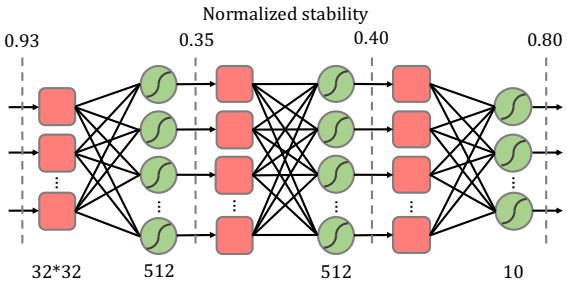

Figure 7: An example of SC MLP with $T = 10\%$

as $c = s \cdot G$, the code c is transmitted via a noisy channel and becomes $\hat{c} = c + e$, where e is the noise. LDPC decoding can recover source message s from the noisy code $\hat{c}$, leveraging the fact that $c \cdot H^{\mathrm{T}} = s \cdot G \cdot H^{\mathrm{T}} = 0$. Viewing each column of H as variable nodes (VNs) and each row of H as check nodes (CNs) in a Tanner graph, the decoding process is an iterative process with a *feedback* dataflow between n VNs and m CNs in the Tanner graph.

An example of the 8-bit unipolar SC LDPC decoding process is presented in Fig. 6. Gray inputs to the rectangle VNs denote the noisy messages, while edges between the rectangle VNs and circle CNs represent the messages generated by the decoding algorithm in [17, 19]. Note that each VN/CN hardware will differ from other VNs/CNs if node degrees differ. Those gray VNs correspond to the redundant bits for parity check, while the red VNs are responsible to output the recovered source code. The correspondent valid output normalized stability values are labeled. Due to different hardware implementations of VNs and CNs, the output bits have distinguished normalized stability with $T = 5\%$. And the value 0.57, 0.67, and 0.75 indicate that 112, 86, and 64 cycles out of 256 cycles are required for each bit to be successfully decoded. Therefore, 112 cycles are required in total, and the latency reduction, 57%, directly corresponds to the lowest output normalized stability.

5.2 Machine Learning

Machine learning [12, 16] is a popular SC application. In this work, we examine a 3-layer *feed-forward* Multi-Layer Perceptron (MLP) on MNIST dataset as in [20], and obtain around 94.7% final accuracy for bipolar SC implementation [7]. Each layer of MLP, with size labelled at the bottom, performs matrix multiplication followed by nonlinear operations: *ReLU* [14] in the first two layers, and *softmax* followed by a one-hot encoded *max* in the third layer.

Propagation of normalized stability for one test sample is reported at the top of Fig. 7 using $T = 10\%$. Due to performing non-scaled addition [20] in as short as 256 cycles, the normalized stability degrades significantly at the first layer. However, at the output, the normalized stability recovers, as MLP performs a classification task, where the output *relative magnitude* matters. Starting from the 52^{nd}

cycle, the classification result for this test sample is correct, leading to an almost 80% reduction in execution time.

Prediction of early termination is an important capability provided by our normalized stability metric. We profile 1000 random training samples with our MLP model and obtain an average output normalized stability of 0.72 (with $T = 5\%$). This implies that our metric predicts that the output accuracy will already be within 5% of the 256-cycle accuracy by cycle $(1 - 0.72) * 256 + 1 = 72$. We validate this by terminating the inference early on the full 10000 test dataset at cycle 72. We find that this yields 91.1% final accuracy, which is indeed within 5% of the 256-cycle accuracy, 94.7%.

5.3 Image Processing

Image processing [1, 11] is another active area for SC; we explore *feed-forward* SC edge detection as a case study in this work. SC edge detection in [11] applies a convolutional kernel G given in Eq. (9), including two sub-kernels. The input image is first convolved with each sub-kernel, and then the absolute values of two results are added to retrieve the edge. The correspondent 10-bit unipolar SC implementation is shown in Fig. 8, where every four adjacent pixels produce one output pixel.

$$G = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} + \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} \tag{9}$$

Propagation of normalized stability for the same image with different adder implementations and T values is shown at the top of Fig. 8. We observe normalized stability in the application degrades gradually, similar to the situation in MLP. Note that the output error and normalized stability here are the mean of the absolute pixel intensity errors and the mean of all individual normalized stability.

Influence of flux normalized stability aids in design space exploration for application-level early termination. We demonstrate this in our edge detection application, varying the choice of adder and T. There are two approaches to increase the flux normalized stability according to Table 2. As our earlier experiments show, counter-based adders yield higher flux normalized stability than combinational ones (also implied by adder profiling in [16]); thus we expect counter-based adders to be more performant in edge detection. We validate this in Fig. 8, showing that early termination (with $T = 5\%$) using counter-based adders saves $(0.84 - 0.81)/(1 - 0.81) = 15.8\%$ more cycles of latency than using combinational adders. Furthermore, applying a larger T, if acceptable to the user, leads to higher flux normalized stability. When early termination is enabled, increasing T from 5% to 10% for the counter-based implementation can save $(0.91 - 0.81) * 1024 = 102$ additional cycles; i.e., tolerating up to 10% application error saves $(0.91 - 0.81)/(1 - 0.81) = 52.6\%$ more cycles than tolerating only 5% error.

Prediction of early termination in edge detection can be performed using our normalized stability metric, similar to our previous MLP example. We profile 20 random training images and observe output normalized stability of 0.82 with $T = 5\%$. This predicts that a reduction of $0.82 * 1024 = 840$ cycles is possible with early termination. We validate this on our edge detection application with another 20 random test images and find that indeed terminating 840 cycles early yields acceptable (i.e., within T) output error of 2.5%.

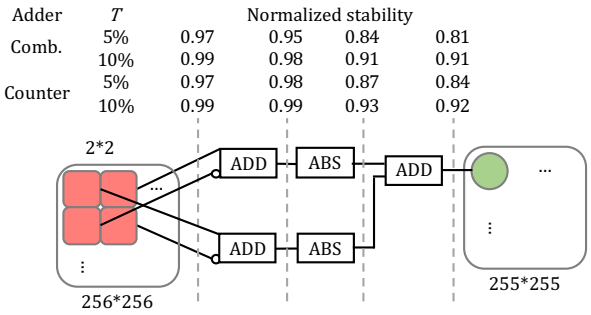

Figure 8: An example of SC edge detection

6 CONCLUSION

Though early termination is an important aspect of efficient stochastic computing, the SC community currently lacks sufficient methodologies for enabling support for early termination in SC systems. To address this, we introduce a new metric, normalized stability, to characterize how long a bit stream has been sufficiently accurate. Our experiments show that normalized stability is able to accurately reflect the potential for early termination for both arithmetic units and applications, demonstrating its efficacy for SC design space exploration and predicting when it is acceptable to terminate early.

REFERENCES

[1] A. Alaghi et al. 2013. Stochastic Circuits for Real-Time Image-Processing Applications. In *DAC*.
[2] A. Alaghi and J. P. Hayes. 2013. Exploiting Correlation in Stochastic Circuit Design. In *ICCD*.
[3] A. Alaghi and J. P. Hayes. 2014. Fast and Accurate Computation using Stochastic Circuits. In *DATE*.
[4] T. H. Chen et al. 2017. Achieving Progressive Precision in Stochastic Computing. In *GlobalSIP*.
[5] T. H. Chen and J. P. Hayes. 2014. Analyzing and Controlling Accuracy in Stochastic Circuits. In *ICCD*.
[6] T. H. Chen and J. P. Hayes. 2016. Design of Division Circuits for Stochastic Computing. In *ISVLSI*.
[7] B. R. Gaines. 1969. Stochastic Computing Systems. In *Advances in Information Systems Science*.
[8] K. Kim et al. 2015. Approximate De-Randomizer for Stochastic Circuits. In *ISOCC*.
[9] J. Li et al. 2017. Hardware-Driven Nonlinear Activation for Stochastic Computing based Deep Convolutional Neural Networks. In *IJCNN*.
[10] P. Li et al. 2012. The Synthesis of Linear Finite State Machine-based Stochastic Computational Elements. In *ASPDAC*.
[11] P. Li and D. J. Lilja. 2011. Using Stochastic Computing to Implement Digital Image Processing Algorithms. In *ICCD*.
[12] Y. Liu and K. K. Parhi. 2016. Computing RBF Kernel for SVM Classification Using Stochastic Logic. In *SiPS*.
[13] S. Mittal. 2016. A Survey of Techniques for Approximate Computing. *Comput. Surveys* (2016).
[14] V. Nair and G. E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*.
[15] K. Parhi and Y. Liu. 2017. Computing Arithmetic Functions Using Stochastic Logic by Series Expansion. *TETC* (2017).
[16] A. Ren et al. 2017. SC-DCNN: Highly-Scalable Deep Convolutional Neural Network Using Stochastic Computing. In *ASPLOS*.
[17] S. S. Tehrani et al. 2008. Fully Parallel Stochastic LDPC Decoders. *TSP* (2008).
[18] H. Sim and J. Lee. 2017. A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks. In *DAC*.
[19] D. Wu et al. 2016. Strategies for Reducing Decoding Cycles in Stochastic LDPC Decoders. *TCASII* (2016).
[20] D. Wu et al. 2020. uGEMM: Unary Computing Architecture for GEMM Applications. In *ISCA*.
[21] D. Wu and J. San Miguel. 2019. In-Stream Stochastic Division and Square Root via Correlation. In *DAC*.
[22] D. Wu and R. Yin. 2020. UnarySim. https://github.com/diwu1990/UnarySim
[23] J. Yu et al. 2017. Accurate and Efficient Stochastic Computing Hardware for Convolutional Neural Networks. In *ICCD*.