When Sorting Network Meets Parallel Bitstreams: A Fault-Tolerant Parallel Ternary Neural Network Accelerator based on Stochastic Computing

Yawen Zhang[1], Sheng Lin[2], Runsheng Wang[1*], Yanzhi Wang[2*], Yuan Wang[1*], Weikang Qian[3*], Ru Huang[1]
[1]*Institute of Microelectronics, Peking University, Beijing, P.R. China*
[2]*Department of Electrical and Computer Engineering, Northeastern University, Boston, USA*
[3]*UM-SJTU Joint Institute, Shanghai Jiao Tong University, Shanghai, P.R. China*
[*]Email: r.wang@pku.edu.cn, yanz.wang@northeastern.edu, wangyuan@pku.edu.cn, qianwk@sjtu.edu.cn

***Abstract*—Stochastic computing (SC) has been widely used in neural networks (NNs) due to its simple hardware cost and high fault tolerance. Conventionally, SC-based NN accelerators adopt a hybrid stochastic-binary format, using an accumulative parallel counter to convert bitstreams into a binary number. This method, however, sacrifices the fault tolerance and causes a high hardware cost. In order to fully exploit the superior fault tolerance of SC, taking a ternary neural network (TNN) as an example, we propose a parallel SC-based NN accelerator purely using bitstream computation. We apply a bitonic sorting network for simultaneously implementing the accumulation and activation function with parallel bitstreams. The proposed design not only has high fault tolerance, but also achieves at least $2.8\times$ energy efficiency improvement over the binary computing counterpart.**

***Index Terms*—Stochastic computing, ternary neural network, bitonic sort, parallel computing**

I. Introduction

The large-scale matrix multiplication operations of deep neural networks (NNs) lead to complex hardware implementations. However, computational and memory resource requirements are becoming more stringent, especially in mobile systems and Internet of Thing (IoT) devices. To overcome this difficulty, hardware architectures for many specific applications have been proposed, such as ternary neural networks (TNNs) [1], [2], where all synaptic weights and neuron activations are ternarized to 0, $+1$, or -1. TNNs can greatly reduce the storage and computational requirements, while the accuracy is only slightly reduced.

On the other hand, stochastic computing (SC) [3], [4], as a promising alternative to traditional binary computing, is a new computational paradigm using probability as a medium and is compatible with modern VLSI design and manufacturing technology [5]. SC has great potential for a dedicated NN accelerator design due to its advantages of strong fault tolerance and low hardware cost [6]–[9], which is very suitable for low-precision NNs with weak fault tolerance, like TNN.

However, traditional SC-based NN accelerators suffer from several problems. First, SC needs a long bitstream to maintain computational accuracy. This leads to a long computation latency since the traditional SC serially inputs the bitstream in the form of time sequences. Second, SC has random fluctuation, leading to inaccurate calculation results. Particularly, the multiplication on near-zero data is quite inaccurate with the bipolar format [6], which is contrary to the sparsity of NNs. Third, SC is not suitable for computation with multiple stages, which requires increasing bitstream lengths to maintain high accuracy. This problem becomes more serious for DNNs with many computation layers. To solve it, usually, the SC will convert the bitstream into a binary number, which is done by circuits such as an accumulative parallel counter (APC), and then convert it back to avoid multiple consecutive arithmetic calculations. However, this results in high binary-bitstream conversion cost [3]. It also largely weakens the fault-tolerance advantage of SC. In addition, although the multiplication is simple in SC, it is very difficult to implement the accumulation and activation function of NN in SC without loss of its accuracy. Usually, binary computing is used for their implementation.

In this work, we present a fault-tolerant parallel SC-based NN accelerator purely using bitstreams, which addresses the problems mentioned above. We redesign the multiplication, accumulation, and activation function of SC-based NN to realize the fully parallel bitstream calculation. This new parallel design avoids the long latency and the high conversion cost to binary numbers. Notably, an efficient and accurate accumulator is proposed using the bitonic sorting network to further reduce energy consumption. Using a TNN as the implementation target, we compare the proposed parallel SC-based NN accelerator with the existing SC-based and binary-based ones. We show that our design has a shorter latency than traditional SC-based design. Furthermore, it has higher energy efficiency and fault tolerance than both the binary and the traditional SC-based designs. Our contributions are as follows:

- A parallel SC-based NN accelerator architecture that purely uses bitstreams for calculation, eliminating the conversion cost between bitstreams and binary numbers, and maintaining the advantage of fault tolerance.
- A parallel SC design for accurate ternary multiplication.
- A parallel SC design for accurate accumulation and activation function using bitonic sorting network.
- A redesign of the convolution kernel structure in the TNN for a complete end-to-end parallel SC-based accelerator.

The rest of the paper is organized as follows. Section II presents the proposed design. Section III shows the experimental results. Section IV concludes the paper.

II. Parallel SC-based NN Accelerator

A. Overview of Different NN Architectures

In extreme situations, such as under the effects of cosmic rays or low supply voltages, soft error is often produced during calculations. As shown in Fig. 1, there are usually two types of soft errors in an NN accelerator, which are SRAM reading error, caused by SRAM read and write noises, and multiply-accumulate (MAC) calculation error, due to the static and dynamic variations in devices and circuits [5].

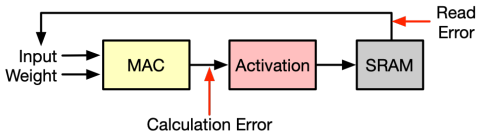

Fig. 1. Illustration of possible soft error occurrence in NN accelerator.

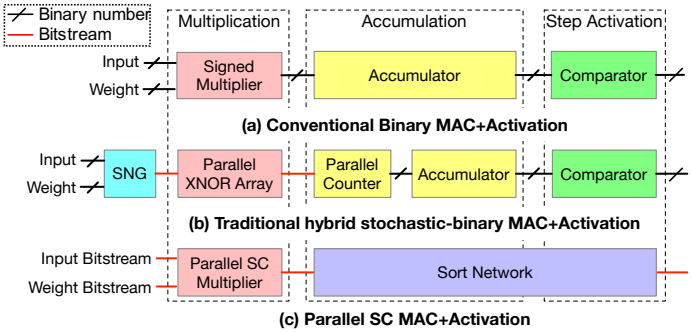

Fig. 2. Comparison of three NN accelerator architectures.

Due to the advantages of uniform weight coding (i.e., all bits have the same weight), SC is considered to be highly fault-tolerant. For the above two types of soft errors in NN accelerators, the SC-based design should have good fault tolerance in theory, but in fact not, as discussed below.

Fig. 2 shows the architectures of a binary, a traditional SC-based, and our proposed parallel SC-based NN accelerator. We analyze the fault tolerance of these three different architectures. It can be found that traditional SC uses APC to convert bitstreams into binary numbers for accumulation. For traditional SC and binary computing, the data read from SRAM is in binary format, which has low tolerance to noise. For example, 1 (01) changes to -1 (11) when an error occurs in the sign bit, which causes a value change of 2. In our proposed parallel SC design (see Fig. 2c), the data is store in the SRAM in the bitstream format, which has strong fault tolerance. Regarding the MAC calculation error, since the conventional SC and binary based designs apply the adder tree, the fault tolerance is poor due to the binary computing. However, the parallel SC design uses a bitonic sorting network to output a parallel bitstream (see Section II-D for details). Thus, it has strong fault tolerance.

In summary, due to the use of binary accumulation, traditional SC-based NN accelerator loses the fault tolerance advantage, which is often overlooked in previous hybrid stochastic-binary NN accelerators. In contrast, the proposed design using fully parallel bitstreams not only keeps the high fault tolerance of SC but also reduces the energy consumption. The details of our design will be presented in the following sections.

B. Bitstream Coding Method

In SC, the conventional binary arithmetic is transformed into the probabilistic computation. Stochastic bitstreams used to carry information are coded in two formats, bipolar and unipolar. For the unipolar format, the value is encoded as the probability of a 1 in the bitstream. Thus, only the values between 0 and 1 can be represented. For the bipolar format, the value is encoded as $2P-1$, where P is the probability of a 1 in the bitstream. Thus, both positive and negative values can be represented. The actual operation becomes a kind of probabilistic calculation in SC. The multiplication is implemented by an AND (resp. XNOR) gate in the unipolar (resp. bipolar) coding format. Since the weight and the input data in TNNs are all composed of -1, 0, and $+1$, bipolar coding is needed to cover the negative numbers.

TABLE I
VARIOUS TERNARY CODING METHOD.

Number	Bipolar SC		Binary computing
	Proposed parallel	Deterministic [10]	
-1	00	0000	11
0	10/01	1100/1010	00
$+1$	11	1111	01

TABLE II
MULTIPLICATION IN SC.

Multiplication	Parallel SC		Traditional SC	
	Input Parallel Bitstream	Output	Input Deter. Bitstream	Output
$0*0$	$01*10\ 01*01$ $10*10\ 10*01$	10	$1010\ \overline{\oplus}\ 1100$	1001
$0*1$	$01*11\ 10*11$	10	$1010\ \overline{\oplus}\ 1111$	1010
$0*(-1)$	$01*00\ 10*00$	10	$1010\ \overline{\oplus}\ 0000$	0101
$1*0$	$11*10\ 11*01$	10	$1111\ \overline{\oplus}\ 1100$	1010
$1*1$	$11*11$	11	$1111\ \overline{\oplus}\ 1111$	1111
$1*(-1)$	$11*00$	00	$1111\ \overline{\oplus}\ 0000$	0000
$(-1)*0$	$00*10\ 00*01$	10	$0000\ \overline{\oplus}\ 1100$	0101
$(-1)*1$	$00*11$	00	$0000\ \overline{\oplus}\ 1111$	0000
$(-1)*(-1)$	$00*00$	11	$0000\ \overline{\oplus}\ 0000$	1111

Traditional SC is subject to stochastic error. A recent work introduces the deterministic coding format, which makes the calculation results of SC accurate [10]. However, in order to produce accurate result for tenary multiplication, bitstream length should be $2\times 2=4$. The deterministic coding for -1, 0, and $+1$ in the bipolar format is shown in Table I. Since our proposed parallel design is accurate, to make a fair comparison, traditional SC discussed below is coded in this accurate deterministic format.

The proposed parallel SC adopts the bipolar coding method, as shown in Table I, and only two bits are needed to accurately represent the ternary value, which has the same coding efficiency as the binary computing and is twice as efficient as the SC deterministic coding method. Unlike the serial computation of traditional SC, the parallel SC computes all the bits in parallel in only one cycle, which also reduces the computational latency.

C. Parallel Multiplication

As shown in Table II, the traditional SC in bipolar coding achieves tenary multiplication by an XNOR gate but requires 4 cycles to complete. Moreover, to ensure the independence of the two input bitstreams, two stochastic number generators (SNGs) are required, each of which includes a random number source and a comparator to compare random numbers with the input binary data. Since SNGs are much larger than an XNOR, they dominate the area/power of the SC multiplier.

Given the parallel bitstream coding shown in Table I, we can obtain the truth table for the tenary multiplier for the coding, as listed in Table II. From the truth table, we can derive a parallel SC multiplier taking two ternary parallel bitstreams

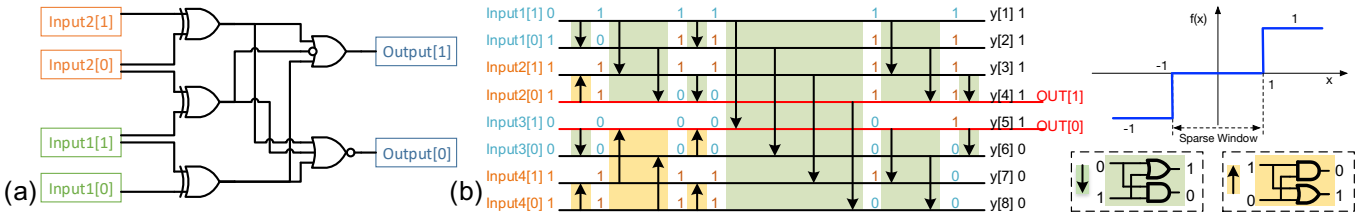

Fig. 3. Parallel SC component design. (a) multiplication; (b) accumulation and activation function.

as inputs (Fig. 3a). The parallel SC multiplier requires only five gates to achieve a completely accurate multiplication in one cycle. Since the data stored in SRAM are bitstreams for the parallel SC multiplier, SNGs are not required for data format conversion, which greatly reduces the area/power of the multiplier.

D. Accumulation and Activation Function Implementation

For the traditional SC-based NN accelerators, accumulating multiple bitstreams can be implemented by a parallel counter and a binary accumulator, which is also known as an APC [11]. The parallel counter adds up multiple bits from different bitstreams in the same clock cycle, and then the result of the parallel counter is added by an accumulator over all the cycles of a bitstream. Although this method is accurate, the conversion to binary numbers eliminates the fault-tolerance advantage. In addition, traditional SC uses a parallel counter and finite state machines (FSMs) to implement activation functions [12], which is complex and inaccurate, especially for the widely utilized step function like ReLU.

In this work, we address these problems by employing a bitonic sorting network to simultaneously implement the accumulation and the activation function. The Batcher's bitonic sorting algorithm [13]–[15] is a parallel sorting network. It is based on the concept of first converting the given sequence into a bitonic sequence and then converting the bitonic sequence into a monotonic sequence. The bitonic sequence is strictly increasing at first, and then after a point, strictly decreasing. Fig. 3b shows the implementation of accumulation and activation function using a bitonic sorting network. Here we use a two-step function as the activation function to implement the ternary activation quantization results, as shown in Fig. 4.

Fig. 3b illustrates an example of four parallel ternary bitstreams, encoding 0, 1, -1, and 1. The sorting network is designed to sort the input bits so that all the 1s will be before all the 0s. The arrow is a comparator consisting of an AND gate and an OR gate. The green (resp. yellow) box indicates descending (resp. ascending) sorting. It can be seen that the final output is sorted in a descending order as 11111000.

Note that by the proposed parallel bitstream coding, the value encoded by a bitstream equals the number of 1s in the stream minus 1 (see Table I). Thus, the accumulation result of k parallel bitstreams equals the number of 1s over all the bitstreams minus k. For example, the accumulation result of the 4 values represented by the input parallel bitstreams in Fig. 3b is 1, which equals the number of 1s over all the bitstreams minus 4. The sorting network does not change the number of 1s over all the bitstreams. However, by sorting all the bits of all the bitstreams, a single output of the sorting network directly represents the comparison of the input bitstream accumulation result with a certain value. For example, as shown in Fig. 3b, if the fourth bit $y[4]$ of the sorting result is 1, it means that the accumulated result of the parallel input bitstreams is greater than or equal to 0 (11110000). Conversely, if $y[4]$ is 0, it means that the accumulated result of the parallel input bitstreams is less than or equal to -1 (11100000). We can select the fourth bit $y[4]$ and the fifth bit $y[5]$ of the sorting result as the parallel output bitstream ($out[1]$ and $out[0]$) of the activation function, which is the same as the two-step activation function. That is, if the accumulated result of the parallel bitstreams is less than or equal to -1 (11100000), the parallel output bitstream is 00; if the accumulated result is greater than or equal to $+1$ (11111000), the parallel output bitstream is 11; and in the remaining case where the accumulated result is 0 (11110000), the parallel output bitstream is 10. Furthermore, since we only need two output bits of the sorting network, some of its comparators can be omitted, as shown in Fig. 3b, which further reduces the area of the sorting network.

In summary, we have proposed a special method based on bitonic sorting network to simultaneously implement the accumulation and activation function over parallel bitstreams. The whole process adopts the form of bitstream without conversion to binary numbers, which fully exploits the high fault tolerance of SC. Moreover, the calculation result is completely accurate without random fluctuation.

III. Experimental Results

To demonstrate the effectiveness of the proposed parallel SC-based NN accelerator, we compare it with the traditional SC-based and binary-based ones, which are based on the previous design methods [6], [11].

A. End-to-End TNN Setup and Implementation Detail

Due to the fact that bitonic sorting network only works on parallel inputs with size as a power of 2, we redesign the TNN structure, where each layer of the TNN uses an even number of convolution kernels, to make full use of the bitonic sorting network, as shown in Fig. 4. The MNIST handwritten digit database is used. The image input is also ternarized besides weights and activations. The accuracy of this end-to-end TNN is 97.36%. We synthesize different TNN accelerators using Synopsys Design Compiler with TSMC 40nm technology. The operating frequency is 100MHz.

B. Hardware Implementation in Each Layer

Table III compares the parallel SC-based TNN accelerator with the traditional SC-based and the binary-based ones, in

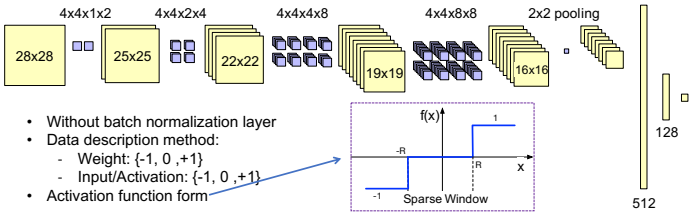

Fig. 4. Structure of the end-to-end TNN.

terms of area, power, time per layer, energy per layer, and energy efficiency. Note that for both parallel SC and binary computing, the data stored in memory occupies two bits. Thus, their time and energy consumption on memory transfers are the same.

Since parallel SC uses the bitonic sorting networks, its area is larger than traditional SC and binary, which is its cost in exchange for other advantages. As shown in Table III, our proposed parallel SC-based TNN accelerator achieves at least $2.5\times$ power improvement and $10\times$ energy efficiency improvement over the traditional SC-based one. Even compared to the binary TNN accelerator, the energy efficiency of the parallel SC-based one is at least $2.8\times$ higher. In addition, traditional SC requires multiple clock cycles to complete a layer of NN calculations. Thus, although the power consumption of traditional SC is lower than that of binary, the total energy consumption is higher. Furthermore, for $4 \times 4 \times 1$ convolution layer, the energy efficiency of the parallel SC-based NN accelerator can reach 833 Tops/W, which achieves $24.5\times$ improvement over the traditional SC-based one, showing the great potential for low-energy-budget applications.

Therefore, the experimental results show that the proposed parallel SC-based TNN accelerator can greatly improve the power consumption and energy efficiency over the traditional SC-based and binary-based ones, at the cost of the area.

C. Accuracy Comparison and Fault Tolerance

Table IV compares fault tolerance results for different TNN accelerators. We applied two soft error models to the TNN accelerator, namely the SRAM read error and the MAC calculation error (see Fig. 1). For both models, we set the flip rate for each bit to be 1%, 5%, and 10%, respectively. The results show that our proposed parallel SC has high fault tolerance to both soft errors due to the fact that APC is not used to convert the bitstream to the binary format. Notice that the fault tolerance of the traditional SC-based NN accelerator is the same as the binary one, due to its hybrid format. That is, except that the multiplication is completed in the form of bitstreams, the other parts are calculated in the binary form (see Fig. 2).

IV. Conclusion

In this paper, we propose a new fault-tolerant parallel SC-based TNN accelerator, where all calculations are implemented in the form of parallel bitstreams. A new parallel SC multiplier is proposed. Using the bitonic sorting network, the accumulation and activation function are simultaneously implemented in parallel SC. Our proposed parallel TNN accelerator design not only maintains high fault tolerance of SC but also shows more than $10\times$ energy improvement over the traditional SC-based one.

TABLE III
Hardware cost comparison for different implementations and TNN layers.

		Area (μm^2)	Power (μW)	Time (μs)	Energy (fJ)	Energy Efficiency (Tops/W)
4x4x1Conv	Binary	294	66	13	824	97
	Trad. SC	708	48	50	2385	34
	Para. SC	379	8	13	96	**833**
4x4x2Conv	Binary	658	104	19	2013	123
	Trad. SC	1325	89	77	6722	37
	Para. SC	1019	19	19	374	**663**
4x4x4Conv	Binary	1451	177	29	5112	145
	Trad. SC	2539	161	116	18599	34
	Para. SC	2612	46	29	1340	**552**
4x4x8Conv	Binary	3221	311	20	6369	165
	Trad. SC	4977	275	82	22528	47
	Para. SC	6530	110	20	2253	**465**

TABLE IV
Accuracy for different accelerators considering soft errors.

	Accuracy						
	w/o Error	w. Read Error			w. Calculation Error		
		1%	5%	10%	1%	5%	10%
Binary	97.36%	96.02%	80.77%	40.26%	97.30%	95.77%	79.05%
Trad. SC	97.36%	96.02%	80.77%	40.26%	97.30%	95.77%	79.05%
Para. SC	97.36%	96.61%	93.06%	81.17%	97.34%	97.19%	94.55%

Acknowledgement

This work was partly supported by NSFC (61522402 and 61421005) and the 111 Project (B18001).

References

[1] F. Li *et al.*, "Ternary weight networks", *CoRR*, pp. 1–5, 2016, [online] Available: http://arxiv.org/abs/1605.04711.

[2] L. Deng *et al.*, "GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," in *Neural Networks*, vol. 100, pp. 49–58, 2018.

[3] W. Qian *et al.*, "An architecture for fault-tolerant computation with stochastic logic," in *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[4] J. P. Hayes, "Introduction to stochastic computing and its challenges," *DAC'15*, pp. 59:1–59:3.

[5] Y. Zhang *et al.*, "Design guidelines of stochastic computing based on FinFET: A technology-circuit perspective," *IEDM'17*, pp. 6.6.1–6.6.4.

[6] K. Kim *et al.*, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," *DAC'16*, pp 124:1–124:6.

[7] A. Ren *et al.*, "SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing," in *ACM SIGOPS Operating Systems Review*, vol. 25, no. 5, pp. 405–418, 2017.

[8] H. Sim *et al.*, "DPS: dynamic precision scaling for stochastic computing-based deep neural networks," *DAC'18*, pp. 13:1–13:6.

[9] Y. Zhang *et al.*, "Parallel convolutional neural network (CNN) accelerators based on stochastic computing," *SiPS'19*.

[10] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," *ICCAD'16*, pp. 102:1–102:8.

[11] H. Sim *et al.*, "Scalable stochastic-computing accelerator for convolutional neural networks," *ASPDAC'17*, pp. 696–701.

[12] J. Li *et al.*, "Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks," *IJCNN'17*, pp. 1230–1236.

[13] K. E. Batcher, "Sorting networks and their applications," *Spring Joint Computer Conference*, pp. 307–314, April 1968.

[14] R. Cai *et al.*, "A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology," *ISCA'19*, pp. 567–578.

[15] M. H. Najafi *et al.*, "Low-cost sorting network circuits using unary processing," in *IEEE Transactions on VLSI Systems*, vol. 26, no. 8, pp. 1471–1480, 2018.