Integration, the VLSI Journal 70 (2020) 70–79

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Fault tolerance in memristive crossbar-based neuromorphic computing systems

Qi Xu [a], Song Chen [b], Hao Geng [c], Bo Yuan [d], Bei Yu [c], Feng Wu [b], Zhengfeng Huang [a,*]

[a] Department of Electronic Science and Technology, Hefei University of Technology, China
[b] School of Microelectronics, University of Science and Technology of China, China
[c] Department of Computer Science and Engineering, The Chinese University of Hong Kong, China
[d] Department of Computer Science and Engineering, Southern University of Science and Technology, China

ARTICLE INFO

Keywords:
Neuromorphic computing system
Memristive crossbar
Fault tolerance
Hierarchical clustering

ABSTRACT

In recent years, neuromorphic computing systems (NCS) based on memristive crossbar have provided a promising solution to enable acceleration of neural networks. However, Stuck-at faults in the memristor devices significantly degrade the computing accuracy of NCS. In this paper, we propose an effective fault tolerant framework for memristive crossbar-based neuromorphic computing systems. First, a fault tolerance-aware hierarchical clustering method is proposed to partition weight connections of a sparse neural network into clusters. Then, for each cluster, memristive crossbar configuration is proposed to determine a suitable size of the crossbar with consideration of both hardware cost and successful mapping rate. Next, an integer linear programming formulation is developed to derive a connection-memristor mapping for fault tolerance. Finally, an efficient matching-based heuristic algorithm is further proposed to speed-up the fault-tolerant mapping process. Experimental results show that the proposed fault tolerant framework can improve the successful mapping rate and simultaneously reduce the hardware cost.

1. Introduction

Neuromorphic computing systems (NCS) based on hardware designs intend to mimic neuro-biological architectures [1]. Different from conventional von Neumann architectures, NCS is often constructed with highly parallel, extensively connected, and collocated computing and storage units, which eliminates the gap between CPU computing capacity and memory bandwidth [2]. However, the implementation of NCS on CMOS technology has been shown to suffer from mismatch between NCS building blocks (neuron and synapse) and CMOS primitives (Boolean logic). To address this problem, the emerging memristive technology is adopted to implement synapse circuit due to the similarity between memristive and synaptic behaviors [3,4]. For example, the memristor is suitable to store the weight of synapse since the resistance of memristor can be programmed by applying current or voltage. In addition, compared with the state-of-the-art CMOS design, memristive crossbar has been proven as one of the most efficient nanostructures that carry out matrix-vector multiplications while hardware cost and computation energy are significantly reduced [1]. However, despite of these tremendous advantages, NCS implementations on memristive crossbars also encounter some design challenges.

First, the crossbar utilization can be low if a large-scale sparse neural network is directly implemented on a memristive crossbar, resulting in highly area-inefficient designs. For instance, in LDPC coding based on message passing algorithm, the network sparsity is higher than 99% [5]. Many previous works have been proposed to enable the efficient realization on memristive crossbars. An iterative spectral clustering method is performed in Refs. [2,6] to group the neural connections into a set of memristive crossbars. However, since the spectral clustering is executed in every iteration, the methods could be time-consuming when the neural network is large. Cui et al. [7] proposed a matrix reordering method to facilitate sparse weight connections clustering. But the crossbar utilization is not considered, so memristive crossbars with low utilization are still generated.

* Corresponding author.
E-mail address: huangzhengfeng@139.com (Z. Huang).

https://doi.org/10.1016/j.vlsi.2019.09.008
Received 24 June 2019; Received in revised form 25 August 2019; Accepted 14 September 2019
Available online 23 September 2019

Second, memristor suffers from various defects and faults, leading to a significant yield loss and errors in NCS. The faults occurred in the fabrication process make the memristor get stuck at high or low resistance state; the computational accuracy for NCS is limited to incorrect weights in memristive crossbar. To tolerate Stuck-at faults, a number of solutions have been proposed. Huangfu et al. [8] proposed a mapping algorithm for tolerating Stuck-at faults by using extra memristive crossbars. Xia et al. [9] proposed a remapping-based fault-tolerant method by using the inherent sparsity of neural networks. Yuan et al. [10] proposed a memetic algorithm for the defect-tolerant logic mapping in crossbar-based nanoarchitectures. Tunali et al. [11] proposed a hybrid fault tolerant logic mapping method. However, both [10,11] are only considering the stuck-at-one faults. Therefore, the methods are not suitable for stuck-at-zero faults. Su et al. [12] proposed an integrated algorithm framework that performs mapping, logic morphing, and logic hardening simultaneously. Logic morphing process exploits the various equivalent forms of a logic function to tolerate faults. Meanwhile, logic hardening process adds redundant vertical wires in crossbars to make the hardened logic function fault tolerable. As a result, the sizes of used crossbar are two or three times larger than the sizes of the logic function matrix. Although the successful mapping rate can be improved, the hardware utilization are significantly reduced. Tunali et al. [13] developed a fault tolerant logic mapping methodology for nano-crossbar arrays, which contains pre-mapping morphing, defect oriented sorting, row by row matching, and backtracking. In addition to mapping-based fault tolerant methods, another retraining-based method are proposed in Refs. [14–16] to tolerate Stuck-at faults. However, when the number of faults is larger than the inherent fault tolerance of the neural network, the performance of retraining method will be degraded. Sometimes the hardware-level fault tolerant scheme does not allow changes to the well trained model [17].

To the best of our knowledge, there is no prior work focusing on both above challenges, which means current state of the arts either only solving sparsity problem in NCS or tackling fault-tolerance issue alone. In this paper, we argue that taking account of both challenges simultaneously will benefit the NCS design. According to this argument, for the first time, we propose a fault tolerant synapse mapping framework, in which both hardware utilization and successful mapping rate are considered. Key technical contributions of this work are listed as follows.

- Given a sparse neural network, A fault tolerance-aware hierarchical clustering method is proposed to partition weight connections into a set of clusters. In order to improve the successful mapping rate of memristive crossbar-based NCS, input neurons that connect to different output neurons are grouped into the same cluster, under the guidance of Jaccard distance metric.
- For connection matrix of each cluster, a non-linear programming is proposed to determine suitable size of the mapped memristive crossbar, considering both hardware cost and successful mapping rate.
- Fault-tolerant mapping is formulated as an integer linear programming (ILP) to map connection matrix to memristive crossbar.
- An efficient matching-based heuristic algorithm is further proposed to speed-up the ILP process.
- A Monte Carlo simulation is exploited to evaluate the performance of the fault tolerant synapse mapping framework on different benchmarks.

The remainder of this paper is organized as follows. Section 2 presents the preliminary and gives the problem formulation. Section 3 introduces the proposed fault tolerant framework for neuromorphic computing systems. Section 4 lists experimental results, followed by conclusion in Section 5.

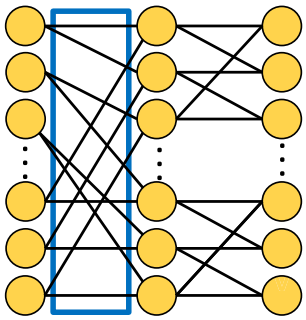

Fig. 1. Sparse neural network.

2. Preliminaries

2.1. Neural network

In a neural network, pre-synaptic (i.e. input) neurons $\boldsymbol{I}$ send signals into the network, while post-synaptic (i.e. output) neurons $\boldsymbol{O}$ receive information through memristors. The memristors will apply different weights on the information during transmission, which can be expressed as $O = WI$. Theoretically, a memristor device can achieve continuous analog resistance states. However, under the current technology, the imperfection of the fabrication process causes memristance variations from device to device [18]. Thus in most memristive crossbar designs, only two resistance states are adopted, i.e., high and low resistance states. Compared with CMOS-based technology, memristor with two resistance states can effectively reduce power consumption thanks to low forming and set voltage [19]. In this paper, the synaptic weight matrix W is represented as $(1, -1)$ connection matrix where "1" indicates a connection exists between two corresponding neurons and "−1" vice versa. Note that "−1" value is only for the convenience of fault tolerant mapping calculation, but not the true weight value. An example of a sparse neural network is shown in Fig. 1, where each neuron is only connected with two neurons in the previous layer.

2.2. Fault models in memristor

The computational accuracy of memristive crossbar-based NCS is limited to faults in memristor. According to Refs. [20,21], most of the memristor faults are caused by insufficient dopants or impurities. Among all kinds of hard faults, stuck-at-one (SA1) faults and stuck-at-zero (SA0) faults appear rather frequently [22]. The SA1 faults, on one hand, are caused by permanent open switch defects and broken word-lines. A memristor device with SA1 fault is always in the high resistance state (HRS). The SA0 faults, on the other hand, are caused by over-forming defects, reset failures and short defects. SA0 faults force the on-chip memristors in a low resistance state (LRS) [14,22]. In nanocrossbar fabrication, it is generally believed that SA1 faults are far more common than SA0 faults [12]. In this paper we use the memristive crossbar matrix C to represent the states of crossbar, in which fault free memristor is denoted as "0" and SA1 and SA0 faults are denoted as "1" and "−1", respectively. Via the memristive crossbar structure of a neural network, the computation of $O = CI$ is in $\boldsymbol{\Theta}(1)$ [23]. Note that we assume that the faults are independently and uniformly distributed as previous work did [10]. An example of memristive crossbar structure is shown in Fig. 5.

To detect and identify the faults, several testing methods are developed. March-C algorithm is proposed to detect Stuck-at faults [22]. To accelerate the test process, a sneak-path technique is proposed to test a set of adjacent memristors simultaneously [24]. These testing methods can pinpoint the exact locations of fault memristors, and hence improve the yield of memristive crossbars with fault tolerance guaranteed.

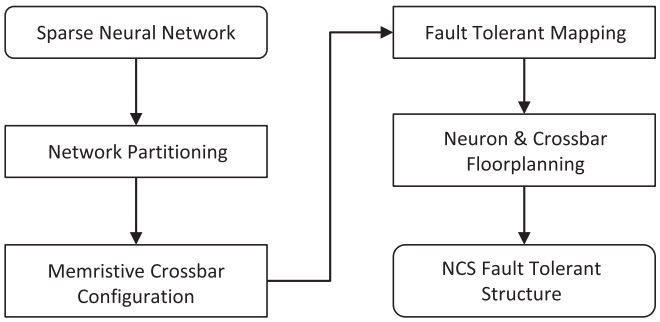

Fig. 2. The proposed fault tolerant framework for NCS.

2.3. Problem formulation

In this work, the successful mapping rate is defined as the probability of finding a valid mapping between connection matrix and memristive crossbar. The utilization of a memristive crossbar is given by the ratio between the utilized connections in the network and the total available connections in the crossbar. The area and the wirelength of floorplan are used to evaluate the hardware cost of NCS. Therefore, the successful mapping rate, the crossbar utilization, the area and the wirelength of floorplan are exploited to quantify the simulation results. We define the fault tolerance problem in NCS as follows:

Input: A sparse neural network and the fault probability distribution in memristive crossbar.

Output: The physical implementation of the fault tolerant neuromorphic computing systems, including a set of memristive crossbars used to store the weights of synapse in the sparse network.

Objective: Boosting the successful mapping rate and simultaneously reducing the hardware cost.

3. Fault tolerant framework for NCS

The overall flow of the proposed fault tolerant framework for NCS is shown in Fig. 2, which mainly consists of four stages: (1) Given a sparse neural network, a fault tolerance-aware hierarchical clustering method is proposed to partition sparse connections into clusters. (2) Then, for the connection matrix of each cluster, a non-linear programming is proposed to determine a suitable size of the mapped memristive crossbar. (3) Next, an ILP based algorithm and a matching-based heuristic method are proposed respectively to derive a connection-memristor mapping for fault tolerance. (4) Finally, a floorplanning is performed to realize the neuromorphic computing systems. Based on the floorplanning results, we estimate the area and wirelength of NCS. In this work, the neurons and the memristive crossbars are considered as blocks. The neuron blocks connect the memristive crossbar blocks through wires.

Fig. 3 shows a floorplan example of a neural network based on memristive crossbars, where input neuron blocks connect to output neuron blocks through memristive crossbar blocks. We exploit sequence pair (SP) to represent floorplans. IARFP in Ref. [25] is adopted to conduct the floorplanning. The chip area is evaluated by the method in Ref. [25], which takes the fixed-outline constraint into account. The wirelength is estimated by the half perimeter wirelength (HPWL) model. In this paper, we focus on the first three stages. Details of these major steps will be explained in the following sub-sections. For convenience, some notations used in this section are listed in Table 1.

3.1. Network partitioning

In real applications, large neural networks are often sparse. If a sparse neural network is directly implemented on a memristive crossbar, the crossbar utilization can be low, resulting in highly area-inefficient designs. Besides, sometimes realizing the sparse neural network on a single memristive crossbar with faults can not generate a fault tolerant solution. Therefore, a fault tolerance-aware hierarchical clustering method is proposed to partition sparse connections into a set of dense clusters. After clustering, some "−1" weights are removed, and thus reducing the possibility of mapping "−1" weights to SA1 faults.

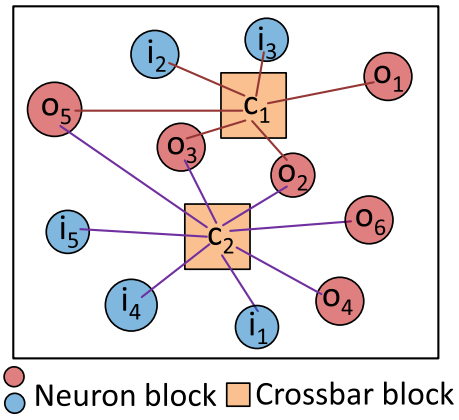

Fig. 3. Floorplan of neurons and crossbars.

Table 1
Notations used in this section.

W	Connection matrix of a sparse neural network
C	State matrix of a memristive crossbar
W_i	Connection matrix of cluster i
C_i	State matrix of the memristive crossbar for W_i to be mapped to
M, N	Number of rows and columns of W_i
M_c, N_c	Number of rows and columns of C_i
p_o, p_s	SA1 and SA0 fault rate

Hierarchical clustering generates clusters in a bottom-up iterative manner [26]. In every iteration, two clusters that are closest in "distance" are merged. The metric of "distance" varies with specific applications in practice. The iterative merging is repeated until all data points are grouped into one cluster. In our proposed framework, to improve the successful mapping rate of neuromorphic computing systems, input neurons connecting to different output neurons tend to be clustered. We propose a distance metric between two input neurons, i_p and i_q as follows:

$$\mathrm{d}(i_p, i_q) = 1 - \frac{n_{10} + n_{01}}{n_{11} + n_{10} + n_{01}}, \tag{1}$$

where n_{11} represents the number of output neurons simultaneously connecting to i_p and i_q, while n_{10} and n_{01} count the number of output neurons only connecting to i_p or i_q. For example in Fig. 4(a), the distance between i_3 and i_4 is $1 - \frac{2}{3} = \frac{1}{3}$. It can be seen that the distance metric in Equation (1) forces input neurons connecting to different output neurons to be clustered. The proposed metric is superior over other metrics, because it is beneficial to reducing mapping errors. For example, i_2 and i_3 are partitioned into a cluster, as shown in Fig. 4(b). We notice that a connection exists between i_3 and o_5 ("1" weight), while i_2 is not connected to o_5. Therefore, based on this staggered distribution, both row and column reordering can avoid mapping the "1" weight to the SA0 fault.

Fig. 4 illustrates the process of hierarchical clustering. A sparse neural network W with five input neurons (data points) is shown in Fig. 4(a). When hierarchical clustering performs, according to the distance calculated by Equation (1), the first iteration step merges the input neurons i_1 and i_5, and then the second iteration step groups the input neurons i_2 with i_3. Next, the input neuron i_4 is clustered with $\{i_1, i_5\}$ in the third iteration step. Finally, the two clusters containing $\{i_1,$

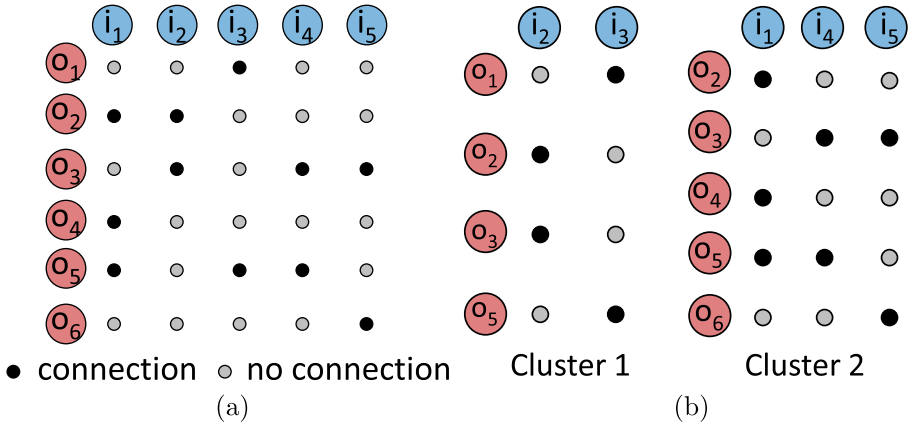

Fig. 4. (a) Connection matrix W of a sparse neural network with five input neurons $\{i_1, \ldots, i_5\}$ and six output neurons $\{o_1, \ldots, o_6\}$; (b) pruned connection matrices of the two clusters W_1 and W_2 generated by hierarchical clustering.

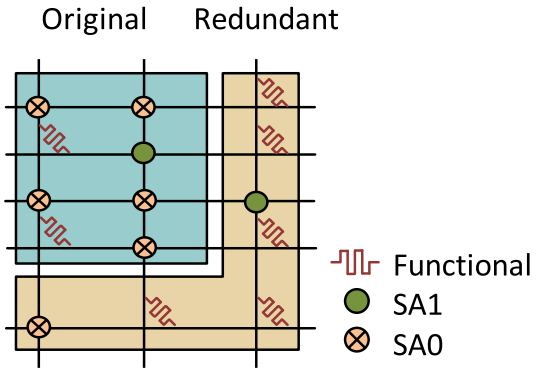

Fig. 5. An example of the mapped memristive crossbar for W_1.

i_4, $i_5\}$ and $\{i_2, i_3\}$ respectively are re-united in the last iteration step.

Besides, the L-method [27] is adopted to determine the optimal number of clusters in our work and thus, a regression problem is formulated. After dealing with the connection matrix W shown in Fig. 4(a) via the L-method, the input neurons are divided into 2 clusters. Then we do the pruning by removing the output neurons without connecting any input neurons. The two pruned connection matrices W_1 and W_2 are shown in Fig. 4(b). Generally, for a sparse network, mapping dense clusters to memristive crossbars can achieve high utilizations.

3.2. Memristive crossbar configuration

In this stage, determining the size of the mapped memristive crossbar C_i for each connection matrix W_i emerges. Because larger crossbar makes it easier to find a valid mapping solution, redundancy has been introduced in previous work [28] during constructing a crossbar. However, the crossbar size is empirically set to 1.5 times of the connection matrix size with ignoring hardware cost. Different from prior art, we propose a non-linear programming to determine a suitable size of the memristive crossbar for each cluster, with consideration of both hardware cost and successful mapping rate. Note that "−1" weight can be mapped to either fault free or SA0 fault point, while "1" weight can be mapped to fault free or SA1 fault point. Additionally, we assume that the faults are independently and uniformly distributed as previous work did [10]. Based on the above preliminaries, our non-linear programming is modeled in Formula (2).

$$\min_{M_c, N_c} \; M_c \times N_c, \tag{2a}$$

$$\text{s.t.} \;\; P_1^r = (1 - p_s \cdot N/N_c)^{n_1^r}, \tag{2b}$$

$$P_0^r = (1 - p_o \cdot N/N_c)^{n_{-1}^r}, \tag{2c}$$

$$P = \prod_{r=0}^{M-1} [1 - (1 - P_1^r \times P_0^r)^{M_c - r}] \geq P_t, \tag{2d}$$

$$M_c \geq M, N_c \geq N. \tag{2e}$$

The objective function (2a) is to minimize the memristive crossbar size. In constraints (2b) and (2c), n_1^r and n_{-1}^r represent the number of "1" and "−1" weights in r-th row of W_i with $n_1^r + n_{-1}^r = N$. As illustrated in constraints (2b) and (2c), when considering the redundant columns in crossbar, P_1^r shows the probability of mapping "1" weights in r-th row of W_i to fault free or SA1 fault points in a row of C_i, while P_0^r describes the probability of mapping "−1" weights in r-th row of W_i to fault free or SA0 fault points in a row of C_i. Therefore, $P_1^r \times P_0^r$ refers to the matching probability of the r-th row of W_i to one row of C_i. Given a crossbar C_i with size $M_c \times N_c$, the matching probability of the r-th row of W_i to a row of C_i can be formulated with $[1 - (1 - P_1^r \times P_0^r)^{M_c}]$. As a result, multiplication of matching probability of all rows of W_i gives the successful mapping rate P from W_i to C_i as shown in the constraint (2d). According to the calculation for P, larger M_c and N_c improve P, meanwhile it increases the hardware overhead. So constraint (2d) (i.e., forcing P to be larger than target value P_t) makes a trade-off between the successful mapping rate and hardware cost. Formula (2) is addressed by a greedy method, with M_c and N_c alternately increasing by 1 until satisfying P_t.

For instance, after solving Formula (2) for W_1 shown in Fig. 4(b), the successful mapping rate P is to be larger than P_t until $M_c = 5$ and $N_c = 3$ (one redundant row and one redundant column). Based on the assumption that the faults are independently and uniformly distributed, an example of the corresponding memristive crossbar C_1 is shown in Fig. 5.

3.3. ILP based mapping

In this section, we discuss how the fault tolerant mapping problem can be formulated as an integer linear programming (ILP). For convenience, some notations used in this section are listed in Table 2.

$$\max \quad \sum_{k=1}^{M} \sum_{j=1}^{M_c} \sum_{p=1}^{N} \sum_{q=1}^{N_c} X_{kjpq} \tag{3a}$$

$$\text{s.t.} \quad \sum_{j=1}^{M_c} r_{kj} = 1, \forall k \in [1, M], \tag{3b}$$

Table 2
Notations used in ILP.

w_{kp}	weight value at k-th row and p-th column of W_i
s_{jq}	state value of memristor at j-th row and q-th column of C_i
r_{kj}	binary variable; if k-th row of W_i is mapped to j-th row of C_i then $r_{kj} = 1$, otherwise $r_{kj} = 0$ $(1 \le k \le M, 1 \le j \le M_c)$
c_{pq}	binary variable; if p-th column of W_i is assigned to q-th column of C_i then $c_{pq} = 1$, otherwise $c_{pq} = 0$ $(1 \le p \le N, 1 \le q \le N_c)$
X_{kjpq}	binary variable; if $r_{kj} = 1$ and $c_{pq} = 1$ then $X_{kjpq} = 1$, otherwise $X_{kjpq} = 0$

$$\sum_{k=1}^{M} r_{kj} \le 1, \forall j \in [1, M_c], \tag{3c}$$

$$\sum_{q=1}^{N_c} c_{pq} = 1, \forall p \in [1, N], \tag{3d}$$

$$\sum_{p=1}^{N} c_{pq} \le 1, \forall q \in [1, N_c], \tag{3e}$$

$$r_{kj} + c_{pq} - 2 \cdot X_{kjpq} \ge 0, \forall k, j, p, q, \tag{3f}$$

$$r_{kj} + c_{pq} - X_{kjpq} \le 1, \forall k, j, p, q, \tag{3g}$$

$$w_{kp} \cdot (\sum_{j=1}^{M_c} \sum_{q=1}^{N_c} s_{jq} \cdot X_{kjpq}) \ge 0, \forall k, p, \tag{3h}$$

$$\sum_{k=1}^{M} \sum_{j=1}^{M_c} \sum_{p=1}^{N} \sum_{q=1}^{N_c} X_{kjpq} \le M \cdot N, \tag{3i}$$

$$r_{kj}, c_{pq}, X_{kjpq} \in \{0, 1\}, \forall k, j, p, q. \tag{3j}$$

Based on the above notations, the fault tolerant mapping problem can be formulated as the following ILP (3). Given a connection matrix W_i with size $M \times N$ and a corresponding memristive crossbar matrix C_i with size $M_c \times N_c$, constraint (3b) ensures that each row of W_i would be mapped to one and only one row of C_i. Constraint (3c) guarantees that at most one row of W_i can be mapped to a row of C_i. Similarly, constraint (3d) is defined to ensure that each column of W_i would be mapped to one and only one column of C_i, and constraint (3e) restricts that at most one column of W_i can be mapped to a column of C_i. In order to maintain the coherency of the mapping, a binary variable X_{kjpq} is introduced. Constraint (3f) ensures that no row (column) of W_i is mapped to a row (column) of C_i when the number of suitable mapping memristors in C_i is not available. $X_{kjpq} = 1$ if and only if $r_{kj} = 1$ and $c_{pq} = 1$, which is defined in constraint (3g). Note that "1" weight can be mapped to either fault free or SA1 fault point, while "−1" weight can be mapped to fault free or SA0 fault point. Thus constraint (3h) is introduced to guarantee the correctness of the weight-memristor mapping. Besides, in order to find a valid mapping between weight matrix W_i and memristive crossbar C_i, the mapping position of each weight element of W_i at C_i must be known. Constraint (3i) ensures that the number of mapping points will not be exceed the number of weight elements. The objective function (3a) is to maximize the number of mapping points. As a result, a valid mapping solution is reached when all weight elements find mapping points. Although the ILP based mapping method is time consuming for the large weight matrix, it can ensure the convergence to a global optimum compared with the heuristic methods in Ref. [29]. As a result, a valid mapping between connection matrix and memristive crossbar is guaranteed to be found if it exists.

Observed from the ILP formulation, the number of variables is $(MM_c + NN_c + MNM_cN_c)$, and the number of constraints is $(M + M_c + N + N_c + 2 \cdot MNM_cN_c + MN + 1)$.

3.4. Heuristic mapping

In this section, a matching-based heuristic algorithm is further proposed to speed-up the connection-memristor mapping process. Given a W_i with size $M \times N$ and a corresponding C_i with size $M_c \times N_c$, two decision vectors that can represent the mapping result need to be determined:

Table 3
Benchmark statistics.

Benchmark	Size	Sparsity	Synapses
b1	784 × 10	56.45%	3414
b2	784 × 10	60.36%	3108
b3	784 × 10	62.95%	2905
b4	141 × 14	57.45%	840
b5	784 × 10	66.06%	2661
b6	481 × 32	69.13%	4752
b7	4096 × 1000	84.99%	614809
b8	4096 × 1000	90.01%	409190

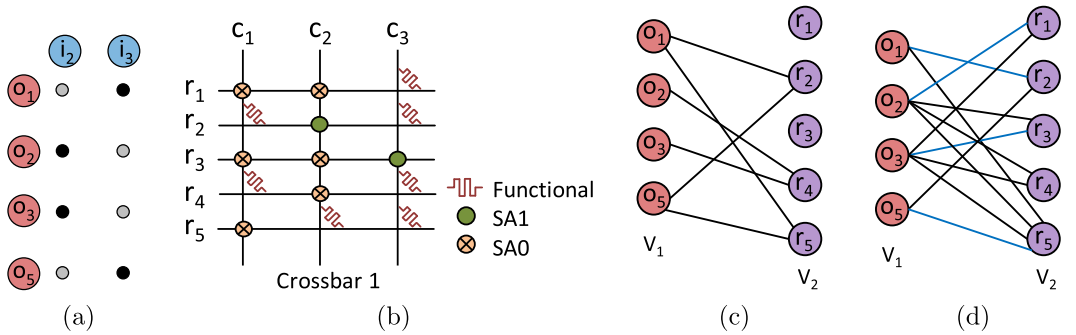

Fig. 6. (a) W_1 of cluster 1; (b) an example of the mapped memristor crossbar C_1 for W_1; (c) a bipartite graph of W_1 to C_1 under $A_c = [1, 2]$, in which a matching cannot be solved; (d) a bipartite graph of W_1 to C_1 under $A_c = [3, 2]$ and the maximum bipartite matching result $A_r = [2, 1, 3, 5]$ (blue lines). (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

(1) $1 \times M$ Row Assignment Vector (A_r): $A_r[k] = j$ if row k of W_i is assigned to row j of C_i ($1 \le k \le M, 1 \le j \le M_c$);
(2) $1 \times N$ Column Assignment Vector (A_c): $A_c[p] = q$ if column p of W_i is assigned to column q of C_i ($1 \le p \le N, 1 \le q \le N_c$).

The fault tolerant mapping is equivalent to the subgraph isomorphism problem, which is an NP-complete problem [30]. Thus most of existing algorithms are not efficient enough to jointly tackle two decision vectors A_r and A_c. In our relaxed problem, A_c is fixed, thus A_r is expected to be efficiently solved by bipartite matching. However, sometimes the algorithm is unfortunately stuck in finding a valid solution under the current A_c. Therefore, column permutations are exploited to update A_c until reaching a valid A_r.

Algorithm 1 Matching-based Heuristic
Input: W_i and C_i.
Output: A_c and A_r.
1: Rearrange columns of W_i in descending order according to the number of "1" weights;
2: Rearrange columns of C_i in ascending order according to the number of SA0 faults; ▷ A_c initialization
3: **for** $j \leftarrow 1$ to K **do**
4: Construct a bipartite graph between rows of W_i and C_i;
5: Maximum bipartite matching; ▷ `Kuhn_Munkres`
6: **if** a maximum bipartite matching is found **then**
7: Record A_r;
8: break;
9: **else**
10: Applying a pairwise column permutation of C_i;
11: Update A_c according to permutation result;
12: **end if**
13: **end for**

The major steps of the proposed matching-based heuristic algorithm are summarized in Algorithm 1. First, in order to reduce the matching runtime, a rearrangement for the columns of W_i and C_i is performed (lines 1–2). Because "1" weight cannot be mapped to SA0 fault, we tend to map the columns with more "1" weights in W_i to the columns with fewer SA0 faults in C_i during the rearrangement. As a result, an initial A_c is obtained. According to our preliminary observations, the rearrangement of matrices process can reduce the mapping runtime by 15.4%. Then given the A_c, in order to represent all the possible row matching from W_i to C_i, a bipartite graph $G(V, E)$ is constructed (line 4). Here vertex set $V = V_1 \cup V_2$, where V_1 is the row set of W_i and V_2 is the row set of C_i. Besides, the edge set $E = \{(p, q) \mid p \in V_1 \text{canbematchedto} q \in V_2\}$. For the bipartite graph, Kuhn_Munkres algorithm is used to find a maximum bipartite matching [31] (line 5). If each row of W_i has a corresponding matching row of C_i, it means that a valid solution is found (lines 7–8). Otherwise column permutations are performed for C_i (lines 10–11). The above process is terminated until a valid solution is found or satisfying the iteration number K.

We exemplify the process of the matching-based heuristic algorithm with Fig. 6. Given a W_1 shown in Fig. 6(a) which is the clustering result in Fig. 4 and a corresponding C_1 illustrated in Fig. 6(b), if A_c equals to Refs. [1,2], a corresponding bipartite graph is constructed as presented in Fig. 6(c). Since both rows o_2 and o_3 of W_1 can only be mapped to row r_4 of C_1, a valid solution cannot be solved under the A_c. Then a pairwise column permutation is performed to reorder the column of C_1. If columns c_1 and c_3 of C_1 are swaped ($A_c = [3, 2]$), a corresponding bipartite graph is constructed as shown in Fig. 6(d). A valid solution can be obtained by Kuhn_Munkres algorithm (blue lines in Fig. 6(d)), which means rows o_1, o_2, o_3 and o_5 of W_1 are mapped to rows r_2, r_1, r_3 and r_5 respectively. Therefore, the combination of $A_r = [2, 1, 3, 5]$ and $A_c = [3, 2]$ is a valid mapping from W_1 to C_1.

4. Experimental results

We implement the proposed framework using MATLAB on a 2.20 GHz 64-bit Windows 7 machine with 12 GB RAM. IARFP [25] is adopted to conduct the floorplanning, which is run on a 12-core 2.0 GHz Linux server with 64 GB RAM. GUROBI [32] is used as the ILP solver. p_o and p_s are set to 9.04% and 1.75%, respectively [14,22]. The value of P_t is set to 99%. Monte Carlo simulations are tested with a sample size of 400. The areas of neurons and memristive crossbars with different sizes are extracted from Refs. [2,33]. The evaluation method in Ref. [20] is adopted to estimate power and delay overhead. In the work, we consider the similar two-level logic design [20], and thus the delay overhead for all testbenches is constant (7 cycles). In addition, since the energy consumption in a crossbar is proportional to the number of the used memristors, the power factor defined as the ratio of utilized memristors in crossbar to the delay overhead can be adopted to describe the power consumption [20]. To evaluate the proposed algorithm, a two-layer feed-forward neural network model for MNIST handwritten digit recognition is first exploited. Three sparse connection matrices of the neural network are generated to evaluate the proposed algorithm (b_1–b_3). Then we also randomly generate three sparse connection matrices with different sizes and sparsities (b_4–b_6). Finally AlexNet is trained for ImageNet [34] dataset, which consists of five convolutional layers, three pooling layers and three fully connected layers. Because more than 90% of the weights are situated in the fully connected layers, hence we mainly focus on the fully connected layers. By using the synapse-granularity pruning, two sparse weight matrices of the last fully connected layers are achieved (b_7–b_8). Table 3 lists all the statistics of different test cases.

4.1. Effectiveness of fault tolerant framework

In the experiment, two different design methodologies are compared to demonstrate the effectiveness of the proposed fault tolerant frame-

Table 4
Performance evaluation of the proposed fault tolerant framework.

Bench	Area	Wire	Util	PF	FC + NLP + ILP		FC + NLP + MH	
	(mm^2)	(mm)	(%)		Succ(%)	RT(s)	Succ(%)	RT(s)
$b1$	1.02	588.20	30.38	44.49	100.0	10.19	100.0	0.20
$b2$	0.98	550.04	30.70	40.38	100.0	10.34	100.0	0.34
$b3$	0.95	546.50	30.61	37.72	100.0	12.77	100.0	0.41
$b4$	0.22	62.21	28.92	17.05	100.0	6.31	96.25	0.08
$b5$	0.92	532.08	26.05	34.56	NA	>1800	94.18	0.68
$b6$	0.89	390.22	22.58	56.37	NA	>1800	90.32	0.79
$b7$	6.08	2181.00	23.51	2927.66	NA	>1800	83.51	10.62
$b8$	5.47	1903.59	22.19	2015.72	NA	>1800	80.17	10.97

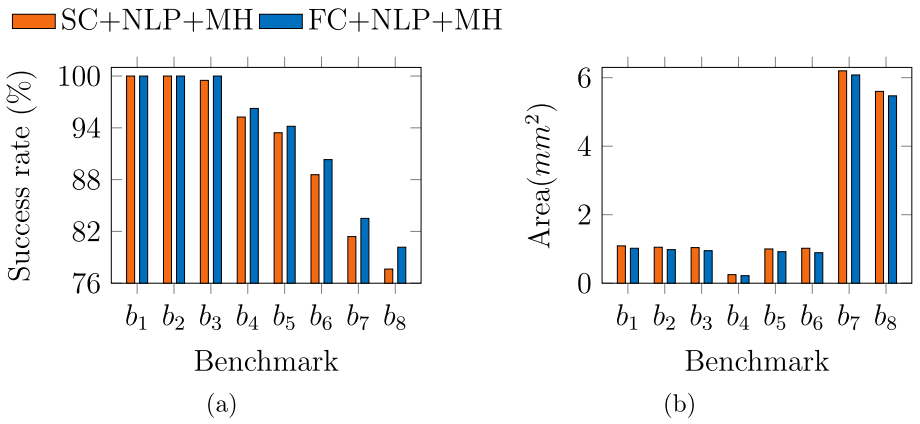

Fig. 7. Comparison between "SC + NLP + MH" and "FC + NLP + MH" on (a) success rate and (b) area.

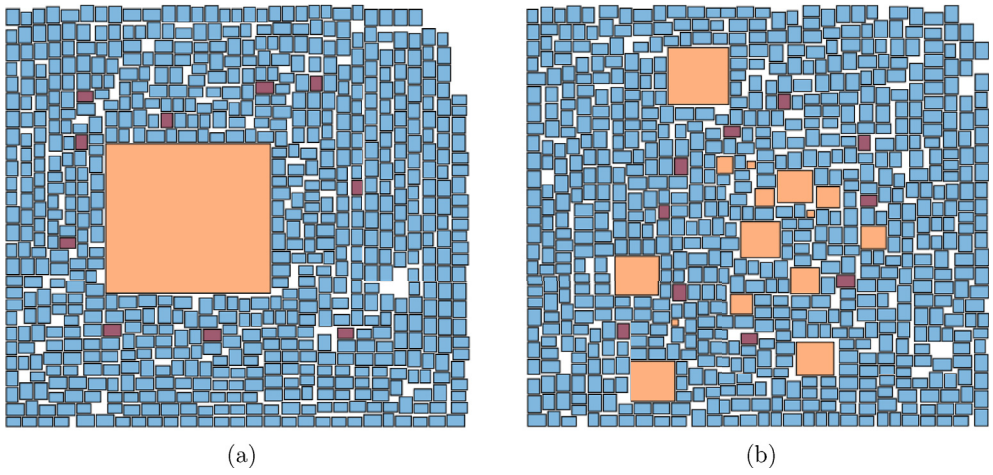

Fig. 8. Floorplan examples of the neural network b_1 generated by (a) "SC + NLP + MH" (area:1.09 mm^2); (b) "FC + NLP + MP" (area:1.02 mm^2).

work for NCS. The experimental results are listed in two columns of Table 4. "FC + NLP + ILP" represents that based on the clustering results generated by the proposed fault tolerance-aware clustering (FC, see Section 3.1), the memristive crossbar size is determined by the proposed non-linear programming (NLP, see Section 3.2), and then the proposed ILP model introduced in Section 3.3 is adopted to derive a connection-memristor mapping (ILP). "FC + NLP + MH" represents that instead of using the ILP model, the proposed matching-based heuristic algorithm presented in Section 3.4 is used to generate a connection-memristor mapping (MH). "Succ" refers to the successful mapping rate which is calculated by the ratio of the number of samples with valid mapping to the total size of samples. "Area" and "Wire" represent chip area and total half-perimeter wirelength overhead, respectively. "Util" lists the average utilization of all mapped crossbars. "PF" denotes the average power factor of all mapped crossbars. "RT" reports the computational time of the mapping process. "NA" represents that the fault tolerant mapping solution cannot be achieved within the time limit (1800s).

As shown in Table 4, "FC + NLP + ILP" can achieve 100% successful mapping rate on b_1–b_4, while "FC + NLP + MH" only obtains 100% successful mapping rate on b_1–b_3, which demonstrates the effectiveness of the proposed ILP model for fault tolerance. But since ILP is an NP-

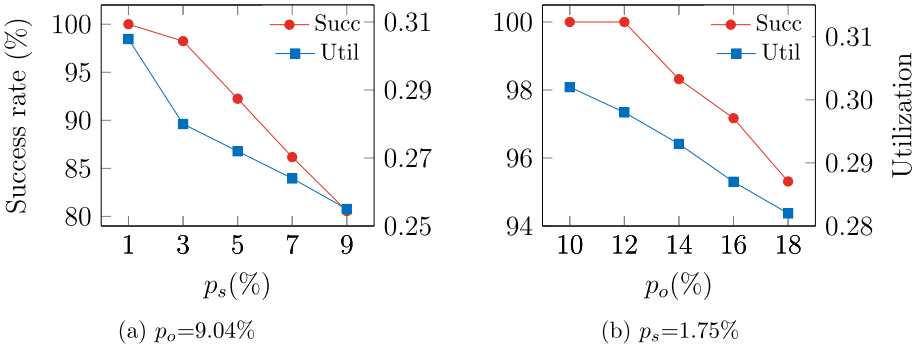

Fig. 9. Effect of the fault rate of SA1 and SA0 on performance (b_1 testbench).

hard problem, its runtime increases dramatically with the size of the neural network. As shown in Table 4, "FC + NLP + ILP" cannot generate a valid mapping solution on b_5–b_8 within the time limit, which indicates that for large weight matrices, ILP based method is very time consuming. On the other hand, our proposed heuristic method is very efficient. Note that since "FC + NLP + ILP" and "FC + NLP + MH" are based on the same clustering results and memristive crossbars, here the area, the wirelength, the hardware utilizations and the power factor are the same.

In addition, to verify the effectiveness of the proposed fault tolerance-aware hierarchical clustering, we define a comparison experiment that uses a single crossbar to implement the neural network (SC). Fig. 7 shows the successful mapping rate and average area cost of the two different methodologies of "SC + NLP + MH" and "FC + NLP + MH". As shown in the figure, the successful mapping rate of "FC + NLP + MH" methodology is quite larger than "SC + NLP + MH" method, which demonstrates the effectiveness of the proposed fault tolerance-aware clustering method. Besides, "FC + NLP + MH" methodology can also reduce the area of NCS. A floorplan of the neural network b_1 generated by the two methodologies of "SC + NLP + MH" and "FC + NLP + MH" are shown in Fig. 8. It can be seen that through consideration of sparse connection, "FC + NLP + MH" methodology can efficiently reduce floorplan area consumption.

Fig. 9 illustrates the trends of the successful mapping rate and utilizations of all crossbars with respect to the changing of p_o and p_s. In the experiment, we perform "FC + NLP + MH" on testbench b_1. When p_o is set to 9.04%, the successful mapping rate and utilizations of crossbars are varied with p_s, shown as in Fig. 9(a). We notice that as p_s increases, both the successful mapping rate and the utilization of crossbars are reduced. That is because, based on the constraint (2b), P_1^r drops with the increases of p_s. As a result, according to constraint (2d), large memristive crossbars are generated to ensure that the successful mapping rate P is larger than the target value P_t. Thus the utilizations of crossbars are decreased. The same trend can also be observed in Fig. 9(b), where p_s is set to 1.75% and the successful mapping rate and utilizations of crossbars are changed with p_o. In addition, in Fig. 9(a) and (b), it can be seen that SA0 fault has a greater impact on the mapping performance than SA1 fault.

4.2. Comparison with previous works

Here we compare three different fault tolerant frameworks for NCS, as listed in three columns of Table 5. In "AutoNCS" [2], an iterative spectral clustering is repeatedly performed to partition the sparse connections into dense clusters. However, the fault tolerance is not considered during the clustering. Based on the cluster results generated by "AutoNCS", we map the connection matrix of each cluster to a memristive crossbar with the same size, and the proposed matching-based heuristic algorithm is applied to derive a connection-memristor mapping. In "Hybrid" [11], a hybrid fault tolerant mapping method is presented to tolerate stuck-at-one faults. But the stuck-at-zero faults are not considered in the work. In order to compare with "Hybrid", we also implement the hybrid method to consider both SA0 and SA1 faults. Without the fault tolerance-aware clustering, we directly map the sparse connection matrix to a single memristive crossbar by the hybrid method. The mapped memristive crossbar is set to the same size as the sparse connection matrix.

In the first experiment, we compare the proposed fault tolerant framework with "AutoNCS" [2]. As shown in Table 5, compared with "AutoNCS", the proposed fault tolerant framework can improve the successful mapping rate of NCS by 36.12%, while the area, the wirelength overhead and the power factor are reduced by 6%, 6% and 4%, respectively. This means the crossbars generated by "AutoNCS" are larger than the proposed fault tolerant framework. Since the larger the crossbar, the huger solution space for connection-memristor mapping. As a result,

Table 5
Comparison with state-of-the-art.

Bench	AutoNCS [2]						Hybrid [11]						FC + NLP + MH					
	Succ (%)	Area (mm^2)	Wire (mm)	Util (%)	PF	RT (s)	Succ (%)	Area (mm^2)	Wire (mm)	Util (%)	PF	RT (s)	Succ (%)	Area (mm^2)	Wire (mm)	Util (%)	PF	RT (s)
b1	62.55	1.03	623.50	39.51	45.72	1.12	100.0	1.09	594.20	43.55	487.71	0.37	100.0	1.02	588.20	30.38	44.49	0.20
b2	61.00	1.02	574.83	36.17	41.63	1.80	100.0	1.05	703.53	39.64	444.00	0.38	100.0	0.98	550.04	30.70	40.38	0.34
b3	71.50	1.01	542.86	35.05	38.73	3.59	100.0	1.04	581.68	37.05	415.00	0.42	100.0	0.95	546.50	30.61	37.72	0.41
b4	100.0	0.23	63.74	36.09	17.45	0.01	94.00	0.25	63.31	42.55	120.00	0.10	96.25	0.22	62.21	28.92	17.05	0.08
b5	66.40	0.95	524.50	30.13	35.48	1.78	90.00	1.00	576.24	33.94	380.14	6.17	94.18	0.92	532.08	26.05	34.56	0.68
b6	34.94	0.97	463.30	26.87	57.50	1.01	41.18	1.02	425.08	30.87	678.86	9.52	90.32	0.89	390.22	22.58	56.37	0.79
b7	30.81	6.53	2352.75	27.33	3049.65	35.41	39.05	6.79	2423.33	31.16	87829.86	42.51	83.51	6.08	2181.00	23.51	2927.66	10.62
b8	28.27	5.84	2049.07	23.89	2078.06	36.39	37.52	6.02	2110.54	27.23	58455.71	40.17	80.17	5.47	1903.59	22.19	2015.72	10.97
average	56.93	2.20	899.32	31.88	670.53	10.14	75.22	2.28	934.74	35.75	18601.41	12.46	93.05	2.07	844.23	26.87	646.74	3.01
ratio	1.00	1.00	1.00	1.00	1.00	1.00	1.32	1.04	1.04	***1.12***	27.74	1.23	***1.63***	***0.94***	***0.94***	0.84	***0.96***	***0.30***

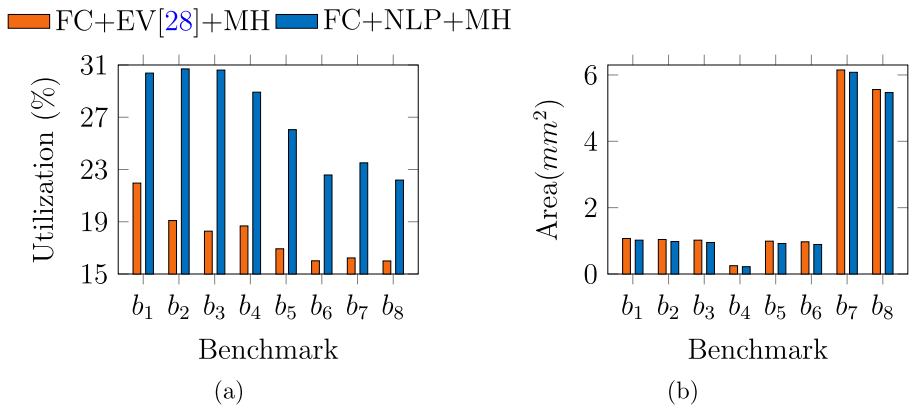

Fig. 10. Comparison between of "FC + EV [28]+MH" and "FC + NLP + MH" on (a) utilization and (b) area.

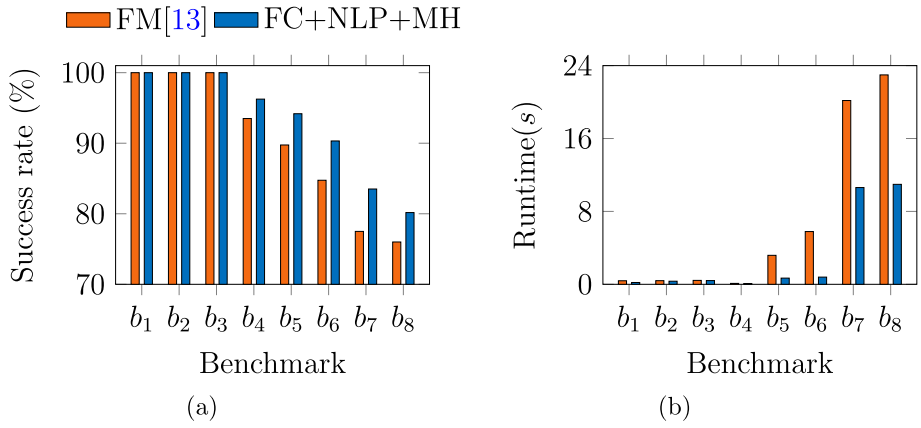

Fig. 11. Comparison between of "FM" [13] and "FC + NLP + MH" on (a) success rate and (b) runtime.

compared with "AutoNCS", the proposed fault tolerant framework can reduce the time of the matching process by 70%.

In the second experiment, we compare the proposed fault tolerant framework with "Hybrid" [11]. As shown in Table 5, compared with "Hybrid", the proposed fault tolerant framework can improve the successful mapping rate of NCS by 17.83%, while the area and the wirelength overhead are reduced by 10% and 10%, respectively. We also observe that "Hybrid" significantly increase the power factor. The reason is that, in "Hybrid", we directly map the sparse connection matrix to a single large memristive crossbar without the clustering. But in "AutoNCS" and "FC + NLP + MH", a set of crossbars are generated to implement the network, and thus the average power factor of crossbars are decreased. Besides, since the connection matrix is mapped to a single crossbar without redundant rows and columns in "Hybrid", the utilization of crossbars generated by the proposed fault tolerant framework is reduced.

In the third experiment, the performance of the proposed memristive crossbar configuration method is analyzed. "FC + EV [28]+MH" represents that based on the clustering results generated by the proposed fault tolerance-aware clustering (FC), the memristive crossbar size is set to an empirical value (EV, $M_c/M = N_c/N = 1.5$) [28], and then the proposed matching-based heuristic algorithm is adopted to derive a connection-memristor mapping (MH). We compare "FC + EV [28]+MH" with "FC + NLP + MH". Fig. 10 shows utilizations of all crossbars and the average area of the floorplan generated by the two methodologies. As shown in the figure, "FC + NLP + MP" methodology can reduce the area and increase the hardware utilization.

In the fourth experiment, we compare "FC + NLP + MH" framework with "FM" [13]. "FM" denotes that the sparse connection matrix is directly mapped to a single crossbar by the heuristic mapping method in Ref. [13], without the fault tolerance-aware clustering. The successful mapping rate and the runtime of the mapping process of the two frameworks are illustrated in Fig. 11. As shown in the figure, "FC + NLP + MH" can improve the successful mapping rate and reduce the mapping runtime. That is because, the fault tolerant mapping is just one stage of the proposed fault tolerant framework for memristive crossbar-based NCS, thus purely mapping algorithm itself is hard to achieve reasonable fault tolerance in NCS.

5. Conclusion

In this paper, we have proposed a synapse fault tolerant framework for memristive crossbar-based neuromorphic computing systems, with consideration of both hardware cost and successful mapping rate. Experimental results demonstrate that, compared with state of the arts, our proposed framework can effectively improve the successful mapping rate and reduce the hardware cost. Memristive crossbar provides an efficient hardware implementation for neuromorphic networks, thus we believe this paper will stimulate more research on fault tolerance design for memristive crossbar-based NCS.

Besides, in a memristor-based NCS, programming the resistance of the memristors suffer from variations. As a result, the actual programming resistance is deviated from its target resistance. In future, we plan to investigate a variation-aware framework to tolerate the stochastic resistance variation.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (NSFC) under grant No. 61904047, 61874156, 61574052, the Fundamental Research Funds for the Central Universities of China under grant No. JZ2019HGBZ0159, and The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.vlsi.2019.09.008.

References

[1] Y. Chen, H.H. Li, C. Wu, C. Song, S. Li, C. Min, H.-P. Cheng, W. Wen, X. Liu, Neuromorphic computing's yesterday, today, and tomorrowan evolutional view, Integr. VLSI J. 61 (2018) 49–61.
[2] W. Wen, C.-R. Wu, X. Hu, B. Liu, T.-Y. Ho, X. Li, Y. Chen, An EDA framework for large scale hybrid neuromorphic computing systems, in: ACM/IEEE Design Automation Conference (DAC), 2015, pp. 1–6.
[3] Q. Xu, S. Chen, B. Yu, F. Wu, Memristive crossbar mapping for neuromorphic computing systems on 3D IC, in: ACM Great Lakes Symposium on VLSI (GLSVLSI), 2018, pp. 451–454.
[4] S. Acciarito, A. Cristini, G. Susi, et al., Hardware design of lif with latency neuron model with memristive stdp synapses, Integr. VLSI J. 59 (2017) 81–89.
[5] Z. Li, C. Liu, Y. Wang, B. Yan, C. Yang, J. Yang, H. Li, An overview on memristor crossabr based neuromorphic circuit and architecture, in: IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SOC), 2015, pp. 52–56.
[6] C.-R. Wu, W. Wen, T.-Y. Ho, Y. Chen, Thermal optimization for memristor-based hybrid neuromorphic computing systems, in: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2016, pp. 274–279.
[7] J. Cui, Q. Qiu, Towards memristor based accelerator for sparse matrix vector multiplication, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2016, pp. 121–124.
[8] W. Huangfu, L. Xia, M. Cheng, X. Yin, T. Tang, B. Li, K. Chakrabarty, Y. Xie, Y. Wang, H. Yang, Computation-oriented fault-tolerance schemes for RRAM computing systems, in: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2017, pp. 794–799.
[9] L. Xia, M. Liu, X. Ning, K. Chakrabarty, Y. Wang, Fault-tolerant training with on-line fault detection for rram-based neural computing systems, in: ACM/IEEE Design Automation Conference (DAC), vol. 33, 2017.
[10] B. Yuan, B. Li, T. Weise, X. Yao, A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures, IEEE Trans. Evol. Comput. 18 (2014) 846–859.
[11] O. Tunali, M. Altun, Logic synthesis and defect tolerance for memristive crossbar arrays, in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2018, pp. 425–430.
[12] Y. Su, W. Rao, An integrated framework toward defect-tolerant logic implementation onto nanocrossbars, IEEE Trans. Comput. Aided Des. Integr Circuits Syst. 33 (2014) 64–75.
[13] O. Tunali, M. Altun, A fast logic mapping algorithm for multiple-type-defect tolerance in reconfigurable nano-crossbar arrays, IEEE Trans. Emerg. Top. Comput. (2017), https://doi.org/10.1109/TETC.2017.2755458.
[14] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, L. Jiang, Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar, in: IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE), 2017, pp. 19–24.
[15] C. Liu, M. Hu, J.P. Strachan, H. Li, Rescuing memristor-based neuromorphic design with high defects, in: ACM/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.
[16] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, V.S. Sathe, Energy-efficient neural network acceleration in the presence of bit-level memory errors, IEEE Trans. Circuits Syst. I (2018) 1–14.
[17] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, H. Yang, Stuck-at fault tolerance in RRAM computing systems, IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS) 8 (2018) 102–115.
[18] Y. Wang, W. Wen, L. Song, H.H. Li, Classification accuracy improvement for neuromorphic computing systems with one-level precision synapses, in: IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2017, pp. 776–781.
[19] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to 1 or -1, 2016. arXiv preprint arXiv:1602.02830.
[20] O. Tunali, M.C. Morgul, M. Altun, Defect-tolerant logic synthesis for memristor crossbars with performance evaluation, IEEE Micro 38 (2018) 22–31.
[21] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, et al., Efficient and self-adaptive in-situ learning in multilayer memristor neural networks, Nat. Commun. 9 (2018) 2385.
[22] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, F.T. Chen, RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme, IEEE Trans. Comput. (2014) 1.
[23] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, H. Yang, RRAM-based analog approximate computing, IEEE Trans. Comput. Aided Des. Integr Circuits Syst. 34 (2015) 1905–1917.
[24] S. Kannan, J. Rajendran, R. Karri, O. Sinanoglu, Sneak-path testing of crossbar-based nonvolatile random access memories, IEEE Trans. Nanotechnol. 12 (2013) 413–426.
[25] S. Chen, T. Yoshimura, Fixed-outline floorplanning: enumerating block positions and a new objective function for calculating area costs, IEEE Trans. Comput. Aided Des. Integr Circuits Syst. 27 (2008) 858–871.
[26] R.C. Dubes, A.K. Jain, Algorithms for Clustering Data, Prentice Hall Englewood Cliffs, New Jersey, 1988.
[27] S. Salvador, P. Chan, Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. IEEE International Conference on Tools with Artificial Intelligence, 2004, pp. 576–584.
[28] O. Tunali, M. Altun, Permanent and transient fault tolerance for reconfigurable nano-crossbar arrays, IEEE Trans. Comput. Aided Des. Integr Circuits Syst. 36 (2017) 747–760.
[29] O. Tunali, M. Altun, A survey of fault-tolerance algorithms for reconfigurable nano-crossbar arrays, ACM Comput. Surv. 50 (2018) 79.
[30] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to NP-Completeness, WH Freeman and Company, San Francisco, 1979.
[31] J. Munkres, Algorithms for the assignment and transportation problems, J. Soc. Ind. Appl. Math. 5 (1957) 32–38.
[32] Gurobi Optimization Inc., Gurobi optimizer reference manual. http://www.gurobi.com, 2014.
[33] B. Liu, Y. Chen, B. Wysocki, T. Huang, The circuit realization of a neuromorphic computing system with memristor-based synapse design, in: Springer International Conference on Neural Information Processing (ICONIP), 2012, pp. 357–365.
[34] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Conference on Neural Information Processing Systems (NIPS), 2012, pp. 1097–1105.