Memristive Crossbar Mapping for Neuromorphic Computing Systems on 3D IC

QI XU, Hefei University of Technology, China
HAO GENG, The Chinese University of Hong Kong, Hong Kong
SONG CHEN, University of Science and Technology of China, China
BEI YU, The Chinese University of Hong Kong, Hong Kong
FENG WU, University of Science and Technology of China, China

In recent years, neuromorphic computing systems based on memristive crossbar have provided a promising solution to enable acceleration of neural networks. However, most of the neural networks used in realistic applications are often sparse. If such sparse neural network is directly implemented on a single memristive crossbar, then it would result in inefficient hardware realizations. In this work, we propose E3D-FNC, an enhanced three-dimesnional (3D) floorplanning framework for neuromorphic computing systems, in which the neuron clustering and the layer assignment are considered interactively. First, in each iteration, hierarchical clustering partitions neurons into a set of clusters under the guidance of the proposed distance metric. The optimal number of clusters is determined by L-method. Then matrix re-ordering is proposed to re-arrange the columns of the weight matrix in each cluster. As a result, the reordered connection matrix can be easily mapped into a set of crossbars with high utilizations. Next, since the clustering results will in turn affect the floorplan, we perform the floorplanning of neurons and crossbars again. All the proposed methodologies are embedded in an iterative framework to improve the quality of NCS design. Finally, a 3D floorplan of neuromorphic computing systems is generated. Experimental results show that E3D-FNC can achieve highly hardware-efficient designs compared to the state of the art.

CCS Concepts: • **Hardware → Emerging architectures**; **Physical design (EDA)**; **3D integrated circuits**;

Additional Key Words and Phrases: Neuromorphic computing, memristive crossbar, 3D floorplanning, hierarchical clustering

ACM Reference format:
Qi Xu, Hao Geng, Song Chen, Bei Yu, and Feng Wu. 2019. Memristive Crossbar Mapping for Neuromorphic Computing Systems on 3D IC. *ACM Trans. Des. Autom. Electron. Syst.* 25, 1, Article 8 (November 2019), 19 pages.
https://doi.org/10.1145/3365576

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants No. 61904047, No. 61874102, and No. 61732020; the Fundamental Research Funds for the Central Universities under Grants No. JZ2019HGBZ0159 and No. WK2100000005; the Beijng Municipal Science & Technology Program under Grant No. Z181100008918013; and The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

Authors' addresses: Q. Xu, School of Electronic Science and Applied Physics, Hefei University of Technology, Hefei, China; email: xuqi@hfut.edu.cn; H. Geng and B. Yu, Department of Computer Scinece and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong; emails: {hgeng, byu}@cse.cuhk.edu.hk; S. Chen and F. Wu, Department of Electronic Science and Technology, University of Science and Technology of China, Hefei, China; emails: {songch, fengwu}@ustc.edu.cn.

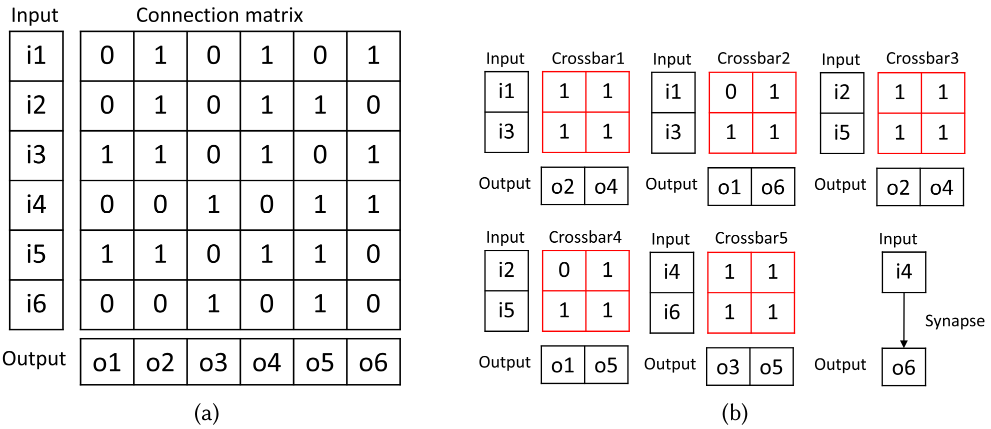
1084-4309/2019/11-ART8 $15.00
https://doi.org/10.1145/3365576

1 INTRODUCTION

Neuromorphic computing systems (NCS) based on hardware designs intend to mimic neurobiological architectures [12]. 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. However, CMOS technology implementation has been shown to suffer from mismatch between NCS building blocks (neuron and synapse) and CMOS primitives (Boolean logic) [7]. Consequently, the conventional CMOS realization of synapse functionality requires a large number of transistors to mimic a single synapse [2]. To address the problem, many neuromorphic designs on device and architecture level have been explored. For example, the emerging memristive technology is adopted to implement synapse circuit due to the similarity between the memristive and the synaptic behaviors [1]. Besides, memristive crossbar has been proven as one of the most efficient nanostructures that carry out matrix-vector multiplications while the hardware cost and the computation energy are significantly reduced compared with the state-of-the-art CMOS design [12]. Despite of these tremendous advantages, NCS implementation on memristive crossbars also encounter some design challenges.

In realistic applications, the size of neural networks is often very large. For instance, AlexNet proposed by Krizhevsky et al. [18] in 2012 contains $650K$ neurons and $60M$ synapses. Most of large neural networks are sparse. In low-density parity check (LDPC) coding based on a message passing algorithm, the network sparsity is higher than 99% [19]. If a sparse neural network is directly implemented on a memristive crossbar, then the crossbar utilization can be low, resulting in highly area-inefficient designs. Xia et al. [27] proposed an architectural simulation platform to evaluate memristive crossbar architectures. Ankit et al. [6] developed a reconfigurable architecture with memristive crossbars for spiking neural network. However, the sparsity of the neural network is not considered in the two works. Thus mapping a sparse neural network to one memristive crossbar will produce high hardware consumption. To overcome the issue, many previous works have been proposed to enable the efficient realization on memristive crossbars. Ankit et al. [7] presented a training framework that transforms deep neural networks to crossbar friendly architectures. Wen et al. [25] proposed a design automation framework for large-scale hybrid neuromorphic computing systems. Wu et al. [26] developed a thermal optimization for memristor-based hybrid neuromorphic computing systems. In References [25, 26], an iterative spectral clustering is performed to group the connections into memristive crossbars. In each iteration, all the already-clustered connections will be removed first, and then the spectral clustering is applied again to cluster the remaining connections of the network. The process will be terminated until the majority of the weight connections are partitioned into clusters. Cui et al. [13] proposed a sparse matrix re-ordering method, which uses row and column permutation matrices to group connections into crossbars. However, crossbar utilization is not considered, and thus memristive crossbars with low utilizations may be generated. A sparse neural network mapping method based on k-means clustering is presented to achieve high crossbar utilizations in Reference [20]. However, the physical design is not implemented.

To achieve increasingly demanding computational tasks, NCS with high complexity and high connectivity is required. But traditional two-dimensional (2D) integration may not meet these requirements, as longer signal transmission distances are introduced due to a large number of connections in 2D integration. Three-dimensional integrated circuits (3D ICs) involve vertically stacking multiple dies connected by through silicon vias (TSVs), providing a promising way to enable high density and high computation speed in NCS [16]. Besides, 3D design in NCS can emulate real biophysical processing in the human brain [4]. In References [25, 26], a 2D placement is



Input	Connection matrix					
i1	0	1	0	1	0	1
i2	0	1	0	1	1	0
i3	1	1	0	1	0	1
i4	0	0	1	0	1	1
i5	1	1	0	1	1	0
i6	0	0	1	0	1	0
Output	o1	o2	o3	o4	o5	o6

(a)

Input	Crossbar1	
i1	1	1
i3	1	1
Output	o2	o4

Input	Crossbar2	
i1	0	1
i3	1	1
Output	o1	o6

Input	Crossbar3	
i2	1	1
i5	1	1
Output	o2	o4

Input	Crossbar4	
i2	0	1
i5	1	1
Output	o1	o5

Input	Crossbar5	
i4	1	1
i6	1	1
Output	o3	o5

(b)

Fig. 1. (a) Connection matrix $\boldsymbol{W}$ of a feed-forward neural network with six pre-synaptic neurons $\{i_1, \ldots, i_6\}$ and six post-synaptic neurons $\{o_1, \ldots, o_6\}$; (b) the connections are mapped to five memristive crossbars with high utilizations and one discrete synapse.

implemented to estimate the NCS hardware cost. Although the 2D NCS architecture may not be equivalent to 3D NCS design, the idea of partitioning the sparse connections into a set of memristive crossbars can be leveraged.

Recently, several 3D NCS designs have been explored. An et al. [5] proposed a 3D NCS combining monolithic 3D integration and vertical resistive random-access memory technology. Ehsan et al. [15] presented a 3D NCS architecture by using redundant TSVs to supply the neuronal membrane capacitance. However, these works only focused on electrical modeling and analysis of 3D NCS without considering the constraints derived from physical synthesis stage. In the conference version [29], a 3D floorplanning framework for neuromorphic computing systems (3D-FNC), is presented in consideration of both crossbar utilization and design cost. However, the clustering and the 3D floorplanning are considered separately. In this article, we argue that taking account of weight clustering and floorplanning in 3D design interactively will benefit the 3D NCS design. That is because layer results of neurons influence the clustering results of weights between neurons, which in turn affect the final floorplan. In accordance with this argument, we propose E3D-FNC, an enhanced 3D floorplanning framework for neuromorphic computing systems, in which the neuron clustering and the floorplanning are considered interactively. Figure 1 presents an example to illustrate how the proposed framework can effectively map the connections into crossbars with high utilizations. As shown in Figure 1(a), if the sparse connection matrix is directly mapped to a crossbar, then the utilization of the crossbar is low. However, through several techniques in our framework (e.g., hierarchical clustering, matrix re-ordering), the mapped crossbars with high utilizations are generated as shown in Figure 1(b). The corresponding 3D floorplan example is shown in Figure 2(b). Key technical contributions of this work are listed as follows.

- An iterative 3D floorplanning framework for NCS is proposed, in which the neuron clustering and the layer assignment of neurons and crossbars are considered interactively.
- A hierarchical clustering based method is proposed to partition weight connections of a sparse neural network into a set of clusters. To enhance the utilization of the mapped memristive crossbars, the pre-synaptic neurons connecting more common post-synaptic neurons are partitioned into the same cluster, under the guidance of the proposed distance metric used in clustering. Besides, to reduce the number of TSVs, the proposed distance metric also forces neurons located at the same layer to be grouped into the same cluster.

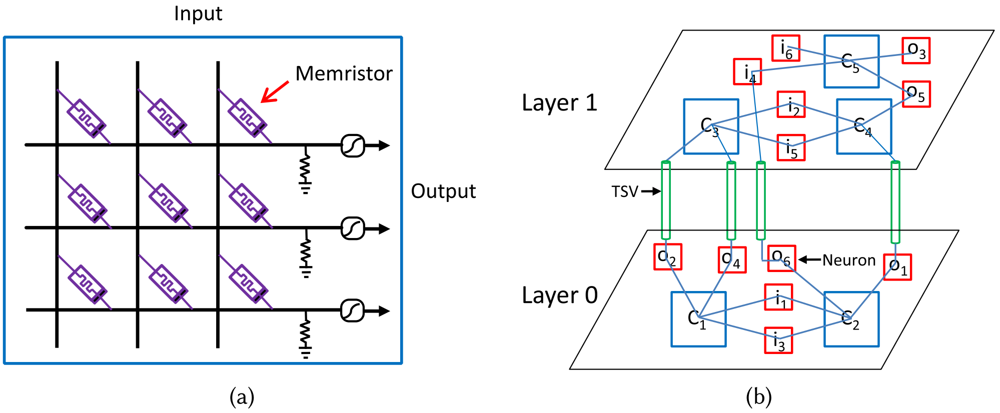


Fig. 2. (a) The structure of a memristive crossbar block; (b) 3D floorplan example of the mapping results of the neural network in Figure 1(a).

- An L-method with a post-processing step is developed to determine the optimal number of clusters.
- Matrix re-ordering is proposed to re-arrange the connection matrix of each cluster. The reordered connection matrix can be easily mapped into suitable sized memristive crossbars with high utilizations or discrete memristors.
- Experimental results show that, compared with state of the art, the proposed framework can enhance the crossbars utilization and reduce the hardware cost.

The remainder of this article is organized as follows. Section 2 presents preliminaries and gives the problem formulation. Section 3 describes the proposed E3D-FNC framework in details. Section 4 lists experimental results, followed by conclusion in Section 5.

2 PRELIMINARIES

2.1 Memristive Crossbar

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 the synapses [21]. The synapses will apply different weights on the information during transmission, which can be expressed as $\boldsymbol{O} = \boldsymbol{W} \cdot \boldsymbol{I}$. An element w_{ij} in the weight matrix $\boldsymbol{W}$ represents the weight of a synapse between the pre-synaptic neuron i in $\boldsymbol{I}$ and the post-synaptic neuron j in $\boldsymbol{O}$. The synaptic weight matrix $\boldsymbol{W}$ is represented as (1, 0)-matrix, where "1" indicates a connection exists between two corresponding neurons and "0" vice versa. Note that "synaptic weight matrix" and "connection matrix" are interchangeable in this article. Since the resistance of memristor can be programmed by applying current or voltage, the memristor is suitable for storing the synapse weight. Therefore, a memristor is used to link the pre-synaptic neuron and the post-synaptic neuron in the crossbar structure. An example of memristive crossbar structure is shown in Figure 2(a). Figure 2(b) shows a two-layer floorplan example of a neural network based on a memristive crossbar, where six input neuron blocks $\{i_1, i_2, i_3, i_4, i_5, i_6\}$ connect to six output neuron blocks $\{o_1, o_2, o_3, o_4, o_5, o_6\}$ through five memristive crossbar blocks.

2.2 Problem Definition

In this work, 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. Specifically, the

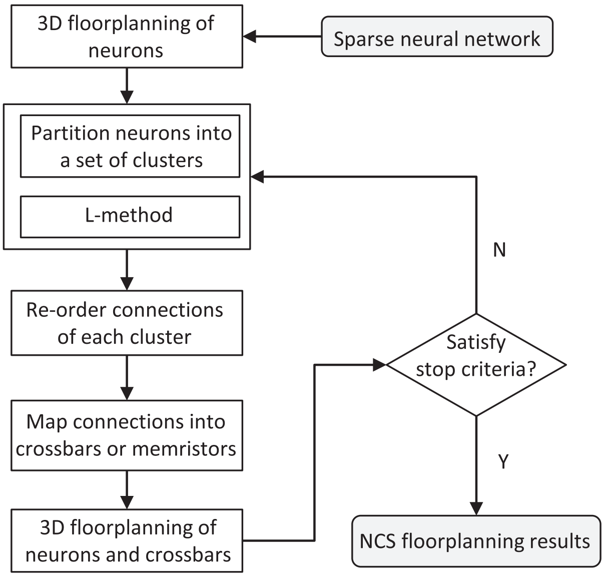

3D floorplanning of neurons
Sparse neural network
Partition neurons into a set of clusters
L-method
N
Re-order connections of each cluster
Satisfy stop criteria?
Map connections into crossbars or memristors
Y
3D floorplanning of neurons and crossbars
NCS floorplanning results

Fig. 3. The flow of E3D-FNC.

higher the better. Besides the crossbar utilization, we also exploit other three metrics, area, wirelength, and the number of TSVs of the 3D floorplan, to quantify the simulation results. We define the enhanced 3D floorplanning for neuromorphic computing systems (E3D-FNC) as follows:

Input: A sparse neural network $\boldsymbol{W}$.

Output: A 3D floorplan for neuromorphic computing systems, including a set of memristive crossbars for the synapses in the sparse network to be mapped to.

Objective: Enhancing crossbars utilization and simultaneously reducing chip area, wirelength, and TSV numbers.

3 E3D-FNC FRAMEWORK

Figure 3 illustrates the overall flow of the proposed E3D-FNC framework, which mainly consists of five stages. Given a sparse neural network, since layers of neurons will influence the clustering result, we first perform the neuron floorplanning to determine layers of the neurons. Then the hierarchical clustering partitions neurons into a set of clusters. The optimal number of clusters is determined by the proposed L-method. To enhance the utilization of the mapped memristive crossbars, the proposed distance metric used in hierarchical clustering forces the input neurons connecting more common output neurons to be partitioned into the same cluster. In addition, to reduce the number of TSVs, neurons located at the same layer are prone to be grouped into the same cluster. Then based on the clustering result of neurons, matrix re-ordering is to re-arrange the connection matrix in each cluster so that the reordered connection matrix is gathered. As a result, the gathered connections can be efficiently mapped to a set of memristive crossbars. Because the clustering results will also in turn affect the floorplan, afterwards we perform the floorplanning of neurons and crossbars again. If the hardware cost of the floorplan is not improved under fixed number of iterations, then the flow will be terminated. Otherwise, the hierarchical clustering is re-performed based on the current layer information of neurons. Finally E3D-FNC results with high crossbars utilization and low hardware cost are generated. For convenience, some notations used in this section are listed in Table 1.

Table 1. Notations Used in This Section

$\boldsymbol{W}$	Connection matrix of a sparse neural network
n, m	Number of rows (input neurons) and columns (output neurons) of $\boldsymbol{W}$
$\boldsymbol{W}_i$	Connection matrix of cluster i
$tier\_no$	The total number of device layers
$\hat{t}$	The optimal number of clusters
$\boldsymbol{P}$	Column permutation matrix for $\boldsymbol{W}_i$
$\boldsymbol{W}'_i$	Reordered connection matrix of cluster i
λ	Aspect ratio of a floorplan

ALGORITHM 1: Hierarchical clustering to cluster neurons

Input: Connection matrix $\boldsymbol{W}$, layer information of all neurons.
Output: A dendrogram of neurons.

```
1: Assign each neuron to a cluster;
2: Calculate distances among clusters;                                   ▷ Equation (1)
3: for $i \leftarrow 1$ to $n-1$ do
4:     Merge two clusters $\{r\}$ and $\{s\}$ with the shortest distance;
5:     $d_{\{k\},\{r,s\}} \leftarrow \min[d_{\{k\},\{r\}}, d_{\{k\},\{s\}}]$;   ▷ Update distances between $\{r,s\}$ and other clusters $\{k\}$
6: end for
7: All neurons are clustered into one cluster;
```

3.1 Hierarchical Clustering

In real applications, large neural networks are often sparse. If a sparse neural network is directly implemented on a memristive crossbar, then the crossbar utilization can be low, resulting in highly area-inefficient designs. In this work, the hierarchical clustering is adopted to partition sparse connections into a set of dense clusters. Hierarchical clustering generates clusters in a bottom-up iterative manner [14]. In every iteration, two clusters with the shortest distance are merged. Implication of "distance" varies with specific applications in practice. The iterative merging is repeated until all data points are formed into one cluster. In our framework, we propose a distance metric between two input neurons, i_p and i_q ($1 \le p, q \le n$), as follows:

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

where n_{11} represents the number of output neurons simultaneously connecting to i_p and i_q, n_{10} and n_{01} count the number of output neurons only connecting to i_p or i_q, and $tier(i_p)$ and $tier(i_q)$ describe the layer where i_p and i_q are located. In each iteration of hierarchical clustering, two groups with the shortest distance are merged. The first part of Equation (1) forces input neurons connecting more common output neurons to be clustered. As a result, the generated clusters can be easily mapped to memristive crossbars with high utilizations. The second part of Equation (1) also guides neurons located at the same layer to be partitioned into the same cluster, which shows the proposed clustering method can reduce the number of TSVs.

The flow of hierarchical clustering is shown in Algorithm 1. Starting from n data points (input neurons), hierarchical clustering first treats each point as a single cluster (line 1). Then the iterative merging steps are performed n-1 times and eventually build a dendrogram (cluster tree) (lines 3–7). During the clustering, the data points that are close will be merged first, and those that are far

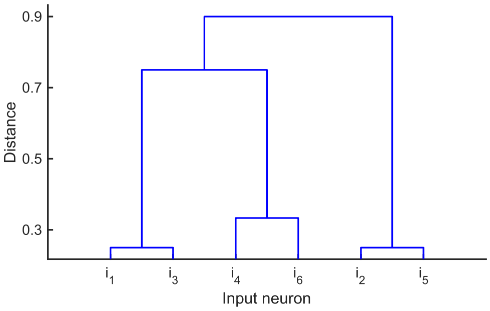


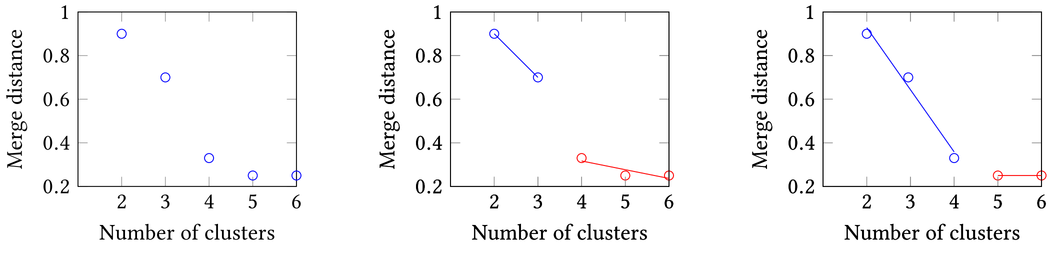
Fig. 4. A hierarchical clustering dendrogram.

away will not be merged until the late iterations [3]. Figure 4 is used to illustrate the process of hierarchical clustering. For the connection matrix $\boldsymbol{W}$ of a feed-forward neural network shown in Figure 1(a), it contains six pre-synaptic neurons and six post-synaptic neurons. In the floorplanning stage, the layers of neurons have been determined, an example is shown in Figure 2(b). When hierarchical clustering performs, according to the distance calculated by Equation (1), the first iteration step merges the pre-synaptic neurons i_1 and i_3, because the distance between them is the smallest (locate at the same layer, and connect three common post-synaptic neurons o_2, o_4, and o_6). Then the second iteration step groups the input neurons i_2 with i_5, and i_4 is merged with i_6 in the third iteration step. Next, the input neuron $\{i_1,\ i_3\}$ is clustered with $\{i_4,\ i_6\}$ in the fourth iteration step. Finally, the two clusters containing $\{i_1,\ i_3,\ i_4,\ i_6\}$ and $\{i_2,\ i_5\}$, respectively, are merged in the last iteration step.

3.2 Optimal Number of Clusters

Although the hierarchical clustering can generate a cluster tree, however, if the number of clusters is not given, the hierarchical clustering cannot build the final clustering result. In this work, the L-method is proposed to find the optimal number of clusters. We summarize the major steps of the L-method for selection of the optimal number of clusters in Algorithm 2. The L-method is derived from the fact that for many clustering algorithms, it is possible to plot an evaluation graph, where the x-axis is the number of clusters and the y-axis is the evaluation metric used by the clustering algorithm. For the hierarchical clustering, the y-axis values are the merge distance between the last two merged clusters at x clusters [30]. The evaluation graph has three distinctive regions: a sharply-sloping region to the left, a curved transition area in the middle, and a flat region to the right. Note that the sharp transition point on the evaluation graph is considered to be the optimal number of clusters [22]. Since the hierarchical clustering merges a pair of clusters in every iteration, the evaluation graph can be produced by running the clustering algorithm only once (line 1).

To illustrate how the L-method works, we use the connection matrix shown in Figure 1(a) as an example. Based on the dendrogram generated by hierarchical clustering, we plot the evaluation graph in Figure 5(a), where the x-axis shows the number of clusters and the y-axis represents the merge distance d calculated by Equation (1). The L-method tries to determine the optimal number by searching for two lines that can best-fit the evaluation graph, and the intersection of these two lines represent the transition point of the graph. In Figure 5(b), the two lines can accurately fit the graph where one line fits the data in the interval $x \in [2, 3]$ and the other line fits the data in the interval $x \in [4, 6]$. It can be noticed that the sharp transition point is located at $x = 3$, meaning that the neurons should be grouped into three clusters.



(a) (b) (c)

Fig. 5. (a) The evaluation graph of the connection matrix shown in Figure 1(a); two best-fit lines are obtained at the transition point (b) $x = 3$ and (c) $x = 4$ by the L-method.

ALGORITHM 2: L-method for selection of cluster numbers

Input: A cluster tree generated by the hierarchical clustering.

Output: The optimal number of clusters $\hat{t}$.

```
1: Construct the evaluation graph based on the cluster tree;
2: for $t \leftarrow 3$ to $n - 2$ do
3:     Solve the regression problem with the transition point $x = t$;     ▷ Equation (2)
4:     Calculate $\text{RMSE}_t$;                                          ▷ Equation (3)
5: end for
6: Determine $\hat{t}$;                                                   ▷ Equations (4) and (5)
```

Mathematically, the L-method can be formulated as the regression problem [3] (lines 2–5). Considering the evaluation graph where the number of clusters varies from 2 to n (n-1 data points), and the data points are partitioned into a left region and a right region at transition point $x = t$. The left region has points with $x \in [2, \ldots, t]$, and the right region has points with $x \in [t + 1, \ldots, n]$. To ensure each region contains at least two points, the value of t ranges from 3 to n-2. Then the regression problem is solved respectively to achieve two best-fit lines for the left and the right regions as follows:

$$\begin{aligned} \text{RMSE}_l &= \min_{s_l, b_l} \|\boldsymbol{d}_l - s_l \cdot \boldsymbol{x}_l - b_l \cdot \mathbf{1}\|_2, \\ \text{RMSE}_r &= \min_{s_r, b_r} \|\boldsymbol{d}_r - s_r \cdot \boldsymbol{x}_r - b_r \cdot \mathbf{1}\|_2, \\ \text{where} & \\ \boldsymbol{x}_l &= (2, 3, \ldots, t)^\top, \quad \boldsymbol{x}_r = (t + 1, t + 2, \ldots, n)^\top, \end{aligned} \tag{2}$$

where $\boldsymbol{d}_l$ and $\boldsymbol{d}_r$ are the merge distance values for left region $\boldsymbol{x}_l$ and right region $\boldsymbol{x}_r$. (s_l, b_l) and (s_r, b_r) represent the slope and bias for left-fit and right-fit lines. RMSE_l and RMSE_r are the root mean square error of the two best-fit lines. Then the total root mean square error (RMSE_t) is defines as follows:

$$\text{RMSE}_t = \frac{t - 1}{n - 1} \cdot \text{RMSE}_l + \frac{n - t}{n - 1} \cdot \text{RMSE}_r. \tag{3}$$

We solve the regression problems for all values of t, and the transition point that achieves the minimum RMSE_t is assumed as the optimal number of clusters (line 6):

$$\hat{t} = \operatorname*{argmin}_{t \in [3, \ldots, n-2]} \text{RMSE}_t. \tag{4}$$

Although the L-method tries to match the human intuition in finding the transition point of the evaluation graph, it can result in an unexpected answer in some cases. For example, Figure 5(b)

ALGORITHM 3: Matrix re-ordering to map the weight connections into crossbars or synapses

```
Input: Connection matrix $\boldsymbol{W}_i$ of cluster $i$ ($1 \leq i \leq \hat{t}$).
Output: The memristive crossbars or synapses.
1: Permute columns of the original connection matrix $\boldsymbol{W}_i$;
2: Map the reordered connection matrix $\boldsymbol{W}_i'$ into a set of groups $G$ with suitable sizes;
3: for $g \in G$ do
4:     if the utilization of group $g > 0.4$ then
5:         The connections in group $g$ are mapped to a suitable sized memristive crossbar;
6:     else
7:         The connections in group $g$ are mapped to synapses.
8:     end if
9: end for
```

and Figure 5(c) show the results fitted by setting $t = 3$ and $t = 4$, respectively. It can be noticed that both fitting curves achieve small error. Thus the choice of $t = 3$ or $t = 4$ by the L-method is arbitrary in this example. To address the issue, a post-processing step is added to check if a sharp transition occurs at the point t or $t + 1$ [30]. That is, the second-order difference for the evaluation graph in the logarithmic domain is calculated as:

$$s(t) = [\log(\boldsymbol{d}_{t+1}) - \log(\boldsymbol{d}_t)] - [\log(\boldsymbol{d}_t) - \log(\boldsymbol{d}_{t-1})], \tag{5}$$

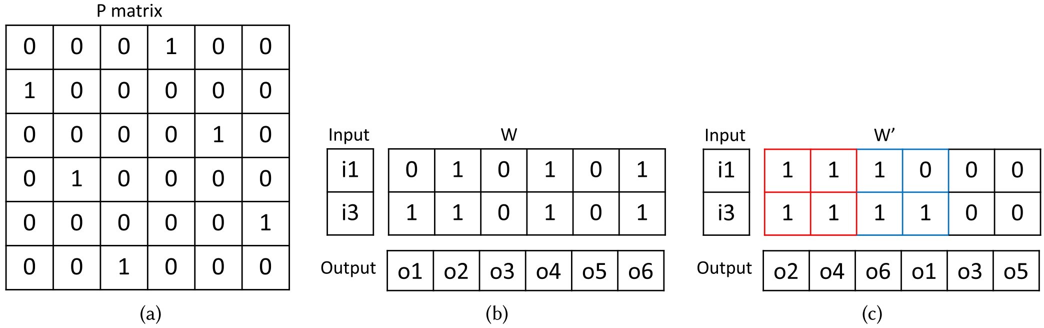
where $\boldsymbol{d}_t$ denotes the merge distance value at point t. A large second-order difference means an abrupt transition in the evaluation graph. Therefore, after obtaining $\hat{t}$ from the L-method, $s(\hat{t})$ and $s(\hat{t} + 1)$ are compared to accurately determine the number of clusters.

After the L-method with the post-processing step determines the number of clusters, the hierarchical clustering builds the final clustering results of neurons. For example in Figure 5, the optimal number of clusters determined by L-method equals to 3, and the absolute value of $s(3)$ is larger than $s(4)$. As a result, the neurons should be partitioned into 3 clusters. The three clusters are $\{i_1, i_3\}$, $\{i_2, i_5\}$, and $\{i_4, i_6\}$, respectively.

3.3 Matrix Re-ordering

Due to the reliability issue, the size of current crossbars cannot be larger than 64 × 64 [25]. However, since the size of the weight matrix in each cluster may violate the size limitation, the cluster cannot be directly mapped to a memristive crossbar. In addition, considering the hardware utilization, it is impossible to partition all connections into memristive crossbars, especially for the neural networks with randomly distributed connections. In this case, the discrete memristors are more efficient for realizing the connections. We recognize the outliers as the connections belonging to none of the crossbars. In this work, the matrix re-ordering method is proposed to map the weight connections of each cluster into crossbars or discrete memristors. First, the connection matrix of each cluster will be re-arranged so that the reordered connection matrix is gathered. Then we map the gathered connections to a set of suitable sized memristive crossbars with high utilizations of crossbars considered. The flow of matrix re-ordering is shown in Algorithm 3.

Given a connection matrix $\boldsymbol{W}_i$ of cluster i, the objective is to find a convenient column permutation matrix $\boldsymbol{P}$ subject to $\boldsymbol{W}_i' = \boldsymbol{W}_i \cdot \boldsymbol{P}$. The reordered connection matrix $\boldsymbol{W}_i'$ can be partitioned into several groups (lines 1 and 2). If the utilization of a formed group is larger than a predefined threshold u_t, then the weight connections in the group will be efficiently mapped to a suitable sized memristive crossbar. Otherwise, the connections (outliers) should be mapped to discrete synapses instead of memristive crossbar (lines 3–7). As a result, a set of suitable sized memristive crossbars



P matrix

0	0	0	1	0	0
1	0	0	0	0	0
0	0	0	0	1	0
0	1	0	0	0	0
0	0	0	0	0	1
0	0	1	0	0	0

(a)

Input	W					
i1	0	1	0	1	0	1
i3	1	1	0	1	0	1
Output	o1	o2	o3	o4	o5	o6

(b)

Input	W'					
i1	1	1	1	0	0	0
i3	1	1	1	1	0	0
Output	o2	o4	o6	o1	o3	o5

(c)

Fig. 6. (a) The column permutation matrix $\boldsymbol{P}$. (b) The original matrix $\boldsymbol{W}_1$ of cluster 1. (c) The reordered matrix $\boldsymbol{W}_1'$.

that have high utilizations, and several discrete memristors are generated. The utilization threshold of the memristive crossbar is set to 0.4 by the experimental results shown in Section 4.2.

We give an example to illustrate the process of matrix re-ordering. Given the cluster $\{i_1, i_3\}$, the original connection matrix $\boldsymbol{W}_1$ is shown in Figure 6(b). By applying the column permutation matrix $\boldsymbol{P}$ shown in Figure 6(a), a reordered matrix $\boldsymbol{W}_1'$ is obtained as shown in Figure 6(c). It can be noticed that the reordered connection matrix $\boldsymbol{W}_1'$ can be efficiently mapped to two memristive crossbars with high utilizations. The corresponding two memristive crossbars are shown in Figure 6(c) with the red and blue boxes.

3.4 Physical Implementation

The hardware cost of E3D-FNC is estimated based on floorplan area (Ac), wirelength (Wire) and TSV numbers (Via). In the floorplanning stage, the neurons, the memristive crossbars and the discrete memristors are considered as blocks. The neuron blocks connect to the crossbar blocks or the discrete memristor blocks through wires. The partitioned sequence pair (P-SP) [24] is used to represent multi-layer 3D floorplans. A fixed-outline multi-layer floorplanner (IAR-MLFP) [11] is adopted to conduct the floorplanning. In IAR-MLFP, an insertion-after-remove method is presented to perturb solutions during floorplanning, which can greatly accelerate searching-based algorithms, since many solutions that fail to meet the fixed-outline constraints are skipped. Each time we remove a block from the multi-layer floorplan randomly, and a proper position including the coordinates and the appropriate layer will be selected for the removed block. As a result, the neuron clustering and the layer assignment can be effectively updated.

The cost function of a floorplan is defined as follows:

$$C_f = \alpha \cdot Ac + \beta \cdot Wire + \gamma \cdot Via, \tag{6}$$

where α, β, and γ are user-defined parameters to determine the importance of chip area, wirelength and TSV numbers. Because of the difference scales of the three objectives, it is difficult to choose the three parameters. Instead of evaluating solutions solely, we adopt a method of calculating the improvement percentage of one solution (S_1) to another (S_2) to compare two solutions [10].

The chip area Ac is evaluated by Equation (7), which takes the fixed-outline constraint into account,

$$E_W + E_H \cdot \lambda + C_1 \cdot \max\{E_W, E_H \cdot \lambda\} + C_2 \cdot \max\{W, H \cdot \lambda\}, \tag{7}$$

where W and H are the maximum width and height of all layers, $E_W = max\{W - W_0, 0\}$ and $E_H = max\{H - H_0, 0\}$ are the excessive width and height of the floorplan when considering the fixed-outline constraint, λ is the aspect ratio of the floorplan, and C_1 and C_2 are user-defined constants.

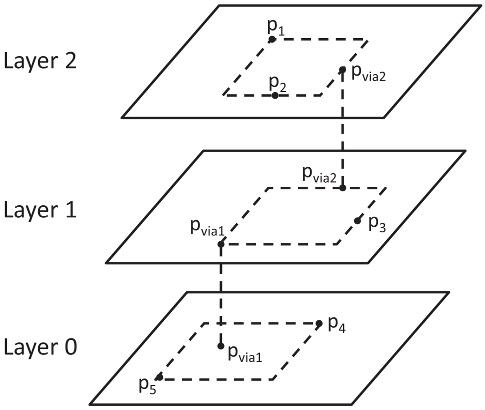


Fig. 7. Decomposition of a net spanning multiple layers.

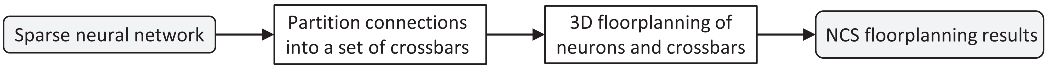

Sparse neural network
Partition connections into a set of crossbars
3D floorplanning of neurons and crossbars
NCS floorplanning results

Fig. 8. The flow of 3D-FNC [29].

Following the experimental configuration in Reference [28], $C_1 = 3$ and $C_2 = 1/16$. Using this equation, we not only penalize greater excessive width and excessive height but keep the area item effective when combined with other objectives.

The wirelength $Wire$ is estimated by the half perimeter wirelength (HPWL) model. In this work, we adopt the calculation model in Reference [9] to decompose the nets spanning multiple device layers into sub-nets, one on each device layer, by introducing dummy pins corresponding to TSVs. For example, as depicted in Figure 7, a net includes pins $\{p_1, p_2, p_3, p_4, p_5\}$, where p_1 and p_2 are on device layer 2, p_3 on layer 1, and p_4 and p_5 on layer 0. By introducing dummy pins p_{via1}, corresponding to the TSV on layer 1, and p_{via2}, corresponding to the TSV on layer 2, we decompose the net into 3 sub-nets: $\{p_1, p_2, p_{via2}\}$, $\{p_3, p_{via1}, p_{via2}\}$, and $\{p_4, p_5, p_{via1}\}$.

Via is the sum of the TSV numbers for all nets. If a net spans layer i to layer j, then $|j - i|$ TSVs are required for the net. As shown in Figure 7, the net crosses three device layers, two TSVs are needed to connect pins.

3.5 Framework Extension

In the 3D-FNC framework [29] shown in Figure 8, the clustering and the floorplanning are considered separately. Given a sparse neural network, the hierarchical clustering generates clusters of neurons, and then the weight connection matrix of each cluster is directly mapped to a memristive crossbar. Finally, the 3D floorplanning of memristive crossbars and neurons are performed to estimate the hardware cost.

In 3D NCS design, the floorplanning should interact with the weight clustering. That is because the weight clustering results influence the floorplan, which also in turn affects the weight clustering. Therefore, an iterative framework E3D-FNC shown in Figure 3 is proposed to improve the quality of NCS design. The algorithmic flow of the proposed iterative framework is summarized in Algorithm 4. Given a sparse neural network, since layers of neurons will influence the clustering result, we first perform the neuron floorplanning to determine layers of the neurons. Then after the memristive crossbars are generated by the hierarchical clustering and the matrix re-ordering method (lines 2 and 3), we perform the floorplanning of neurons and crossbars again to estimate

ALGORITHM 4: Interactive process of weight connection clustering and floorplanning.

```
1: repeat
2:     Perform the clustering based on the current layer information of neurons;   ▷ Equation (1)
3:     Map connections into a set of crossbars;                                  ▷ Matrix re-ordering
4:     Re-perform the floorplanning of neurons and crossbars;
5:     Evaluate the hardware cost of the current floorplan;                        ▷ Equation (6)
6: until the cost is not improved under fixed number of iterations
7: Output the 3D floorplan of the memristive crossbar-based NCS.
```

the hardware cost (lines 4 and 5). The procedure is repeated until the hardware cost of the floorplan is not improved over a predefined threshold iteration number.

4 EXPERIMENTAL RESULTS

In this work, a mixed programming is implemented by integrating `MATLAB` functions into `C++` files using `MATLAB` API interface. The hierarchical clustering, the L-method and the matrix re-ordering are implemented in `MATLAB`, and 3D floorplanning is implemented in C++ language on a 12-core 2.0-GHz Linux server with 64GB RAM. The wirelength is estimated by the half perimeter wirelength (HPWL) model. The areas of neurons and memristive crossbars with different sizes are extracted from Reference [8, 23]. Since the device defects, process variations, and IR-drop decrease the reliability of crossbar, the size of current memristor crossbars cannot be larger than 64×64 [25]. In the experiment, the allowable sizes of the utilized crossbars range from 32 to 64 at a step size of 4. First the experiment is tested on three Hopfield networks (b_{11}–b_{13}). Then a two-layer feed-forward neural network for MNIST handwritten digit recognition is exploited. Four corresponding sparse connection matrices of the neural network are generated to evaluate the proposed algorithm (b_{21}–b_{24}). Next a convolutional neural network (CNN) for CIFAR-10 [17] dataset is trained, which consists of two convolutional layers, two pooling layers and three fully connected layers. By using the synapse-granularity pruning, two sparse weight matrices of the last fully connected layers are achieved (b_{31}–b_{32}). Since in convolutional layer each neuron is connected to only a small region of the input volume, a sparse local connection matrix of the convolutional layer is gained without the weight sharing (b_{33}). Finally AlexNet is trained for ImageNet [18]. Two sparse weight matrices of the last fully connected layers (b_{41}–b_{42}) and a sparse weight matrix of the convolutional layer (b_{43}) are also generated. Table 2 lists all the statistics of different test cases. The layer number *tier_no* is set to 2 and the aspect ratio λ equals to 1.

4.1 Comparison with Previous Work

In this section, we compare E3D-FNC with some previous works to see the effectiveness of the proposed methodologies. First, to see the effect of the proposed hierarchical clustering, we compare E3D-FNC with AutoNCS [25]. In AutoNCS [25], the iterative spectral clustering is used for grouping connections into memristive crossbars. In each iteration, all the already-clustered connections are removed first, and then the spectral clustering is applied again to cluster the remaining connections of the network. Therefore, AutoNCS [25] cannot partition the sparse neural network into clusters globally. Based on clustering results, a 2D placement is implemented in AutoNCS to estimate the NCS hardware cost. Although the 2D NCS architecture may not be equivalent to our 3D NCS design, the idea of partitioning the sparse connections into a set of memristive crossbars in Reference [25] can be leveraged. In this experiment, we just compare our clustering method with the iterative spectral clustering in AutoNCS. To provide a fair comparison, we also implement a 3D floorplanning based on the iterative spectral clustering results of

Table 2. Benchmark Statistics

Benchmark	Type	Size	Sparsity
b_{11}	Hopfield network	300 × 300	69.97%
b_{12}	Hopfield network	400 × 400	63.99%
b_{13}	Hopfield network	500 × 500	68.93%
b_{21}	Fully connected layer	784 × 10	56.45%
b_{22}	Fully connected layer	784 × 10	60.36%
b_{23}	Fully connected layer	784 × 10	62.95%
b_{24}	Fully connected layer	784 × 10	66.06%
b_{31}	Fully connected layer	192 × 10	79.06%
b_{32}	Fully connected layer	192 × 10	85.73%
b_{33}	Convolutional layer	144 × 144	87.94%
b_{41}	Fully connected layer	4,096 × 1,000	84.99%
b_{42}	Fully connected layer	4,096 × 1,000	90.01%
b_{43}	Convolutional layer	169 × 169	95.46%

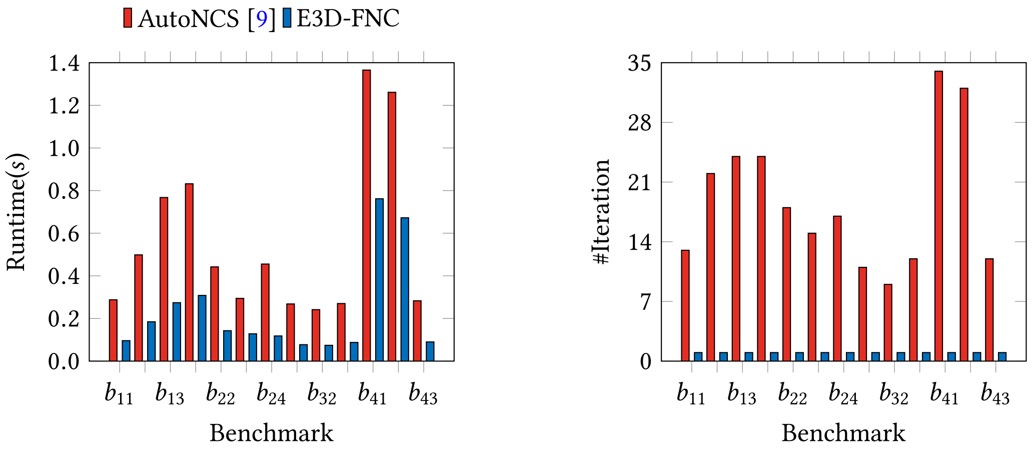
AutoNCS. The experiment is tested on all benchmarks. The average statistic results are listed in the columns "E3D-FNC" and "AutoNCS [25]" in Table 3, respectively. "Area," "Wire," and "#TSV" represent chip area, total half-perimeter wirelength overhead, and the number of TSVs, respectively. "Util" gives the average utilization of all mapped crossbars. As shown in Table 3, E3D-FNC behaves even much better with an average TSV numbers of 526 and an average crossbar utilization of 0.43 that surpass AutoNCS with 6.3% less TSV numbers and 9.3% higher crossbar utilization. In addition, in E3D-FNC, the hierarchical clustering (HC) is only performed once to build the final clustering results. However, in AutoNCS, the spectral clustering is executed in every iteration to partition connections into clusters. We use Figure 9(a) to compare the runtime of two clustering methods. From the figure, we can notice that AutoNCS could be time consuming when the neural network is large. Besides, we use Figure 9(b) to compare the number of clustering iterations of AutoNCS and E3D-FNC. From the figure we notice that the number of clustering iterations in E3D-FNC is always equal to 1, while the spectral clustering in AutoNCS will be performed several iterations until the majority of the weight connections are partitioned into clusters.

Second, to verify the effectiveness of the proposed matrix re-ordering, we compare the E3D-FNC with the generalized sparse matrix re-ordering method (GSMR) in Reference [13]. In GSMR, only row and column permutation matrices are used to break down sparse matrices into sub-clusters. Then the sub-clusters will be directly mapped to suitable size crossbars without considering the crossbar utilization. Thus many memristive crossbars with small sizes and low utilizations are generated. The experiment is tested on all benchmarks. Columns "E3D-FNC" and "GSMR [13]" in Table 3 list the statistic results. It can be noticed that compared with GSMR [13], the proposed hierarchical clustering and the matrix re-ordering method can increase the utilization of crossbars by 44.2%, which demonstrates the proposed hierarchical clustering and matrix re-ordering can achieve hardware-efficient design. In Figure 10, we depict the utilization distributions of all crossbars in testbenches b_{11}, b_{22}, and b_{43}. The sizes of crossbars are between 32 and 64.

Third, we compare E3D-FNC with 3D-FNC framework presented in the conference version [29]. The differences between E3D-FNC and 3D-FNC are illustrated in Section 3.5. The results are, respectively, shown in the columns "E3D-FNC" and "3D-FNC [29]" in Table 3. As shown in Table 3, compared with 3D-FNC [29], E3D-FNC can significantly reduce the number of TSVs by 5.7%, while the utilizations of all mapped crossbars are comparable.

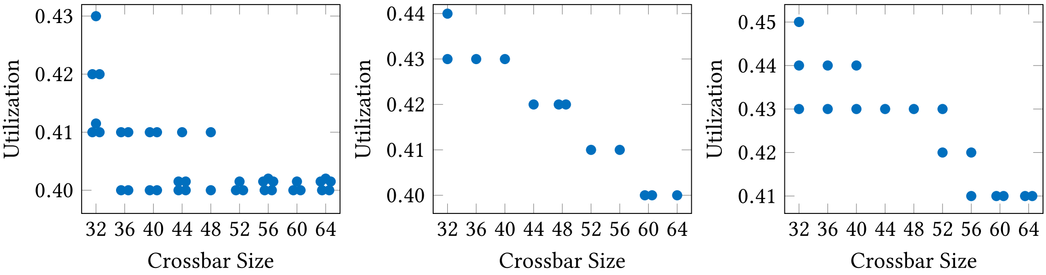
Table 3. Comparison between Our Framework and State-of-the-Art Works

Bench	AutoNCS [25]				GSMR [13]				3D-FNC [29]				E3D-FNC			
	Area (μm^2)	Wire (μm)	#TSV	Util	Area (μm^2)	Wire (μm)	#TSV	Util	Area (μm^2)	Wire (μm)	#TSV	Util	Area (μm^2)	Wire (μm)	#TSV	Util
b_{11}	207837.00	289351.37	468	0.36	206169.00	292223.56	473	0.22	206334.00	291427.96	469	0.41	207936.00	287056.54	440	0.40
b_{12}	277036.00	460382.02	629	0.40	276637.00	461600.62	630	0.23	276693.00	461297.09	624	0.42	278784.00	452993.74	588	0.42
b_{13}	345927.00	646519.02	781	0.36	346417.00	650737.63	786	0.21	346626.00	649306.55	788	0.40	349281.00	638917.65	736	0.41
b_{21}	501259.00	435276.31	424	0.43	499378.00	445324.46	449	0.26	499849.00	434074.20	422	0.47	504100.00	437364.43	419	0.47
b_{22}	486391.00	418012.25	418	0.37	486116.00	420575.61	436	0.23	487204.00	416554.00	414	0.42	490000.00	413301.50	405	0.41
b_{23}	478533.00	404309.65	438	0.37	475094.00	408949.23	427	0.22	476100.00	403701.62	442	0.40	478864.00	401717.32	398	0.41
b_{24}	462310.00	399762.79	472	0.38	459141.00	388871.54	439	0.20	459684.00	407742.72	467	0.40	463761.00	382108.37	410	0.41
b_{31}	127873.00	155323.42	150	0.43	127617.00	155630.99	148	0.31	127745.00	155784.77	146	0.46	128130.00	153785.56	132	0.46
b_{32}	116695.00	146518.48	142	0.41	116461.00	146563.70	144	0.28	116578.00	146808.91	138	0.43	117046.00	145211.58	126	0.42
b_{33}	188291.00	247599.96	390	0.44	188101.00	248328.20	393	0.32	188670.00	246387.22	385	0.47	189618.00	242743.06	376	0.49
b_{41}	3508631.00	1797070.07	1255	0.38	3505098.00	1798633.63	1259	0.23	3515695.00	1786488.69	1250	0.41	3532357.00	1763562.38	1182	0.42
b_{42}	3197104.00	1552320.84	1206	0.36	3193885.00	1559752.71	1211	0.20	3199323.00	1549268.10	1197	0.41	3218635.00	1526372.51	1158	0.40
b_{43}	234394.00	303479.50	491	0.40	234159.00	304073.97	499	0.27	234630.00	302885.02	487	0.44	235572.00	297236.51	463	0.43
avg.	779406.23	558148.17	559	0.39	778021.00	560097.37	561	0.24	779625.46	557825.14	556	0.43	784160.31	549413.17	526	0.43
ratio	0.994	1.016	1.063	0.907	0.992	1.019	1.067	0.558	0.994	1.015	1.057	1.00	1.00	1.00	1.00	1.00



(a) (b)

Fig. 9. Comparison between the two clustering methods of AutoNCS [25] and E3D-FNC.



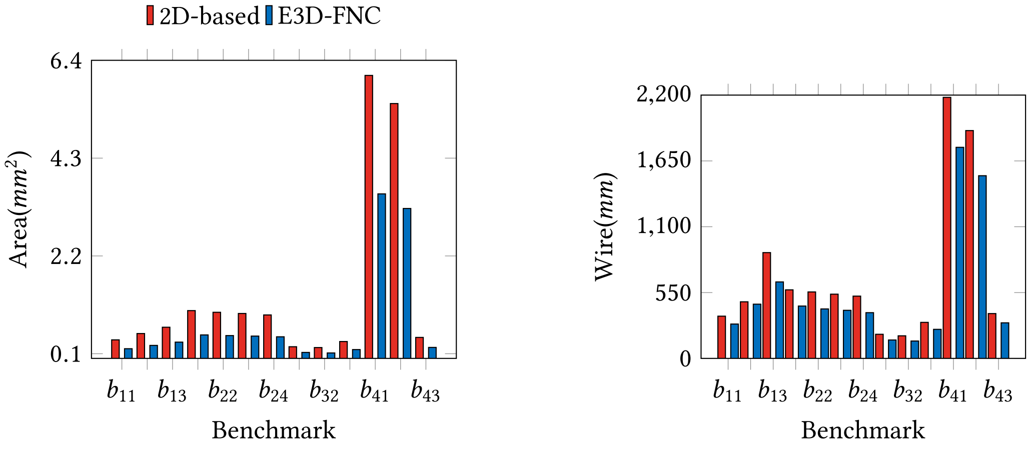
(a) (b) (c)

Fig. 10. The utilization distributions of all crossbars in testbenches (a) b_{11}, (b) b_{22}, and (c) b_{43}.

4.2 Impact of E3D-FNC on Hardware Cost

In the fourth experiment, we show the advantage of the proposed E3D-FNC design in terms of hardware cost when compared to 2D-based design. We compare the area and wirelength of the E3D-FNC design with that of 2D-based design. The experiment is tested on all benchmarks. As shown in Figure 11, compared with 2D-based design, E3D-FNC can significantly reduce the area and the wirelength overhead.

In the fifth experiment, to see the effectiveness of the L-method, we compare the floorplanning results of E3D-FNC with and without the L-method. In the proposed E3D-FNC, based on the evaluation graph generated by hierarchical clustering, the optimal number of clusters is determined by L-method. Without the L-method, the number of clusters is determined by the way, in which iteratively increasing the number of clusters by one until the size of the largest crossbar is below the size limitation. The maximum size of crossbars is set to 64 ×64 in the experiment. The average statistic results are listed in the columns "E3D-FNC" and "E3D-FNC w/o. LM" in Table 4, respectively. We can see that in the case without L-method, the utilization of mapped crossbars are reduced by 30.2%. In addition, area and wirelength are increased by 2.3% and 2.6%, which demonstrates the hierarchical clustering with L-method can result in highly hardware-efficient designs.



(a) (b)

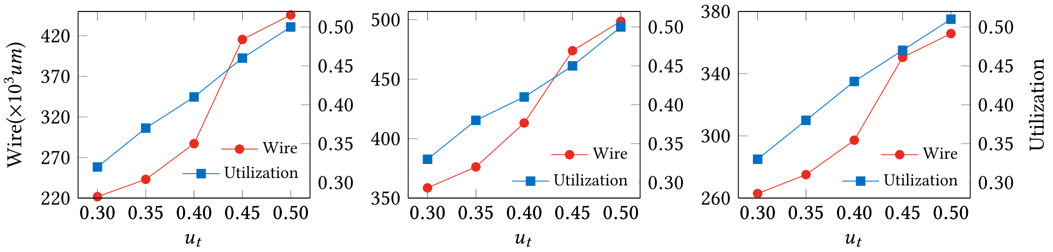
Fig. 11. Comparison between the 2D-based design and the E3D-FNC design.

Table 4. Effectiveness of the Proposed E3D-FNC

Bench	E3D-FNC w/o. LM				E3D-FNC			
	Area (μm^2)	Wire (μm)	TSV	Util	Area (μm^2)	Wire (μm)	TSV	Util
b_{11}	214382.00	295955.29	438	0.22	207936.00	287056.54	440	0.40
b_{12}	286590.00	467172.44	590	0.26	278784.00	452993.74	588	0.42
b_{13}	358758.00	657957.40	738	0.27	349281.00	638917.65	736	0.41
b_{21}	517492.00	451023.32	415	0.35	504100.00	437364.43	419	0.47
b_{22}	503230.00	426072.52	403	0.30	490000.00	413301.50	405	0.41
b_{23}	492655.00	414491.93	406	0.31	478864.00	401717.32	398	0.41
b_{24}	475772.00	393162.77	408	0.33	463761.00	382108.37	410	0.41
b_{31}	132358.00	157937.76	136	0.30	128130.00	153785.56	132	0.46
b_{32}	120674.00	148723.87	125	0.28	117046.00	145211.58	126	0.42
b_{33}	195548.00	250025.32	378	0.36	189618.00	242743.06	376	0.49
b_{41}	3603004.00	1800597.19	1176	0.27	3532357.00	1763562.38	1182	0.42
b_{42}	3286226.00	1559952.71	1160	0.27	3218635.00	1526372.51	1158	0.40
b_{43}	242461.00	305856.37	460	0.33	235572.00	297236.51	463	0.43
avg.	802242.31	563763.76	526	0.30	784160.31	549413.17	526	0.43
ratio	1.023	1.026	1.00	0.698	1.00	1.00	1.00	1.00

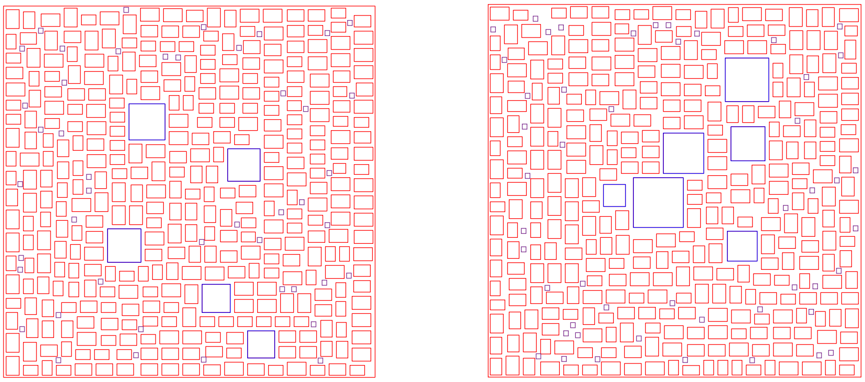
In addition, in matrix-reordering stage, the value of the utilization threshold u_t is chosen through experimental results. The experiment is performed on b_{11}, b_{22}, and b_{43}, and the results are shown in Figure 12. We noticed that as u_t increases, although the utilizations of mapped crossbars enhance accordingly, the wirelength of floorplan is sharply increasing. That is because, the larger u_t, the more connections are mapped to discrete synapses instead of memristive crossbars, resulting a higher hardware cost. This trend can also be observed in other testbenches. Therefore, according to Figure 12, $u_t = 0.4$ achieves the relatively high utilization and low wirelength cost.

A two layer floorplan example generated by E3D-FNC is shown in Figure 13.



(a) b_{11} (b) b_{22} (c) b_{43}

Fig. 12. Effect of the value of utilization threshold u_t on performance.



(a) (b)

Fig. 13. Three-dimensional floorplan example of neural network b_{21}: (a) layer 0 and (b) layer 1 (area: 0.5041 mm^2).

5 CONCLUSION

In this article, we have proposed an enhanced 3D floorplanning framework for neuromorphic computing systems, in which the neuron clustering and the layer assignment are considered interactively. A set of algorithms, e.g., hierarchical clustering, L-method and matrix re-ordering have been developed. Because the floorplanning interacts with the weight clustering, all the proposed methodologies are embedded in an iterative framework to improve the quality of 3D NCS design. Experimental results show that, compared with state of the art, the proposed E3D-FNC can achieve highly hardware-efficient designs. Memristive crossbar gives hope for the anticipated efficient implementation of artificial neuromorphic networks, thus we expect to see a lot of researches to provide more efficient physical synthesis solutions.

In this work, once the dense weight clusters are generated, the clusters will be directly implemented on memristive crossbars. However, the faults occurred in the fabrication process can make a memristor get stuck at high or low resistance state, which leads to a significant yield loss and errors in NCS. Therefore, there exists a mapping relationship between weight clusters and memristive crossbars. In future we plan to derive a weight-memristor mapping for fault tolerance.

ACKNOWLEDGMENTS

The authors thank the Information Science Laboratory Center of USTC for hardware and software services, as well as Professor Tsung-Yi Ho from National Tsing Hua University for providing test benches.

REFERENCES

[1] Simone Acciarito, Alessandro Cristini, Gianluca Susi, et al. 2017. Hardware design of LIF with Latency neuron model with memristive STDP synapses. *Integration* 59 (2017), 81–89.

[2] Filipp Akopyan, Jun Sawada, Andrew Cassidy, et al. 2015. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE J. Technol. Comput. Aid. Des.* 34, 10 (2015), 1537–1557.

[3] Mohamed Baker Alawieh, Fa Wang, and Xin Li. 2018. Identifying Wafer-level systematic failure patterns via unsupervised learning. *IEEE J. Technol. Comput. Aid. Des.* 37, 4 (2018), 832–844.

[4] Hongyu An, M. Amimul Ehsan, Zhen Zhou, Fangyang Shen, and Yang Yi. 2019. Monolithic 3D neuromorphic computing system with hybrid CMOS and memristor-based synapses and neurons. *Integration* 65 (2019), 273–281.

[5] Hongyu An, M. Amimul Ehsan, Zhen Zhou, and Yang Yi. 2017. Electrical modeling and analysis of 3D synaptic array using vertical RRAM structure. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED'17).* 1–6.

[6] Aayush Ankit, Abhronil Sengupta, Priyadarshini Panda, and Kaushik Roy. 2017. Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks. In *Proceedings of the Design Automation Conference (DAC'17).* 27.

[7] Aayush Ankit, Abhronil Sengupta, and Kaushik Roy. 2017. TraNNsformer: Neural network transformation for memristive crossbar based neuromorphic system design. *Proceedings of the International Conference on Control, Automation and Diagnosis (ICCAD'17).*

[8] Andrew S. Cassidy, Paul Merolla, John V. Arthur, and et al. 2013. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'13).* 1–10.

[9] Song Chen, Liangwei GE, Mei-Fang Chiang, and Takeshi Yoshimura. 2009. Lagrangian relaxation based inter-layer signal via assignment for 3-D ICs. *IEICE Trans. Fundam. Electr. Commun. Comput. Sci.* 92, 4 (2009), 1080–1087.

[10] Song Chen and Takeshi Yoshimura. 2008. Fixed-outline floorplanning: Enumerating block positions and a new objective function for calculating area costs. *IEEE J. Technol. Comput. Aid. Des.* 27, 5 (2008), 858–871.

[11] Song Chen and Takeshi Yoshimura. 2010. Multi-layer floorplanning for stacked ICs: Configuration number and fixed-outline constraints. *Integration* 43, 4 (2010), 378–388.

[12] Yiran Chen, Hai Helen Li, Chunpeng Wu, Chang Song, Sicheng Li, Chuhan Min, Hsin-Pai Cheng, Wei Wen, and Xiaoxiao Liu. 2018. Neuromorphic computing's yesterday, today, and tomorrow–an evolutional view. *Integration* 61 (2018), 49–61.

[13] Jianwei Cui and Qinru Qiu. 2016. Towards memristor based accelerator for sparse matrix vector multiplication. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'16).* 121–124.

[14] Richard C. Dubes and Anil K. Jain. 1988. *Algorithms for Clustering Data.* Prentice Hall. Englewood Cliffs, NJ.

[15] M. Amimul Ehsan, Hongyu An, Zhen Zhou, and Yang Yi. 2018. A novel approach for using TSVs as membrane capacitance in neuromorphic 3-D IC. *IEEE J. Technol. Comput. Aid. Des.* 37, 8 (2018), 1640–1653.

[16] Md Amimul Ehsan, Zhen Zhou, and Yang Yi. 2017. Neuromorphic 3D integrated circuit: A hybrid, reliable and energy efficient approach for next generation computing. In *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI'17).* 221–226.

[17] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images.* Technical Report. Citeseer.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Conference and Workshop on Neural Information Processing Systems (NIPS'12).* 1097–1105.

[19] Zheng Li, Chenchen Liu, Yandan Wang, Bonan Yan, Chaofei Yang, Jianlei Yang, and Hai Li. 2015. An overview on memristor crossabr based neuromorphic circuit and architecture. In *Proceedings of the International Conference on Very Large Scale Integration and System-on-Chip (VLSI-SoC'15).* 52–56.

[20] Jilan Lin, Zhenhua Zhu, Yu Wang, and Yuan Xie. 2019. Learning the sparsity for ReRAM: mapping and pruning sparse neural network for ReRAM based accelerator. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC'19).* 639–644.

[21] Beiye Liu, Yiran Chen, Bryant Wysocki, and Tingwen Huang. 2012. The circuit realization of a neuromorphic computing system with memristor-based synapse design. In *Neural Information Processing.* 357–365.

[22] Stan Salvador and Philip Chan. 2004. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04).* 576–584.

[23] Jae-sun Seo, Bernard Brezzo, Yong Liu, and et al. 2011. A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'11).* 1–4.

[24] Pun Hang Shiu, Ramprasad Ravichandran, Siddharth Easwar, and Sung Kyu Lim. 2004. Multi-layer floorplanning for reliable system-on-package. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'04)*, Vol. 5. V–69.

[25] Wei Wen, Chi-Ruo Wu, Xiaofang Hu, Beiye Liu, Tsung-Yi Ho, Xin Li, and Yiran Chen. 2015. An EDA framework for large scale hybrid neuromorphic computing systems. In *Proceedings of the Design Automation Conference (DAC'15)*. 1–6.

[26] Chi-Ruo Wu, Wei Wen, Tsung-Yi Ho, and Yiran Chen. 2016. Thermal optimization for memristor-based hybrid neuromorphic computing systems. In *Proceedings of theAsia and South Pacific Design Automation Conference (ASPDAC'16)*. 274–279.

[27] Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Xiling Yin, Wenqin Huangfu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, et al. 2016. MNSIM: Simulation platform for memristor-based neuromorphic computing system. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'16)*. 469–474.

[28] Qi Xu, Song Chen, and Bin Li. 2016. Combining the ant system algorithm and simulated annealing for 3D/2D fixed-outline floorplanning. *Appl. Soft Comput.* 40 (2016), 150–160.

[29] Qi Xu, Song Chen, Bei Yu, and Feng Wu. 2018. Memristive crossbar mapping for neuromorphic computing systems on 3D IC. In *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI'18)*. 451–454.

[30] Wangyang Zhang, Xin Li, Sharad Saxena, Andrzej Strojwas, and Rob Rutenbar. 2013. Automatic clustering of wafer spatial signatures. In *Proceedings of the Design Automation Conference (DAC*'13). 71.

Received January 2019; revised May 2019; accepted September 2019