An EDA Framework for Large Scale Hybrid Neuromorphic Computing Systems

[1] Wei Wen, [2] Chi-Ruo Wu, [3] Xiaofang Hu, [1] Beiye Liu, [4] Tsung-Yi Ho, [5] Xin Li, [1] Yiran Chen

[1] University of Pittsburgh, Pittsburgh, PA, USA; [2] National Cheng Kung University, Tainan, Taiwan; [3] City University of Hong Kong, Hong Kong;

[4] National Chiao Tung University, Hsinchu, Taiwan; [5] Carnegie Mellon University, Pittsburgh, PA, USA

[1] {wew57, bel34, yic52}@pitt.edu, [2] gtrw@eda.csie.ncku.edu.tw, [3] xiaofanhu2-c@my.cityu.edu.hk, [4] tyho@cs.nctu.edu.tw, [5] xinli@cmu.edu

ABSTRACT

In implementations of neuromorphic computing systems (NCS), memristor and its crossbar topology have been widely used to realize fully connected neural networks. However, many neural networks utilized in real applications often have a sparse connectivity, which is hard to be efficiently mapped to a crossbar structure. Moreover, the scale of the neural networks is normally much larger than that can be offered by the latest integration technology of memristor crossbars. In this work, we propose AutoNCS – an EDA framework that can automate the NCS designs that combine memristor crossbars and discrete synapse modules. The connections of the neural networks are clustered to improve the utilization of the memristor elements in crossbar structures by taking into account the physical design cost of the NCS. Our results show that AutoNCS can substantially enhance the utilization efficiency of memristor crossbars while reducing the wirelength, area and delay of the physical designs of the NCS.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*

General Terms

Algorithms, Design

Keywords

Neuromorphic Computing Systems; Neural Networks; Spectral Clustering; Memristor Crossbar; Sparsity

1. INTRODUCTION

The word "neuromorphic computing" was originally created to denote VLSI realization of neuro-biological architecture and then extended to various types of systems that accelerate the computation of neural network and machine learning algorithms [1]. The structure that mixes data storage and computing in neuromorphic systems eliminates the well-known von Neumann bottleneck in conventional microarchitecture, which refers to the increasing gap between the computing capacity and memory bandwidth of microprocessors [2].

Directly emulating a neuromorphic algorithm on von Neumann computers often incurs high memory and computation costs due to the complexity of connections in the distributed networks and the frequent updates on the weights of the connections [3]. Thus, FPGA and VLSI designs of neural circuits and synapse networks have been implemented with conventional CMOS technology to accelerate many specific algorithms [4][5]. In addition, the discovery of the emerging memristor device inspires the approaches of using memristors to build synapse circuit due to the similarity between the memristive and synaptic behaviors [2]. The low programming energy, small footprint, and non-volatility make the nanoscale memristor device become a promising candidate for the implementations of large-scale neuromorphic systems.

In designs of memristor-based neuromorphic systems, the weights of the connections are represented by the resistance of the memristor devices. From topological point of view, there are two approaches of constructing a neural network: using discrete synapses [2] and using memristor crossbars [1]: A discrete synapse makes a point-to-point connection between two neurons while a crossbar structure connects all its input neurons to all its output neurons. In fact, crossbar structure offers the highest connection density that can be obtained in two-dimensional VLSI circuit.

Although memristor crossbar is believed to be a game changing technology for neuromorphic system realization, how to efficiently design such a system with minimized (or even practical) hardware cost is still an important research topic barely touched. For example, once the application is given, how to partition the application into a set of memristor crossbars will significantly affect the implementation cost. This fact is extremely important when the mapped neural network model is sparse so that the utilization rate of some mapped memristor crossbars can be low. In such a case, directly mapping some weight connections to discrete synapses could be more efficient than mapping them to a (fully connected) memristor crossbar in terms of the area cost and delay.

In this work, we propose AutoNCS – an electronic design **Auto**mation (EDA) framework for large-scale hybrid **N**euromorphic **C**omputing **S**ystems. In particular, AutoNCS can perform the following functions to improve the implementation efficiency of NCS: 1) Given a neural network model, AutoNCS is able to partition the connections into a set of fixed-size memristor crossbars from a predefined library and discrete synapses; 2) AutoNCS can iteratively cluster the connections and map them to crossbars to minimize routing complexity in the corresponding physical designs, under realistic design constraints such as crossbar size limit etc.; 3) Based on clustering, AutoNCS also includes a customized placement & routing process to achieve a minimum area and wirelength of the designed neuromorphic system.

To the best knowledge of the authors, AutoNCS is the first EDA flow that aims the design automation of ASIC-like memristor-based neuromorphic systems, which are designated to specific neural network models. Simulation shows that compared to the brute-force implementation using the maximum-size crossbars, AutoNCS achieves on average 47.80%, 31.97%, and 47.18% reductions in wirelength, area and delay, respectively, over the three simulated test benches.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06…$15.00
http://dx.doi.org/10.1145/2744769.2744795

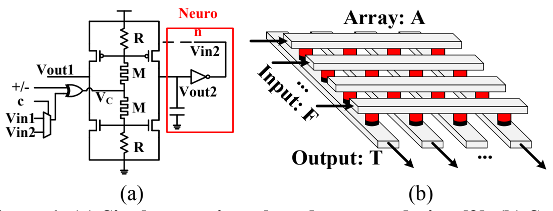


Figure 1. (a) Single memristor based synapse design [2]. (b) Synapse network based on memristor crossbar [1].

2. PRELIMINARY

2.1 Memristor based Neural Networks

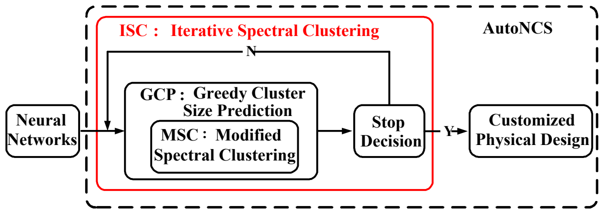
In a neural network, a set of synapses connect the input neurons **F** and the output neurons **T**. The relationship between the inputs and outputs of the synapse network can be expressed as **T** = **AF**. An element $a_{i,j}$ in the connection matrix **A** denotes the synaptic strength (weight) of the synapse connecting the *i*th neuron in **F** and the *j*th neuron in **T** (As the topology of neural networks is represented by a connection matrix, "connection matrix" and "network" are interchangeable in this paper). Since a memristor's resistance can be programmed by carefully adjusting the duration and amplitude of the applied voltage or current, we may use memristors to implement the synapses in a neural network. Figure 1(a) illustrates an example of a memristor-based synapse design including an output neuron. The weight of the synapse is stored as the resistance of the memristor. Here the output neuron is realized by an integrate-and-fire circuit built on a capacitor [2].

If a neural network is nearly fully connected, the number of the synapses in the network increases quadratically following the increase in the number of the connected neurons. Directly implementing such a network using discrete synapses becomes very costly because of the high routing overhead of the connections. To solve this issue, memristor crossbar is proposed [1], as shown in Figure 1(b). Every input neuron is connected to all output neurons via a memristor that is sandwiched between a horizontal wire and a vertical wire. Peripheral circuits are also required to perform additional functions including memristor training etc. As the size of a crossbar raises, IR-drop, device defect, and process variation introduce increasing impacts on the reliability of memristor crossbar programming and computing. A recent study shows that, considering the process variations and IR-drop, the current technology can only supply reliable memristor crossbars with a size no larger than 64×64 [6].

2.2 Sparse Neural Network Realization

The size of neural networks used in realistic applications is often very large. For instance, the deep neural network adopted in [7] for image classification has more than 4000 input nodes. Similar scale is also required by the neural network designed for LDPC coding in IEEE 802.11 [8]. If such a large network is implemented with the smaller-size memristor crossbars, the network inputs and outputs shall be partitioned and grouped into the inputs and outputs of different crossbars.

Large neural networks are often very sparse. In LDPC coding based on message passing algorithm, for example, the network sparsity is higher than 99% [8]. Here the sparsity of a network is defined as one minus the ratio between the number of actual connections and all possible connections in the network. In fact, such a high sparsity is also close to the biological facts that in neocortex, neurons are typically connected to only 10^{-9} to 10^{-7} of all the neurons and these connections are limited in the neighborhood of 1cm^2 of the tissue [9]. However, when the sparsity of a network is high, using memristor crossbars to implement such a network becomes inefficient because the utilization rate of the connections in the crossbars could be low. It may be more efficient to realize these sparse connections using smaller-size crossbars or even discrete synapses. The tradeoffs between the selection of the crossbars with different sizes, the crossbar utilization rates and the impacts on physical design cost inspire this work.


ISC : Iterative Spectral Clustering
AutoNCS
N
GCP : Greedy Cluster Size Prediction
MSC : Modified Spectral Clustering
Neural Networks
Stop Decision
Y
Customized Physical Design
Figure 2. Overview of AutoNCS.

3. AutoNCS FRAMEWORK

In this work, we developed an efficient EDA flow to design a custom memristor-based analog NCS for specific neural networks with fixed connection topology. We note that our design maintains the topology of the original NCS by mapping connections into crossbars and discrete synapses. Note that ADC/DAC are not included in the implementation because they are normally deployed outside of the NCS and shared with external devices [19].

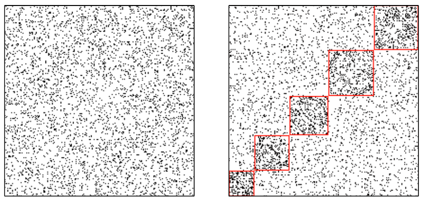
As discussed in Section 2.2, memristor crossbars are suitable for implementing dense synapse connections while discrete synapses are more efficient for realizing sparse synapse connections. In a neural network, however, synapse connections are often scattered over the whole network. Directly mapping the network to the memristor crossbars generally causes low utilization of the crossbars. In this work, we propose an iterative process based on spectral clustering algorithm to consolidate synapse connections into clusters and map them to memristor crossbars for high utilization rate of the connections in the crossbars.

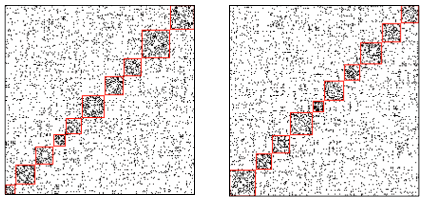
The proposed AutoNCS framework consists of the following four components: 1) Modified spectral clustering (MSC) that groups the connections in a network into dense clusters that can be efficiently mapped to memristor crossbars; 2) Greedy cluster size prediction (GCP) that constrains the largest cluster size within the maximum available crossbar scale; 3) Iterative spectral clustering (ISC) that repeatedly performs clustering on the networks to group the connections into the clusters, and minimize the outliers that need to be mapped to the discrete synapses; and 4) a customized physical design method to realize the neuromorphic systems based on the clustering result. The overview of AutoNCS is depicted in Figure 2.

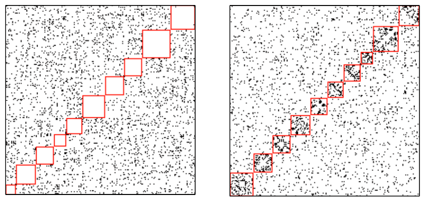
3.1 Definition of Terminologies and Variables

For ease of explanation, we define the following terminologies and variables that are frequently referred in this paper:

- ***Outliers*** – the connections that are not included in any crossbars/clusters in the implementation.
- ***Crossbar size (s)*** – the dimension of a crossbar. For simplicity, only crossbars with square shape are utilized in this work. A crossbar with a size of s offers s^2 connections.
- ***Crossbar utilized connections (m)*** – the number of the connections used for the network implementation in a crossbar.
- ***Crossbar utilization (u)*** – the ratio between the utilized connections in the network implementation and the total available connections in a crossbar, such as $u = m/s^2$.
- ***Crossbar preference (CP)*** – a criterion used to estimate the relative circuit cost reduction resulted from replacing discrete synapses with a crossbar. For a crossbar with a size s, utilized connections m, and an utilization rate u, its *CP* should satisfy the following criteria: (a) given a fixed s, *CP* monotonically increases with m or u because a larger m or u implies more discrete synapses can be replaced by crossbars, leading to a small-


(a) Original (b) Clustered
Figure 3. Clustering results of applying MSC.


(a) GCP (b) Traversing
Figure 4. GCP and Traversing.


(a) Outliers (b) Clustered outliers
Figure 5. Iteration based on MSC+GCP.

er routing cost; (b) given a fixed m, CP monotonically decreases with s as a larger s causing a higher area cost. Based on the above criteria, we propose a definition of $CP = m / s = u \times s$.

3.2 Modified Spectral Clustering (MSC)

Spectral clustering algorithm is generally adopted to partition an undirected graph to different clusters in order to minimize the between-cluster similarity and maximize the within-cluster similarities [10]. Implication of “similarity” varies with specific applications in practice. In this work, we leverage spectral cluster algorithm to group the connections between the input and output neurons and fit them into the crossbars as many as possible. Hence, we redefine the similarity in spectral clustering algorithm as the number of connections. The goal in our modified spectral clustering (MSC) becomes minimizing the (between-cluster) connections that need to be mapped to discrete synapses and maximizing the (within-cluster) connections that fit into the crossbars. Note that the elements in the given connection matrix are binary where ‘1’ represents a connection linking two neurons and ‘0’ indicates the two neurons are not connected. The process of MSC is shown in Algorithm 1 where k-means is also a traditional clustering algorithm to partition a data set to k clusters [10][11].

Figure 3(a) and (b) respectively shows the connection matrix of a real 400×400 neural network before and after applying MSC. Empty spaces show no connections between the neurons while red squares are the formed clusters. It can be clearly observed that after MSC, the connections in the network are effectively grouped into several clusters. The connections in the clusters can be efficiently mapped to memristor crossbars with a high utilization.

In the design of neuromorphic systems based on memristor crossbars, there are at least two realistic challenges that need to be solved in the clustering:

(1) Cluster size limitation: the sizes of the clusters are limited by the maximum size of the available crossbars. Classic spectral clustering algorithm, however, does not consider this limit;
(2) High outlier ratio: It is almost impossible to group the majority of the connections into clusters by performing MSC only once, especially for the neural networks with randomly distributed connections. For instance, the outliers in Figure 3(b) still count for 57% of total connections in the network.

Algorithm 1: Modified Spectral Clustering (MSC)

```
Input: connection matrix $W \in \mathbb{R}^{n\times n}$ with elements $w_{ij}$ $(1 \le i,j \le n)$,
    and number $k$ of clusters to construct.
1: Compute the degree matrix $D$: for each element on the diagonal
    $d_{ii} = \sum_{j=1}^{n} w_{ij}$, for other elements $d_{ij} = 0$ $(i \ne j)$;
2: Compute unnormalized Laplacian matrix: $L = (D - W)$;
3: Compute $k$ generalized eigenvectors $u_1, \dots, u_k$ of the generalized
    eigenproblem $Lu = \lambda Du$ corresponding to the $k$-smallest
eigenvalues;
4: Let $U \in \mathbb{R}^{n\times k}$ be the matrix containing the columns $u_1, \dots, u_k$;
5: For $i$=1 to $n$, let $y_i \in \mathbb{R}^k$ be the $i$-th row of $U$;
6: Cluster the points $y_i$ ($i$=1, . . .,$n$) with the k-means algorithm
    into clusters $C_1, \dots, C_k$;
Output: clusters $A_1, \dots, A_k$ with $A_i = \{j \mid y_j \in C_i\}$.
```

Two techniques, greedy cluster size prediction (GCP) and iterative spectral clustering (ISC) are proposed in Section 3.3 and 3.4, respectively, to conquer the above two challenges.

3.3 Greedy Cluster Size Prediction (GCP)

The limit of crossbar size can be passively imposed by exhaustively increasing the value of k in MSC until the size of the largest crossbar is below the size limit. We refer to this method as *traversing algorithm*. However, such a method could be very time consuming when the network is large. Rather than scanning the number of total clusters k, we propose to directly limit the largest cluster size obtained in the k-means algorithm: if the size of the obtained cluster is beyond the limit, the cluster will be automatically broken into two smaller sub-clusters. The centroids of these two sub-clusters are updated accordingly and k is incremented by 1. Algorithm 2 gives the details of this proposed greedy cluster size prediction (GCP).

We applied GCP to the test case in Figure 3 by setting the maximum cluster size to 64. The result is shown in Figure 4(a). It can be observed that the maximum cluster size is constrained below the preset limit, demonstrating the effectiveness of GCP.

Figure 4(b) illustrates the clustering result from traversing algorithm, which is very close to the GCP result in Figure 4(a). However, the computation time of traversing algorithm (190ms) is almost double of the one of GCP (106ms). These results clearly demonstrate the effectiveness and efficiency of GCP in limiting the cluster size during the clustering process.

3.4 Iterative Spectral Clustering (ISC)

To minimize the outliers in the clustering process, we propose an iterative spectral clustering (ISC) scheme to recursively group the connections into the clusters.

Algorithm 2: Greedy Cluster Size Prediction (GCP)

```
Input:    connection matrix $W \in \mathbb{R}^{n\times n}$ and limited max cluster size $s$.
1: Initialization: compute all generalized eigenvectors $u_1, \dots, u_n$ of
    the generalized eigenproblem $Lu = \lambda Du$. Let $U \in \mathbb{R}^{n\times n}$ be the matrix
    whose columns are an ascending sequence of $u_1, \dots, u_n$ sorted by
    their eigenvalues;
2: Predict cluster number $k$=$n$/$s$, initialize $k$ cluster centroids $B$ as zeros;
3: do
4:      for $i$=1, . . .,$n$, let $y_i \in \mathbb{R}^k$ be vector corresponding to the $i$-th
        row of $U_k$, $U_k$ is the first $k$ columns of $U$; flagOuter=0;
5:      do
6:          under $B$, cluster the points $y_i$ ($i$=1, . . .,$n$) with k-means
            algorithm into clusters $C_1, \dots, C_k$ and update $B$;
7:          flagInner=0;
8:          for all $j$=1, . . .,$k$
9:              if size of $C_j > s$
10:                 cluster $C_j$ to 2 sub-clusters by k-means;
11:                 $k$=$k$+1; flagInner=1; flagOuter=1;
12:                 update $B[j]$ and $B[k]$ to centroids of those 2 sub-
                    clusters;
13:             end if
14:         end for
15:     while flagInner==1
16: while flagOuter ==1
Output: clusters $A_1, \dots, A_k$ with $A_i = \{j \mid y_j \in C_i\}$.
```

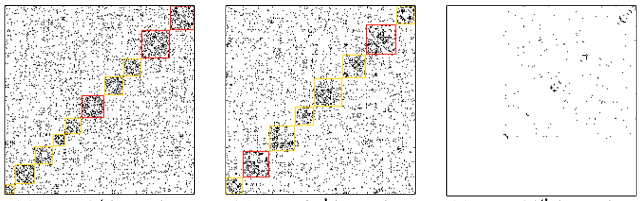

(a) The 1st iteration (b) The 2nd iteration (c) The 11th iteration

Figure 6. Results of ISC iterations.

As shown in Figure 4(a), after the first clustering operation, the within-cluster connections are concentrated along the diagonal of the connection matrix. Applying MSC+GCP on the already-clustered network may not result in further large reduction of the outliers as it will break the clusters that were already grouped. We name this phenomenon as *cluster concealing*. To eliminate the disturbance of the existing clusters, we propose to remove all the connection clusters from the already-clustered connection network, and create a "remaining" network that is composed of only the outliers. We then apply MSC+GCP to the remaining network for clustering the outliers. This procedure is repeated until there are no enough connections can be efficiently clustered, say, the crossbar utilization (u) is below a predefined threshold. Figure 5(a) shows the remaining network obtained by removing the within-cluster connections in Figure 4(a). Applying another round of MSC+GCP produces a new connection matrix in Figure 5(b) where the outliers become sparser than that in Figure 5(a).

However, if a formed cluster is sparse, i.e., its crossbar preference (CP) is low, it might not be worth implementing this cluster on a crossbar. In the iterations of ISC, we only remove the clusters with a high CP (and map them to the crossbars) and leave the rest clusters in the remaining network. This strategy can effectively prevent the occurrence of the crossbars with low utilization in the implementation. It can also result in globally enhanced CPs for all the crossbars by combining the outside-cluster connections with the remaining within-cluster connections for further re-clustering. We refer to this scheme as "partial selection strategy". The details of ISC are summarized in Algorithm 3.

The ISC result of the same test case is shown in Figure 6. Red squares mark the clusters with high CPs that will be removed by the end of the current iteration; Yellow squares mark the clusters to be kept in the remaining network due to the low CPs. After the 11th iteration, most of the connections are clustered, leaving an almost empty remaining network, i.e., < 5% outlier ratio. Here we empirically remove only the top 25% clusters with the high CPs among all the clusters in each iteration.

Algorithm 3: Iterative Spectral Clustering Algorithm (ISC)

```
Input: connection matrix $W \in \mathbb{R}^{n\times n}$ with elements $w_{ij}$ $(1 \le i, j \le n)$,
    crossbar size set $S$ in specification, and utilization threshold $t$
1: Initialization: remaining connection matrix $R$=$W$, average crossbar
    utilization $u$=1; iteration number $m$=1;
2: do
3:      cluster $R$ into clusters $A_1, \ldots, A_k$ by Algorithm 2 with
        limited size max($S$);
4:      compute $CP_i$ for cluster $A_i$ ($i$=1,2,…,$k$);
5:      set $q$ to the quartile of $CP_i$ ($i$=1,2,…,$k$);
6:      if the size of the crossbar with $CP$=$q$ is smaller than min($S$)
7:          break;
8:      end if
9:      for $i$=1,2,…,$k$
10:         if $CP_i \ge q$
11:             realize connections within $A_i$ by a minimum
                satisfiable crossbar in $S$;
12:             delete connections within $A_i$ from $R$;
13:         end if
14:     end for
15:     set $u$ average utilization of crossbars placed in iteration $m$;
16:     $m$=$m$+1;
17: while $u \ge t$
18: realize connections remained in $R$ by discrete memristors
Output: circuit implementation topology.
```

3.5 Physical Implementation and Cost Evaluation

We estimate the NCS hardware cost based on the placement area and wirelength [12]. Compared to discrete synapses, memristor crossbars have higher connection density and more flexible routability. However, the highly congested placement around the crossbars may result in undesired routing detours due to the limited routing resources, incurring the increase in hardware cost.

To measure the total area and wirelength, a physical design implementation including the placement of crossbars and neurons as well as the wiring routing shall be performed. Unfortunately, we cannot directly apply the existing circuit placement algorithms to our problem because of the following differences in problem formulations: (1) various wire weights between memristors and crossbars, (2) mixed-size cells including neurons, memristors, and crossbars, and (3) cells are not required to align into rows. Hence, we propose an analytical model to solve the above challenges by combining some existing models [13-17].

In the phase of placement and routing, the crossbars and neurons are considered as cells. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be the x and y-coordinates set of n cells, $C = \{c_1, \ldots, c_n\}$ be the set of cells, $E = \{e_1, \ldots, e_m\}$ and $W = \{w_1, \ldots, w_m\}$ be the set of m wires and the set of m wire weightings, respectively. We adopt an analytical method to conduct the placement with gradient-based optimization method. To minimize the wirelength with density constraint, the placement problem can be formulated as a penalty function given by $\min WL(x,y) + \lambda D(x,y)$, where $WL(x,y)$ is the wirelength cost function, λ is the penalty parameter, and $D(x,y)$ is the cell density function.

Since the *half-perimeter wirelength* (HPWL) is nonconvex and difficult to minimize, the *weighted-average* (WA) wirelength model [13] was adopted to approximate the HPWL. Furthermore, user-defined various wire weights between memristors and crossbars are included in our wirelength computation to shorten some wires that have high weights and hence, are critical to the system performance. The RC delay is used to estimate the wire weights in this paper in the WA wirelength model given by:

$$WL(x,y) = \sum_{e\in E} w_i \left(\frac{\sum_{v_i\in e} x_i \cdot \exp(x_i/\gamma)}{\sum_{v_i\in e} \exp(x_i/\gamma)} - \frac{\sum_{v_i\in e} x_i \cdot \exp(-x_i/\gamma)}{\sum_{v_i\in e} \exp(-x_i/\gamma)} + \frac{\sum_{v_i\in e} y_i \cdot \exp(y_i/\gamma)}{\sum_{v_i\in e} \exp(y_i/\gamma)} - \frac{\sum_{v_i\in e} y_i \cdot \exp(-y_i/\gamma)}{\sum_{v_i\in e} \exp(-y_i/\gamma)} \right) \cdot \text{(1)}$$

Here γ is a user-defined parameter to control the smoothness. Hence, the cell density model can be applied to place the cells by reducing the cell overlap given by:

$$D(x,y) = \sum_{c_i, c_j \in C} O_x(c_i, c_j)\, O_y(c_i, c_j)\ . \qquad (2)$$

Here $O_x(c_i,c_j)$ is the sigmoid based density model [14] that repre-

Algorithm 4: Placement

```
Input: the locations of cells and wires with wire weighting
1: Initialization: cells location, $\lambda_0 = \frac{\sum|\partial W(x,y)|}{\sum|\partial D(x,y)|}$ and m = 0
2: do
3:      solve $\min WL(x,y) + \lambda_m D(x,y)$;
4:      $m = m + 1$;
5:      $\lambda_m = 2\lambda_{m+1}$;
6: while the sum of overlap is larger than user defined threshold
7: Process the remaining overlap between cells;
Output: optimized locations of cells.
```

sents the overlap function between cell c_i and c_j along the x direction. Similarly, $O_y(c_i,c_j)$ is the overlap function along the y direction. To reserve the space for routing, we consider the virtual width as the product between ω and the cell width. Here ω is a user-defined parameter to determine the routing resources. The pseudo code of our placement algorithm is shown in Algorithm 4. Line 1 initializes the cell placement with regular location. Line 3-7 increases the importance of cell density function iteratively to optimize the cell placement. The conjugate gradient algorithm [15] is used to solve the penalty function at line 4. Line 8 pushes away the cells to legalize the remaining overlap between cells.

To estimate the wirelength, we modify the maze routing [16] with the virtual capacity [17]. A grid graph model [18] is constructed with bin width θ, which is a user-defined parameter as the input of the routing phase. The virtual capacity is used to estimate the number of wires in each edge in grid graph. The routing order is determined by the distance from the center of gravity of all cells to its closest pin of wires. If the distance is the same for more than two wires, we will use wire weighting as the tie breaker. During maze routing, certain wires may fail to be routed by this routing order. In that case, the virtual capacity will be relaxed for rerouting failed wires until all wires are routed. After the placement and routing, the layout is derived and the physical cost is evaluated by:

$$Cost = \alpha \times L + \beta \times A + \delta \times T. \tag{3}$$

Here α, β, and δ are user-defined parameters to determine the importance of total wirelength (L), chip area (A), and average wire delay (T).

4. EXPERIMENTS

4.1 Testbenches

Three testbenches of random quick response code patterns are used in our experiments. We use M and N to denote the amount of patterns in the training set and the dimension of each pattern, respectively. The patterns in each testbench are stored in a sparse Hopfiled network with a size of N. The (M, N) factors of the three testbenches 1-3 are (15, 300), (20, 400) and (30, 500), respectively. The corresponding sparsities of the three networks are 94.47%, 93.59% and 94.39%, respectively. All testbenches offer a recognition rate above 90%.

4.2 The Iterative Spectral Clustering

In our experiments, the allowable crossbar sizes range from 16 to 64 at a step of 4. We define the baseline design as a full crossbar design (denoted as "FullCro") that uses only crossbars with a size of 64 to implement the neural network. Obviously, FullCro has low crossbar utilization. We also define "fanin+fanout" of a neuron to denote the total number of fanins and fanouts of it. The value of fanin+fanout roughly measures the congestions around the neurons. The iteration of ISC stops when the average crossbar utilization is below that of the baseline design, implying no benefit to continue performing clustering.

Figure 7-9 summarize the detailed analysis on the efficacy of ISC in testbenches 1-3, respectively. As shown in Figure 9(a), the outlier ratio of testbench 3 drops quickly over the iterations: after 14 iterations, 95% of connections are clustered and ready to be mapped to the crossbars. Figure 9(b) presents the average crossbar utilization normalized to our baseline and the average crossbar preference over the iterations. Both of them keep decreasing during the process of ISC. It implies that connection clustering generally becomes more and more difficult over the iterations. However, the slight rises of the normalized crossbar utilization at some iterations can be observed, showing the effectiveness of partial selection strategy. When the number of iterations reaches 14, the normalized crossbar utilization becomes less than 1 and the ISC stops. Figure 9(c) shows the distribution of the utilized crossbars in the final implementation. The sizes of most of crossbars are between 32 and 64. In Figure 9(d), we depict the distributions of the fanin+fanout's of all the neurons from/to only the crossbars ('Crossbar") and from/or only the discrete synapses ("Synapsis"), and the x-axis are the N neurons in order of their fanin+fanout. After ISC, the fanouts and fanins of the majority of the neurons come from the crossbars and many of them do not even connect to any discrete synapses. Figure 9(d) also show the distribution of the total fanin+fanout of all the neurons from/to both the crossbars and the discrete synapses ("Sum"). The average total fanin+fanout of all the neurons after ISC ("Avg. sum") is only 80% of the one in the baseline design. Note that all the results in Figure 9(d) have been normalized to the baseline design. Similar results are observed in testbench 1 and 2.

4.3 Hardware Cost Evaluation

To evaluate the hardware cost, we implemented our proposed placement and routing method using C/C++ on a 64-bit Linux machine and a 3.4 GHz processor with 32GB RAM. In the physical cost function, α, β, and δ are all set to 1. The delays and areas of the memristor crossbars with different sizes, discrete synapses, and neurons are extracted from [15] and [2], and carefully scaled to 45nm technology node.

The total wirelength, the placement area, and the wire delay between FullCro and AutoNCS for all three testbenches are summarized in Table 1. Compared to FullCro, AutoNCS on average reduces the total wirelength by 47.80%, the placement area by 31.97%, and the wire delay by 47.18%, respectively, over all the testbenches.

Figure 10(a)-(d) show the optimal placement and routing results of the testbench 3 in FullCro and AutoNCS, respectively. In Figure 10, the unit of both x- and y-axis is the pitch of a memristor cell in the crossbar; in (a) and (c), the squares with different sizes denote crossbars and memristors with different scales; in (b) and (d), routing information is expressed by a wire congestion map with difference colors showing the numbers of wires. In optimal

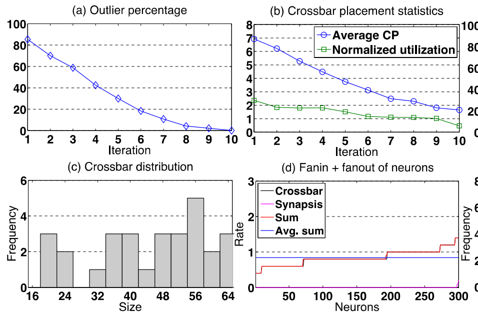


Figure 7. ISC Exp. Result of Testbench 1.

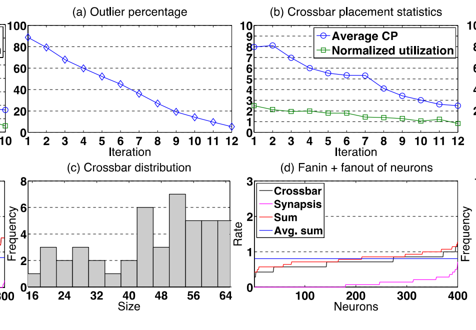


Figure 8. ISC Exp. Result of Testbench 2.

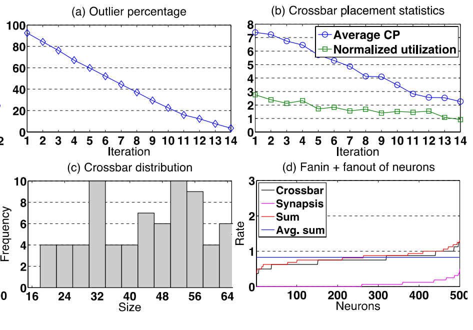


Figure 9. ISC Exp. Result of Testbench 3.

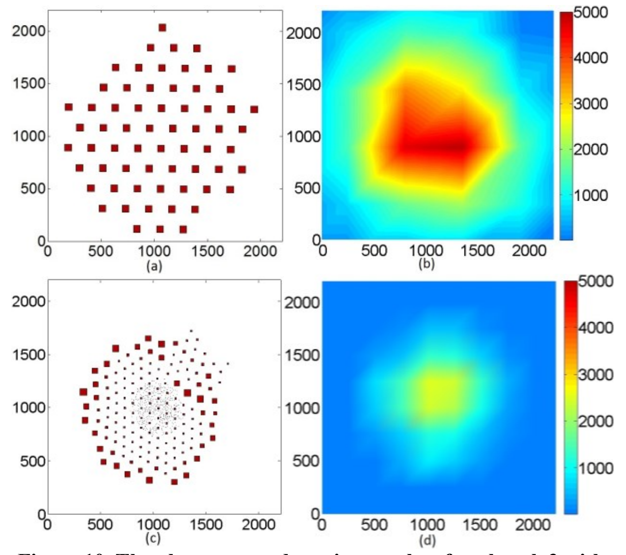


Figure 10. The placement and routing results of testbench 3 without clustering are shown in (a) and (b). The results with AutoNCS are shown in (c) and (d). Scale bar: 140 µm.

FullCro, crossbars with the maximum size are uniformly placed, resulting in heavy wire congestion in the center as Figure 10 (b) illustrates. However, in AutoNCS, large crossbars on the periphery realized the majority of connections, leaving only sparse connections implemented by small crossbars and discrete synapses in the inner place. This topology reduces wirelength, area and average delay substantially. Table 1 also shows that wirelength and area reductions increase with the scale of NCS, which implies the scalability and adapability of our AutoNCS to large-scale NCS. The delay keeps steady because it is determined by the crossbar size distribution, which is similar under different scales of NCS.

5. CONCLUSION

In this work, we proposed an EDA framework, namely, AutoNCS, to automate the implementation of large-scale neuromorphic systems. AutoNCS iteratively clusters the connections in neural networks into memristor crossbars for routability enhancement and optimizes the physical design process for placement area and routing wirelength reductions. Our experiments on three testbenches with different sizes show that AutoNCS can reduce the wirelength, area, and delay by on average 47.80%, 31.97%, and 47.18%, respectively, compared with the brute-force implementations using only the maximum-size crossbars.

6. Acknowledgement

This work was supported in part by AFRL FA8750-15-2-0048, DARPA D13AP00042, NSF XPS-1337198, and ECCS-1202225. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL, DARPA, NSF, or their contractors.

Table 1. The Physical Design Cost Evaluation.

Test bench		Total wirelength (μm)	Area (μm^2)	Delay (ns)
1	**AutoNCS**	131,934.3	7,608.80	1.05
	FullCro	233,080.0	9,667.20	1.95
	Reduc. (%)	43.40%	21.29%	46.15%
2	**AutoNCS**	380,549.6	14,211.54	1.05
	FullCro	676,416.0	20,168.60	1.95
	Reduc. (%)	43.74%	29.54%	46.15%
3	**AutoNCS**	575,760.9	20,943.93	0.99
	FullCro	1,316,590.0	38,136.23	1.95
	Reduc. (%)	56.27%	45.08%	49.23%

7. REFERENCES

[1] M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of BSB recall function using memristor crossbar arrays," *Design Automation Conference*, 2012, pp. 498-503.

[2] B. Liu, Y. Chen, B. Wysocki, and T. Huang, "The circuit realization of a neuromorphic computing system with memristor-based synapse design," in *Neural Information Processing*, 2012, pp. 357-365.

[3] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, *et al.*, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, p. 54.

[4] S. L. Bade and B. L. Hutchings, "FPGA-based stochastic neural networks-implementation," in *FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on*, 1994, pp. 189-198.

[5] K. Wawryn and B. Strzeszewski, "Low power VLSI neuron cells for artificial neural networks," *IEEE International Symposium on Circuits and Systems*, 1996, pp. 372-375.

[6] J. Liang and H. S. Wong, "Cross-point memory array without cell selectors—device characteristics and data storage pattern dependencies," *IEEE Transactions on Electron Devices*, vol. 57, pp. 2531-2538, 2010.

[7] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, 2012, pp. 3642-3649.

[8] "IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1-2793 , 2012.

[9] C. D'Este, M. Towsey, and J. Diederich, "Sparsely-connected recurrent neural networks for natural language learning," in *First Workshop on Natural Language Processing and Neural Networks. Beijing, China*, 1999, pp. 64-69.

[10] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, pp. 395-416, 2007.

[11] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888-905, 2000.

[12] A. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 734–747, 2005.

[13] M.-K. Hsu et al., "TSV-Aware Analytical Placement for 3D IC Designs," Proc. ACM Design Autom. Conf., pp. 664–669, 2011.

[14] S. Chou et al., "Structure-Aware Placement for Datapath-Intensive Circuit Designs," *Design Automation Conference*, 2012, pp. 762–767.

[15] S. Chou et al., "NTUplace3: An Analytical Placer for Large-Scale Mixed-Size DesignsWith Preplaced Blocks and Density Constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1228–1240, 2008.

[16] C. Lee and N. J. Whippany, "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Elctronic Computers*, pp. 346–365, 2009.

[17] Y. Zhang et al., "FastRoute3.0: A Fast and High Quality Global Router Based on Virtual Capacity," *International Conference on Computer‐Aided Design*, pp. 344–349, 2008.

[18] M. Pan and C. Chu, "FastRoute: A Step to Integrate Global Routing into Placement," *International Conference on Computer‐Aided Design*, pp. 464–471, 2006.

[19] X. Liu, M. Mao, H. Li, Y. Chen, H. Jiang, J. J. Yang, Q. Wu, M. Barnell, "A Heterogeneous Computing System with Memristor-Based Neuromorphic Accelerators.", *High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1-6.