Thermal-Aware Layout Optimization and Mapping Methods for Resistive Neuromorphic Engines

Chengrui Zhang*‡, Yu Ma*§†‡ and Pingqiang Zhou*‡
*School of Information Science and Technology, ShanghaiTech University, Shanghai, China
†University of Chinese Academy of Sciences, Beijing, China
‡Shanghai Engineering Research Center of Energy Efficient and Custom AI IC, Shanghai, China
§Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai, China
{zhangchr, mayu, zhoupq}@shanghaitech.edu.cn

Abstract— Resistive neuromorphic engines can accelerate spiking neural network tasks with memristor crossbars. However, the stored weight is influenced by the temperature, which leads to accuracy and endurance degradation. The higher the temperature is, the larger the influence is. In this work, we propose a cross-array mapping method and a layout optimization method to reduce the thermal effect with the consideration of input distribution, weight value and layout of memristor crossbars. Experimental results show that our method reduces the peak temperature up to $10.4K$ and improves the endurance up to $1.72\times$.

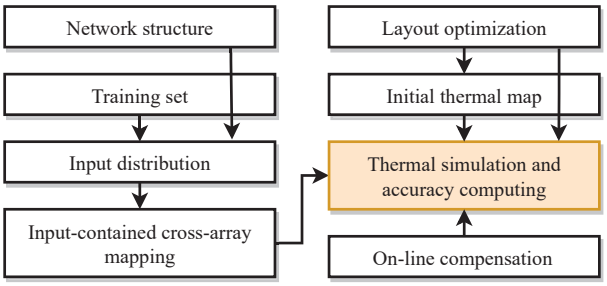

Fig. 1. Proposed optimization framework.

I. Introduction

Spiking neural network (SNN) simulates the activity of neurons [1] to perform brain-inspired tasks, such as image recognition and inference. SNN first utilizes 1-bit spikes as the input of each layer. Then it executes multiply-and-accumulate (MAC) operations with the input and weight, which is the most expensive step in the inference process. The MAC operation can be accelerated efficiently by resistive neuromorphic engines with Kirchhoff's law where the input and weight are programmed as voltage and conductance respectively. In order to implement this idea, memristor crossbar array is the most attractive structure in neuromorphic engines.

Before the inference process, the engine programs the weight into the conductance of memristor crossbars. Ideally, the conductance will not change and we can use this conductance to do MAC operation accurately. However, memristor has the conductance variation issue due to different reasons, such as the drift issue [2], [3], the variation issue [4], [5] and etc. Temperature can also cause serious conductance variation problem, as shown in Fig. 2(a). The HRS conductance of memristor at $400K$ decreases to half compared to its original value at $300K$. [6]–[8] show that the conductance variation results in a serious loss of accuracy. Besides, high temperature reduces the endurance of memristor crossbar [9].

In order to reduce the thermal effect, several methods are proposed from three perspectives. [10], [11] utilize two off-line training ways to change the weight distribution, which can reduce the total weight value to reduce total conductance. [7], [12] propose thermal-aware mapping methods considering thermal effect to reduce the effect on large weight [7] and even the power distribution of the crossbar to reduce the maximum temperature [12]. The third is to compensate the conductance decrease dynamically by bitwidth downgrading [8] or current compensation [12]. However, all these methods only consider the weight itself, ignoring the fact that the input distribution also influences the power consumption in neuromorphic engines [3], which will further affect the thermal distribution. Besides, the previous mapping methods do not consider the row or column exchange among the memristor arrays, as they do the exchange within an array [12] or treat all arrays as a whole [2], [7], limiting the performance of mapping methods.

In this paper, we improve the temperature reduction technology of previous works by proposing a novel mapping algorithm and a layout optimization method. We firstly propose an effective mapping method, which executes a finer re-ordering method with the consideration of the exchange among different memristor crossbars. In addition, we explore the effect of input distribution in the thermal problem, and utilize this distribution to reduce the thermal effect. Furthermore, we propose a novel layout to reduce the thermal effect as the previous layouts centralize the high power density components, which aggregates thermal problem. Our optimization framework is shown in Fig. 1. The contributions of this paper are summarized below.

- We firstly investigate the input distribution of each layer and analyze how the input distribution can affect the thermal distribution. Then we estimate the input distribution of each layer with training set.
- We then model the mapping problem mathematically and propose a novel cross-array mapping method based on the estimated input distribution.
- We propose a novel layout which can decrease the thermal effect and reduce the transmission loss.
- We evaluate our proposed method in VGG9 and VGG11. Experimental results show that our method can reduce the peak temperature up to $10.4K$ and improve the endurance up tp $1.72\times$.

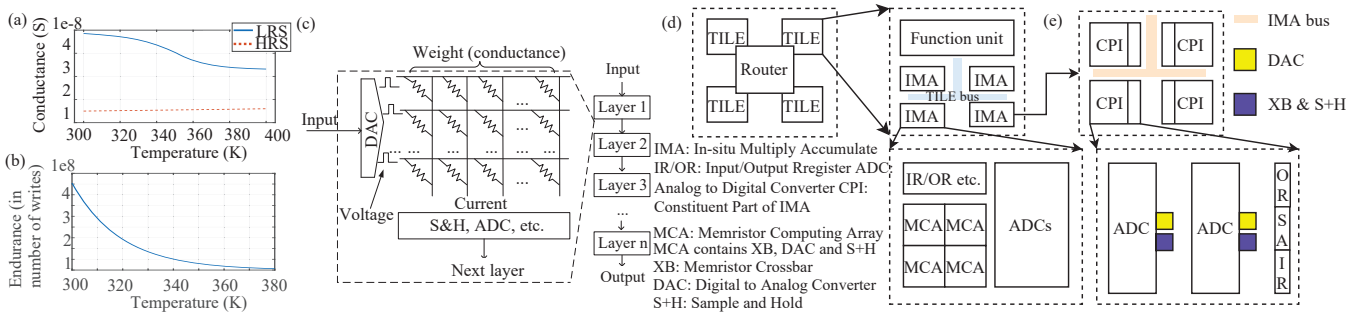

Fig. 2. (a) The conductance variation of memristor with thermal effect [6] (HRS (LRS) denotes high (low) resistance state). (b) The endurace variation of memristor with thermal effect [9]. (c) Memristor crossbar. (d) ISAAC structure [13]. (e) An example of our proposed layout–called "MA" for each IMA.

The rest of this paper is organized as follows. Sec. II introduces the background of our work. Then we propose our thermal-aware layout optimization method in Sec. III. After that, we propose a novel input-contained weight mapping method in Sec. IV. At last, we show our experimental results in Sec. V, followed by the conclusion in Sec. VI.

II. Background

A. Spiking neural network

In order to simulate neurons, SNN simulates a preset number of time steps, $T_{\max}$. Firstly, the input of the SNN is encoded to spikes. In each time step, every neuron updates the membrane potential (MP) as Equation (1):

$$z_i^l(t) = \sum_j w_{ij}^l \Theta_j^{l-1}(t) + b_i^l \tag{1}$$

where Θ_j^{l-1} represents the j-th input of layer l, w_{ij}^l is the weight and b_i^l is the bias of the i-th neuron in layer l. Then, if the MP of the i-th neuron exceeds threshold V_{th}, the neuron sends a spike to the next layer.

$$\Theta_{t,i}^l = U\left(V_i^l(t-1) + z_i^l(t) - V_{th}\right) \tag{2}$$

where $U(x)$ is a step function and $V_i^l(t-1)$ is the MP in time $t-1$. The MP is reset to the resting potential [14]:

$$V_i^l(t) = \begin{cases} V_{rest,i}^l & \Theta_i^l(t) = 1 \\ V_i^l(t-1) + z_i^l(t) & Otherwise. \end{cases} \tag{3}$$

After running the simulation for $T_{\max}$ time steps, the algorithm is terminated and the neuron with the largest spikes is the output.

B. Resistive neuromorphic engines

Equation (1) is the most costly process (MAC operation) in SNNs. Resistive neuromorphic engines can be used to accelerate this kind of processes. Fig. 2(c) shows the ISAAC architecture of the engine [13] based on memristor crossbar (see Fig. 2(b)). The input is converted to the voltage and the weight is programmed to the conductance of the memristor crossbar. Then the engine can execute the MAC operation by detecting the current of each column according to Kirchhoff's law.

C. Thermal effect

Although the programmed conductance should not change in the inference process, the conductance varies with the temperature. This leads to the accuracy degradation of SNNs. Fig. 2(a) shows the relationship between the conductance and temperature. We can find that the conductance of HRS has the semiconducting property. The conductance will increase when the temperature increases. The LRS has the opposite property: with the temperature increasing, the conductance will decrease. The relationship between temperature and endurance of memristor is shown in Fig. 2(b). We can find that the endurance slides rapidly as temperature grows [9].

III. Thermal-aware Layout Optimization

In this section, we propose our layout optimization work to reduce the temperature of each tile by dis-centralizing the high power density components. The motivation comes from the architecture of previous thermal-related works [7], [8], [12]. We find that the layout of memristor chip plays an important role in thermal analysis and the ISAAC architecture they use is not suitable if we take thermal issue into consideration. The centralized high power density components will aggravate the thermal problem in ISAAC. As shown in Fig. 2(c), we find that it centralizes the XBs and DACs – the top-2 power density terms [13]. This centralized layout for high power density components will cause higher temperature than dis-centralization condition, which will aggravate the thermal problem. Therefore, it is necessary to take thermal problem into account when we do layout design.

In order to reduce the thermal effect, we propose an intuitive idea to dis-centralize the high power density components. The goal of our layout optimization is to reduce the thermal effect on memristor crossbar. Therefore, we give priority to the thermal reduction, and we can accept a little cost of other evaluating indicators, like area, delay, etc.

Our proposed layout is shown in Fig. 2(d). We adjust the layout of inner components in each IMA. In this new layout, one tile contains 8 IMAs and the corresponding peripheral circuits, and one IMA contains 4 identical parts called constituent part of IMA (CPI). In the proposed placement method, we do not centralize the XBs and DACs as shown in Fig. 2(c). Instead, we dis-centralize these components, and take 2 ADCs, 2 DACs and 2 XBs as a whole to avoid the aggregation of high power density components. We call this layout "MA" in Fig. 2(d).

In addition to the ability of temperature reduction, this layout can also reduce the transmission impairment of analog data. As the analog data flow of MAC operation in ISAAC is DAC→XB→ADC, the combination of the corresponding DAC, XB and DAC in Fig. 2(d) can reduce the transmission distance of analog signal, especially for the transmission path from XB to ADC.

IV. Thermal-aware Weight re-ordering method

In addition to the layout optimization method, another way to reduce the thermal effect is to change the weight distribution of memristor crossbar [7], [12]. In this section, we firstly propose two important re-ordering observations in Sec. IV-A, which can improve the re-ordering performance. Then, we present a novel weight re-ordering work flow to reduce the thermal effect by considering the input distribution and the exchange among memristor arrays. Our work flow is divided into 3 parts, initial thermal map construction in Sec. IV-B, spiking analysis in Sec. IV-C and re-ordering method in Sec. IV-D.

A. Motivation

In order to reduce the thermal effect, one important step implemented by previous works is to map the neural network's weights to memristor cells with different arrangements. However, we find that the problem formulation of previous methods cannot get good enough mapping performance. We present two important observations to improve re-ordering performance in memristor arrays.

1) The first observation of re-ordering is to consider both the input distribution of each memristor array and the weights value.

As the neural networks are trained for the specific tasks, the input of these networks is more likely to have the characteristic of these tasks. For example, the input of a face recognition network is more likely to be a face, which means that the convolution kernel that detects the eye will generate more spikes than which detects the nose. This difference of spikes will be transferred to the next layer due to the sequential structures in the neural network. Therefore, the power consumption of next layer will be affected. This observation shows that the power of the memristor array depends not only on the weight value, but also on the input distribution of general case input. The relationship between power, input and weight is shown in Eq. (4).

$$P_i = V_i^2 * G_i \propto Input_i^2 * Weight_i \tag{4}$$

P_i, V_i and G_i are the power, input voltage and conductance matrix of memristor array of layer i. $Input_i$ and $Weight_i$ represent the input and weight matrix of layer i. In memristor arrays, the input is the voltage and the weights are programmed into memristor conductance. We use Kirchhoff's law to get the MAC results.

2) The second observation of re-ordering is to consider the re-ordering in different memristor arrays.

The reason is that previous mapping methods only do row or column re-ordering for the whole weight of one layer or within a memristor array. We call this method 'in-array re-ordering method'. However, it ignores the fact that the memristor array cannot be infinite large [15] and the weights are always represented by several memristor arrays [13]. Therefore, the re-ordering process not only can be implemented in the whole weight, but also can be implemented among different memristor arrays. We need to execute the row or column re-ordering between arrays to search larger solution space and get better results. We call this method 'cross-array re-ordering method'.

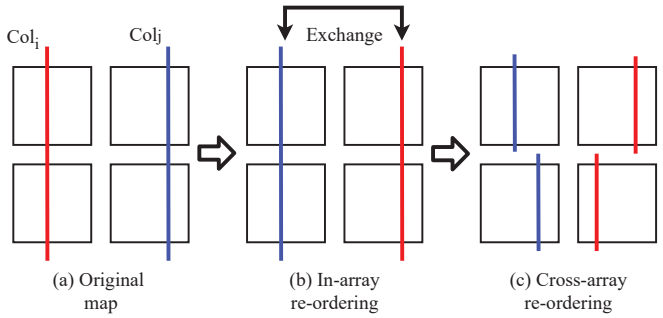

Fig. 3. The column re-ordering difference.

Fig. 3 shows the difference between the 'in-array' method and our method. We assume that the weight matrix can be represented by 4 memristor arrays, which are the four squares shown in Fig. 3(a). Col_i (the red line) and Col_j (the blue line) means the i^{th} and j^{th} column of W_0. Then the 'in-array' method in Fig. 3(b) can only exchange the sequence of the whole column, but the 'cross-array' method in Fig. 3(c) has the ability to exchange with other columns within each memristor array. Therefore, the 'cross-array' method has more solution space to get better re-ordering results than 'in-array' method.

B. Construction of initial thermal map

In order to determine the mapping method, we firstly construct the initial thermal map of each tile. The reason is that different states of memristor have different sensitivity to the temperature, as shown in Fig. 2(a). The LRS changes a lot when temperature increases while the HRS almost remains. Therefore, we need to execute weight re-ordering based on the initial thermal map to avoid mapping a large weight to a high-temperature region.

Suppose the ambient temperature and layout of each tile are denoted as $T_{ambient}$ and $Layout_0$, respectively. The power of components except the memristor array in each tile is denoted as P_0. Then we use Hotspot [16] to simulate the grid steady temperature map TM_0 and get the initial temperature condition $\mathbf{T_{init}}$ of each memristor array.

C. Input distribution analysis

After getting the initial thermal map, we analyze the spike input condition of memristor arrays in each layer of SNN. Fig. 4 shows one inference step in SNN from layer i to layer $i+1$, and it explains why we need to analyze the input value. Map1 and Map2 mean the feature maps (or output) of layer i. C_1 and C_2 represent the input vectors to the memristor weights W_1 and W_2 of layer $i+1$, respectively. C_1 comes from output feature map MAP_1 of layer i, and is the same as C_2.

Since different kernels extract different features, the output results of these kernels are also different. Here, we suppose

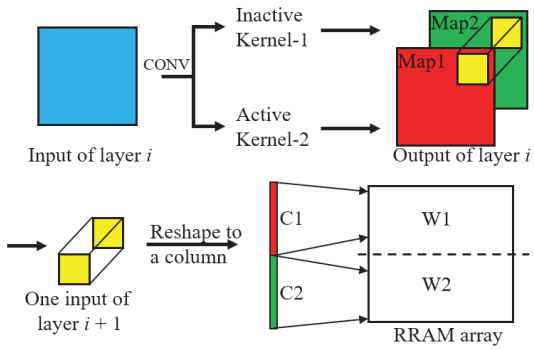

Fig. 4. One inference step in SNN from layer i to layer $i+1$.

that layer i only has two kernels which are active Kernel-1 and inactive Kernel-2. Kernel-1 generates more output spikes than Kernel-2. Then we find that in layer $i+1$, C_1 will have more spikes than C_2. This condition will transmit to the next layer due to the sequential structure of neural network. Assume that the weight matrix W_1 is equal to W_2, then based on Eq. (4), the crossbar which is mapped with W_1 consumes more power than that of W_2.

Therefore, it is necessary to collect the input distribution of all layers. Suppose that our network has n layers, the input spike of i^{th} layer can be represented by a vector $K_i, i = 1, 2, \cdots, n$. Their values are collected by feeding all training set to the well-trained model and counting the input spikes of each layer. Here we assume that the training sets of over $50,000$ images can represent the general input case (or we can create an input dataset to represent the general input case).

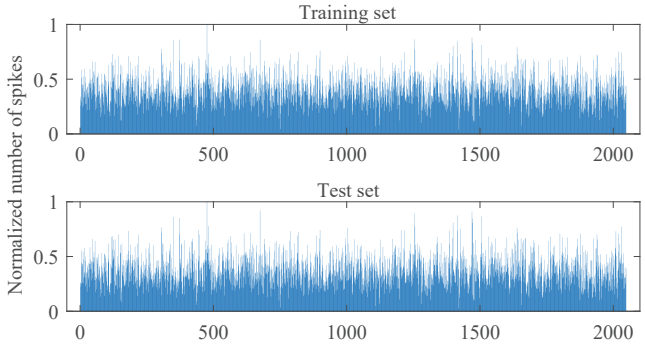

Fig. 5. The normalized input distribution of the second fully-connected layer of VGG11 in training set and test set.

Fig. 5 shows the normalized input distribution of the 2^{nd} fully-connected layer in $VGG11$ for both training and test sets. The x-axis of this figure is the input sequence, we reshape the input sequence to vector to show the input distribution of this layer. The y-axis is the normalized number of spikes received by this layer. From this figure we can find that the input spikes differ within one layer, and the difference of the spike distribution between the training and test sets are small.

D. Input-contained re-ordering method

1) Re-ordering process: After the initial thermal and input spike analysis in Sec. IV-B and Sec. IV-C, we get the initial thermal map T_{init} and the input spike situation K of all the memristor arrays. In this part, we propose a novel re-ordering algorithm to further reduce the thermal effect. The process of our algorithm is shown in Alg. 1.

The input of this algorithm is the initial thermal map T_{init}, input spikes K, weight matrix W and initial weight location L. Suppose that our network has n layers, the K, W and L are all dictionaries with n elements.

The process of our re-ordering method can be divided into 3 parts. The first part is the pre-processing part. We use the input spikes generated by training sets to estimate the real input distribution of each layer. Then, we use this estimation to get the input-contained weight, as shown in Line 3. Symbol $\bigotimes$ represents a new multiplication operator and Eq. (5) is an example of this operator.

$$\begin{bmatrix} x \\ y \end{bmatrix} \bigotimes \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ax & bx \\ cy & dy \end{bmatrix} \tag{5}$$

After the pre-processing part, we get the input-contained weight W_{bias}, then we use it to do weight re-ordering by 'cross-array' method and get the re-ordering weight and location matrices W_{re} and L_{re}. The detailed re-ordering algorithm is shown in Sec. IV-D2 and Alg. 1-RE-ORDER.

The third part is the mapping part. We divide the input-contained weight W_{bias} into N small weight matrices w with $k \times k$ size, and map them to the array. The mapping rule is that for all w_i, $i = 1, 2, \cdots, N$, the w_i with larger sum matches the lower temperature area. The results are stored in Map.

In general, we get the final mapping result, and program the weight into memristor crossbar by Map. The performance is shown in Sec. V-B.

2) Re-ordering algorithm: In order to illustrate our proposed algorithm, we first model the problem mathematically. Since the thermal problem highly depends on the power distribution, here we want to even the power distribution to reduce the peak temperature. The problem model is formulated in Eq. (6).

$$\min \quad (max(P) - min(P)) \tag{6}$$

$$\text{s.t.} \quad \textit{Every row (or column) in } w_i \textit{ must be in the same row (or column) of origin weight;}$$

Here, $P = \{p_1, p_2, \cdots, p_N\}$ and $p_i = w_i.sum(), i = 1, 2, \cdots, N$. P represents the set of power of all small input-considered weight, and p_i represents the total power of i^{th} small input-considered weight. These two constraints aim to make sure the mapping results satisfy the rule of MAC operation in memristor array.

Since this problem is at least an NP-Complete problem, we use a heuristic method to solve it. The algorithm is shown in Alg. 1-RE-ORDER and Fig. 6.

Fig. 6 shows the re-ordering algorithm flow and an example of how we execute the column exchange. Suppose that the W_{bias} can be divided into four parts as shown in Fig. 6(a), we then execute row exchange in W_{bias}, trying to reduce the difference between each part(Alg. 1-Line $11-13$ do the coarse exchange and Line $14-18$ do the finer exchange). After that, we divide each part into several smaller blocks as shown in Fig. 6(a)-Step 2, and do column exchange to reduce the difference among these blocks. The same operation is also performed in these blocks and the blocks of these blocks, etc. Our method ends the iteration after executing Step 2 for a

Algorithm 1 Re-ordering Process

Input: Initial thermal map T_{init}; Input spikes K; Weight matrix W; Initial weight location L; Array size D;

Output: Re-ordering weight matrix W_{re}; Re-ordering location matrix L_{re}; Weight-to-memristor map Map

1: **function** PROCESS(T_{init}, K, W, L);
2: % Pre-processing, get the input-considered weight
3: $W_{bias\_i} \leftarrow K_i \bigotimes W_i$ for all i;
4: % Re-ordering
5: W_{re}, L_{re} = RE-ORDER($W_{bias}, L, Param$)
6: % Mapping process
7: $Map \leftarrow$ Based on L_{re} and T_{init}, map weights to the MCA by order;
8: **return** W_{re}, L_{re}, Map
9: **end function**
10: **function** RE-ORDER($W_{bias}, L, Param$);
11: $Sum \leftarrow$ sum of rows or cols in W, the selection of row or col depends on $Param.type$;
12: $Sort \leftarrow$ sort the Sum with descending order;
13: $Part \leftarrow$ Divide the W_{bias} into several parts, and map each row or col to these parts by the order of $Sort$, as shown in Fig. 6(b).
14: **for** $i : 1 \rightarrow Param.iter$ **do**
15: Find $Part_{max}$ and $Part_{min}$, which have the largest (smallest) sum value of $Part$.
16: $diff \leftarrow Part_{max} - Part_{min}$;
17: Find 1 row or col in $Part_{max}, Part_{min}$, respectively, and their difference is close to $diff/2$, exchange them.
18: **end for**
19: Repeat line $11-18$ for each part and finer re-ordering as shown in Fig. 6(a).
20: **end function**

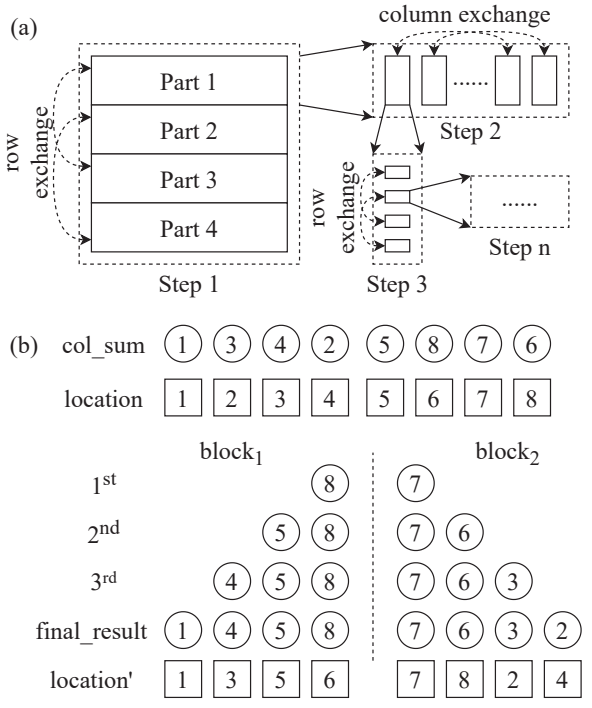

Fig. 6. Illustration of proposed re-ordering method.

minimum block size 128×128, but this algorithm is suitable for finer blocks.

Fig. 6(b) shows an example of column exchange. Suppose that a weight matrix has 8 columns, we first compute the sum of each column in col_sum and define the origin location vector $location$. Here, we try to divide the W_{bias} into 2 blocks. We firstly pick the largest 2 elements in col_sum and send them into these 2 blocks ($8 \rightarrow block_1$ and $7 \rightarrow block_2$). Then, we pop another 2 largest elements in col_sum and send it to these 2 blocks in an opposite order compared to the former step ($5 \rightarrow block_1$ and $6 \rightarrow block_2$). Repeating this step, we get a coarse re-ordering result. The $block_1$ contains $1, 4, 5, 8$ and $block_2$ contains $7, 6, 3, 2$. This result means $block_1$ and $block_2$ contain the $1^{st}, 3^{rd}, 5^{th}, 6^{th}$ and $7^{th}, 8^{th}, 2^{nd}, 4^{th}$ columns in the origin weight, respectively. We output the combination of $block_1$ and $block_2$, and the $location'$ as the re-ordering result (W_{re}, L_{re}).

V. Experimental results

A. Methodology

In this section, we evaluate our proposed thermal-aware optimization framework, based on the practical configuration of memristor-based DNN accelerator with 128×128 array size [13]. In order to satisfy the ISAAC working conditions [13], here we set the resolution of each memristor cell to 2 bits. Under this resolution, the workflow of ISAAC can be satisfied. The working frequency is set to $1.2GHz$ [13] and the resistance range of the memristor is $[5K\Omega, 500K\Omega]$ with the input voltage range $[0V, 0.9V]$. We simulate the temperature by the thermal simulator Hotspot [16]. Besides, we assume that the memristor-based accelerator is fully running and the computing of convolutional layers is parallel. In this simulation, we also assume that the tile-to-tile effect can be ignored. Other detailed parameters of Hotspot including heat sink specification are set to the same as previous works [7], [9] and the ambient temperature is set to 300K [12]. In addition, the area of a single tile for both ISAAC and our method is the same ($0.4055mm^2$).

We validate our method on two modern SNN models transformed from $VGG9$ and $VGG11$, using PyTorch framework [17] in CIFAR10 (for $VGG9$) and CIFAR100 (for $VGG11$) dataset [18]. These SNN models are transformed by ANN-to-SNN conversion and STDB (Spike Timing Dependent Backpropagation) methods implemented in [1]. The design configuration for evaluation is shown below:

- *Base*: The baseline of hardware-based neuromorphic structure that maps the neural network weight to the memristor cells.
- *MA*: Our proposed layout modification scheme with discentralized memristor arrays.
- *TOPAR-C*: The column re-ordering method [12] by balanced largest first differencing method (BLDM).
- *MP-FLP*: This method includes all proposed methods (layout modification and re-ordering methods).

B. Performance

Fig. 7 indicates the normalized power range and the average power range reduction of different layers of $VGG11, VGG9$ networks. After mapping the re-ordered weights to each

memristor array, we calculate the average power of each array, and record their power ranges (i.e. maximum power - minimum power) with different methods. The height of these bars in Fig. 7(a) denotes the normalized value of these ranges. From this figure we can find that our method gets better performance on both $VGG11$ and $VGG9$ compared to TOPAR-C. Fig. 7(b) compares the performance of our method between convolutional (CONV) and fully-connected (FC) layers. We can find that our method can shorten the power range on an average of $\sim 20\%$ in CONV and $\sim 15\%$ in FC.

Fig. 8 shows the temperature distribution of the hottest tile in the third fully-connected layer of $VGG9$. From this figure we can find that (1) the thermal distributions in *MA* and *MP-FLP* methods can make the temperature distribution more well-balanced than the *Base* and *TOPAR-C* methods, (2) The *MP-FLP* method can reduce the peak temperature effectively.

The peak temperature distributions of different methods in $VGG11$ and $VGG9$ are shown in Fig. 9(a). We can find that our method can reduce more peak temperature than *TOPAR-C*. The peak temperature reduction for $VGG11$ and $VGG9$ are $5.0K$ and $10.4K$, respectively. This reduction is larger than what *TOPAR-C* does–$2.1K$ and $4.2K$ reduction in $VGG11$ and $VGG9$, respectively. This reduction of peak temperature prolongs the endurance of memristor crossbar [9], [12].

For endurance evaluation, we use memristor with 4.14×10^8 possible number of writes at $300K$ and validate endurance enhancement in terms of the total amount of possible writes under different temperature. Since the lifetime of memristor is defined as the first failure time with the highest temperature, we evaluate endurance enhancement with peak temperature [12]. The endurance comparison is shown in Fig. 9(b). Compared to the *Base* situation, our method can improve the endurance of memristor by $1.30\times$ and $1.72\times$ in $VGG11$ and $VGG9$, respectively.

VI. CONCLUSION

Temperature influences the conductance of memristors, which is used to program the weight of SNNs in resistive neuromorphic engines. In this work, we propose a cross-array mapping method and a layout optimization method to reduce the thermal effect with the consideration of input distribution, weight value and layout of memristor crossbar. Experimental results show that our method can reduce the peak temperature up to $10.4K$ and improve the endurance up to $1.72\times$.

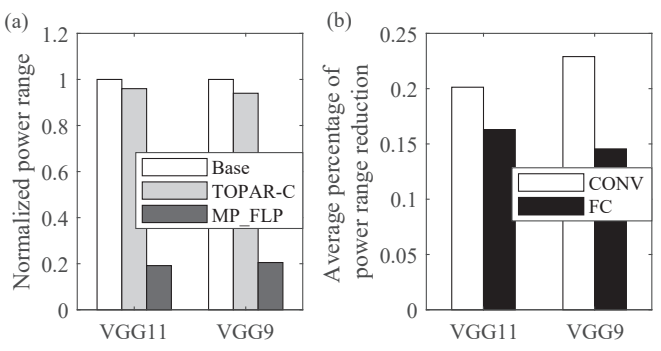

Fig. 7. (a) The normalized power range. (b) The average percentage of power range reduction.

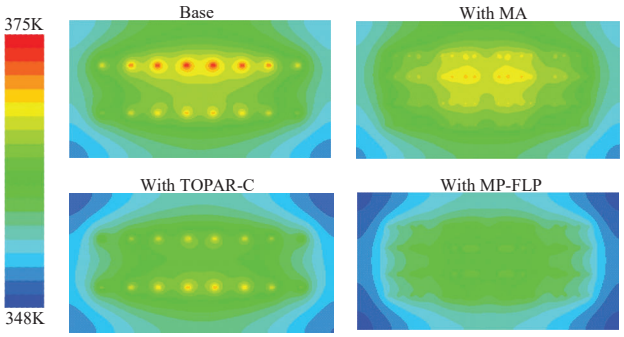

Fig. 8. The temperature distribution of the hottest tile in the third fully-connected layer of $VGG9$.

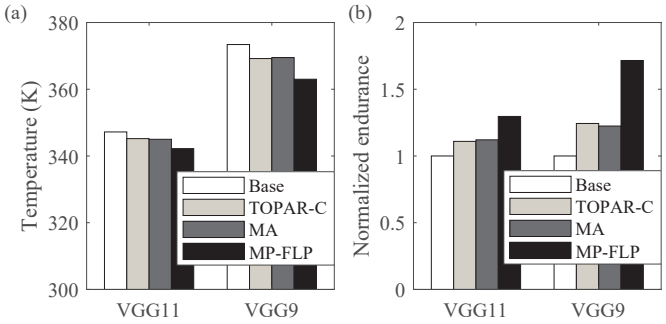

Fig. 9. (a) Peak temperature. (b) Normalized endurance.

REFERENCES

[1] N. Rathi *et al.*, "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," *arXiv*, 2020.

[2] S. Zhang *et al.*, "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects," in *AICAS*, 2020, pp. 11–15.

[3] Y. Ma and P. Zhou, "Efficient techniques for training the memristor-based spiking neural networks targeting better speed, energy and lifetime," in *ASP-DAC*, 2021, pp. 390–395.

[4] C. Zhang and P. Zhou, "A quantized training framework for robust and accurate reram-based neural network accelerators," in *ASP-DAC*, 2021, pp. 43–48.

[5] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *DATE*, 2017, pp. 19–24.

[6] C. Walczyk *et al.*, "Impact of temperature on the resistive switching behavior of embedded HfO_2-based rram devices," *TED*, pp. 3124–3131, 2011.

[7] M. V. Beigi and G. Memik, "Thermal-aware optimizations of reram-based neuromorphic computing systems," in *DAC*, 2018, pp. 1–6.

[8] X. Liu *et al.*, "HR^{3}AM: A heat resilient design for rram-based neuromorphic computing," in *ISLPED*, 2019, pp. 1–6.

[9] M. V. Beigi and G. Memik, "THOR: Thermal-aware optimizations for extending reram lifetime," in *IPDPS*, 2018, pp. 670–679.

[10] S. Zhang *et al.*, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *DATE*, 2019, pp. 1751–1756.

[11] G. L. Zhang *et al.*, "Robustness of neuromorphic computing with rram-based crossbars and optical neural networks," in *ASP-DAC*, 2021, pp. 853–858.

[12] H. Shin *et al.*, "A thermal-aware optimization framework for reram-based deep neural network acceleration," in *ICCAD*, 2020, pp. 1–9.

[13] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, pp. 14–26, 2016.

[14] E. Vatajelu *et al.*, "Special session: Reliability of hardware-implemented spiking neural networks (SNN)," in *VTS*, 2019, pp. 1–8.

[15] X. Qi *et al.*, "Fault tolerance in memristive crossbar-based neuromorphic computing systems," *Integration*, pp. 70–79, 2020.

[16] W. Huang *et al.*, "Hotspot: A compact thermal modeling methodology for early-stage VLSI design," *TVLSI*, pp. 501–513, 2006.

[17] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv*, 2019.

[18] Krizhevsky *et al.*, "CIFAR-10 and CIFAR-100 datasets," 2009.