Learning the Sparsity for ReRAM: Mapping and Pruning Sparse Neural Network for ReRAM based Accelerator

Jilan Lin[1,2], Zhenhua Zhu[2],Yu Wang[2] and Yuan Xie[1]
[1]Dept. of Electrical and Computer Engineering, UCSB, Santa Barbara, U.S.
[2]Dept. of Electronic Engineering, Tsinghua University, Beijing, China

***Abstract*— With the in-memory processing ability, ReRAM based computing gets more and more attractive for accelerating neural networks (NNs). However, most ReRAM based accelerators cannot support efficient mapping for sparse NN, and we need to map the whole dense matrix onto ReRAM crossbar array to achieve $O(1)$ computation complexity. In this paper, we propose a sparse NN mapping scheme based on elements clustering to achieve better ReRAM crossbar utilization. Further, we propose crossbar-grained pruning algorithm to remove the crossbars with low utilization. Finally, since most current ReRAM devices cannot achieve high precision, we analyze the effect of quantization precision for sparse NN, and propose to complete high-precision composing in the analog field and design related periphery circuits. In our experiments, we discuss how the system performs with different crossbar sizes to choose the optimized design. Our results show that our mapping scheme for sparse NN with proposed pruning algorithm achieves $3-5\times$ energy efficiency and more than $2.5-6\times$ speedup, compared with those accelerators for dense NN. Also, the accuracy experiments show that our pruning method appears to have almost no accuracy loss.**

I. Introduction

While the neural network (NN) gains unprecedented success and becomes the hottest topic in the machine learning area, the huge demand of computation resources and limited memory bandwidth restrict its development. With the ability of in-memory computing which can break the memory wall in the von-Neumann architecture, Resistive Random Access Memory (ReRAM) [1] provides a promising alternative for accelerating NN. The crossbar structure of ReRAM can efficiently reduce the computation complexity of the matrix-vector multiplication from $O(n^2)$ to $O(1)$ and eliminate the parameter fetching procedure. Therefore, designing ReRAM based NN accelerators attracts lots of researchers' attentions [2]–[4].

However, most ReRAM based accelerators focus on the mapping for regular dense NN instead of the pruned sparse NN. Learning the sparsity of NN through weights pruning [5] is a powerful method of NN compressing, which significantly reduces the redundant parameters and bring considerable relief from the pressure of storage and bandwidth. And some work leveraged the sparsity to tolerant the defects in ReRAM devices [6]. However, to achieve the $O(1)$ complexity of matrix operations, the proper order of column and row must remain in the crossbar structure, meaning that we still need to store the sparse matrix in the dense way. As Fig. 1(b) shows, the gray cells are at High Resistance State (HRS) and representing zero values, while black cells in Low Resistance State (LRS) represent non-zero values. This makes the effort of network pruning in vain because zeros in NN cannot be eliminated.

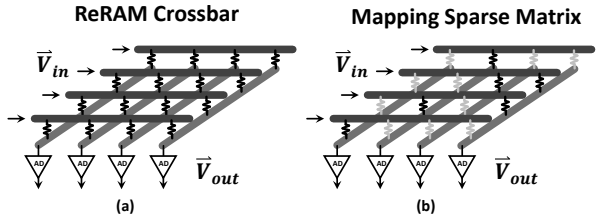

Fig. 1. (a) Matrix-vector computing based on ReRAM crossbar array. (b) Mapping sparse matrix on ReRAM crossbar array with lots of ReRAM cells at HRS and representing zero (shown as grey cell).

Overcoming the intrinsic contradiction between dense crossbars and the sparse matrix is extremely difficult. Researchers have made some exploration to make the mapping of sparse matrix more efficient. For graph processing, Song tried to using ultra small ReRAM processing element (PE), like 8×8 or 4×4 crossbar, to traverse the adjacency matrix [7], which, however, will consume lots of energy for large scale NN, because it has to read/write the weights repeatedly. Wang focused on mapping structured sparse NN [8] with block building approach [9], but not looking into common irregular sparse NN, which can prune more redundant parameters. Besides, some work proposed to train a sparse NN that fits the hardware structures of ReRAM crossbar [10]. It may be a good way but still cannot deal with the mapping problem for exist sparse NN.

In addition, the model quantization is another remarkable compressing approach along with the sparsity. Since current ReRAM devices cannot achieve high precision fixed-point value [1], mapping quantized NN is also necessary for ReRAM based accelerators. Previous work proposed to split high-precision value in several low-bit crossbars [2], [11]. However, such precision composing method will bring too much interface cost since they choose to complete the precision composing procedure in digital part and every split ReRAM crossbar need separate analog-digital converters.

In this paper, we propose a novel mapping scheme for sparse NN, along with software-level pruning algorithm and hardware extension for low-cost high-precision computation. The main contributions of this work are as follows:

1) We propose a sparse NN mapping scheme based on k-means clustering, which shuffles the column in weight matrix and eliminates all-zero crossbars.
2) We propose crossbar-grained pruning algorithm to reduce the crossbars with low utilization and rescue the accuracy loss by retraining NN.
3) We analyze how quantization precision influences the

This work was supported by the Project of Science and Technology on Reliability Physics and Application Technology of Electronic Component Laboratory (No.61428060401162806001), the National Key Research and Development Program of China (2017YFA0207600), and NSF 1719160, 1725447, 1730309.

ASPDAC '19, January 21–24, 2019, Tokyo, Japan

ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3287715

accuracy of sparse NN, to find the optimized computing precision for ReRAM. We propose to complete bit-composition in the analog field to reduce the cost of analog/digital interfaces and design related circuits.

4) The simulation results show that our mapping scheme with proposed pruning algorithm achieves $3-5\times$ energy efficiency and $2.5-6\times$ speedup, compared with PRIME. Also, the accuracy experiments show that our pruning method appears to have almost no accuracy loss.

II. Preliminary and Motivation

In this section, we will introduce the background of sparse neural network and ReRAM crossbar based computing. And then we will present our motivation in detail.

A. Sparse Neural Network

As a type of machine learning algorithm, NN typically consists of input/output layers and multiple hidden layers. The calculation of each layer can be expressed as follows:

$$\mathbf{y} = f(W^T\mathbf{x} + \mathbf{b}) = f(\sum_{i=1}^{n} W_i x_i + \mathbf{b}) \tag{1}$$

where $\mathbf{x}, \mathbf{y}$ are the input and output data, respectively. W is the weight matrix and $\mathbf{b}$ is the bias. f is the activation function.

Current NN often contains enormous parameters, which brings huge storage and computation pressure. Therefore, sparsity becomes an efficient approach to compress the model. By weights and neurons pruning, the density, or the parameters of the NN model will be reduced about $10\times$, with very small or even no accuracy loss [5], [8].

B. RRAM and RRAM Computing System

As a non-volatile memory, ReRAM stores information with resistive cells. Multiple ReRAM cells construct the crossbar structure, which can be used to perform analog matrix-vector multiplication with reducing the computation complexity from $O(n^2)$ to $O(1)$ [12], as shown in Fig.1 (a). The relationship between input voltage vector (V^i) and the output voltage vector (V^o) can be expressed as follows:

$$\begin{bmatrix} V_1^o \\ \vdots \\ V_M^o \end{bmatrix} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,N} \\ \vdots & \ddots & \vdots \\ c_{M,1} & \cdots & c_{M,N} \end{bmatrix} \begin{bmatrix} V_1^i \\ \vdots \\ V_N^i \end{bmatrix} \tag{2}$$

where $c_{i,j}$ is the parameter to be mapped to ReRAM cell. Note that the conductance of ReRAM cell can only represent positive value, two ReRAM crossbars are required to represent a matrix with both positive and negative parameters.

With the advantages of efficient in-memory computing ability, previous work has proposed several ReRAM-based Computing Systems, like PRIME [2] and ISAAC [3].

C. Motivation

In ReRAM crossbar based computing systems, to remain the $O(1)$ computation complexity of the matrix-vector multiplication, we must map the whole matrix into the crossbar as Eq. 2 shows even though there may be lots of zeros, otherwise it will cause unpredictable computing error.

For the sparse NN, which contains plenty of zeros in weight matrix (usually >70% sparsity), the mapping is still the same as that of the dense NN. The '0's shall occupy their corresponding positions to make the parallel voltage inputs added into correct cells and receive expected output results, as shown in Fig. 1(b). Therefore, we may say that the crossbar structure ruins almost every advantage that the sparsity brings, because the redundancy of zero elements still exists in the ReRAM crossbar. To deal with the contradiction between the dense structure of ReRAM crossbar and the sparsity of weight matrix, a software and hardware co-optimization scheme is needed for implementing sparse NN on ReRAM.

III. Sparse NN Mapping Scheme

To reduce the redundancy in ReRAM crossbar and make use of the sparsity of NN, we next introduce our mapping scheme for sparse NN with as fewer crossbar arrays used as possible. We first introduce the observed facts that motivate us to propose column exchanging based mapping algorithm. After, we will explain the mapping procedure in detail and discuss the overhead it causes.

There are two observations that help us explore the efficient mapping design. First, to achieve a higher accuracy in complicated tasks, the development of NN is going deeper and larger-scale. In common-used neural networks, the weight matrix can be very large. For instance, the first full connection layer of VGG-16, which contains more than 90% parameters of the whole network, has the weight matrix with the size of 25088×4096. Obviously, such a massive matrix cannot be mapped on one single ReRAM crossbar and needs to be split into smaller blocks. Meanwhile, in the current tape-out chip of ReRAM based computing systems, the size of fabricated ReRAM crossbar is quite small, like 32×32 or 64×64 [13], [14]. The second observation is that for those extremely large layers, although there are massive parameters, the pruning results show that these layers will be really sparse. Again, take above FC layer in VGG-16 for example, after pruning, the density of it is 4% [5], which means that only 4% elements remain and 96% of the matrix is zero. Further, since we need two crossbars to represent positive and negative part of the matrix, the density will be half smaller if half parameters are positive and the others are negative, which means about 98% parameters of the matrix become zero.

The observations inspire us that those smaller blocks of the split weight matrix are probably all-zero, or have all-zero columns/rows. If we eliminate such zero parts, we could save considerable resources. However, for the reason mentioned in Section II-C, even there is only one non-zero element in the column/row of ReRAM crossbar, we need to keep this column/row to make the computation of matrix in the correct order. Here we proposed the k-means clustering based column exchanging scheme for mapping. In fact, some work applied the spectrum clustering algorithm to gather the neurons for pruning [10]. But here our goal is to make the non-zero elements more

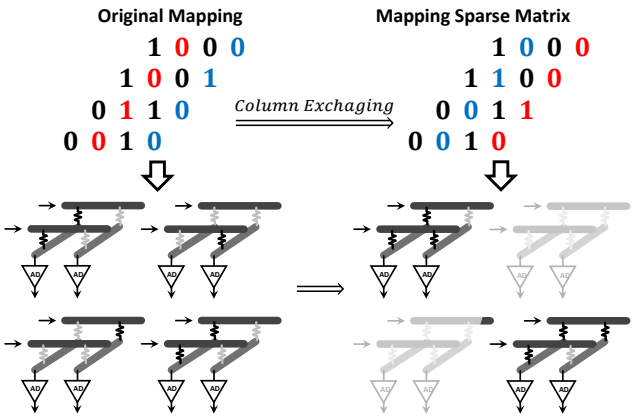

Fig. 2. A illustration of our column exchanging based mapping algorithm. (a) Column exchanging for sparse matrix. (b) The whole mapping scheme

concentrated through column exchanging. Thus we can gain more all-zero rows and save more crossbars.

k-means is a wide-used method for cluster analysis in data mining. Given n samples, k-means algorithm aims to cluster them into k labels, with the minimal within-cluster variance. In our application, we have n columns in the original large matrix and crossbar size is L, so we need to partition the n columns, or say n vectors into n/L labels, with L vectors in each label. k-means algorithm will not assign specific number of samples in each label, so our solution is to take the most concentrated L vectors as clustering results, and then through a recursive procedure we repeat the clustering for remaining vectors. Also, since k-means is sensitive to initial value, we choose to repeat our algorithm to avoid local optimal results.

Algorithm 1 shows our complete mapping scheme which requires weights matrix, ReRAM crossbar size and a pre-splitting function f_{split} as input. Clustering the extremely large matrix, like 25088×4096, is never a wise decision, because it consumes much more time and will not achieve better results for smaller pieces. Therefore, we propose to pre-split the matrix before k-means clustering as shown in line 1-2 with the pre-splitting function f_{split}. Line 3-16 describe our key operation for k-means based column exchanging. We cluster the vector in split matrix and pick up L columns into the target set R repeatedly, and through the recursive calls we will shuffle all the columns into specific sets we need. After we exchange the columns according to clustering results, the size of the split matrix will shrink as we eliminate the all-zero parts. Then we can further split the shrunk matrix into crossbar size and finish the mapping procedure as described in line 17-20.

Mention that we use a splitting function f_{split} to decide the split size, and here is how it works: We first produce plenty of random matrices to start different splitting strategies and choose the best one by iteration. Through those experienced parameters, we infer the splitting size for a given matrix. So in our experiments the f_{split} is actually an interpolating function.

Fig. 3 shows our system skeleton. The flowchart of our system is similar to PRIME but we add extra indexing units, since we disorder the columns and rows of a regular matrix. The index registers are only added for the output vectors because

Algorithm 1 k-means base column exchanging for mapping sparse weight matrix.

Require: Weight matrix W, splitting function f_{split}, ReRAM crossbar size L.

1: Get the matrix size (x, y) of W. Let $ext = y \bmod L$, and add ext zero columns for W with length of x;
2: Get the sparsity sp of W, and the splitting size $(x_{split}, y_{split}) = f_{split}(x, y, sp)$. Split W into m small blocks W_i;
3: Initialize clustering result set $R_i = \{\}, i = 1, 2, ..., m$, which will store the tuples of columns in a crossbar;
4: **for** $i = 0$ to m **do**
5: Initialize unclustered set $U = \{v\}$, where v is each column in W_i;
6: **while** $U! = \emptyset$ **do**
7: $n = |U|$;
8: Cluster U into n subset C_j using k-means with hamming distance;
9: **for** $j = 0$ to n **do**
10: **if** $|C_j| \geq L$ **then**
11: Find the nearest L vectors $\{v\} \subset C_j$;
12: Add $\{v\}$ into R_i, and remove them from U;
13: **end if**
14: **end for**
15: **end while**
16: **end for**
17: **for** $\{v\}$ in R_i **do**
18: Compose the block matrix $W_{i,block}$ with $\{v\}$;
19: Eliminate all-zero parts and get $W_{i,block}$ shrunk, then map it into ReRAM crossbar;
20: **end for**

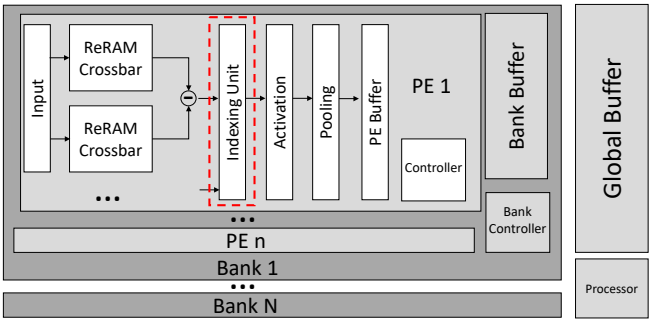

Fig. 3. The skeleton of our system flowchart.

we can schedule the corresponding input voltages after NN mapping. Note that the index units are needed for ReRAM crossbars but not every matrix element. Besides, Due to the pre-splitting procedure, we do not need to index among the huge matrix but inside the split small block.

IV. Crossbar-Grained Pruning

In the section above, we introduce our proposed sparse mapping scheme based on column exchanging strategy, which will gather non-zero elements together as concentrated as possible in order to eliminate the zero crossbars. However, we could still find that in some crossbars, the utilization of

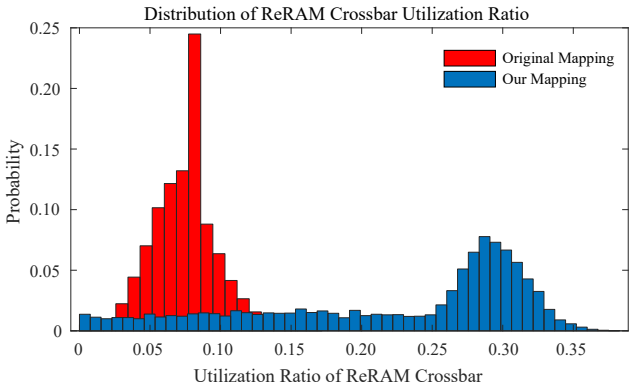

Fig. 4. The distribution of utilization ratio for ReRAM crossbar, with mapping a sparse VGG-16.

ReRAM cells is much lower, and there may exist only few, or even just one non-zero element(s) in one row. We implement our mapping algorithm on 64×64 ReRAM crossbar for VGG-16. Fig. 4 shows the distribution of utilization ratio of those crossbars, compared with original mapping in the dense way. The percentile in x-axis represents the ratio for non-zero cells in the ReRAM crossbar. We can easily find that in more than 20% crossbars, there are only 15% ReRAM cells storing valid parameters and 85% of them are zero.

Obviously, if we can further delete those low utilization crossbars, we can achieve much more hardware resource reduction. Here we will introduce our crossbar-grained NN pruning to further compress the sparse NN and save ReRAM crossbars. The key idea is to throw away those ReRAM crossbars with only few parameters. First, we implement column exchanging for NN mapping according to Section III. Then we start crossbar-grained pruning for NN, which will remove multiple weights in the crossbar rows. We adopt the pruning criterion in Mao's work [15]: Compute the Salience $S_i = \sum_{w \in G_i} |w|$, i.e. the sum of L1-norm weights, to decide which group of weights should be deleted. Here the pruning grain G_i is the ReRAM crossbar rows. After weights pruning, we finetune the NN to rescue the accuracy, expecting the least accuracy loss. To achieve better pruning results, we can repeat the procedure of "Pruning - Finetuning - Pruning", and get the final NN model step by step.

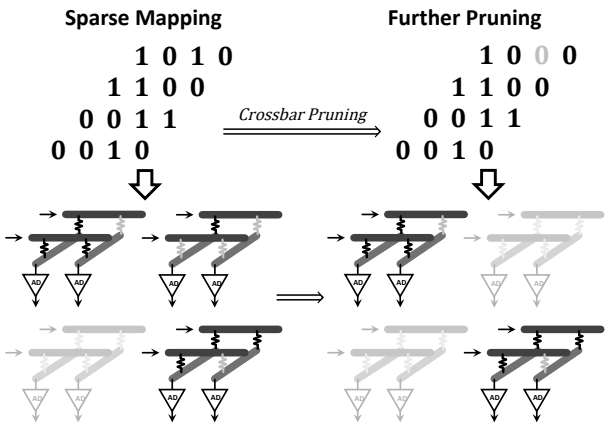

Fig. 5. ReRAM crossbar-grained pruning for sparse NN.

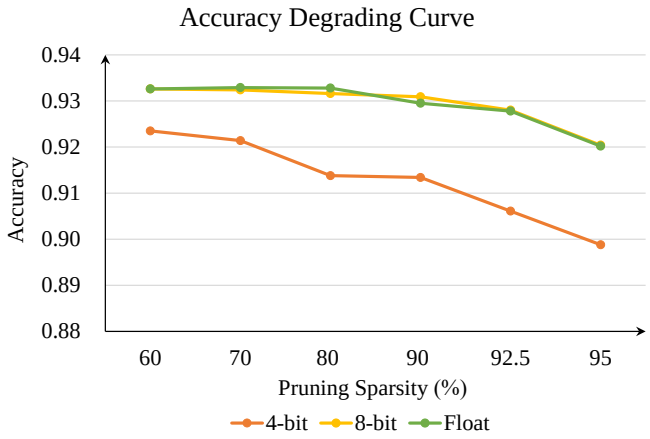

Fig. 6. The curve showing the accuracy degrading with gradually deeper NN pruning sparsity, and different quantization precision.

Finally, mention that our pruning algorithm is conducted on sparse NN, where the model is already compressed. Therefore, the further pruning may damage the NN performance on accuracy. In our experiments section, we will discuss the above issue in detail.

V. Precision Composition

Quantization for NN often accompanies weights pruning, which is another powerful method for NN compression [16]. Quantizing NN will reduce the storage with low-bit weights, and computing fixed-point operations is easier from hardware level compared with float operation. Besides, since current ReRAM devices are difficult to represent full-precision or high-precision value [1], [17], quantizing the NN and composing high-precision operation with multi crossbars are quite necessary. Obviously, the quantization will lead to accuracy loss and this impact will be severer for sparse NN. Previous work, like PRIME, has some discussions on the precision for ReRAM based architectures, but it was not meant for sparse NN and the precision composition cost too many interfaces. In this section, we first explore how NN precision affects the accuracy for sparse NN, and then we will introduce our precision composing circuits which computes the results in the analog field and minimizes the interface cost.

First, we evaluate the accuracy degrading caused by quantization with different degrees of sparsity. Fig. 6 plots the accuracy results, according to our experiments of VGG-16 on CIFAR-10 dataset, with quantization of 4-bit, 8-bit and float precision. From Fig. 6 we can observe the degrading curve of accuracy with deeper pruning. We find that under 8-bit precision, the degrading curve is almost the same as the curve under full precision, which achieves no accuracy loss with $\leq$90% pruning and $<$1% accuracy loss with $\leq$95% pruning. At the same time, the accuracy is lower even at the beginning phase of pruning as we use 4-bit precision, and it starts to degrade badly with 90% pruning. Therefore, 8-bit precision is enough to represent the whole NN.

Here we will introduce our circuits design for precision composing. The key idea is that we decompose the high

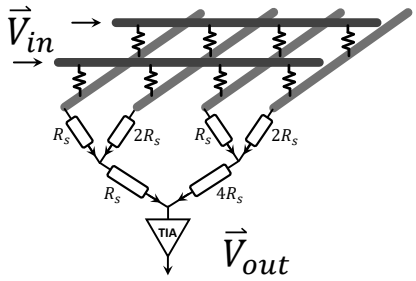

Fig. 7. The tree circuits design for bit composing.

precision value among a single crossbar and finish the bit carrying in the analog field. Taking 1-bit ReRAM crossbar for instance, we map the 8-bit value in 8 columns, instead of 8 crossbars, like the storage mode in traditional memory. Then after the voltage inputs, the output currents from each column will represent the computing result in different binary bits. What we need is to complete the bit carrying operation and add the results. Most work did this in digital part, which means we need 8 times analog-digital interfaces [2] [18]. Fig. 7 shows our solution. We design a tree circuits to make each current from different digital columns multiply their binary coefficients, and through a trans-impedance amplifier, which shall be realized operational amplifier, we can get our final computing result.

VI. Simulation Results

A. Simulation Setup

We implements our design, including our sparse NN mapping algorithm along with proposed precision composing circuits design on PRIME, which means we modify the PRIME design and its data scheduling. We mainly look into the energy efficiency and speedup as our evaluation metrics. We assume 2-bit ReRAM cell and simulate the energy cost through NVSim [19]. The ReRAM crossbar in baseline PRIME is size 256×256 with 4-bit precision. Besides, to evaluate the performance of ReRAM crossbar grained pruning, we will compare the accuracy loss after pruning for different NNs.

To throughly exam our design for different NN structures, we choose to conduct our simulation on CNN (Convolutional Neural Network) and RNN (Recurrent Neural Network). CNN mainly consists of convolutional layers, like ResNet series [20], which only have one fully connected layer, while RNN is a typical fully connected neural network, like LSTM [21]. The benchmarks in our simulation are LeNet-5, AlexNet, VGG-16, ResNet-20 and LSTM-5. The LSTM-5 model we use is a 5-layer bi-directional LSTM RNN, and the length of hidden unit is 800. The sparsity, which represents the degree of pruning, of each NN can be found in Table I.

TABLE I
The Sparsity of Each NN

NN	LeNet-5	AlexNet	VGG-16	ResNet-18	LSTM-5
Sparsity	92%	89%	92.5%	75%	85%

B. Energy Results with Sparse Mapping

Before we conduct our experiments on different NNs, let us look into how the crossbar size affects the system's energy efficiency. We take VGG-16 to implement our experiments and get our energy results through simulation. As shown in Fig. 8, the x-axis represents deeper pruned NNs with different crossbar size, and the y-axis plots the normalized energy efficiency compared with PRIME. We can find that as the ReRAM crossbar gets smaller, we may first save more energy, but the energy cost becomes larger with 16×16 crossbar, which may be caused by the huge amount of interfaces when mapping such a large NN into those really small ReRAM crossbar. Therefore, we will take 32×32 ReRAM crossbar in our next experiment.

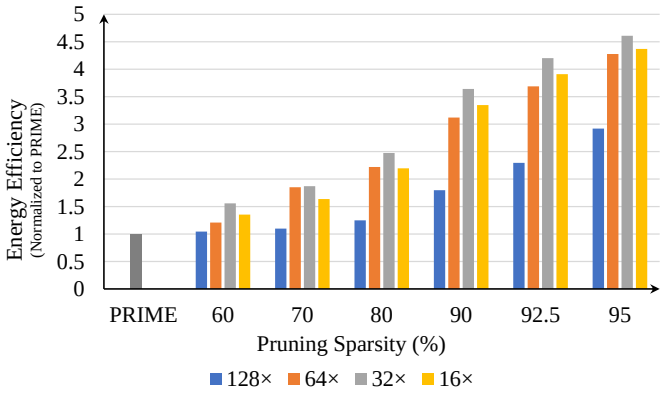

Fig. 8. The energy results with gradually deeper NN pruning sparsity, and different ReRAM crossbar size.

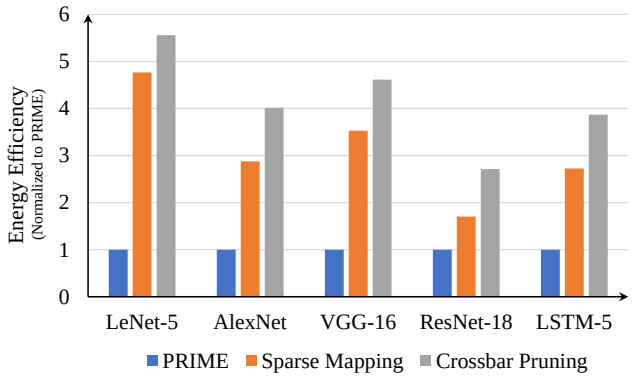

Fig. 9. The Energy Results with Different NNs.

Fig. 9 plots the energy performance, and the results have been normalized to PRIME. As seen, our system for sparse NN achieves $3-5\times$ energy efficiency improvement for popular NNs after our crossbar grained pruning, which makes the energy efficiency further improved. Through the results we find that the system performs much better in NN with large layers, like AlexNet, VGG or LSTM, because such layer often occupy too much energy cost and will be quite sparse after pruning. Meanwhile, for the ResNet which mainly contains convolutional layers, it is more difficult to cut down the parameters, but the energy efficient will be improved a lot after our crossbar grained pruning. The system can achieve more energy savings as the NN get sparser, as we learn from Fig. 8. Finally, we have to note that the excellent result of LeNet-5 is that mapping such a small NN into 256×256 crossbar will cause huge resources wasted, while our small crossbar show its charm to LeNet-5.

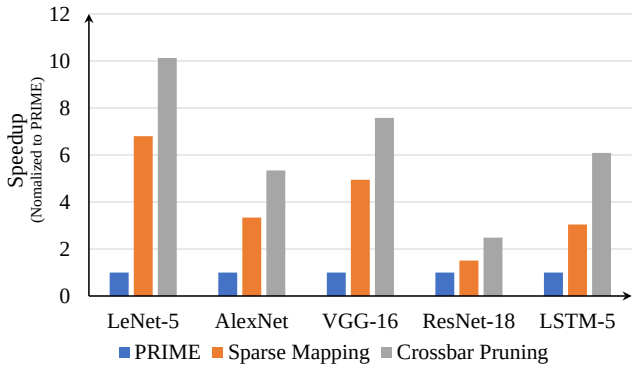

Fig. 10. The Speedup Results with Different NNs.

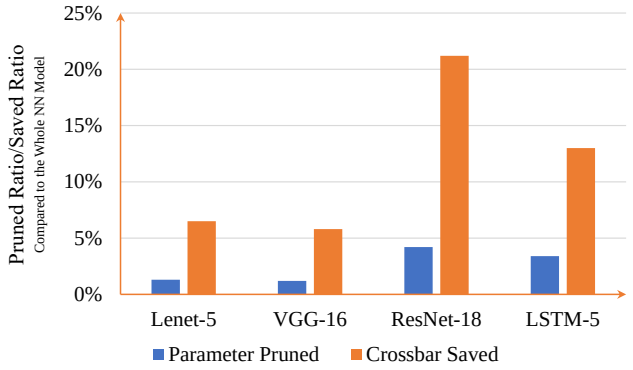

Fig. 11. The NN parameters we pruned V.S. the crossbar resources we saved, both of which are normalized to the whole NN model.

The system's speedup is mainly brought by the data re-using in those ReRAM crossbar we saved, and faster time-cycle for smaller crossbar. Fig. 9 plots the time consuming results for different NN, where we can learn that our system achieve more than $5\times$ speedup for most NNs. However, The results for ResNet is not so impressing, the reason is the same, for convolutional layers, compressing with pruning and mapping sparse irregular weights matrix is more difficult.

C. Accuracy Results with Crossbar-Grained Pruning

We conduct our pruning experiments through LeNet-5 on MNIST dataset, VGG-16 and ResNet-18 on CIFAR-10 dataset, and LSTM-5 on 1000h LibriSpeech.

Fig. 11 shows the comparison between the parameters we removed and the crossbars we saved, both of which have been normalized to the whole NN model. We find that through pruning a small amount of parameters, we can save plenty of crossbar resources. Table II presents the accuracy for those NNs, through which we can find that for three CNNs and LSTM, our crossbar-grained pruning achieves almost no accuracy loss ($<1\%$). Besides, note that the performance for accuracy highly depends on the redundancy in NN, and in our experiments, we adjusted the pruning rate gradually for the trade-off between accuracy and hardware efficiency.

TABLE II
ACCURACY RESULTS AFTER CROSSBAR PRUNING

Nerual Networks	LeNet-5	VGG-16	ResNet-18	LSTM-5
Original	**99.23%**	93.64%	**92.37%**	**89.24%**
Normal Pruning	99.13%	93.62%	92.07%	88.49%
Crossbar Pruning	99.15%	**93.72%**	91.78%	88.01%

VII. CONCLUSIONS

In this paper, we propose a novel sparse NN mapping scheme based on weight columns clustering, to achieve better ReRAM crossbar utilization. Further, we propose crossbar-grained pruning algorithm to reduce the crossbars with low utilization. Finally, since most current ReRAM device cannot achieve high precision, we analyze the effect of quantization precision for sparse NN, and propose to complete high-precision composing in analog field and design related periphery circuits. The simulation results show that compared with those accelerators for dense NN, our mapping scheme for sparse NN with proposed pruning algorithm achieves $3-5\times$ energy efficiency and more than $2.5-6\times$ speedup. Also, our pruning algorithm appears to have almost no accuracy loss.

REFERENCES

[1] H.-S. P. Wong *et al.*, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[2] P. Chi *et al.*, "PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *International Symposium on Computer Architecture*, 2016, pp. 27–39.

[3] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[4] L. Song *et al.*, "PipeLayer: A pipelined reram-based accelerator for deep learning," in *HPCA*, 2017, pp. 541–552.

[5] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.

[6] L. Xia *et al.*, "Fault-tolerant training with on-line fault detection for rram-based neural computing systems," in *DAC*, 2017, p. 33.

[7] L. Song *et al.*, "GraphR: Accelerating graph processing using reram," in *HPCA*. IEEE, 2018, pp. 531–543.

[8] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," 2016, pp. 2074–2082.

[9] P. Wang *et al.*, "SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory," in *DAC*, 2018.

[10] A. Ankit *et al.*, "Trannsformer: Neural network transformation for memristive crossbar based neuromorphic system design," in *ICCAD*, 2017.

[11] L. Ni *et al.*, "On-line machine learning accelerator on digital RRAM-crossbar," in *ISCAS*, 2016, pp. 113–116.

[12] L. Xia *et al.*, "Switched by input: power efficient structure for RRAM-based convolutional neural network," in *DAC*, 2016, p. 125.

[13] F. Su *et al.*, "A 462 GOPS/J RRAM-based nonvolatile intelligent processor for energy harvesting ioe system featuring nonvolatile logics and processing-in-memory," in *VLSI*, 2017, pp. T260–T261.

[14] W.-H. Chen *et al.*, "A 65nm 1mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *ISSCC*, 2018, pp. 494–496.

[15] H. Mao *et al.*, "Exploring the granularity of sparsity in convolutional neural networks," *IEEE CVPRW*, vol. 17, 2017.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[17] S.-S. Sheu *et al.*, "A 5ns fast write multi-level non-volatile 1 k bits RRAM memory with advance write scheme," in *VLSI*, 2009, pp. 82–83.

[18] L. Ni *et al.*, "An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary RRAM crossbar," in *ASP-DAC*, 2016, pp. 280–285.

[19] X. Dong *et al.*, "NVSIM: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.

[20] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[21] F. A. Gers *et al.*, "Learning to forget: Continual prediction with LSTM," 1999.