Deep Neural Network Mapping and Performance Analysis on Tiled RRAM Architecture

Xinxin Wang[1], Qiwen Wang[1], Fan-Hsuan Meng[1], Seung Hwan Lee[1], and Wei D. Lu[1,*]

[1]Electrical Engineering and Computer Science, University of Michigan

*Email: wluee@umich.edu

***Abstract*—Representative deep neural networks (DNNs) have been successfully mapped on an RRAM-based tiled in-memory computing (IMC) architecture. Effects of finite array size and quantized partial products (PPs) due to ADC precision constraints have been analyzed. Methods were developed to solve these challenges and preserve DNN accuracies and IMC performance gains in the tiled architecture. Popular models VGG-16, MobileNet, and RNN/LSTM have been successfully implemented and tested on ImageNet dataset and text classification tasks, respectively.**

Keywords—RRAM, Deep Neural Network, In-Memory Computing, AI Accelerator

I. Introduction

Deep neural networks have shown impressive performance in many artificial intelligence (AI) tasks. However, DNNs also come with high computation cost and complexity when implemented on conventional processor architectures, which are poorly matched for such workload. Dedicated AI accelerators are desirable for meeting the computational demand of DNNs. In most deployment scenarios, only inference operations on pre-trained model are needed, and efficiency of such operation is important for wider adaption. In particular, for edge computing applications where power and latency constrains are severe, efficient inference is crucial.

Among approaches for DNN accelerators, resistive random-access memory (RRAM) based in-memory computing (IMC) accelerators have received increasing attention in recent years. RRAM is a type two terminal non-volatile resistive device whose conductance level can be changed by applying electrical signal in analog or digital fashion. RRAM array can perform vector-matrix multiplication (VMM) in analog domain by accumulating the total current or charge at each column. Furthermore, the high density and non-volatile nature of RRAM devices enable entire DNN models to be stored on chip, thus eliminating the expensive off-chip memory access.

Previous studies have been limited to small neural networks models and simple tasks such as MNIST and CIFAR-10, or without considering physical limitations of the RRAM array [1]. The results may not capture many of the challenges that may arise when implementing large-scale models. For example, device and circuit non-idealities limit the maximum usable array size, making it impractical to implement an entire layer of large-scale models on a single array.

In this paper, we discuss a reconfigurable and scalable tiled RRAM architecture to fulfill the requirement of large-scale storage for mapping DNN weights. A simulation framework is developed to evaluate the performance of different DNNs mapped on this hardware accelerator with the existence of the non-ideal factors.

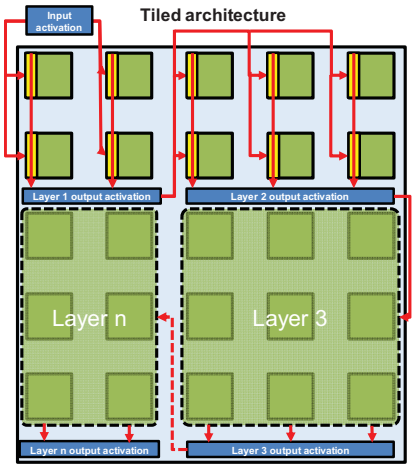

Fig. 1. Schematic of a DNN mapped on tiles of RRAM crossbar arrays.

This work was supported in part by SRC and DARPA through the Applications Driving Architectures (ADA) Research Center, and by the National Science Foundation through grant CCF-1900675.

II. Implementation of Tiled Architecture

The schematic of the tiled architecture for implementing large scale DNN is shown in Fig. 1. In this architecture, multiple moderately sized RRAM arrays are linked through digital interface, thus a large amount of crossbar arrays can be integrated on a single chip. This allows large-scale DNN models to be implemented across multiple RRAM arrays while minimizing effect from device and circuit non-idealities. However, a layer of a typical DNN model is usually much larger than the practical array size. Strategies for mapping of

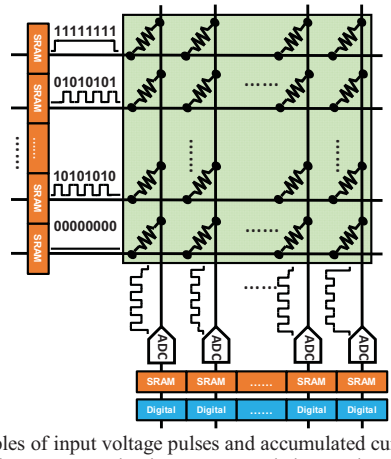

Fig. 2: Examples of input voltage pulses and accumulated currents during VMM operation. Input activations are encoded as pulses in bit-serial approach.

different types of DNN layers onto crossbar arrays using this approach can be found in [4].

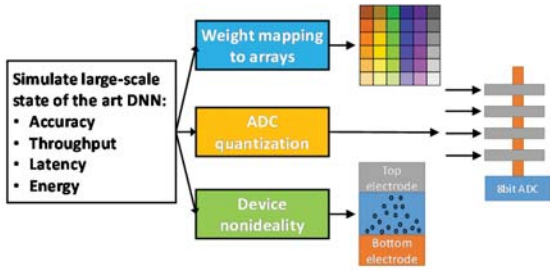

Fig. 3: The purpose of the simulation framework is to map different neural networks to the tile-based architecture, and to simulate inference through the crossbar and the effects of quantization, device variability and other nonlinearity factors.

As the number of conductance levels of RRAM devices and the area/energy of periphery circuitry are both limited, DNN models where quantized input activations and weights are used. In the inference stage, the input activations are programmed as voltage pulses and are fed to different rows of the crossbar arrays. To optimize the latency, the bit-serial approach is adopted. As an example, we implement 8-bit models and set the unit pulse width as 10ns. Plain pulse-width modulation would require 2550ns to complete a single VMM operation. By applying bit-serial modulation, the latency for single VMM operation can be reduced to 80ns, therefore significantly improving the throughput.

After collecting the currents/charges from the crossbar, the analog signals need to be digitized by ADCs to produce the VMM results. Since the input vector and the weights are divided among multiple arrays, each array only outputs a partial-product (PP). To keep the design general, ADCs are located at each crossbar so that the PPs are quantized first and then summed in digital domain to produce the final layer output. However, due to limited ADC accuracy, digitization of the PPs can introduces significant additional error in the VMM process and can severely degrade DNN model performance. This effect is particularly pronounced for highly optimized models (e.g. MobileNet) that cannot tolerate VMM errors well. In addition, device non-idealities including finite on/off ratio and conductance variations can also introduce errors to the computation. These effects can be carefully analyzed through our simulation framework, as shown in Fig. 3.

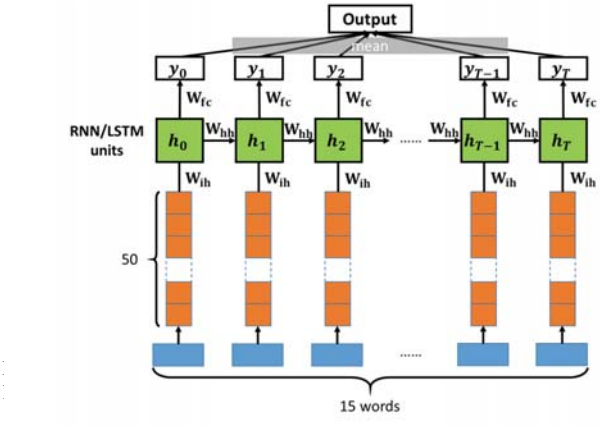

Fig. 5: RNN/LSTM models for text classification.

III. Implementation of Simulation Framework

The simulation framework of the tile-based DNN accelerator is implemented in a hierarchical structure, as shown in Fig. 4. In the bottom level, we defined Crossbar Class and Activation Class to store the crossbar arrays and the activation vectors, respectively. Each Crossbar object contains the information of an array, including the conductance levels of each device and the device non-ideality parameters such as on-off ratio and device variability. Each Activation object is a 2-D array, with each column dimension denoting the vector length and the row dimension denoting the number of vectors.

In the second level, the weight mapping functions and activation mapping functions are defined. Weight mapping methods for fully connected (FC) layer, regular convolution (Conv) layer and depthwise convolution (DW Conv) layer are discussed in [4]. Activations at the same time step will form a vector and elements in different vectors will be programmed as voltage pulses and fed into crossbar arrays in bit-serial manner.

In the third level, we defined different layer functions to implement in the hardware, including FC layer, regular Conv layer and DW Conv layer. In the crossbar operation stage, these functions will perform VMM with the Crossbar objects and the Activation Objects and produce the output PPs. Afterwards, the PPs are quantized with the ADCs at each crossbar. Some information in the PPs will be lost in the quantization stage and the mult-and-add stage is not able to restore them. To mitigate the accuracy degradation due to quantization effects of the PPs, the ADCs need to support multiple quantization ranges, *i.e.* at the extreme case for each column at each crossbar. This approach is termed column-wise ADC quantization [4]. We note that for inference operations for a given model the ADC ranges only need to be set once before the model operation, and does not need to be dynamically re-configured during inference. Afterwards, the quantized PPs will be scaled with the corresponding ranges and then added up with the other PPs to produce the final output activation. At last, the final quantized 8-bit output product is obtained. Nonlinear activation functions including ReLu and ReLu6 can also be added at this stage.

In the top level, the users can build their own model in a user defined file. In this file, the pretrained model should be loaded

and the weights in the model should be mapped onto crossbar arrays by calling the weight mapping functions defined in the second level. Different layers in the model can be built by calling different layer functions in the third level. The layer functions will automatically call the activation mapping functions. The mapped Activation and Crossbar objects are stored in the global variables. They will be loaded with the layer functions and multiplied with each other in the crossbar operation stage in the simulation framework. At the end of inference on each image in the dataset, the predicted result will be recorded.

IV. Simulation of ImageNet and NLP models

As a general implementation of hardware neural network accelerators, the tiled architecture can support models from a broad range of categories. Simulations have been performed for image classification tasks based on the tiled architecture, using popular models including VGG-16 [2] and MobileNet [3] using our framework, and mitigation methods were developed to preserve prediction accuracy on the tiled architecture after considering the PP quantization effects and other non-ideality factors, as discussed in [4]. Here, we will show that RNN and LSTM models for text classification can also be mapped on the tiled-architecture and simulated through our simulation framework. The structures of the models are shown in Fig. 5. Each word in the input sentence is embedded to a vector with 50 elements. At each time step, the RNN or LSTM unit accepts an input vector and a previous hidden vector, which will be multiplied with input-to-hidden weights (W_{ih}) and hidden-to-hidden weights (W_{hh}), respectively. The two results will then be added up to produce the output of the current time step, which also works as the previous hidden vector for the next time step. The current output vector will go through an additional fully connected layer, and a 2-element vector will be produced. At last, all the 2-element vectors corresponding to different time steps will be averaged and the input sentence can be labeled according to the final output.

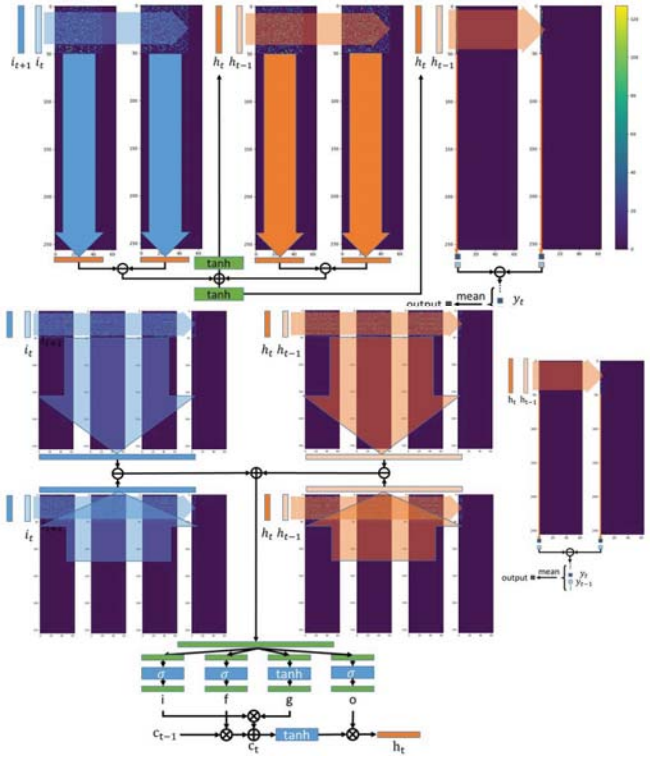

Fig. 7: Mapping results and data flow of the LSTM model.

The mapping results and data flow of these two models are shown in Fig. 6 and 7, respectively. In the simulations, dual array mapping discussed in [4] is used to map signed weights. The VMM operations are implemented with crossbar arrays. Other functions, including subtracting PPs corresponding to the positive weights and the negative weights, adding the output activations corresponding to W_{ih} and W_{hh}, tanh function, sigmoid function, multiplication and addition of intermediate results, are implemented with digital circuits.

We first simulated RNN and LSTM model accuracies for text classification with floating point models. Using the floating models as baseline, digital models with input activation and weight values quantized from 8-bit to 2-bit are simulated, under different ADC quantization conditions. The results in Table I show that even with 4-bit model and layer-wise 4-bit ADC, the accuracies do not degrade much. Highest accuracies can be achieved with 4-bit model and column-wise 8-bit ADC for RNN and 8-bit model and layer-wise 8-bit ADC for LSTM. These accuracies are in fact higher than the floating models, likely a result of over-fitting in the floating model. Finally, the models are simulated under 100 on/off ratio and 2% variation of the device, and rounded ADC quantization scales, as shown in Table II. The performances in most cases are still high. One reason that the performances of the RNN and LSTM model in this application are robust against quantization is that this task is relatively simple, and the models are in general over-parameterized. While for highly optimized models and for more challenging tasks, quantization effects need to be carefully analyzed and mitigated, as discussed in [4].

TABLE I. Accuracies of RNN and LSTM in different cases

Accuracy	Floating point model	Activation/ weight bits	Digital Model	Column-wise 8-bit ADC	Layer-wise 8-bit ADC	Layer-wise 6-bit ADC	Layer-wise 4-bit ADC	Layer-wise 2-bit ADC
RNN	93.32%	8	93.18%	93.13%	93.19%	93.16%	92.76%	87.41%
		6	93.19%	93.23%	93.17%	93.07%	92.39%	87.52%
		4	92.84%	93.84%	92.87%	92.93%	92.10%	86.48%
		2	54.63%	54.55%	54.39%	54.28%	54.12%	61.46%
LSTM	95.38%	8	95.34%	95.37%	95.39%	95.37%	95.06%	57.91%
		6	95.33%	95.31%	95.29%	95.31%	95.09%	56.14%
		4	95.32%	95.34%	95.37%	95.32%	95.06%	54.36%
		2	48.46%	47%	47.14%	50.1%	52.23%	43.27%

TABLE II. Accuricies of RNN and LSTM with device non-idealities and rounded quantization scales

Accuracy	Floating point model	Activation/ weight bits	Digital Model	Column-wise 8-bit ADC	Layer-wise 8-bit ADC	Layer-wise 6-bit ADC	Layer-wise 4-bit ADC	Layer-wise 2-bit ADC
RNN	93.32%	8	93.18%	93.12%	93.15%	93.29%	87.15%	50%
		6	93.19%	93.04%	93.17%	92.53%	59.52%	50%
		4	92.84%	92.64%	92.59%	88.02%	54.24%	50%
		2	54.63%	52.62%	52.18%	50.88%	50.75%	50%
LSTM	95.38%	8	95.34%	95.38%	95.28%	95.25%	78.04%	50%
		6	95.33%	95.38%	95.46%	93.84%	54.86%	50%
		4	95.32%	95.24%	95.28%	83.64%	49.97%	50%
		2	48.46%	52.67%	54.51%	49.17%	50%	50%

V. Conclusion

In this paper, we discussed strategies to implement DNN inference models using the tiled architecture by considering practical crossbar size limitations, quantization effects and other non-ideality factors. The simulation framework for evaluating model accuracies based on the tiled architecture is presented in detail. To demonstrate the generality of the architecture for tasks beyond image classification, RNN and LSTM models were mapped on the same architecture and demonstrated that the tiled-architecture and the simulation framework can support models from different categories.

References

[1] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 6.1.1-6.1.4.

[2] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," pp. 1–8, Dec. 2013.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017.

[4] X. Wang, Q. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A Deep Neural Network Accelerator Based on Tiled RRAM Architecture," *2019 IEEE Int. Electron Devices Meet.*, p. 14.4, 2019.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[6] F. Cai, J. M. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. P. Flynn and W. D. Lu, "A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations," *Nat. Electron.*, vol. 2, no. 7, pp. 290–299, Jul. 2019.

[7] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner and W. D. Lu, "A general memristor-based partial differential equation solver," *Nat. Electron.*, vol. 1, no. 7, pp. 411–420, Jul. 2018.