technische universität
dortmund

This is SPATEM! A Spatial-Temporal Optimization Framework for Efficient Inference on ReRAM-based CNN Accelerator

Yen-Ting Tsou*, Kuan-Hsun Chen†, Chia-Lin Yang*, Hsiang-Yun Cheng‡, Jian-Jia Chen§, Der-Yu Tsai*

*National Taiwan University, Taiwan,
†University of Twente, Netherlands
‡Academia Sinica, Taiwan
§Technical University of Dortmund, Germany
Corresponding author: Chia-Lin Yang, yangc@csie.ntu.edu.tw

Citation: TBD

BibTeX:

```
@inproceedings{spatem,
  author={Y.-T. Tsou and K.-H. Chen and C.-L. Yang and H.-Y Cheng and J.-J. Chen and D.-Y. Tsai},
  booktitle={27th Asia and South Pacific Design Automation Conference (ASP-DAC)},
  title={This is SPATEM! A Spatial-Temporal Optimization Framework for Efficient Inference
  on ReRAM-based CNN Accelerator},
  year={2022},
  volume={},
  number={},
  pages={},
  doi={}
}
```


CS 12 computer science 12

This is SPATEM! A Spatial-Temporal Optimization Framework for Efficient Inference on ReRAM-based CNN Accelerator

Yen-Ting Tsou*, Kuan-Hsun Chen†, Chia-Lin Yang*, Hsiang-Yun Cheng‡, Jian-Jia Chen§, Der-Yu Tsai*
*National Taiwan University, Taiwan, †University of Twente, Netherlands
‡Academia Sinica, Taiwan, §Technical University of Dortmund, Germany
Corresponding author: Chia-Lin Yang, yangc@csie.ntu.edu.tw

***Abstract*—Resistive memory-based computing-in-memory (CIM) has been considered as a promising solution to accelerate convolutional neural networks (CNN) inference, which stores the weights in crossbar memory arrays and performs in-situ matrix-vector multiplications (MVMs) in an analog manner. Several techniques assume that a whole crossbar can operate concurrently and discuss how to efficiently map the weights onto crossbar arrays. However, in practice, the accumulated effect of per-cell current deviation and Analog-to-Digital-Converter overhead may greatly degrade inference accuracy, which motivates the concept of Operation Unit (OU), by which an operation per cycle in a crossbar only involve limited wordlines and bitlines to preserve satisfactory inference accuracy.**

With OU-based operations, the mapping of weights and scheduling strategy for parallelizing CNN convolution operations should take the cost of communication overhead and resource utilization into consideration to optimize the inference acceleration. In this work, we propose the first optimization framework named SPATEM, that efficiently executes MVMs with OU-based operations on ReRAM-based CIM accelerators. It decouples the design space into tractable steps, models the expected inference latency, and derives an optimized spatial-temporal-aware scheduling strategy. By comparing with state-of-the-arts, the experimental result shows that the derived scheduling strategy of SPATEM achieves on average 29.24% inference latency reduction with 31.28% less communication overhead by exploiting more originally unused crossbar cells.

I. Introduction

Convolutional Neural Networks (CNNs) have been widely used for a wide range of tasks in artificial intelligence applications. These neural networks often consist of lots of parameters and matrix-vector multiplications (MVMs) about the parameters, motivating tremendous studies on computing-in-memory (CIM) neural network accelerators to execute neural network applications efficiently [1]. The CIM architectures using resistive random access memory (ReRAM) have been a promising solution for accelerating Convolutional Neural Networks. The crossbar structure performs MVMs by storing weights in crossbar cells, supplying inputs in wordline drivers and reading outputs in bitlines, which greatly reduce data movements and boost the energy efficiency of neuromorphic computation.

Despite the promising potential, recent work about technology of ReRAM [2], [3] has shown that the imperfect circuits and devices might weaken the performance of the ReRAM-based CNN accelerator. The accumulated effect of per-cell current deviation and ADC overhead might degrade inference accuracy when operating entire crossbar array within a single cycle. To achieve satisfactory inference accuracy, rather than utilizing all wordline inputs and bitline cells, only limited wordlines and bitlines in a crossbar should be operated at once, namely Operation Unit (OU) [3], [4]. For example, only 9 wordlines and 8 bitlines can be turned on at a time within a 512×256 crossbar array in a state-of-the-art ReRAM macro designed for CNN acceleration [2]. Nevertheless, the cycle time can be reduced due to less Analog-to-Digital-Converter overhead [4], [5]. Such an OU-based acceleration raises new challenges during deployment.

By assuming that the entire crossbar array can be operated per cycle, state-of-the-arts [1], [6] map all weights of a layer onto the same crossbar and and replicate weights onto more crossbars to exploit parallel computing ability. ISAAC replicates weights to balance the layer-wise throughput because of the data dependency between adjacent layers [1], whereas HitM decides the number of weight replication by formulating the layer-wise crossbar allocation problem into bin-packing problem [6]. However, considering that only one OU per crossbar can be activated per cycle in practice, executing a convolution layer or fully connected layer in a single crossbar requires multiple cycles, whereas many available crossbar cells are in fact unused. To fully exploit the available resource, the OU-induced challenge is in twofold:

- *Spatial issues:* MVMs must be split into different partitions and assigned onto different crossbar cells to maximize the parallel degree. How to pack multiple partitions on each crossbar is essentially the key of achieving highly parallel computing.
- *Temporal issues:* The order of MVMs must comply with the data dependencies between layers. In order to minimize inference latency, the balance between input consumption and output production should be considered.

However, these issues are not independent. A solution solely for resolving spatial issues, e.g., only mapping weights onto crossbars, may put unnecessary restrictions on possible MVM execution orders and impose additional communication overhead between adjacent layers. An unoptimized order of MVMs might also result in an unbalanced input consumption rate to output production rate, such that the inference might be hindered by the synchronization over dependent partitions. To fully accelerate CNN, a joint solution taking both spatial and temporal issues into consideration without imposing restrictions to each other is thus desired.

In this work, we propose an optimization framework **SPATEM** that executes MVMs by OU-based operations in parallel. The goal is to reduce CNN inference latency by tackling the spatial and temporal issues jointly. It decouples the spatial deployment into three steps: partitioning MVMs with an analytical cost model based on the longest execution path, packing them onto virtual crossbars ideally and mapping them onto physical crossbars. For the temporal deployment, an ordering step is conducted to derive an global MVM sequence in the perspective of input windows to fulfill data dependency between adjacent layers as soon as possible. Afterwards, an optimized scheduling strategy can be systematically derived under given CNN structures and hardware constraints.

Our contributions in a nutshell:

- We explore the design space of inference optimizations under OU-based operations. To exploit parallelism capability, relevant spatial and temporal issues are considered (see Section III).
- An optimization framework (SPATEM) consists of 1) a spatial module employing several algorithms considering the trade-off over computation parallelism, resource utilization and communication cost, and 2) a temporal module deriving an MVM execution sequence under the layer dependencies (see Section IV).
- We extensively evaluate the inference of SPATEM on different relevant aspects for five pre-trained CNNs, comparing with state-of-the-arts [1], [6] (see Section V).

II. Background and System Models

In this section, we introduce the considered CNNs and the ReRAM-based accelerator. Specifically, we highlight the practical limit on ReRAM-based accelerators, which motivates the usage of OU-based operations and its relevance to this work.

A. Convolutional Neural Network (CNN)

Given a trained CNN model, we denote the number of layers as L. For each layer l, the number of input windows is denoted as W_l, respectively. Also, for layer l, the number of input feature map elements needed in an input window is denoted as IR_l, while the number of output feature map elements generated by the convolution of an input window is denoted as OW_l. They are typically composed of three types of layers: convolution layer, pooling layer and fully connected layer. A convolution layer consists of dot-products between input windows sliding through input feature map and filters, to extract features from the data. Pooling layers subsample the feature map to reduce the size of feature map while maintaining the extracted features. At the end of the CNNs, fully connected layers employ classic deep learning algorithm which consists of neurons and synaptic weights to classify the input based on the features extracted from convolution layers.

Most computations in CNNs are independent, which can be parallelized to reduce the inference latency. For example, the convolution of different input windows does not depend on the input or output of each other. However, there are data dependencies between adjacent layers, since the output of a layer are the input of its subsequent layer. Such dependent layers cannot be executed concurrently.

B. ReRAM-based CNN Accelerator

We consider the CIM architecture of ReRAM-based CNN accelerator (Fig. 1) proposed in [1], which consists of multiple processing engines (PEs). PEs consist of multiple computing units (CUs) and communicate through packet routing interconnection. CUs are composed of multiple crossbar arrays which can efficiently accelerate matrix-vector multiplications in CNN [3], [4]. We denote the number of available crossbars as C. A digital-to-analog converter (DAC) is connected to each wordline of the ReRAM crossbar array to convert the input feature map data into input voltages. By storing weights as the conductance in cells and supplying inputs as the wordline voltages, the dot-production can be performed between inputs supplied from wordlines and weights in the same bitline. Afterwards, the accumulated currents on the bitlines (sum-of-products results) are fed to the analog-to-digital converters (ADCs). In addition, there is one on-chip eDRAM buffer in each PE to store needed input and output feature maps generated by the PE. A non-linear activation unit and a pooling unit are also included in the PE to execute the corresponding layers in CNNs. Like in [1], once the weight values are preloaded into crossbar cells statically, the crossbar cells will not be modified during the whole inference execution.

C. Operation Unit

In practice, only limited wordlines and bitlines in a crossbar array can be activated per cycle [2], [3], [5]. To this end, we consider the concept of Operation Unit (OU) proposed in [3], [4] as a hardware constraint, which enforces the computation to be operated in a unit of OU for each crossbar, by which only the limited number of wordlines and bitlines that can be turned on. Nevertheless, the Analog-to-digital overhead can be reduced [4], [5] and the parallel computing of MVMs can be still conducted on multiple crossbars. We denote the number of OUs needed for layer l's convolution of an input window as OU_l.

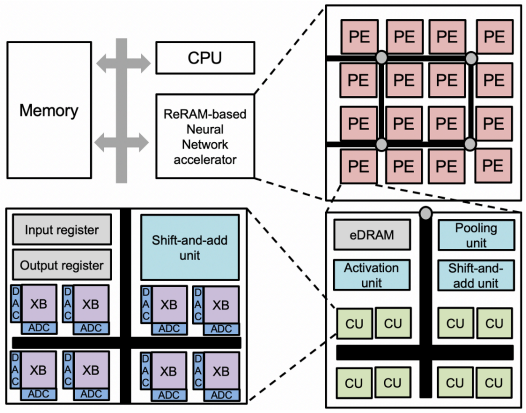

Fig. 1. Architecture of ReRAM-based CNN accelerator, based on [1].

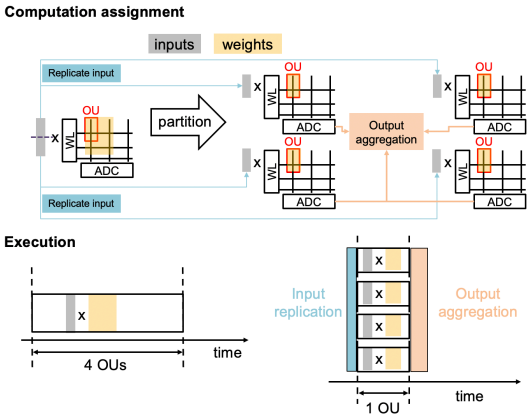

Fig. 2. Partitioning weights onto 4 crossbars to execute concurrently.

For example, if a layer l's MVM mapped to 1 crossbar requires 4 OU operations sequentially, i.e., $\text{OU}_l = 4$, the MVM might be partitioned into 4 parts and executed in parallel, as shown in Fig. 2, whereas the induces communication overhead, i.e., input replication and output aggregation, should also be taken into consideration.

III. Design Space Exploration

Given a trained CNN model with L layers, required number of OU_l for each layer and hardware configuration of ReRAM-based CIM accelerator with C available crossbars, the design space for the deployment of CNN inference on ReRAM-based accelerator can be divided into two parts:

- The *spatial deployment* copes with the mapping of weight values and multiplication results onto crossbar cells. In our studied problem, one crossbar cell stores one weight value and generates one multiplication result.
- The *temporal deployment* deals with the execution order of MVMs. In our studied problem, MVMs are partitioned into multiple parts on different crossbars, so the execution order should preserve the data dependencies between adjacent layers.

To optimize the inference latency jointly, both spatial and temporal deployments require extremely high complexity if a brute-force approach is trivially adopted. The eventual scheduling strategy will be generated as a script like [1], followed by each hardware component on the accelerator. In the following, we discuss our insights about how to tackle the complex solution space.

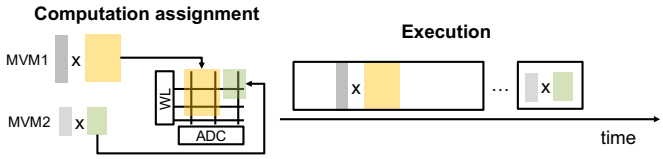

Fig. 3. Example of packing partial weights from two MVMs onto the same crossbar, by which the same crossbar can be utilized at different time points.

A. Spatial Deployment

In summary, there are three steps to decide the spatial deployment in our framework. The partitioning step decides how to partition the MVMs and how many parts should be partitioned. The packing step decides how to pack partitioned MVMs onto virtual crossbars, which ideally omits the communication overhead. The assigning step decides the assignment of virtual crossbars onto physical crossbars. In the following, we discuss their difficulties and required treatments:

1) Partitioning Step: The convolution and fully connected layers in CNNs can be executed in parallel by ReRAM-based accelerator in the form of MVM. After transforming them into MVM by flattening, the filters are concatenated to form weight matrices and input windows are flattened to form input vectors. Different size of the weight matrix and input vectors in the MVMs will result in various execution latency with various crossbar and cell requirements. Considering the overhead of replication and aggregation, there are 4 distinct dimensions to partition all MVMs of a layer, as follows:

- **Partition weight matrix in height** (*hwm*) equally splits the weight matrix into upper and lower parts. The output of the partitioned MVMs needs an aggregation, since the partial values of the same input window and the same weight matrix column need to be added to generate the output feature map.
- **Partition weight matrix in width** (*wwm*) equally splits the weight matrix into left and right parts. The input vector needs a replication, since both partitioned weight matrices require the whole input vectors.
- **Partition input windows** (*iw*): replicates the whole weight matrix and equally divide input windows to the replicated weight matrices. Each replicated weight matrix is only multiplied by half of the input windows.
- **Partition input bits** (*ib*): replicates the whole weight matrix and equally divide the input bits of all windows into the replicated matrices. Each replicated weight matrix is multiplied by half of the bits over all input windows. The output of the partitioned MVMs needs an aggregation to generate the output feature map.

One partitioning decision should consider the above 4 degrees. Each degree represents how many parts the MVMs of the layer is split into in the respective dimension.

2) Packing Step: One crossbar can store the weights from multiple layers to exploit unused cells. As shown in Fig. 3, one crossbar can execute multiple MVMs with different weights at different times. After the partitioning decision is made, there are lots of partitioned MVMs. To exploit each crossbar, this packing step has to group suitable partitioned MVMs like tiles together on each virtual crossbars to improve the resource utilization.

However, the positions of the assigned crossbars also affect the communication overhead induced by the distance between dependent MVMs. To simplify the solution space, we leverage on virtual crossbars omitting taking crossbar positions into consideration, and impose the following two constraints: 1) MVMs of the same layer must be in different virtual crossbars, so that the decided partitions can be deployed to multiple crossbars without redundant replication and aggregation, and 2) the number and the cell requirements of virtual crossbars must not exceed the given hardware configuration.

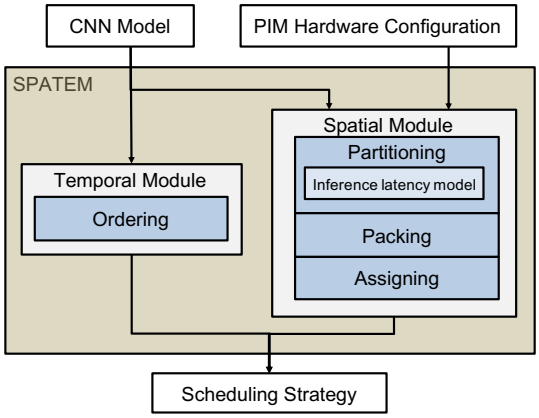

Fig. 4. Overview of the proposed framework – SPATEM

3) Assigning Step: After the packing decision is conducted, partitioned MVMs have been packed on virtual crossbars. The assigning step subsequently map these virtual crossbars onto physical crossbars provided in the given ReRAM-based CIM architecture. Considering the communication overhead between actual crossbars, the decision should account for the communication amount and distance to ideally optimize the inference latency. Since the communication amount between partitioned MVMs are determined after the partitioning step, the assignment can arrange these partial MVMs in descending order.

B. Temporal Deployment

Since hardware components on the ReRAM-based CIM architecture follow the generated script to execute all MVM computations in a predefined order, an ordering step is needed to arrange the MVM execution sequence under their data dependencies. An important observation is that, no matter how the spatial deployment is, the dependencies between adjacent layers are not changed. Hence, the ordering step can be conducted independently without involving the decisions of the spatial deployment.

Since the MVM execution sequence decides the order of input consumption and output production, the ordering step should ensure that the precedence constraint imposed by the data dependencies between MVMs must be preserved. Otherwise, a deadlock might potentially take place to hinder the parallel computation. To prevent from such deadlocks, the MVM execution sequence must avoid circular waiting by making all dependencies follow the MVM sequence, e.g., concatenating layer-wise MVM sequences.

IV. SPATEM: Spatial-Temporal Optimization Framework

Fig. 4 illustrates an overview of the spatial-temporal optimization framework, namely **SPATEM**. It consists of two independent modules for spatial and temporal deployments. With the given inputs, i.e., a pretrained CNN model and CIM hardware configuration, spatial and temporal modules address for the aforementioned steps involving the presented insights. Afterwards, the output of the framework, the scheduling strategy, is generated to drive the corresponding hardware components that steer inputs and outputs. In the following, we present an estimation model for inference latency used in the partitioning step, and the detailed design of each step respectively.

A. Inference Latency Estimation Model

To quantify the expected inference latency for a layer with possible partitioning decisions, we propose an estimation model for end-to-end inference latency. Since the execution of a CNN inference on

ReRAM-based CIM accelerator is pipelined, except for the first layer, the critical path contains computation latency of one MVM and communication latency about the output of the MVM for each layer. As for the first layer, the critical path includes the computation latency of all MVMs in the longest part and the communication latency about fetching input for the first MVM in the part. Given a possible partition for layer l derived from the partitioning step, which has partition degrees $\{hwm, wwm, iw, ib\}$, we define the maximal number of partitioned MVMs this partition can have as max_l, that is:

$$max_l = \lceil \frac{W_l}{iw} \rceil$$

As the partitioning step partitions each MVM in the same layer evenly, which results in the same latency for all partitioned MVMs, we define lat_l as the latency of one partitioned MVM by the following:

$$lat_l = \text{IR}_l \times \frac{wwm * ib}{hwm} + \frac{\text{OU}_l}{hwm \times wwm \times ib} + \text{OW}_l \times \frac{hwm * ib}{wwm}$$

Since at this step, the positions of the final assignment are still unknown, the communication overhead is estimated by the amount of transferred data multiplied with the average communication latency between 2 crossbars in the ReRAM-based CIM architecture, where the average communication latency ($avgLat$) is derived from the latency value between 2 random crossbars from the CIM configuration.

The inference latency estimation (IL) is calculated by the computation time about layer l (T_c^l), the overhead of the required communication about layer l's output (T_o^l) and the overhead of fetching input for the first layer (T_{fetch}) as follows:

$$\text{IL} = \sum_{l=1}^{L} (T_c^l + T_o^l) + T_{fetch}$$

where

$$T_c^l = \begin{cases} max_l \times lat_l & , l = 1 \\ lat_l & , otherwise \end{cases}$$

$$T_o^l = \text{OW}_l \times \frac{hwm * ib}{wwm} \times avgLat$$

$$T_{fetch} = \text{IR}_1 \times \frac{wwm * ib}{hwm} \times avgLat$$

B. Spatial Module

In the following, we present the design principle of three three steps: partitioning, packing and assigning, respectively.

1) Partitioning Step: To minimize the inference latency for each layer, we need to know what is the best partition strategy for the layer under a given number of crossbars. However, a crossbar can be used by multiple layers at different time points. To check if the number of required crossbars exceeds the given hardware constraint, we have to wait until the subsequent step decides which partitions should be packed on which crossbars. Although it is possible to determine the best strategy for each layer by checking all possible packing strategies, and choosing the combination that all partitions of layers together yield the minimal inference latency while satisfying the given number of crossbars, this straightforward method is definitely not tractable. Instead, we develop a dynamic programming algorithm to determine the best partition strategy of each layer. Instead of taking the given number of available crossbars into consideration, we relax the number of available crossbars to $L \times C$, such that each layer can use as many crossbars as the number of available crossbars C given in the CIM architecture. Hence, all possibilities of the best partition strategies can be preserved and recorded as a look-up table for the subsequent packing step to incorporate the demand of all layers under the real number of available crossbars.

Algorithm 1: Partitioning Algorithm

Input: Number of layers L, number of available crossbars C
Output: Best partitioning strategies table ILR
Initialize ILR$[x][y]$ to 0, $\forall x \in [0, L \times 4)$, $y \in [0, L \times C)$;
for $l \leftarrow 0$ ***to*** $L - 1$ **do**
  for $d \leftarrow \{hwm = 0, wwm = 1, iw = 2, ib = 3\}$ **do**
    for $r \leftarrow 0$ ***to*** $L \times C$ **do**
      $tmp_$ILR = ILR$[l \times 4 + d][r]$;
      while *True* **do**
        increase layer l's dimension d with new requirement tmp_r and latency reduction $tmp_$ILR.
        if $tmp\_r > L \times C$ **then**
          break;
        else
          if *tmp_ILR* $>$ *ILR*$[l \times 4 + d + 1]$*[tmp_r]* **then**
            ILR$[l \times 4 + d + 1][tmp_r]$ = $tmp_$ILR
          end
        end
      end
    end
  end
end

By treating the available crossbars as the capacity of the knapsack, the partition dimensions as bounded items and the latency reduction as the item value, Alg. 1 shows the pseudo code as to solve a bounded knapsack problem. The algorithm uses a 2-D array to store the best strategies in inference latency reduction (*ILR*):

$$\text{ILR} = \text{the reference latency} - \text{the estimated latency},$$

where the reference is calculated by the strategy with the least dimension degrees, which is to partition every layer's weight matrix to fit in crossbars without weight replication.

Each entry of the data structure with indices $l \times 4 + d$ and r represents the partitioning strategy with the best ILR, which can only utilize first $l \times 4 + d$ partitions to split MVMs under the available number r of crossbars. Please note that here r is up to $L \times C$ due to the assumption that each layer can adopt up to C crossbars.

To fill the entry with indices $l \times 4 + d + 1$ and r, the algorithm generates partitioning strategies by increasing the dimension d of layer l's partition in all partitioning strategies with index $l \times 4 + d$, and records the one with the most inference latency reduction among strategies which requires r crossbars.

2) Packing Step: By taking the best partitioning strategies found in the above step, this step employs a two-level greedy algorithm to pick the partitioning strategy. First, the algorithm adopts *worst-fit bin packing*, starting from the entry with the most relaxed resource requirement in table ILR, where $r = L \times C$, to greedily pack the partitions to fit in resource limitation based on the partitioning strategies. The partitioned MVMs provided by the table entry are sequentially packed to the most emptiest virtual crossbars from the first layer to the last layer, while reducing structural hazards by assigning MVMs of close layers to different virtual crossbars.

Although one crossbar can be utilized by multiple MVMs, the required number of crossbars can only be decided via the above packing procedure. If the initial strategy is not feasible, this greedy approach iteratively follows the next best strategies in the look-up table, requiring less number of crossbars to find a feasible strategy, which fits the given available crossbars C.

3) Assigning Step: This step assigns virtual crossbars to physical crossbars to shorten the communication distance between corresponding producer and consumer by two greedy algorithms. The first greedy algorithm hierarchically merges virtual crossbars in a bottom-up manner. Similar to the concept of using virtual crossbars in the

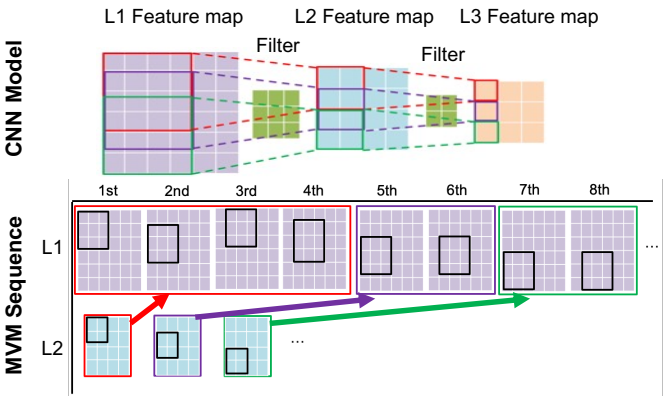

Fig. 5. Illustration of the ordering step.

partitioning step, we also adopt virtual CU and virtual PE to derive the actual mapping of all crossbars, CU, and PE on the CIM architecture.

Initially, virtual crossbars with the least amount of shared data is picked as many as the number of physical CUs provided in the CIM configuration to form virtual CUs. Afterwards, each virtual crossbar is iteratively assigned to the virtual CU which currently has the most amount of dependent data to reduce the communication across CUs, by which the intra-CU data can be reused as much as possible. A similar method is also applied to merge virtual CUs to Virtual PEs to reduce the data communication across PEs. The second greedy algorithm assigns virtual PEs to physical PEs. It picks virtual PEs with the most amount of shared data and assigns the virtual PE onto the physical PE in the descending order.

This step reduces the communication overhead required by the data dependencies between MVM as much as possible and avoid hindering the MVM execution sequence decided by the independent ordering step in the next subsection. Eventually, a feasible mapping between CNN computation and hardware resource given in the ReRAM-based CIM architecture can be derived for the final scheduling strategy.

C. Temporal Module

We design an ordering step to derive a global MVM execution sequence in the perspective of input windows to fulfill data dependency between adjacent layers as soon as possible. The global MVM execution sequence is listed in a column-major order of the feature map from the last layer of MVMs and iteratively arrange MVM sequence of the former layer to fulfill the dependency of the later layer as soon as possible.

Fig. 5 demonstrates the ordering step on 3 layers, i.e., L1, L2 and L3. To arrange L3 in column-major order, we arrange L2 to generate the feature map in the order of the frames in red, purple and green for L3. Therefore, we append the MVM with the input window in the L2 red frame to the L2 sequence, and iteratively append the MVMs with input windows in the L1 red frame to the L1 sequence. After arranging MVMs through the first layer of the red frames, we arrange MVMs of the purple frames and the green frames progressively.

D. Spatial-Temporal Scheduling Strategy

Once the computation mapping from the spatial module and the MVM execution sequence are derived from the temporal module, we can generate a script to load weights into corresponding memristor cells, and deploy the partitioned MVMs accordingly. On top of the configured CIM architecture, the computation of MVMs should start from the first layer of the given CNN. The communication may take place within the steering of feature map between producer and consumer crossbars. A crossbar can only execute when all MVM executions prior to its execution on the global sequence are ready and all dependent inputs are placed in the eDRAM.

V. Evaluation

In this section, we compare the inference latency of the scheduling strategy obtained by SPATEM, with two previous work which are the baseline method [1] and the state-of-the-art method [6]. We evaluate all three methods to inference five pre-trained CNNs, including Lenet [7], DeepID [8], Deepface [9], Caffenet [10] and Overfeat [11] on the OU-based ReRAM CIM accelerator configured in [1]. We analyze the inference latency improvement with computation parallelism degree, resource utilization and communication cost of the three methods to show that the proposed algorithms can optimize inference latency with consideration of those factors respectively.

A. Experimental Setup

The evaluation is carried out on an in-house event-driven OU-based ReRAM CIM simulator that uses the configuration of [1]: The number of PEs, CUs per PE and crossbars per CU are 12x14, 12 and 8 respectively (see Fig. 1). The size of a crossbar array is 128x128, where only 9 wordlines and 8 bitlines can operate per cycle, resulting in 3-bit ADC resolution due to lower range of values [2], [4]. The resolution of DAC and crossbar cells are 1-bit and 2-bit respectively. The clock rate of the accelerator is 1.2GHz. The bus-width is 384-bit and the on-chip embedded DRAM has 64GB capacity.

In the literature, the scheduling strategies proposed by the baseline method, noted as ISAAC [1], and the state-of-the-art, noted as HitM [6] both consider that whole crossbar can operate per cycle. To fairly evaluate our work, we implement their original methodologies but only adapt their throughput estimation models with the considered OU concept, i.e., instead of 1-cycle latency for arbitrary MVMs, the number of MVM execution cycles is derived by the number of OUs needed to perform the MVM.

B. Evaluation Result

We first show the inference latency of the three scheduling strategies. Fig. 6 shows that SPATEM achieves 29.24% improvement in average. In general, the larger the given CNN structure, the lower the improvement. This is because with the same number of available crossbars given by the CIM architecture configuration, smaller CNNs may obtain much more spatial freedom to deploy their MVMs on available crossbars. Since the improvements vary across different CNNs, we further analyze each affecting factor of inference latency, i.e., parallelism degree, communication cost and resource utilization.

1) Parallelism Degree: Fig. 7 shows that the average number of active crossbars when executing smaller CNNs, i.e., Lenet and DeepID, is significantly small compared to other networks. Even if small networks can be fully parallelized, it may not activate a large amount of crossbars. We also note that allowing storing weights of different layers in a single crossbar through the packing step significantly increases the computation parallelism since there are more effective crossbars to be allocated for computations of layers.

2) Communication Cost: As shown in Fig. 8, SPATEM indeed reduces the data transfer between PEs, resulting in a significant reduction of communication cost. We observe that the amount of data when executing Deepface has a different trend from other large CNNs. One explanation is that Deepface possesses a great number of neurons in fully connected layers, producing lots of intermediate data with partitioning weights. Note that the intra-PE data transfer does not incur extra latency, because all data generated by the computations within a PE will be stored in the embedded DRAM.

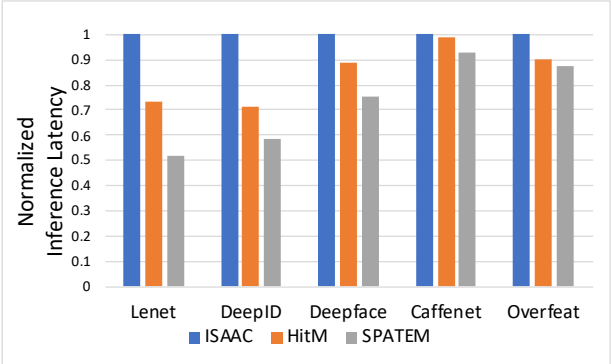

Fig. 6. Normalized inference latency

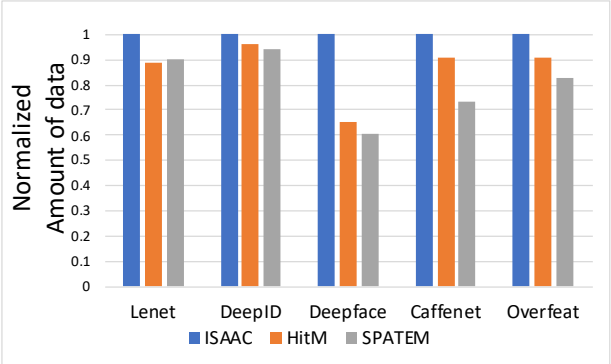

Fig. 8. Normalized amount of data transfer between PEs

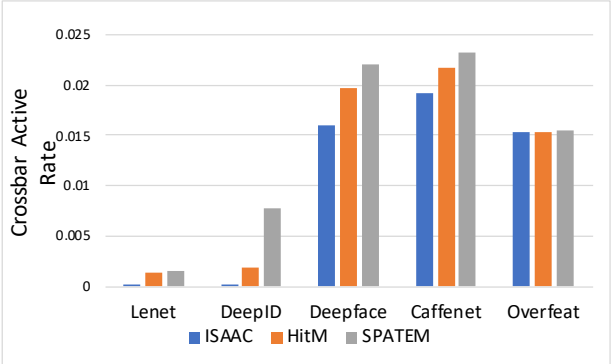

Fig. 7. Average percentage of active crossbars during execution

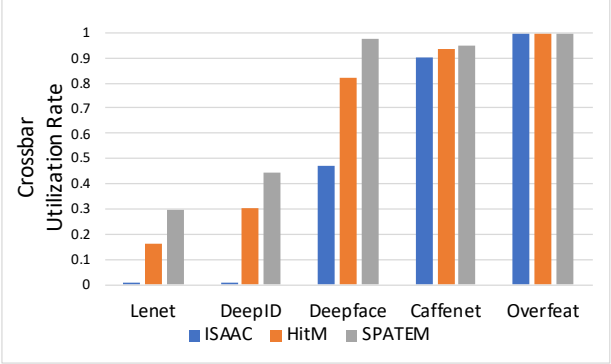

Fig. 9. Rate of Crossbar utilization

3) Resource Utilization: When applying SPATEM, those originally unused crossbars by the state-of-the-art methods can be exploited now, which results in better inference latency with 3.19x more utilized resource in average as shown in Fig. 9. When the size of NNs is smaller like Lenet and DeepID, the improvement is not significant.

VI. Conclusion

Several inherent issues of ReRAM may degrade CNN inference accuracy on the CIM architecture. To keep the accuracy acceptable, only limited wordlines and bitlines are allowed to operate concurrently in each crossbar, forming operating unit (OU). Our work shows that spatial and temporal issues must be overcome on ReRAM-based accelerators. Our framework decouples the design space into tractable steps, models the expected inference latency for partitioned MVMs, and addresses each step thoughtfully. Comparing to the state-of-the-arts, we show that the derived scheduling strategy over five representative CNNs achieves 29.24% inference latency reduction on average, by utilizing 3.19x more originally unused crossbar cells with 31.28% less communication overhead.

Acknowledgement

We thank the DFG (405422836 and 124020371), the MXIC Technology (109-S-C24), the MOST (107-2923-E-001-001-MY3 and 109-2221-E-002-147-MY3), the NTU (110L880603) and the Delta Electronics (110HT907002) for supporting this work.

References

[1] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *International Symposium on Computer Architecture*, 2016, pp. 14–26.

[2] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang, T.-H. Hsu, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *International Solid - State Circuits Conference*, 2018, pp. 494–496.

[3] M.-Y. Lin, H.-Y. Cheng, W.-T. Lin, T.-H. Yang, I.-C. Tseng, C.-L. Yang, H.-W. Hu, H.-S. Chang, H.-P. Li, and M.-F. Chang, "DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning," in *International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[4] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *International Symposium on Computer Architecture (ISCA)*, 2019, pp. 236–249.

[5] G. Yuan, P. Behnam, Z. Li, A. Shafiee, S. Lin, X. Ma, H. Liu, X. Qian, M. N. Bojnordi, Y. Wang, and C. Ding, "FORMS: Fine-grained polarized reram-based in-situ computation for mixed-signal dnn accelerator," *arXiv preprint arXiv:2106.09144*, 2021.

[6] B. Li, Y. Wang, and Y. Chen, "Hitm: High-throughput reram-based pim for multi-modal neural networks," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1891–1898.

[9] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2014.