PIM-HLS: An Automatic Hardware Generation Tool for Heterogeneous Processing-In-Memory-based Neural Network Accelerators

Yu Zhu[1], Zhenhua Zhu[1], Guohao Dai[2], Fengbin Tu[3], Hanbo Sun[1], Kwang-Ting Cheng[3], Huazhong Yang[1], Yu Wang[1]
[1]Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China
[2]Qingyuan Research Institute, Shanghai Jiao Tong University, Shanghai, China
[3]Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong
yu-wang@tsinghua.edu.cn

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247755

***Abstract*—Processing-in-memory (PIM) architectures have shown great abilities for neural network (NN) acceleration on edge devices that demand low latency under severe area constraints. Heterogeneous PIM architectures with different PIM implementation approaches such as RRAM-based PIM and SRAM-based PIM can further improve the performance. However, the automatic generation of heterogeneous PIM architectures faces the following two unresolved problems. First, existing work has not considered the design for heterogeneous PIM-based NN accelerators with multiple memory technologies. Second, for PIM with insufficient memory on edge devices, it is challenging to find the optimal runtime weight scheduling strategy in an $O(L!)$ optimization space for the NN with L layers.**

In this paper, we propose PIM-HLS, an automatic hardware generation tool for heterogeneous PIM-based NN accelerators. Aiming at the problems above, we first point out that heterogeneous PIM can improve the performance under severe area constraints. Then we optimize the architectures for each NN layer by taking the advantage of different memory technologies. We also define the optimization problem of runtime weight scheduling and mapping for the first time, and propose a dynamic-programming-based weight scheduling algorithm to reduce the optimization space to $O(L^2)$. We implement PIM-HLS to automatically generate the hardware code and the instructions. Results show that we achieve an averagely $5.9\times$ speedup with 72.8% less area compared with state-of-the-art PIM designs.

I. Introduction

With the development of neural networks (NNs), intelligent edge devices start to become important in our daily life. But new edge applications such as augmented reality and tactile internet demand ms-level ultra-low latency under mm^2-level severe area constraints [1], bringing significant challenges to the computing capability of edge devices. For traditional CMOS-based NN accelerators, the numerous data movements between the memory and computing units harm the performance significantly [2].

In recent years, processing-in-memory (PIM) architectures have become a promising way to enhance the performance of NN acceleration on edge devices [3]. Unlike CMOS-based von Neumann architectures, typical PIM architectures perform computations within the memory arrays, avoiding the overhead of data movements. Currently, different kinds of memory technologies provide various PIM implementation approaches [4]–[8], bringing advantages in different aspects. As shown in Figure 1(a), SRAM-based PIM architectures based on [4]–[6] have better performance, while RRAM-based PIM architectures based on [7], [8] are better at memory density. Fortunately, the development of heterogeneous memory integration techniques [9] provides opportunities to integrate different PIM implementation approaches in one chip. Therefore, we

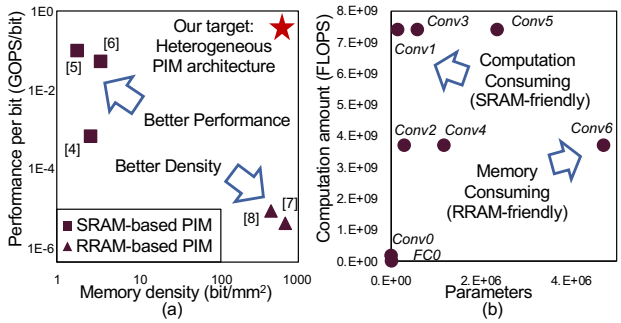

Fig. 1. (a) The memory density and normalized performance of architectures based on recent PIM work. (b) The parameter and computation amount of the layers in VGG-8.

are inspired to take their advantages simultaneously to further improve the performance for edge applications.

Intelligent edge devices face the contradiction between high performance requirements and severe area constraints. As many intelligent edge devices use one specific NN model [3], it is urgent to perform customized hardware design and optimizations in order to fully exploit the performance. However, because PIM-based architectures perform *in-situ* computations, the optimization space is significantly different from CMOS-based von Neumann architectures, making traditional automatic hardware generation tools inapplicable. The automatic generation of PIM-based NN accelerators mainly faces the following two problems that remain unresolved.

On the one hand, existing PIM design work only focuses on homogeneous architectures with a single implementation approach, ignoring the potential benefits brought by heterogeneous integration. As shown in Figure 1(b), the amounts of parameters and computations of different layers in one network vary a lot. Similarly, the PIM implementation approaches will affect the area and computing performance. So they need to be specifically optimized for each layer to achieve the optimal performance. For instance, applying SRAM-based PIM to the most time-consuming layer in VGG-8 [10] will reduce 77% latency with 1.3% more area overhead compared with using RRAM-based PIM only. However, introducing layer-wise hardware optimization will cause the design space to grow exponentially with the number of layers, e.g., 10^{37} for ResNet-18, making the optimization problem challenging.

On the other hand, existing PIM work assumes that all the weights can be mapped to the PIM accelerator, which is impractical for intelligent edge devices due to the severe area constraints. As the erase and rewrite overhead of PIM-based architectures is usually large [11], it is significant to optimize

the runtime weight scheduling and mapping strategy to minimize the overhead. Despite when to map each layer to the PIM accelerator, determining whether to split or duplicate the layers to take full use of the memory resource also contributes to the optimization space. As a result, the optimization space of weight scheduling is tremendous, e.g., 10^{15} for ResNet-18.

Inspired by traditional high level synthesis (HLS) tools, we propose PIM-HLS[1], an automatic hardware optimization and generation tool for PIM-based NN accelerators. PIM-HLS takes the NN structure as the input, automatically generating the optimized hardware description language (HDL) code with the instructions for each module of the architecture. The HDL code and instructions can be used directly for cycle-accurate simulation and functional verification. The contributions of this paper include:

- We point out that heterogeneous PIM with multiple PIM implementation approaches can improve the performance. By analyzing each layer's demands, we propose criteria-based sorting to optimize the PIM implementations for each layer. Such design methodology improves the performance by $3.7\times$ compared with homogeneous designs.
- We define the optimization problem of runtime weight scheduling and mapping for the first time, and find the optimal strategy in polynomial time. We apply data flow graph (DFG) topological sorting to guarantee the data dependencies, and propose a dynamic-programming (DP) -based weight scheduling algorithm to optimize the scheduling strategy. The optimization space is reduced from $O(L!)$ to $O(L^2)$ for the network with L layers.
- We implement PIM-HLS to generate the HDL code and instructions automatically. We compare the performance of the architectures generated by PIM-HLS with state-of-the-art PIM-based architectures under different area constraints. Results show that the architectures generated by PIM-HLS achieve an averagely $5.9\times$ speedup with 72.8% less area compared with state-of-the-art PIM work.

II. Preliminaries

A. High Level Synthesis

HLS tools are used to generate HDL code from high-level language code for CMOS-based architectures. Traditional HLS tools first analyze the data flow and computation flow of the high-level language code, then perform evaluations and design space explorations to generate optimized control flow and hardware architectures under given constraints. With the development of NN algorithms, recent work proposes HLS frameworks to generate accelerators specifically for NNs. DeepBurning [12] designs HLS frameworks for FPGA implementations, supporting multiple kinds of NNs. AutoDNNchip [13] proposes an HLS tool for NNs on both FPGAs and ASICs. In summary, the existing work only aims at CMOS-based homogeneous architectures. For heterogeneous PIM architecture generation, how to assign different PIM implementation approaches to NN layers with various amount of parameters and computations becomes a significant problem.

[1]The code is available in https://github.com/Hazuyuki/PIM-HLS.git

B. Design Automation of PIM-based Architectures

Recent work tries to develop design automation tools to accelerate the design flow of PIM architectures. Modeling and simulation work such as MNSIM [14] and NeuroSIM [15] models the latency, area, and power of PIM architectures, helping hardware designers evaluate the designs conventionally. Design space exploration work such as Gibbon [16] automatically search for the optimal hardware parameters and network structures to achieve the best performance, but existing work focuses on homogeneous PIM architectures. Besides, existing work assumes that all the weights of one network can be stored on the PIM accelerator, which is impractical for edge devices. Due to the contradiction between the increasing amount of NN parameters and the severe area limitations of intelligent edge devices, how to schedule and map the weight data during the runtime becomes an urgent challenge.

III. Overview of PIM-HLS

Figure 2(a) demonstrates the overall framework of PIM-HLS. PIM-HLS mainly contains three stages, i.e., preprocessing, optimization, and generation.

In the preprocessing stage, PIM-HLS first takes the NN description file as the input. The file provides the information of each layer, and the dimensions of all the input and activation tensors. Then the file is fed into the DFG analyzer to determine the data dependencies among layers and generate the DFG. At the same time, we employ PIM simulator [14] to realize layer-wise latency and area evaluations under multiple hardware settings, i.e., different PIM implementation approaches, different array sizes, and different interface numbers for each array. The device-level hardware data can be defined by users, and the evaluation results will be used as the fundamental data for the following optimization stage.

In the optimization stage, the heterogeneous PIM optimizer will optimize the hardware design for the NN layers. The optimizer will first perform PIM implementation approach selection before scheduling (Section IV-A), then finely adjust the hardware parameters (i.e., the array size and the interface numbers) after scheduling (Section IV-C), in order to optimize the performance under a given area constraint. The scheduler will determine the weight mapping and scheduling strategy under a given area constraint (Section IV-B).

Finally in the generation stage, the control flow generator will generate the instructions according to the scheduling optimization results. Also, the hardware generator will generate the HDL code using pre-defined hardware templates, as shown in Figure 2(b). The basic hardware architecture of PIM-HLS refers to MNSIM [14]. Different layers of the network is mapped to different tiles, and connected with the network on chip (NoC). Unlike MNSIM which has the same hardware configurations for all the tiles, PIM-HLS regards the tiles as the minimum granularity of the hardware optimization. Therefore, PIM-HLS performs finer-grained design space explorations. Notice that we use pseudo PIM arrays that consist of register files with delay-configurable multiply accumulate (MAC) units instead of real PIM intellectual properties (IPs). Users can either directly use the pseudo PIM arrays for cycle-accurate

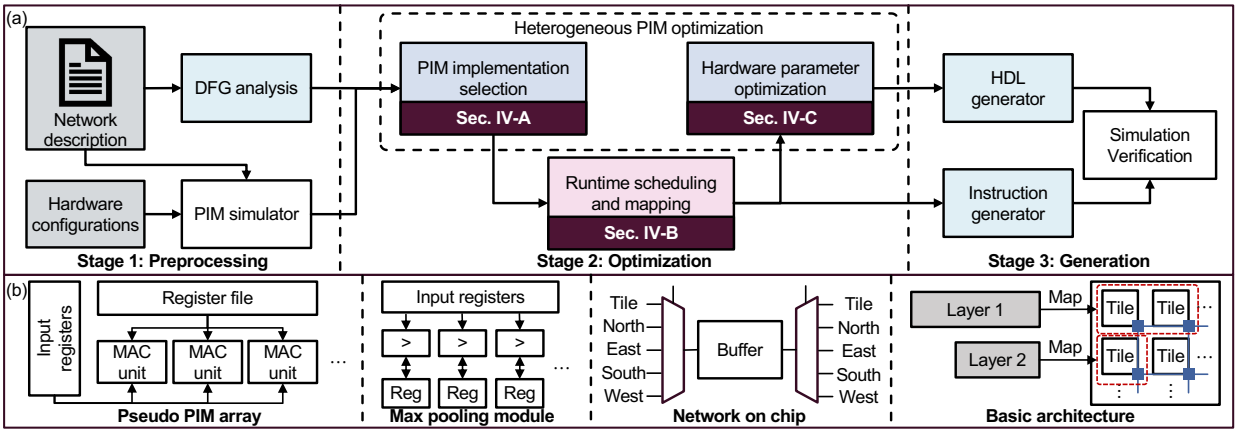

Fig. 2. (a) Overview of PIM-HLS. (b) Our main hardware templates and basic architecture.

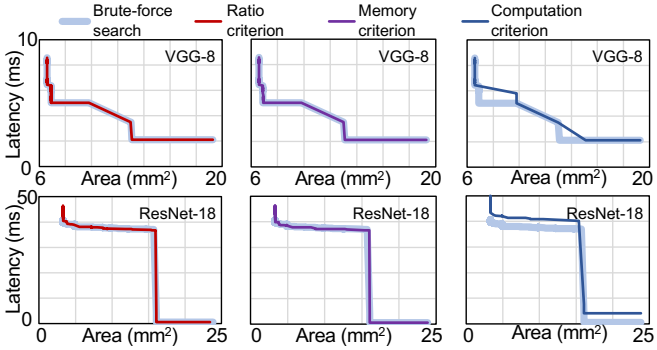

Fig. 3. The Pareto frontier of the candidates generated by the three criteria compared to the brute-force search.

simulation and functional verification, or simply replace them with real PIM IPs for early-stage chip design.

IV. Hardware and Scheduling Optimization

In this section, we will introduce the optimization stage of PIM-HLS. The purpose of this stage is to optimize the heterogeneous PIM architecture design and the weight scheduling and mapping strategy to achieve the best performance under a given area constraint. As discussed in Section I, the entire optimization space is enormous, e.g., $> 10^{52}$ for ResNet-18. To find the optimal design in such a large design space, we split the optimization stage into three steps, i.e., the coarse-grained PIM implementation approach selection, the runtime mapping and scheduling optimization, and the fine-grained hardware parameter optimization, as shown in Figure 2(a).

A. Coarse-Grained PIM Implementation Approach Selection

Motivation. As described in Section I, different layers of the network prefer different PIM implementation approaches, enabling heterogeneous PIM architectures to improve the performance with little overhead.

Problem Definition. In this step, our goal is to generate candidate designs that assign different PIM implementation approaches to the layers, denoted by $R_i = a$ for assigning implementation approach a to layer i. Assuming that the network has L layers and we support K kinds of implementation approaches, $O(K^L)$ candidates will be generated for the scheduling step. When the network becomes deeper, the number of the candidates will rise exponentially.

Observation. Different PIM implementation approaches have different advantages on memory density and computation capability, and different layers have various demands on memory space and computation capabilities. We observe that it will be beneficial to match the demands with the advantages of PIM. E.g., for VGG-8 implementation, assigning SRAM-based PIM to the most time-consuming layer will improve the performance with little area overhead, as described in Section I. In other words, matching more computational-demanding layers to PIM implementation approaches with higher computation capability benefits to the performance with little area overhead in this case.

Method. To match the demands on memory and computation of the layers to the advantages of different PIM implementation approaches, we propose multiple criteria to quantify them. The proposed criteria include the memory (i.e., memory density for PIM and memory demands for the layers), the computation (i.e., computation capability for PIM and computation demands for the layers), and the ratio between memory and computation. We evaluate the layers' demands and the advantages of PIM with the criteria, in order to select the optimal PIM implementation approaches for the layers.

Taking the memory criterion as an example, our method will first calculate the memory consumption of each layer (denoted by Cl_i for layer i) and the memory density of each PIM implementation approach (denoted by Cr_a for approach a). To assign the PIM implementation approaches that have larger memory density to the layers consuming larger memory, we sort the layers and the approaches in the order of the memory consumption and the memory density, respectively. Finally, we search for all the candidate designs that satisfy Equation 1:

$$\begin{aligned} &\forall i, j < L, R_i = a, R_j = b, \\ &(Cr_a - Cr_b)(Cl_i - Cl_j) \geq 0, \end{aligned} \tag{1}$$

which guarantees the implementations with higher memory density to be assigned to the layers requiring larger memory.

The optimization space of the proposed criteria-based method is approximately $O(L^{K-1})$. Because there are only a few PIM implementation approaches to be considered in real circumstances, while the neural networks can have tens or hundreds of layers (i.e., $L \gg K$), the proposed method

is much faster than the naive brute-force search in practice. E.g., the number of candidates will be 262144 and 18 for the naive method and the proposed method, respectively, when considering SRAM-based PIM and RRAM-based PIM for ResNet-18. We also evaluate the proposed method on various networks to figure out whether it can find the optimal design. As illustrated in Figure 3, the candidates generated by the criteria of the memory and the ratio of memory and computation are mostly on the Pareto frontier of the whole search space, proving the practicability of the two criteria and the criteria-based method.

B. Runtime Weight Scheduling and Mapping

Motivation. With the severe area constraints of intelligent edge devices, PIM architectures may not be able to store all the parameters of the network. So it is necessary for PIM architectures to support runtime weight scheduling and mapping for the deployment of NNs on edge devices.

A straightforward method for runtime weight scheduling and mapping is to map as many layers as possible for each time, in order to reduce the number of memory writes. However, this greedy algorithm faces the problem of the low memory utilization, resulting in severe performance loss. E.g., the first three layers of VGG-8 consume $3.6Mb$ memory, while the fourth layer takes $4.7Mb$. If using the greedy algorithm to map VGG-8 to the $8Mb$ RRAM-based PIM architecture, the first three layers will be firstly mapped, and 55% memory will be idle during computation. However, because the second layer of VGG-8 consumes much latency, we can choose to firstly map the first two layers, and duplicate the weights of the second layer into maximally six copies. The total computation latency can be reduced by 42% in this way.

Problem Definition. To simplify the scheduling and mapping problem, we assume that we choose several layers to be mapped to the accelerator simultaneously for each time, called a layer group. Only when the computation of one layer group is finished can we map and compute the next layer group. So, the scheduling problem can be transformed to assigning the layers to multiple layer groups to achieve the optimal performance, with consideration of the area constraints and data dependencies. Because the maximum number of the layer groups is at least L, and there are L layers to be scheduled, the design space of the scheduling problem can reach $O(L!)$.

Observation. We observe that the layers must be scheduled sequentially due to the data dependencies of NN computation. Assuming that the ith layer L_i is in the jth layer group G_j, then L_{i+1} can only be assigned to G_j or G_{j+1}. In other words, if G_j contains the layers $L_{p+1} \sim L_i$, $p \leq i$, the scheduling strategy of the layers $L_1 \sim L_p$ has no aftereffect to the scheduling strategy of L_{i+1}. This inspires us that the scheduling optimization problem can be transformed to a dynamic programming problem, as formulated in Equation 2,

$$Lat_0 = 0, Lat_{i+1} = \min_{0 \leq p \leq i} (Lat_p + F(p+1, i+1)), \quad (2)$$

where Lat_p represents the entire latency of the best scheduling strategy for layers $L_1 \sim L_p$, and $F(a, b)$ denotes the mapping and computation latency of the layer group that contains layers $L_a \sim L_b$.

Algorithm 1 Runtime Weight Scheduling and Mapping

Input: DFG: The data flow graph; L_i: Information of layer i; A: The area constraint.
Output: G_i: Layer groups.

```
1: Topological_Sort(DFG, L_i);
2: for i = 1 to L do
3:     if Area(L_i) > A then
4:         Split(L_i, ⌈Area(L_i)/A⌉);
5:     end if
6: end for
7: Lat_0 = 0;
8: for i = 1 to L do
9:     Lat_i = ∞;
10:    for p = 0 to i − 1 do
11:        Temp_group = {L_{p+1}, L_{p+2}, ..., L_i};
12:        Duplicate(Temp_group)
13:        if Lat_i > Lat_p + F(Temp_group) then
14:            Lat_i = Lat_p + F(Temp_group)
15:            Group_start_i = p
16:        end if
17:    end for
18: end for
19: Compute G according to Group_start_i;
```

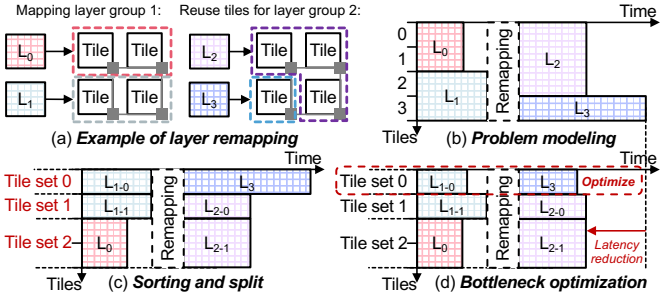

Fig. 4. (a) An example of mapping two layer groups to four tiles. (b) The problem is how to achieve the minimum latency by adjusting the hardware parameters of each tile. (c) Sort the layers by the computing capabilities and divide the tiles into tile sets. (d) Optimize the bottleneck tile sets.

Method. Based on the observation above, we propose a DP-based weight scheduling algorithm to reduce the optimization space to $O(L^2)$, with considering of layer split and duplication. The proposed scheduling and mapping algorithm is shown in Algorithm 1. To guarantee the data dependencies, we first perform topological sorting for the layers according to the DFG (**line 1**). For the layers that consume larger area than the area constraint, we split their weights into multiple pieces (**line 2** $\sim$ **line 6**). During the DP-based weight scheduling algorithm (**line 8** $\sim$ **line 18**), we first perform layer duplication, duplicating the most time-consuming layer in the layer group until the area is insufficient, and then find the optimal Lat_i with Equation 2 and record the choice in $Group\_start_i$. After the DP-based algorithm, we iterate $Group\_start_i$ backwards to determine the optimal layer groups, i.e., the optimal scheduling and mapping strategy.

C. Fine-Grained Hardware Parameter Optimization

Motivation. Despite the PIM implementation approaches, the hardware parameters also contribute to the tradeoff between the performance and area. Especially for analog PIM such as RRAM-based PIM, the high-precision analog-digital

converters (ADCs) and digital-analog converters (DACs) consume large area overhead. But more ADCs and DACs for each array will also enable higher parallelism. Setting different hardware parameters for different layers will improve the performance under certain area constraints. E.g., when deploying VGG-8 and the total array area is limited to $10mm^2$, the latency will be reduced by 65% if only applying $4\times$ more ADCs to the most consuming layer for RRAM-based PIM architectures. However, because of the runtime weight scheduling and mapping proposed in Section IV-B, multiple layers have to be mapped to the same arrays to reuse the hardware resource as shown in Figure 4(a), preventing us from separately optimizing the hardware parameters for each layer.

Problem Definition. As described in Section III, we regard the tiles as the minimum granularity of the hardware optimization. To model the optimization problem, we set layers to different tiles in Figure 4(b). The abscissa in Figure 4(b) represents the execution time, and the ordinate represents the tiles. For each layer, its position in the figure determines when and to which tiles it should be mapped. Our goal is to optimize the hardware configurations of each tile to achieve the minimum latency under a given area constraint. Simply performing a brute-force search is time-consuming as the search space can reach $O(C^T)$, where C is the number of design choices for each tile and T is the tile number.

Observation. To reduce the search space, we make an assumption and propose two observations. We assume that the layers of one layer group are executed in a pipeline, which means the computation latency of one layer group can be estimated by the latency of the most time-consuming layer. Based on this assumption, we observe that if multiple tiles store the weights for one layer, we should give these tiles the same hardware parameters, as the entire latency is determined by the most time-consuming tile. The second observation is that the layers with higher computation demands should be mapped to tiles with higher computation capabilities, in order to reduce the bottleneck latency. Therefore, we regard the more computational-demanding layers as the more important layers, and optimize their latency at a higher priority.

Method. Based on the two observations, we heuristically map the layers to the memory tiles and optimize the hardware parameter for the bottleneck layers. First, as shown in Figure 4(c), we sort the layers in each layer group by their computation amounts. Therefore, the computational-demanding layers of different layer groups will reuse the same tiles, which need to be optimized at a high priority. Second, we split the tiles into tile sets, ensuring that every tile of the same tile set works for only one layer in each layer group. Thus we can perform optimizations for tile sets instead of tiles, and reduce the search space from $O(C^T)$ to $O(C^L)$, where L represents the number of the layers of the network. Finally, as illustrated in Figure 4(d), we only optimize the hardware parameters of the tile sets, which are assigned to the bottleneck layers. Because there are at most L tile sets, and we perform at most C times of trials to optimize each tile set, the bottleneck-optimization algorithm further reduces the optimization space to $O(CL)$. Compared with MNSIM [14] evaluation results, our latency estimation has an averagely 3.5% error due to the initiation interval of the pipeline and the inter-tile data communications. The result shows that the proposed pipeline assumption is practical for the optimization.

V. Evaluations

A. Evaluation Setup

PIM-HLS employs the open-source PIM simulator MNSIM [14] to evaluate the performance and area of different NN layers with different hardware configurations. For the PIM implementation approaches, we consider SRAM-based PIM and RRAM-based PIM, and the device parameters refer to [17] and [18], respectively. We use homogeneous PIM-based architectures based on state-of-the-art work [2] and [17] as baselines, which are also evaluated by MNSIM for fair comparison. We use two typical convolutional neural networks (CNNs) for evaluation, i.e., VGG-8 [10] and ResNet-18 [19]. And PIM-HLS can also be used for other CNN models.

B. Evaluation Results

We compare the architectures that are optimized and generated by PIM-HLS with the RRAM-PIM baseline [2] and the SRAM-PIM baseline [17]. The two baselines consume $39mm^2$ and $47mm^2$ area on VGG-8 dataset, and $93mm^2$ and $62mm^2$ area on ResNet-18 dataset, according to the MNSIM evaluation. As illustrated in Figure 5, to show the abilities of PIM-HLS under different area constraints, we provide the latency data of architectures generated by PIM-HLS under $16mm^2$ and $40mm^2$ area constraints. The former constraint represents a typical area for edge devices [20], while the latter is comparable to the baselines. The results show that our heterogeneous PIM-designs under $16mm^2$ area constraint can achieve $4.8\times$ performance improvement, while saving $3.7\times$ area on average at the same time. When the area constraint is relaxed to $40mm^2$, which is comparable with the baselines, the performance improvements come up to $11.7\times$ and $7.5\times$ on the two datasets. Despite the heterogeneous PIM architectures with both SRAM and RRAM, PIM-HLS can also optimize the hardware parameters and perform scheduling for RRAM-based architectures and SRAM-based architectures. As shown in Figure 5, PIM-HLS enables the deployment of the networks when the memory capacity is insufficient, while bringing $10.0\times$ and $6.3\times$ speedup for RRAM-based architectures and $10.1\times$ and $6.0\times$ speedup for SRAM-based architectures with comparable area to the baselines.

C. Ablation Study

In this section, we discuss the throughput improvement brought by the optimizations proposed in Section IV under typical area constraints of edge devices [20]. We evaluate the throughput of different PIM implementation approaches (i.e., RRAM, SRAM, and Heterogeneous in Figure 6) with different optimizations (i.e., -S for only applying the scheduling optimization, and -SP for applying both the scheduling and parameter optimization in Figure 6).

As shown in Figure 6(a), when the area constraint rises from $10mm^2$ to $20mm^2$, the throughput of RRAM-PIM starts

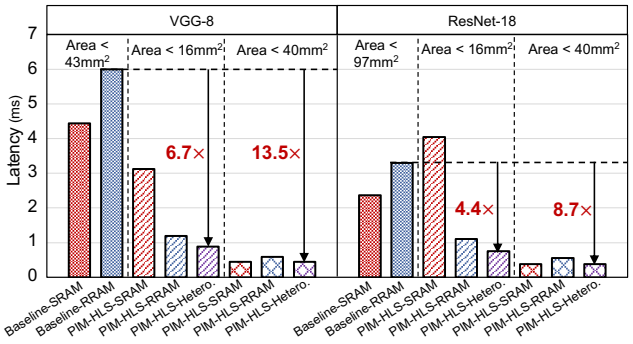

Fig. 5. Latency comparisons between state-of-the-art PIM architectures and architectures generated by PIM-HLS.

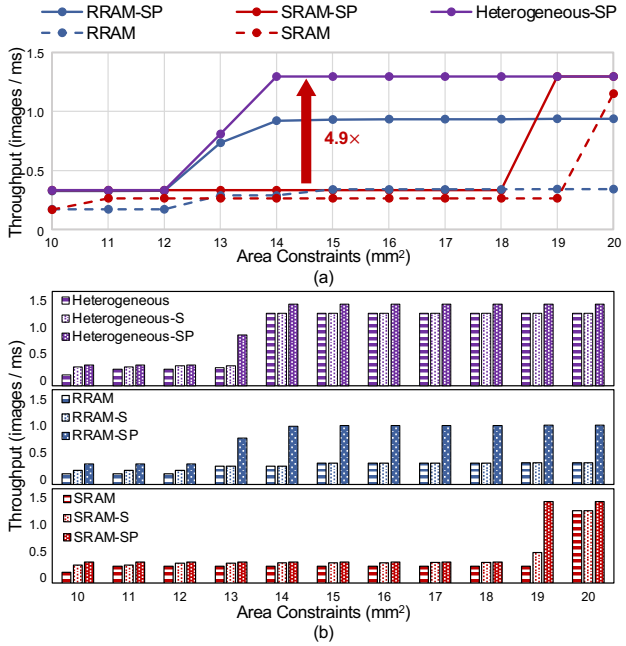

Fig. 6. The throughput improvements brought by (a) PIM implementation approach selection; (b) weight scheduling and parameter optimization.

to rise before SRAM-PIM due to RRAM PIM's high storage density. But the SRAM-based designs achieve higher throughput when the area is sufficient due to SRAM-PIM's high computation capability. The heterogeneous PIM architectures achieve the best throughput within a small area, showing the advantages of both RRAM and SRAM.

According to Figure 6(b), with insufficient area, the weight scheduling and mapping optimization brings $1.3\times$, $1.7\times$, and $1.7\times$ speedups for RRAM, SRAM, and heterogeneous PIM, respectively. As SRAM-PIM is more area-constrained, the scheduling optimization provides higher improvements for SRAM-PIM than RRAM-PIM. Also, due to the large overhead of high-precision interfaces in RRAM-PIM, the hardware parameter optimization provides more throughput improvement for RRAM-PIM (i.e., average $2.4\times$) than the SRAM-PIM and the heterogeneous designs (i.e., $1.2\times$ for both of them).

VI. Conclusions and Future Work

In this work, we propose PIM-HLS, an automatic generation tool for heterogeneous PIM-based NN accelerators. We stress out the key problems of heterogeneous PIM design and runtime weight scheduling and mapping, and find the optimal architecture design and scheduling strategy under a given area constraint. Results show that we achieve an averagely $5.9\times$ speedup with 72.8% less area compared to state-of-the-art PIM designs. In the future, we will adapt PIM-HLS to support the optimization for multiple NN models on one PIM-based accelerator, thus enabling the automatic generation of PIM-based accelerators for a wider range of applications.

VII. Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. U19B2019, 61832007, 62104128, U21B2031), Tsinghua EE Xilinx AI Research Fund, Tsinghua-Meituan Joint Institute for Digital Life, and Beijing National Research Center for Information Science and Technology (BNRist).

References

[1] C. S. M. Babou *et al.*, "Home edge computing (hec): Design of a new edge computing technology for achieving ultra-low latency," in *International conference on edge computing*, pp. 3–17, 2018.

[2] P. Chi *et al.*, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, pp. 27–39, 2016.

[3] F. Tan *et al.*, "A reram-based computing-in-memory convolutional-macro with customized 2t2r bit-cell for aiot chip ip applications," *IEEE TCAS-II*, vol. 67, no. 9, pp. 1534–1538, 2020.

[4] P.-C. Wu *et al.*, "A 28nm 1mb time-domain computing-in-memory 6t-sram macro with a 6.6 ns latency, 1241gops and 37.01 tops/w for 8b-mac operations for edge-ai devices," in *ISSCC*, vol. 65, pp. 1–3, 2022.

[5] F. Tu *et al.*, "A 28nm 29.2 tflops/w bf16 and 36.5 tops/w int8 reconfigurable digital cim processor with unified fp/int pipeline and bitwise in-memory booth multiplication for cloud deep learning acceleration," in *ISSCC*, vol. 65, pp. 1–3, 2022.

[6] F. Tu *et al.*, "A 28nm 15.59 μj/token full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," in *ISSCC*, vol. 65, pp. 466–468, 2022.

[7] J.-M. Hung *et al.*, "An 8-mb dc-current-free binary-to-8b precision reram nonvolatile computing-in-memory macro using time-space-readout with 1286.4-21.6 tops/w for edge-ai devices," in *ISSCC*, 2022.

[8] C.-X. Xue *et al.*, "16.1 a 22nm 4mb 8b-precision reram computing-in-memory macro with 11.91 to 195.7 tops/w for tiny ai edge devices," in *ISSCC*, vol. 64, pp. 245–247, 2021.

[9] A. Kaul *et al.*, "Beol-embedded 3d polylithic integration: Thermal and interconnection considerations," in *ECTC*, pp. 1459–1467, 2020.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] W. Li *et al.*, "A 40nm rram compute-in-memory macro featuring on-chip write-verify and offset-cancelling adc references," in *ESSCIRC*, 2021.

[12] Y. Wang *et al.*, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *DAC*, 2016.

[13] P. Xu *et al.*, "Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics," in *FPGA*, pp. 40–50, 2020.

[14] Z. Zhu *et al.*, "Mnsim 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems," in *GLSVLSI*, pp. 83–88, 2020.

[15] X. Peng *et al.*, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE TCAD*, vol. 40, no. 11, pp. 2306–2319, 2020.

[16] H. Sun *et al.*, "Gibbon: efficient co-exploration of nn model and processing-in-memory architecture," in *DATE*, pp. 867–872, 2022.

[17] B. Yan *et al.*, "A 1.041-mb/mm 2 27.38-tops/w signed-int8 dynamic-logic-based adc-less sram compute-in-memory macro in 28nm with reconfigurable bitwise operation for ai and embedded applications," in *ISSCC*, vol. 65, pp. 188–190, 2022.

[18] C.-X. Xue *et al.*, "24.1 a 1mb multibit reram computing-in-memory macro with 14.6 ns parallel mac computing time for cnn based ai edge processors," in *ISSCC*, pp. 388–390, 2019.

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

[20] F. Wu *et al.*, "A review of convolutional neural networks hardware accelerators for aiot edge computing," in *UCET*, pp. 71–76, 2021.