DyPIM: Dynamic-Inference-Enabled Processing-In-Memory Accelerator

Tongxin Xie[1], Tianchen Zhao[1], Zhenhua Zhu[1]✉, Xuefei Ning[1], Bing Li[2], Guohao Dai[3], Huazhong Yang[1], Yu Wang[1]✉
[1]Dept. of EE, BNRist, Tsinghua University, [2]Capital Normal University, [3]Shanghai Jiao Tong University
✉ zhu-z14@outlook.com, yu-wang@tsinghua.edu.cn

***Abstract*—Dynamic neural network is an emerging research topic in deep learning. Dynamic networks selectively skip redundant computations conditioned on the input during inference (i.e., dynamic inference). And they have demonstrated superior trade-offs between accuracy and inference efficiency. However, memory I/O turns irregular and dominant because of the fine-grained computation skip in dynamic networks. Processing-In-Memory (PIM) can perform Matrix-Vector Multiplications inside the memory, eliminating the data movement of network parameters. So, it is promising to address the memory I/O challenge. However, deploying dynamic networks on PIM architectures faces severe performance degradation caused by (1) Pipeline stall when deciding on computation to be skipped. (2) Mismatch between fine-grained algorithm computation skip and coarse-grained hardware computing granularity. (3) Improper proxy of hardware performance during training. To tackle these problems, we propose DyPIM, the dynamic inference-enabled PIM accelerator with software-hardware co-optimizations. At the algorithm level, a PIM-friendly dynamic network with a standalone mask generation network and a throughput-optimal training technique is proposed. At the hardware level, a PIM architecture supporting dynamic networks is proposed, with a pipeline controller to process the dynamic dataflow. Peripheral circuits are also designed in processing units to enable non-contiguous activating of non-zero wordlines to better utilize the computation skip. Experiments show that DyPIM can achieve $1.52\times$ to $2.74\times$ speedup and $2.05\times$ to $3.95\times$ throughput improvement over the existing PIM architectures for ResNet networks.**

I. Introduction

In recent years, convolutional neural networks (CNNs) have demonstrated remarkable potential across various fields, such as computer vision [1] and natural language processing [2]. To improve the computational efficiency of CNNs, researchers have been actively exploring dynamic neural networks. By selectively skipping redundant computations of the model during inference conditioned on the input (i.e., dynamic inference), dynamic networks can achieve superior trade-offs between accuracy and efficiency [3]. For example, BranchyNet [4] utilizes the Early-Exit strategy, which partitions the model into multiple stages and processes simpler inputs with fewer stages compared to more complex ones, and achieves $2\times$ to $6\times$ speedup on CPU and GPU. DGNet [5] introduces dynamic pruning, which dynamically skips specific pixels and channels by applying masks on the features during the convolution process, and can provide 59.7% computation reduction without loss of accuracy.

Although having demonstrated superiority in reducing computation while preserving accuracy, dynamic networks working on a finer granularity (e.g., spatially dynamic networks) suffer from irregular I/O pattern caused by sparse activations and weights. Moreover, as the computation overhead decreases in dynamic networks, the hardware becomes increasingly memory bounded. In such cases, memory I/O significantly impacts the overall hardware efficiency [6].

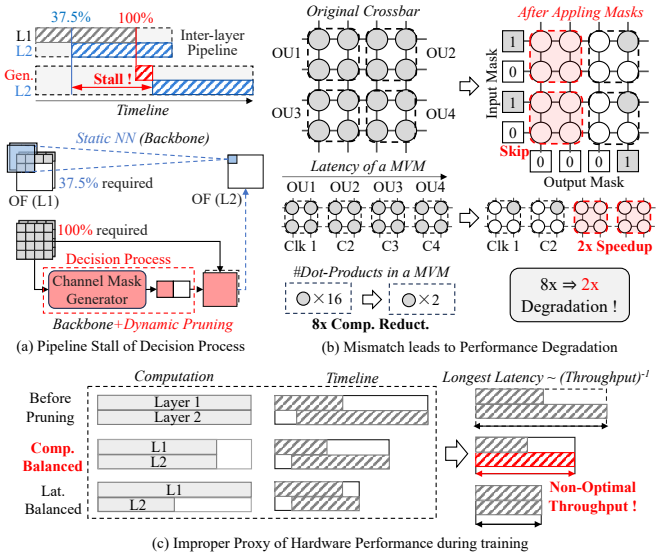

Fig. 1. Challenges in Deploying Dynamic Networks to PIM

Processing-In-Memory (PIM) architectures [7]–[10] based on emerging Non-Volatile Memories (NVMs) is a promising solution to the memory I/O problem in neural networks. PIM architectures can perform Matrix-Vector Multiplications (MVMs) inside the memory, allowing for inference of CNNs without the need for extensive data movements of weights. However, there exist challenges when deploying dynamic inference on PIM accelerators. First of all, dynamic networks stall the inter-layer pipeline of PIM architectures when deciding the computations to be skipped, causing PIM latency deterioration. Inter-layer pipeline starts computation once the required input data of a layer are ready, rather than waiting for the previous layer to complete all the calculations [9]. But the decision process of dynamic networks takes the entire intermediate feature map as inputs to generate decision results. And the following layers require decision results to select the part of calculation to be executed. Thus different convolutional layers must be computed sequentially, which hinders the inter-layer pipeline and leads to over $2.5\times$ latency increase in the deployment of DGNet [5].

Secondly, the mismatch between the granularity of algorithm computation skip and MVM execution in PIM accelerators leads to the gap between computation reduction and actual hardware speedup. In the fine-grained dynamic inference like dynamic pruning, computation reduction comes from the sparsity in the spatial and channel dimensions of the pruned feature map [5]. But when conducting MVM in a PIM memory array, dot-product computations on weight are conducted at a coarser

979-8-3503-4859-0/DATE24/© 2024 EDAA

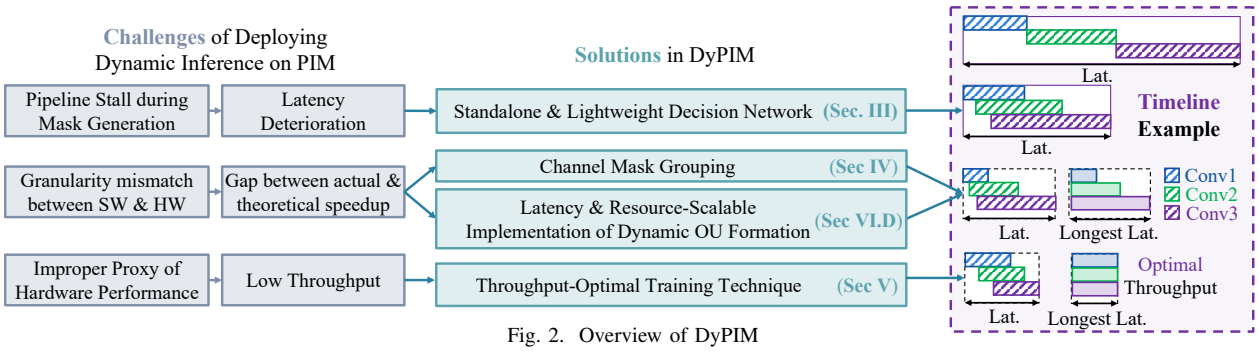

Fig. 2. Overview of DyPIM

granularity containing multiple pixels and channels, which is called Operation Unit (OU). OU is the maximum region of weights that can be computed together in a cycle [11]. The size of OU is decided by the number of wordlines and bitlines that can be activated simultaneously. Fig.1(b) shows an example of OU-based MVM computation with 2×2 OU. Weights in four OUs are computed in order in four cycles. An OU is skipped only if all the weights in it are skipped, so the latency reduction ($2\times$) is much lower than the computation reduction ($8\times$).

Finally, training techniques of dynamic networks are mainly designed to reduce the overall computation, which is not a suitable proxy for the hardware performance of PIM architectures. This improper proxy can lead to nearly $2\times$ throughput degradation. In traditional architectures like CPU and GPU, the processing latency is proportional to the amount of computation. However, in the PIM architecture, to avoid the write overhead of NVM, weights are typically fixed within the PIM crossbar. Due to varying parameter quantities across different layers and the varying number of PIM crossbars employed, the compute capability differs across layers. Consequently, the amount of computation cannot serve as a proper proxy for PIM processing latency. As shown in Fig.1(c), balanced pruning on the computation of two layers will cause unbalanced latency, and thus non-optimal throughput.

To tackle these challenges, we present DyPIM, a software-hardware co-design strategy that can enable and optimize dynamic inference in PIM architectures, as is shown in Fig.2.

The contributions of our work are as follows:

- We propose a pipeline-enabled dynamic network that utilizes a standalone and lightweight decision network to predict channel masks for the backbone network. The decision network calculates the channel mask directly from the original image, which does not require the intermediate feature. As a result, it can recover the inter-layer pipeline.
- We propose a granularity alignment strategy to better utilize computation skip. At the algorithm level, channel grouping is utilized to merge the granularity of computation skip. At the hardware level, computing granularity is broken down by selectively activating non-contiguous wordlines to form OUs [11]. Our implementation of the OU formation strategy can reduce the number of required adders from $N-1$ to K^2-1, and the cycles for process from $\lceil log_2 N \rceil$ to $\lceil log_2 K^2 \rceil$ in an $N \times N$ crossbar mapped with a $K \times K$ kernel.
- We designed a throughput-optimal training technique, which can achieve balanced latency across layers by reducing the computation of the layer with the longest latency during training process. It also prevents unnecessary computation reduction on layers with shorter latency for better accuracy.

Experiments show that DyPIM can achieve $1.52\times$ to $2.74\times$ speedup and $2.05\times$ to $3.95\times$ throughput speedup over the baseline ResNet networks.

II. Background

A. PIM Architectures

PIM architectures include different levels of hierarchies. A typical architecture contains multiple tiles connected together through point-to-point or network-on-chip connection [10], while each tile possesses several processing elements (PEs) composed of crossbars of NVMs. During weight mapping, kernels are divided into smaller sub-kernels which can be directly mapped to a PE, and crossbars within the PE are mapped with different bits of the sub-kernel [8].

B. Dynamic Pruning Algorithms

Dynamic pruning methods select the most relevant part of the network for inference based on specific inputs. Fig.3(a) demonstrates a typical integration form [5], [6] of dynamic pruning with ResNet [12] as the backbone static network.

In the residual block (ResBlock) in the backbone network, spatial mask generator is plugged in to calculate a binary mask over the spatial dimension of OF, which decides the pixels to be evaluated. This spatial mask is designed to prune the redundant background information. Channel mask generator is plugged in to calculate a binary mask over the channel dimension of OF to decide channels to be calculated. The channel mask is used to select the channels that contribute most to the final results.

The spatial mask M_s and channel mask M_c are calculated from the input feature (IF), i.e., $M_s = f_s(\text{Conv}_{1\times1}(IF))$, $M_c = f_s(\text{FC}(\text{AvgPool}(IF)))$. Step function f_s is non-differentiable and can not be directly trained. To make mask generators trainable, Gumbel-Softmax [13] is commonly adopted to replace the step function during training process [5], [6].

III. Pipeline-Enabled Dynamic Network for PIM

For network structure in Fig.3(a), channel mask generator takes all pixels in IF as input, which will stall the inter-layer pipeline in PIM architectures. In contrast, spatial mask generator takes only one pixel in the IF as input at a time, which does not face the same challenge. So, our network design

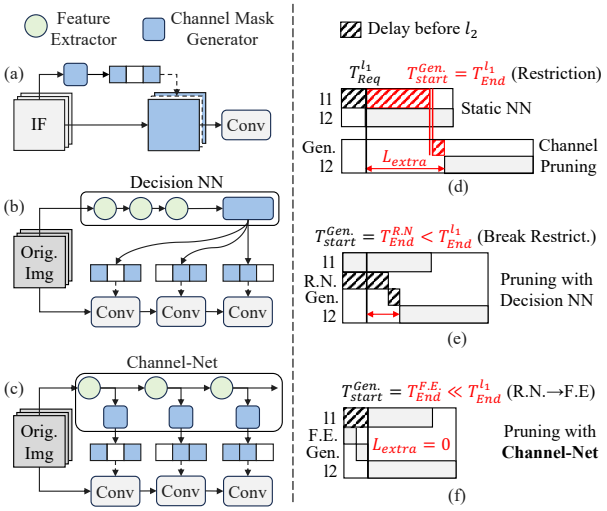

Fig. 3. Left: Different Structures for Channel Mask Generation; Right: An Example of Pipeline Stall

is mainly focused on how to enable the pipeline from the stall of channel mask generation.

A. Problem Analysis: An Example

Firstly, we take a static network with two convolutional layers l_1 and l_2 as the example. In PIM architectures, inter-layer pipeline allows a layer to start computation once the required input data from the previous layer are ready [9]: $T^{l_2}_{start} = T^{l_1}_{req}(Static)$, where $T^{l_2}_{start}$ stands for the start time of l_2, and $T^{l_1}_{req}$ stands for the time when the required input data from the l_1 layer are ready, as shown in Fig.3(d).

Next, dynamic pruning is introduced in the dimension of channels. Assuming that channel masks are generated from l_1, and applied to l_2. Then l_2 should also wait for the channel mask from generator to start computation:

$$T^{l_2}_{start} = \max(T^{Gen}_{end}, T^{l_1}_{req}),\ T^{Gen}_{end} = T^{Gen}_{start} + L_{Gen} \quad (1)$$

where T^{Gen}_{start}, T^{Gen}_{end}, L_{Gen} are the start time, end time, and computing latency of the channel mask generator. And the generator takes the entire output feature of l_1 as input to produce the channel mask. Thus, $T^{Gen}_{start} = T^{l_1}_{end}$, where $T^{l_1}_{end}$ is the time when l_1 finishes computation.

Based on the former analysis, the generator is restricted to start computing after l_1 finishes computing. This restriction brings extra latency (L_{extra}) to the start time of l_2 almost as much as the total computation time of l_1, which is the main cause of pipeline stall, as shown in Fig.3(d).

$$\begin{aligned} T^{l_2}_{start} &= T^{Gen}_{end} = T^{l_1}_{end} + L_{Gen} >> T^{l_1}_{req}(Static) \\ L_{extra} &= T^{l_2}_{start} - T^{l_1}_{req} = (T_{end} - T_{req})^{l_1} + L_{Gen} \end{aligned} \quad (2)$$

Facing this problem, simply reducing the latency of the generator is not an effective approach, as the restriction is left unchanged:

$$L_{extra} > (T_{end} - T_{req})^{l_1} + 0 \quad (3)$$

DPACS [6] shares channel masks across several ResBlocks, reducing the mask generation overhead. However, since sharing channel masks across too much ResBlocks will lead to accuracy drop, there still exist multiple ResBlocks that need to generate channel masks. As a result, pipeline stall can not be avoided.

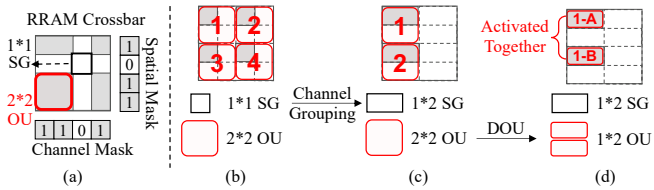

Fig. 4. Left: Illustration of Granularity Mismatch; Right: Example of Granularity Alignment

B. Our Approach: Channel-Net

In order to break the restriction of $T^{Gen}_{start} = T^{l_1}_{end}$ in all ResBlocks of the backbone network without compromising accuracy, we replace the plug-in generator with a standalone decision network [14], as shown in Fig.3(b). The key idea is to generate the channel mask using the original input image, rather than the intermediate OF of l_1. To be specific, a ResNet-20 model is used to generate channel mask using the original image. As ResNet-20 can be computed faster than l_1, the generator can start computing earlier than l_1 finishes computing: $T^{Gen}_{start} = T^{R.N.}_{end} < T^{l_1}_{end}$, where $T^{R.N.}_{end}$ represents the finish time of the ResNet-20. Therefore, the mask generation and l_2 can start earlier than $T^{l_1}_{end}$. However, the naive standalone decision network generates the channel mask at the cost of additional processing latency of ResNet-20, which is as much over 25% of the total latency.

To tackle this problem, we propose Channel-Net that introduces branch structures to the decision network, as is shown in Fig3(c). In Channel-Net, the original image is processed by a series of lightweight feature extractors in sequence. Generators are inserted between feature extractors as branches. Channel masks for the backbone ResBlocks at different depths are computed at different branches step by step. In this way, mask for the first backbone ResBlock can be computed right after the first feature extractor of the decision network, which is much faster than waiting for the entire decision network to be completed: $T^{Gen}_{start} = T^{F.E.}_{end} << T^{R.N.}_{end} < T^{l_1}_{end}$, where $T^{F.E.}_{end}$ is the finish time of the first feature extractor. By leveraging Channel-Net, Equation 2 can be rewritten as Equation 4.

$$L_{extra} = max(0, (T^{F.E.}_{end} + L_{Gen}) - T^{l_1}_{req}) = 0 \quad (4)$$

With Channel-Net, if the mask generation is finished before the required input data of l_2 are ready, l_2 can start computing and the inter-layer pipeline between l_1 and l_2 is recovered, as shown in Fig.3(f). Experimental results show the extra delay of Channel-Net is only 5.16% of the total latency. Therefore, the extra latency of channel mask generation can be reduced by 57.17×, and the inter-layer pipeline is recovered.

IV. Granularity Alignment Strategy

A. Problem Analysis

In fine-grained dynamic algorithms like dynamic pruning, computation reduction comes from the sparsity in the spatial and channel dimension of the pruned feature map:

$$Comp.\ Reduct. = ((1 - SR_s) * (1 - SR_c))^{-1} \quad (5)$$

where SR_s and SR_c are the sparsity ratio (i.e., percentage of 0s) of the spatial and channel mask. Assuming that values in the spatial/channel mask are independent and identically distributed (i.i.d.) random variables. Then the probability of a value being

0 in the spatial/channel mask, i.e., $P_s(0)/P_c(0)$, is equal to the spatial/channel sparsity ratio, i.e., SR_s/SR_c.

From the hardware perspective, after mapping network parameters onto RRAM crossbars, the computation skip of dynamic network means deactivating some memory cells. For example, if a pixel of IF is masked, computations of weights in a wordline can be skipped. And if a channel of OF is masked, computations of weights in a bitline can be skipped. At the algorithm level, the minimum granularity of sparsity (SG) is to mask one pixel of IF and one channel of OF. As a result, the SG corresponds to a single memory cell in the crossbar, as shown in Fig.4(a). Denoting the height and width of SG as SG_H and SG_W, respectively. Then we have $SG_H = SG_W = 1$. While for the hardware, when conducting MVM in a crossbar array, dot-product computations on weight are conducted at the granularity of OU, which contains multiple rows and columns of memory cell, as shown in Fig.4(a). Because of the granularity mismatch between SG and OU, one OU usually contains multiple SGs. The SG number within an OU can be calculated as: $\#M_s = \left\lceil \frac{OU_H}{SG_H} \right\rceil, \#M_c = \left\lceil \frac{OU_W}{SG_W} \right\rceil$, where OU_H and OU_W are the height and width of OU. $\#M_s$ and $\#M_c$ are the number of spatial & channel mask minimum granularity within an OU.

An OU can be skipped if and only if all the weights in it are masked. In other words, if all spatial mask values on an OU are zero, or all channel mask values on an OU are zero, then it can be skipped. As a result, the probability that an OU can be skipped, i.e., $P_{OU}(Skip)$, is as follows:

$$\begin{aligned} P_{OU}(Skip) &= P(\sum M_s = 0 \bigcup \sum M_c = 0) \\ &= 1 - P(\sum M_s > 0)) * P(\sum M_c > 0) \\ &= 1 - (1 - (SR_s)^{\#M_s}) * (1 - (SR_c)^{\#M_c}) \end{aligned} \tag{6}$$

Since OUs are activated and computed sequentially, the PIM hardware speedup comes from the number of skipped OUs:

$$\begin{aligned} Speedup &= \frac{\#OU_{all}}{\#OU_{all} - \#OU_{skip}} = \frac{1}{1 - P_{OU}(Skip)} \\ &= ((1 - (SR_s)^{\#M_s}) * (1 - (SR_c)^{\#M_c}))^{-1} \end{aligned} \tag{7}$$

where $\#OU_{all}$ is the total number of OUs to be computed in one layer, and $\#OU_{skip}$ is the number of skipped OUs.

To maximize the PIM hardware speedup in Equation 7, $\#M_c$ and $\#M_s$ should be as small as possible, i.e., approach to one. However, the mismatch between the size of SG and OU leads to high $\#M_c$ and $\#M_s$, causing severe speedup degradation, as shown in Fig.4(b).

B. Granularity Alignment: Channel Grouping and Dynamic OU Formation

In order to improve the PIM performance, we align the size of sparsity granularity to the size of OU from two dimensions. As a result, $\#M_c$ and $\#M_s$ are reduced to one and the computation reduction is fully converted to the hardware speedup.

To align the widths of OU and SG, we propose the channel grouping method from the algorithm level. Every N channels are merged into a channel group, which share the same mask value. As a result, SG_W is raised and leads to smaller $\#M_c$. In this work, N is aligned to the width of OUs, thus $\#M_c$ can be reduced to 1, as shown in Fig.4(c).

To align the heights of OU and SG, it is not easy to raise SG_H from the algorithm level. As spatial mask changes for each sliding window, SG_H will varies at different position, which is hard to control. Alternatively, we reduce OU_H through a hardware-friendly dynamic OU formation (DOU) strategy from the hardware level. DOU split an OU into several sub-OUs that can be activated together in one cycle. It is realized by dynamically choosing the wordlines with non-zero masks to form OUs, instead of statically grouping contiguous wordlines as OUs. As the height of this new sub-OU is one, $\#M_s$ is reduced to one, as shown in Fig.4(d). More hardware details will be discussed in Sec.VI-D.

V. Throughput-Optimal Training Technique

In pipeline-enabled PIM architectures, the throughput depends on the layer with the longest latency. As a result, to achieve optimal throughput with the same computation reduction, latency of different layers should be balanced.

In traditional architectures like CPU and GPU, the processing latency is proportional to the amount of computation. So existing training techniques of dynamic networks [5], [6] are designed to achieve balanced computation across layers while reducing the overall computation. But in PIM architectures, the compute capability differs across layers as mentioned before. Consequently, these training techniques can not ensure balanced latency even if the computation is balanced.

To achieve balanced latency on PIM architectures, we use the longest latency after pruning as the proxy of hardware performance, instead of the overall computation reduction. Specifically, we first inference on the backbone network and record the latency Lat_i of each layer i. Then we compute the lowest target sparsity ratio for different layers in order to achieve the throughput improvement target $T.I._{tar}$.

$$SR^i_{tar} = \max(0, 1 - \frac{\max(Lat_j)}{Lat_i * T.I._{tar}}) \tag{8}$$

During training, gradients are backpropagated to the weight in the layers that vary most from its sparsity target, through the design of loss function:

$$L_{Perf} = (\text{ReLU}(\max(SR^i_{tar} - SR^i))^2 \tag{9}$$

where SR^i is the sparsity ratio of the i^{th} layer.

Furthermore, we also restrict on the upper bound of sparsity to prevent unnecessary computation reduction, so as to achieve optimal accuracy.

$$\begin{aligned} L_{Acc} &= \beta * (\text{ReLU}(SR^{\arg\max_i(SR^i)}_{tar} - \max(SR^i)))^2 \\ Loss &= \alpha * \text{CrossEntropy}(y_p, y) + L_{Perf} + L_{Acc} \end{aligned} \tag{10}$$

where y_p and y are the prediction and the groundturth. α is the trade-off coefficient between accuracy and hardware performance. β represents the strictness of the upper bound on the sparsity ratio for optimal accuracy. With the loss function, our network is then trained with Gumbel-Softmax [13].

VI. Hardware architecture

A. Overall Architecture

The overall DyPIM accelerator architecture is shown in Fig.5. It can support the layer-wise, spatial-wise and channel-wise dynamic inference with the designed control modules in the accelerator architecture.

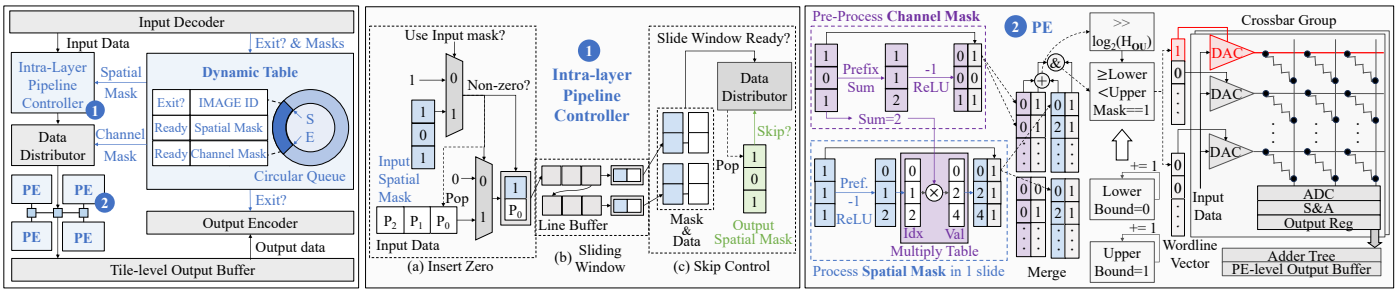

Fig. 5. Left: Tile Architecture with Layer-wise Control Flow; Middle: Intra-layer Pipeline Controller; Left: PE design

B. Tile Architecture with Layer-wise Control Flow

In DyPIM, decision results encompass spatial masks, channel masks, and exit decision. Exit decisions indicate whether the output data proceed to the subsequent stage or is directed to the classifier for early-exit. To efficiently utilize the decision results in the tile-level structure, a dynamic table is designed, as illustrated in the left part of Fig.5. Decision results are detected by the input encoder, and then buffered in the dynamic table. It is implemented with a circular queue, where each entry comprises the ID and decision results of the input image.

C. Intra-layer Pipeline Controller

If a pixel in OF is skipped, the entire MVM will be skipped. And the pixel will not be transmitted to the subsequent layer, leading to sparse inter-layer dataflow. To handle the sparse input flow and skip MVM for masked pixels, an intra-layer pipeline controller is designed, as shown in the middle part of Fig. 5. Firstly, sparse input dataflow is recovered by inserting zeros with the assistance of input spatial mask. Then line-buffer [9] is utilized to perform sliding window on the recovered inputs. Spatial mask values are also integrated with corresponding inputs in the line buffer to provide mask values for each sliding window. Finally, if computation of a pixel in OF is skipped, then the input data won't be transferred to PE for computation.

D. PE Architecture with Dynamic OU Formation

In the original implementation of DOU [11], prefix sum is asserted on the input mask M_w to get the orders (f) of non-zero values, which is calculated as:

$$f(M)[x] = \text{ReLU}(\mathcal{P}(M)[x] - 1) \tag{11}$$

where $\mathcal{P}(M)$ stands for the prefix sum of M.

Non-zero wordlines are grouped and activated in order to form OUs. In cycle Clk, the j^{th} word line is activated to form the OU to be computed if it satisfies:

$$M_w[j] \ \& \ (Clk-1) * OU_H \leq f(M_w)[j] < Clk * OU_H \tag{12}$$

But for each sliding window, prefix sum operation is asserted on M_w, which has the same length as the crossbar size. Based on the algorithm [15] used in [11], if the crossbar is sized $N \times N$, $(N-1)$ adders are needed to compute prefix sum within $\lceil log_2 N \rceil$ cycles, resulting in bad scalability of area and latency.

As input channel mask M_c and input spatial mask M_s are related to the input mask M_w, it is possible to conduct shorter prefix sum on channel and spatial masks separately:

$$f(M_w)[c * K^2 + s] = f(M_c)[c] + f(M_s)[s] * \sum M_c \tag{13}$$

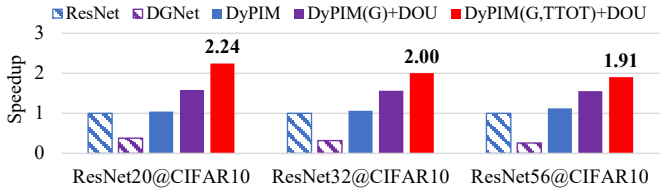

Fig. 6. Hardware Speedup under 50% Computation Budget on CIFAR-10 dataset ((G denotes channel grouping technique, $TOTT$ denotes throughput-optimal training technique, DOU represents dynamic OU formation))

where K is the kernel size.

As shown in the right part of Fig. 5, the channel mask is pre-processed for each input image to get $f(M_c)$ and the multiply table of $\sum M_c$. For each sliding window, spatial masks are processed to get $f(M_s)$, which requires (K^2-1) adders for $\lceil log_2 K^2 \rceil$ latency of prefix sum, which is constant under crossbars with different size. To be specific, We prepare $f(M_c)$ and the multiply table of $\sum M_c$ for each input image. And for each sliding window, we only need to calculate $f(M_s)$. For a typical $3*3$ kernel, only 8 adders are needed to compute prefix sum in 4 cycle. Then $f(M_s)[s] * \sum M_c$ is directly read from multiply table to get $f(M_w)$.

VII. Evaluation

A. Experiment Setup

We evaluate DyPIM using ResNet-20 [12], ResNet-32, and ResNet-56 on the CIFAR-10 and CIFAR-100 datasets [16]. The hardware performance is evaluated using MNSIM 2.0 [9]. Dynamic control flows are added to enable the simulation of dynamic inference algorithms. For hardware configuration, we adopt a crossbar size of 128×128 and an OU size of 16×8, in line with an existing RRAM accelerator [17].

B. Hardware Speedup Results

Under various computation budgets, DyPIM can achieve $1.52\times$ to $2.74\times$ speedups on ResNet models. Fig. 6 illustrates the speedups under 50% computation budgets. The results indicate that all techniques can achieve speedup improvement. DyPIM's full implementation can achieve $1.91\times$ to $2.24\times$ speedups, which basically aligns with the computation budgets. But speedups degrade in deeper models, where inter-layer delays are more dominant and limit the speedups.

C. Throughput Improvement

Under various computation budgets, DyPIM can achieve $2.05\times$ to $3.95\times$ throughput improvement on ResNet models. Fig. 7 illustrates the throughput improvement of various

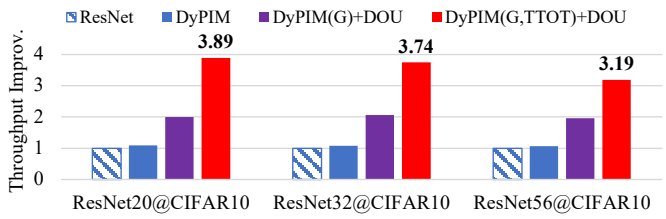

Fig. 7. Throughput Improvement under 50% Computation Budget on CIFAR-10 dataset (G denotes channel grouping technique, $TOTT$ denotes throughput-optimal training technique, DOU represents dynamic OU formation)

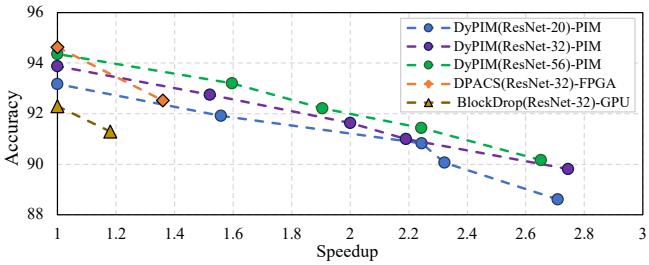

Fig. 8. Trade-offs between Accuracy and Speedup on CIFAR10 dataset

models under 50% computation budgets. Remarkably, DyPIM, when equipped with the throughput-optimal training technique, achieves up to 3.89× throughput improvement, which is nearly twice the computation reduction. However, ResNet-56 achieves a non-optimal improvement in throughput, as the increasing model depth makes it difficult to accurately control the density of each layer.

D. Trade-offs between Accuracy and Speedup

Fig. 8 presents the trade-offs between accuracy and speedup under different computation budgets on CIFAR-10 dataset. On ResNet-32, DyPIM can achieve 1.52× speedup with only 1.13% accuracy loss. It remarkably outperforms DPACS [6] and BlockDrop [18] in the trade-off between accuracy and speedup. Moreover, Fig. 9 showcases the trade-offs between accuracy and speedup on the CIFAR-100 dataset.

E. Overhead Analysis

Under the hardware settings, employing Channel-Net instead of inserted decision modules for channel-wise dynamic inference introduce approximately 6.8% more area and 5.5% more energy consumption. Also, our design of the hardware architecture only introduces 0.51% more area to the tile architecture.

VIII. Summary and Conclusions

This article proposed DyPIM, a software-hardware co-design system combining a PIM-friendly dynamic inference algorithm along with a dynamic-inference-enabled PIM accelerator, which is the first PIM architecture for dynamic inference algorithms. We used Channel-Net to avoid the pipeline stall of decision modules, along with a channel-grouping and a dynamic OU formation strategy to align the granularity of algorithm and OU. We also proposed a throughput-optimal training theme to achieve optimal throughput under the same computation reduction. Experiments show that DyPIM can achieve 1.52× to 2.74× speedup and 2.05× to 3.95× throughput improvement over over the existing PIM architectures for ResNet networks. In the future, we will research on the combination of layer-wise dynamic inference with spatial-wise and channel-wise dynamic pruning to improve the performance of DyPIM on deeper networks, as well as more stable training technique for better sparsity control.

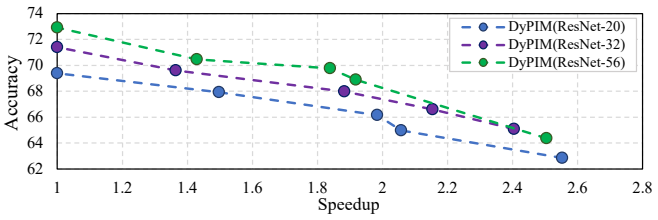

Fig. 9. Trade-offs between Accuracy and Speedup on CIFAR100 dataset

Acknowledgements

This work was supported by the National Key R&D Program of China (2023YFB4502200), the National Natural Science Foundation of China (No. 62104128, U19B2019, 61832007, U21B2031, 62204164), 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] J. Deng, W. Dong, R. Socher et al., "Imagenet: A large-scale hierarchical image database," in CVPR, 2009.
[2] A. Vaswani, N. Shazeer, N. Parmar et al., "Attention is all you need," NeurIPS, 2017.
[3] Y. Han, G. Huang, S. Song et al., "Dynamic neural networks: A survey," TPAMI, 2021.
[4] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in ICPR, 2016.
[5] F. Li, G. Li, X. He et al., "Dynamic dual gating neural networks," in ICCV, 2021.
[6] Y. Gao, B. Zhang, X. Qi et al., "Dpacs: Hardware accelerated dynamic neural network pruning through algorithm-architecture co-design," in ASPLOS, 2023.
[7] P. Chi, S. Li, C. Xu et al., "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," ISCA, 2016.
[8] Z. Zhu, H. Sun, Y. Lin et al., "A configurable multi-precision cnn computing framework based on single bit rram," in DAC, 2019.
[9] Z. Zhu, H. Sun, T. Xie et al., "Mnsim 2.0: A behavior-level modeling tool for processing-in-memory architectures," TCAD, 2023.
[10] H. Sun, T. Xie, Z. Zhu et al., "Minimizing communication conflicts in network-on-chip based processing-in-memory architecture," in DATE, 2023.
[11] T.-H. Yang, H.-Y. Cheng, C.-L. Yang et al., "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in ISCA, 2019.
[12] K. He, X. Zhang, S. Ren et al., "Deep residual learning for image recognition," in CVPR, 2016.
[13] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," arXiv preprint arXiv:1611.01144, 2016.
[14] Z. Chen, Y. Li, S. Bengio et al., "You look twice: Gaternet for dynamic filter selection in cnns," in CVPR, 2019.
[15] W. D. Hillis and G. L. Steele Jr, "Data parallel algorithms," Communications of the ACM, 1986.
[16] A. Krizhevsky, G. Hinton et al., "Learning multiple layers of features from tiny images," 2009.
[17] W.-H. Chen, K.-X. Li, W.-Y. Lin 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.
[18] Z. Wu, T. Nagarajan, A. Kumar et al., "Blockdrop: Dynamic inference paths in residual networks," in CVPR, 2018.