OverlaPIM: Overlap Optimization for Processing In-Memory Neural Network Acceleration

Minxuan Zhou[§]
University of California, San Diego
La Jolla, USA
miz087@ucsd.edu

Xuan Wang[§]
University of California, San Diego
La Jolla, USA
xuw009@ucsd.edu

Tajana Rosing
University of California, San Diego
La Jolla, USA
tajana@ucsd.edu

***Abstract*—Processing in-memory (PIM) can accelerate neural networks (NNs) for its extensive parallelism and data movement minimization. The performance of NN acceleration on PIM heavily depends on software-to-hardware mapping, which indicates the order and distribution of operations across the hardware resources. Previous works optimize the mapping problem by exploring the design space of per-layer and cross-layer data layout, achieving speedup over manually designed mappings. However, previous works do not consider computation overlapping across consecutive layers. By overlapping computation, we can process a layer before its preceding layer fully completes, decreasing the execution latency of the whole network. The mapping optimization without overlap analysis can result in sub-optimal performance. In this work, we propose OverlaPIM, a new framework that integrates the overlap analysis with the DNN mapping optimization on PIM architectures. OverlaPIM adopts several techniques to enable efficient overlap analysis and optimization for the whole network mapping on PIM architectures. We test OverlaPIM on popular DNN networks and compare the results to non-overlap optimization. Our experiments show that OverlaPIM can efficiently produce mappings that are 2.10× to 4.11× faster than the state-of-the-art mapping optimization framework.**

I. Introduction

Processing in-memory (PIM) has emerged as a promising computing solution to boost the performance of many critical applications, like deep neural networks. Compared to conventional architectures, PIM provides extensive parallelism without off-chip data movements, leading into higher throughput, energy efficiency, and scalability. PIM can exploit the abundant memory resources to realize high compute and memory throughput at the same time. The large capacity of memory brings other benefits for workloads which we can allocate exclusive memory resources for different parts of the application, significantly reducing the data loading overhead [7], [14], [15]. Neural network inference is a good candidate that can exploit these benefits where we can allocate exclusive memory for different layers. Therefore, most previous large-scale PIM accelerators adopt the spatially distributed acceleration for neural network inference [7], [14].

The performance of DNN acceleration depends on not only the hardware architecture of the accelerator but also the application mapping which determines the execution order and distribution of DNN operations on the hardware. Figure 1(a) shows an example of two mappings for a 2D convolution

[§]Equal contribution

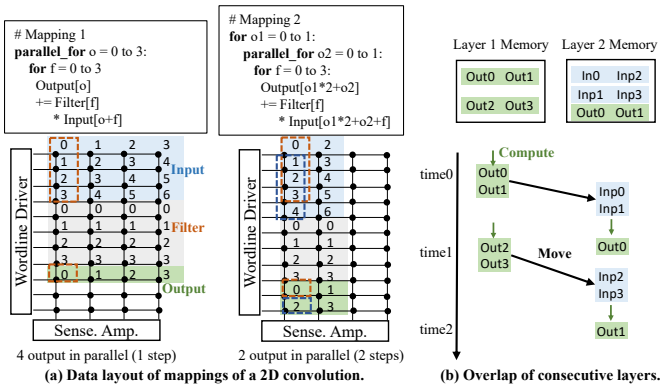

Fig. 1. The mapping problem of DNN on PIM accelerator.

on a PIM-enabled memory block. This example uses the in-memory bit-serial row-parallel processing as the PIM technology [3], [4] which provides state-of-the-art performance due to the extremely parallel in-memory computations. We describe each mapping using the syntax of Timeloop [10]. Mapping 1 parallelizes computations of all 4 outputs by spreading data in different memory columns; mapping 2 parallelizes two output tiles where each tile sequentially processes 2 outputs. As shown, different mappings vary in latency, data layout, and the order of producing outputs. Considering the more complex operations (e.g., 3D convolution can be represented as a 7-level nested loop) used in DNNs, the design space of mapping DNNs onto the hardware accelerator is extremely large.

DNN mapping is critical to the performance of hardware accelerators so that existing accelerators optimize the mapping based on hardware configurations (e.g., row-stationary mapping of Eyeriss [2]). Recent works recognize large design space for DNN mapping by proposing optimization frameworks to find the optimized mapping through either exhaustive search [10], [12], [13] or solving an optimization problem [6]. However, the ASIC-based framework, which assumes the accelerator processes one layer at a time, is not applicable to spatially distributed acceleration on PIM architecture. PIM-based framework [13] considers the scheduling of all DNN layers on PIM architecture by integrating a global optimization with the per-layer optimization. Although such a PIM-based framework takes the whole-network mapping into account for optimization, it misses the computation overlap enabled by spatially distributed DNN acceleration, which is a critical optimization.

978-3-9819263-7-8/DATE23/© 2023 EDAA

Specifically, the existing PIM-based framework assumes DNN layers are executed sequentially in the spatially distributed PIM memory, where each layer should wait for the full completion of its preceding layer(s) to start the execution. In practice, the execution of a layer can start earlier when part of its input has been computed from the preceding layer(s). In other words, we can overlap the execution of different layers to improve the performance. Figure 1(b) shows an example of executing two consecutive layers in two memory blocks. We divide the computation of each layer into multiple time steps, where each time step processes several operation spaces in parallel (spatially processed in different memory blocks). In the example, Layer 2 can overlap the computation of its output 0 with Layer 1 at the time step 1, where Layer 1 completes the computation of data space output0/1 (input0/1 in Layer 2). In real DNNs, such overlapping can bring significant performance benefits. When considering the overlapping, the existing PIM-based framework, which only considers the end-to-end sequential latency, may generate the sub-optimal mapping.

In this work, we propose and implement a novel PIM-based DNN mapping framework, OverlaPIM, that integrates overlap analysis into mapping optimization. There exist several challenges in implementing overlap optimization. First, we can only generate the overlap information based on fine-grained data analysis which compares data spaces between two layers. However, the size of fine-grained data spaces can be extremely large so previous mapping frameworks avoid the full analysis for all operation spaces. To tackle this challenge, we propose a lightweight algorithm that can efficiently generate fine-grained data spaces for overlap analysis. The second challenge is the search for mapping becomes slow with overlap analysis. In order to speed up the search for good mappings, we propose a transformation mechanism that transforms a searched mapping into overlap-friendly mappings with a trivial overhead for analysis. In this case, we effectively increase the search space of the framework in a similar amount of time. We implement the proposed framework in an open-source DNN mapping framework and compare the result against state-of-the-art mapping optimization without the consideration of overlap. Our evaluation of popular DNNs shows OverlaPIM can produce mappings that are 2.10× to 4.11× faster than the mappings optimized by existing methods [10], [13].

II. Background and Motivation

A. DNN Mapping and Optimization

DNN mapping determines the operation scheduling and data allocation of a specific DNN for a computing platform. For example, Timeloop [10] parameterizes the 7-D loop and near-exhaustively searches through possible mappings considering loop decomposition (i.e., temporal tiling and spatial tiling) and permutation. Each mapping has a direct relation to a specific data allocation and operation schedule on given hardware, which has hierarchical storage. The mapping optimization for customized accelerators only optimizes a DNN layer because such accelerators process one layer at a time. The per-layer optimization may lead to sub-optimal performance for PIM architectures with sufficient resources to process many (even all) layers simultaneously. The cross-layer optimization is solved in a recent work [13] by applying a 2-level optimization algorithm to find the efficient data layout of all layers in the memory.

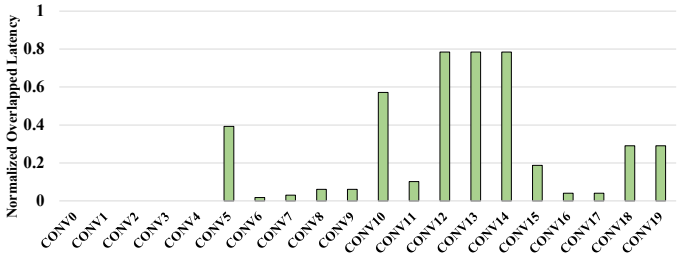

Fig. 2. The normalized overlapped latency for all layers in ResNet-18 optimized by existing framework [10] - higher means better overall performance.

B. PIM Acceleration for DNN

In this work, we focus on the bit-serial row-parallel PIM architectures based on DRAM which exploit special memory commands (e.g., activate-activate-precharge [4]) to implement universal bit-wise operations (e.g., majority-based addition [1]). Figure 1 shows the architecture for a PIM-based memory block supporting bit-serial row-parallel in-memory computations. The PIM block contains an array of memory cells and peripheral that controls bit-lines (rows) and word-lines (columns). We allocate memory rows to different bits of operand and result vectors to support an in-memory computation. The example only shows single-bit values for input, filter, and output. In practice, each data may take multiple rows. Once aligning operand and result vectors, the memory issues a sequence of universal bit-wise operations to generate the result vector. Such bit-serial operations achieve extensive parallelism because we can simultaneously process all columns in memory rows in different memory blocks.

Furthermore, considering the large memory size, the PIM-based accelerators can accommodate the whole DNN network, different from ASIC-based accelerators that usually process one layer at a time. The whole network execution avoids costly off-chip data communication. PIM DNN accelerators [7], [14] achieve state-of-the-art performance due to these benefits.

C. Overlap-based Optimization for PIM DNN Acceleration

Although previous work [13] proposed an optimization framework that can generate efficient mappings for the whole DNN in a PIM architecture, it still misses a key feature for the PIM DNN acceleration - computation overlapping. In practice, the memory of the preceding layer cannot generate all its outputs at the same time, making outputs ready at different times. In this case, the following layer can consume some inputs (outputs from the preceding layer) earlier than others, overlapping the corresponding computations with the preceding layer. Such overlapping can bring better performance than the performance considered by the existing framework.

Figure 2 shows an experiment of PIM acceleration for ResNet-18 where we use the state-of-the-art framework, Timeloop [10], to search for the best mapping (the lowest

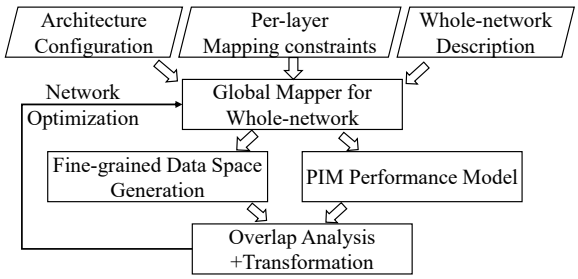

Fig. 3. The overview of OverlaPIM.

latency) layer by layer. We modified Timeloop to analyze the computation overlap of consecutive layers and reduce the overlapped computations from the original latency. We reduce the latency only if the input for all operation spaces of the following layer becomes ready in a specific time step. Therefore, not all overlapped data spaces lead to a latency reduction. We calculate the normalized latency of the overlapped computation in the experiment, where a higher value indicates a better performance with more overlapped computations. As shown in the results, the overlapped latency varies significantly from layer to layer if we naively search for the best non-overlap performance mapping for each layer. Specifically, 13 out of 20 layers only have trivial overlapping ($< 20\%$) while other layers can overlap a significant portion of their computations (29% to 78%). Therefore, it is great potential to optimize the performance by optimizing the DNN mapping on PIM based on overlaps of computations between consecutive layers.

III. Overlap-based Mapping Optimization

In this work, we propose OverlaPIM, a PIM mapping framework for DNN with the consideration of overlapping between different layers. We implement our framework in an open-source mapping framework, Timeloop [10], which uses near-exhaustive search to find efficient mappings. Figure 3 shows key components of OverlaPIM to enable the new functionalities related to overlap-based optimization. First, we add the interface to support the whole network mapping including the combined description of mapping constraints and layer information for the whole network. The new interface enables us to configure the whole network in the optimization for the overlap analysis between consecutive layers. Second, we add a new PIM performance model to enable the accurate evaluation of mapping on PIM architectures because the original Timeloop [10] performance model only considers the compute, read, and write latency. These are insufficient for PIM performance evaluation which requires data movement. Third, we implement a fine-grained data space generation that produces detailed data spaces over time on different memory components for overlap analysis and optimization. Furthermore, we propose a transformation algorithm that can significantly increase the search capability of whole DNN optimization with trivial overhead.

During the execution, the optimization mapper generates mappings based on the configuration, including architecture configuration and mapping constraints. For each mapping, it generates the fine-grained temporal and spatial data spaces, as well as the PIM performance evaluation. Then, the framework calculates the overlap of consecutive layers based on their fine-grained data spaces and recalculates the performance considering the overlapped computations. The framework also transforms the current mapping into overlap-friendly mappings to increase the search space. The framework continues to update the best mapping based on the overlap-based performance until meeting the termination requirements (similar to Timeloop [10]). The following subsections illustrate details for each component of OverlaPIM.

A. PIM Performance Model

In this work, we focus on spatially distributed PIM acceleration for the whole DNN based on bit-serial row-parallel processing [3], as introduced in Section II-B. We allocate a fixed amount of memory (e.g., 2 channels) for each DNN layer, for which we place the filter data as well as the input data for the first layer in the memory based on the DNN mapping. After each layer execution, we move its output to the memory locations of the input for the next layer.

We cannot simply change the latency of operation in the Timeloop's original performance model, which only evaluates the number of read/write operations, to evaluate PIM architectures. Read/write operations of PIM acceleration are replaced by the data movements, including the output-input data transfer and the data movements for reductions of partial sums located in different columns. Thus, we implement a new performance evaluation model in Timeloop to support PIM architectures.

B. Fine-grained Data Space Generation

Analysis of the output/input overlapping between layers requires the comparison of the detailed output/input data spaces for each memory hierarchy level across DNN layers. However, only a small portion of data spaces are collected for size measurement in Timeloop [10] due to runtime limitation. Since Timeloop [10] generate data spaces from recursive function calls, collecting all data spaces is unacceptably expensive for both runtime efficiency and memory consumption and makes overlap analysis impossible. To reduce the complexity, we propose a lightweight algorithm to infer all spatial and temporal data spaces through analytical observation and formulation.

First, OverlaPIM exploits the same 7D-loop representation of Timeloop for DNN layer. We use convolution as the example, where R and S are the height and width of weight, P and Q are the height and weight of output, C is the number of input channels, K is the number of output channels, and the number of inputs or batch size is represented by N. With this parameterized representation, we define the output data space as a 4-D tensor $[N, K, P, Q]$ and the input data space as $[N, C, P+R-1, Q+S-1]$ for further input/output overlapping analysis on different mappings. Figure 4 shows an example of data spaces for a mapping on a two-level memory. For simplicity, we ignore the dimension of N in later discussion.

Each mapping, with a specific loop decomposition and permutation, can be translated into data spaces that are spatially and temporarily distributed. Specifically, the spatial distribution (i.e., $parallel\_for$) means data spaces are split and allocated

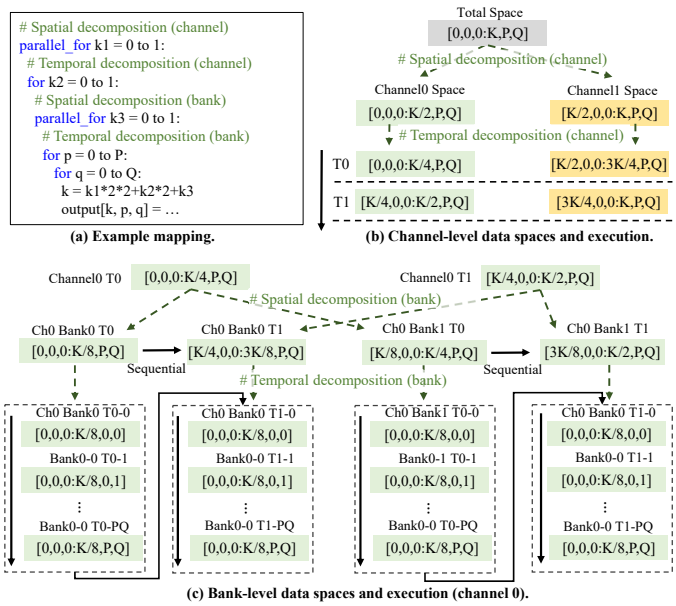

Fig. 4. Fine-grained Data Space Generation

to different hardware instances (e.g., banks). The temporal distribution (i.e., for) happens when a data space of a hardware instance is further decomposed into smaller data spaces. These smaller data spaces are sequentially processed by the corresponding hardware component in multiple temporal steps, where the instance process a temporal data space in each step. As shown in Figure 4, the whole output data space is spatially decomposed into two channel-level spaces, each of which is further decomposed into two temporal steps. Similarly, each channel-level temporal step is decomposed into 2 bank-level spaces, where each consists of $P * Q$ temporal steps. In total, each bank in this example consists of $2 * P * Q$ temporal steps.

We propose a lightweight data space analysis based on the observation that the size of data spaces maintains the same at each hardware level. The dimension value of data spaces changes periodically corresponding to the inner loop iteration, where each loop level increments one data space dimension. Therefore, our analysis algorithm runs in two steps. First, we analyze the nested loop of the mapping to split the whole data space (input or output) into small data spaces until the target level (e.g., bank-level). The loop analysis runs in an up-down loop order to generate small data spaces by splitting data spaces from the upper loop level. Second, we map the generated data spaces into the correct temporal and spatial locations. The spatial index for each data space can be computed straightforwardly by tracking all spatial loops (i.e., $parallel\_for$). For the temporal index, we deduce a formula to translate the indices of loop iterations into the temporal steps at each hardware level. Assuming k is the index of the n^{th} temporal loop iteration and i is the global index of 0-(n-1) temporal loops, the temporal index of a data space in the n^{th} loop can be found by:

$$S_k^i(n) = S_{num_n}^{i-1}(n) + (\prod_{j=q}^{n-1} num_j) * k$$

where num_j is the number of iterations in the j^{th} temporal loop, and q is the lower bound of loops in the target hardware level where the current loop iteration belongs.

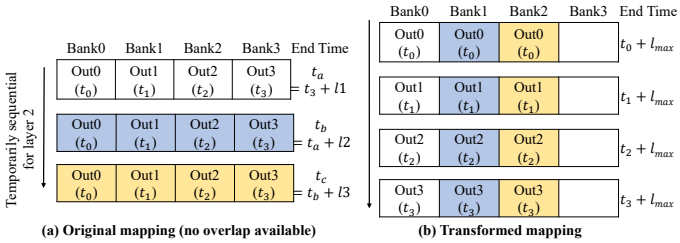

Fig. 5. Overlap-based transformation.

As compared to Timeloop's original logic, which depends on recursive function calls, our method can more efficiently compute all data spaces in $O(n)$ time complexity, where n is the total number of data spaces. If we implement the data space generation in the Timeloop's recursive function calls, the analysis for one mapping takes around 600 seconds. Our proposed analytical calculation only takes less than 60 seconds.

C. Overlap-based Performance Analysis

With the fine-grained data spaces for two consecutive layers, Layer n and $n+1$, we analyze the overlap and estimate the overlapped performance. We denote O_t^i and I_t^i as the whole output and input operation spaces in t^{th} temporal step of all hardware instances for Layer i. We first find the ready time of I_t^{n+1}, which is the time when all data in I_t^{n+1} are finished by the previous layer (Layer n). For each $I_{t_i}^{n+1}$, we need to check all O_t^n and find the latest time step t_o that $O_{t_o}^n$ has an overlap with $I_{t_i}^{n+1}$. If t_o is earlier than the end time of Layer n, we can compute $O_{t_i}^{n+1}$ right after t_o because the whole $I_{t_i}^{n+1}$ has been calculated. In this case, the computation of $O_{t_i}^{n+1}$ is overlapped with computations in Layer n after t_o.

Our new evaluation considers the overlapped data spaces (computations) based on the hardware constraints to calculate the overlapped performance as the new metric for optimization. In our framework, we traverse through all data spaces at a target storage level to find the ready timestamps. In the PIM architecture, we conduct the overlap analysis at the bank level because the analysis at an upper (e.g., channel) produces too coarse-grained data spaces while a lower level (e.g., column) has too many instances that will make the analysis intractable.

D. Overlap-based Mapping Transformation

One way to search for the best mapping, considering the overlap, is to simply add overlap analysis in the normal search process (e.g., Timeloop's search algorithms). However, the search time of this method can be extremely long. The most time-consuming part of the search process is overlap analysis which generates evaluation statistics. Based on our experiments, the overlap analysis increases the evaluation time for a mapping by 2.06× - 41.49 × as compared to the original Timeloop, depending on the number of data spaces. Therefore, it is critical to find a new method to search for more mappings in each overlap analysis. If we can generate the analysis statistics for different mappings in trivial time, we can significantly enlarge the search space, hence improving the search result.

We propose an overlap-based mapping transformation that can directly generate the evaluation results for different mappings based on the mapping that is analyzed in detail. Figure 5 shows an example of mapping transformation, where we mark the ready time for the input of each data space (t_x in each parenthesis). The left part shows the original mapping, where no overlap is available because the latest ready time of all data spaces in a time step is the end time of the previous layer (i.e., t_3). For transformation, we reorganize the data spaces and reschedule data spaces at the same ready time, as shown in the right part, significantly decreasing the end time of the layer.

The transformation runs in two steps. First, it sorts the data spaces in the ascending order of ready time of input. After the sorting, the algorithm allocates the memory resources for each data space based on the ready time. In practice, the number of data spaces that have the same ready time can be larger than the number of memory resources. Therefore, the memory allocation algorithm needs an optimization process. We should note that the transformation is not overhead-free because it might change the locations of partial sums that require data movements for reduction. Therefore, the algorithm uses a round-robin manner to schedule the data space with the same partial sum to the same memory location. Since the transformation does not require re-analysis of the data space and the complexity of the algorithm is $O(logN)$ bounded by the search, the transformation only introduces trivial overhead during the search process.

E. Overlap Optimization for the Whole DNN

OverlaPIM supports the mapping search for all layers in the entire DNN model. To evaluate the whole network, we require descriptions of all layers and their corresponding mapping constraints and architecture configurations as inputs. These inputs can be generated automatically through our self-designed toolkit by providing information of architecture design and constraints. Since DNNs are Directed Acyclic Graphs (DAG) of various layers, the workload parameters for each layer will be auto-generated according to the topological order for analysis.

Given that overlapping performance depends on both Layer n and Layer $n+1$, finding the best performance mappings for all layers through searching and comparing mappings would be prohibitively expensive. For example, if we search for k mappings in each layer, the total possible combination of mappings for all N layers would be k^N. The exhaustive search for optimal mappings would be unacceptably expensive in this case. Thus, for each Layer $n+1$, the overlapping optimization search is done based on the best mapping found for Layer n. Our evaluation shows that such a linear method can produce high-performance mappings. We leave a more in-depth investigation of global optimization for future work.

IV. Experiments

A. Experimental Setup

1) Implementation: We implement the proposed framework in Timeloop [10] as illustrated in Section III.

2) Baseline: We compare OverlaPIM to several baselines based on state-of-the-art PIM mapping optimization [13], implemented in Timeloop [10]. Specifically, the "Best Original" indicates the mapping optimized by the original framework and does not consider overlap; "Best Original Overlap" means the same mapping as the "Best Original" but the performance considers the overlapped computation, which is analyzed by OverlaPIM; "Best Overlap" is the mapping optimized based on the execution time considering overlap in the search process (no transformation); "Best Transform" indicates the optimized mapping generated by considering the transformation during the overlap-based optimization. The searches explore the same number of valid mappings for all methods in the main loop. Note that "Best Transform" effectively checks more mappings with the light-weight transformations for each valid mapping.

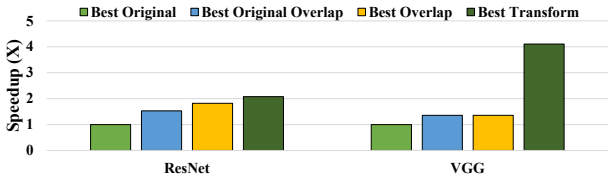

Fig. 6. Overall performance comparison over different algorithms.

3) Architecture Configuration: We use the HBM [8] with the support of majority-based bit-serial computation [1] as the base technology for the PIM architecture. We allocate the fixed number of HBM channels (8 banks/channel, 32MB bank) for each layer, ranging from 1 channel to 4 channels. The whole system has 4 8GB HBM2 stack, with a total of 128 channels. We assume all 4 HBM2 stacks are connected through a host machine with a 256GB/s bus. We extract the timing of HBM from previous work [9] including the detailed latency for memory commands (e.g., activate, precharge, etc.) and internal/external bandwidth of HBM.

4) Workloads and Mapping Constraints: We evaluate the efficiency of OverlaPIM on two popular DNN networks, ResNet-18 [5] and VGG-16 [11]. Because certain groups of mapping may be preferred for a specific architecture configuration, we carefully construct mapping constraints (natively supported by the original Timeloop) for different layers to reduce the overall search time. We note that OverlaPIM is general to all workloads and architectures supported by the original Timeloop [10].

B. Overall Comparison

Figure 6 shows the overall results of different mapping optimization algorithms. For ResNet-18, the overlapped latency of the original best mapping (Best Original Overlap) is $1.51\times$ faster than the end-to-end latency without overlapping (Best Original). The mapping optimized by the overlapped latency (Best Overlap) can provide another $1.22\times$ speedup over "Best Original Overlap". If we adopt the transformation in the search process, the best mapping (Best Transform) further improves the performance of "Best Overlap" by $1.14\times$. OverlaPIM behaves differently in VGG-16 where "Best Original Overlap" and "Best Overlap" have similar performance ($1.36\times$ faster than "Best Original"). However, the overlap-based optimization with transformation produces significantly better mappings than other algorithms, which is $4.11\times$ faster than "Best Original".

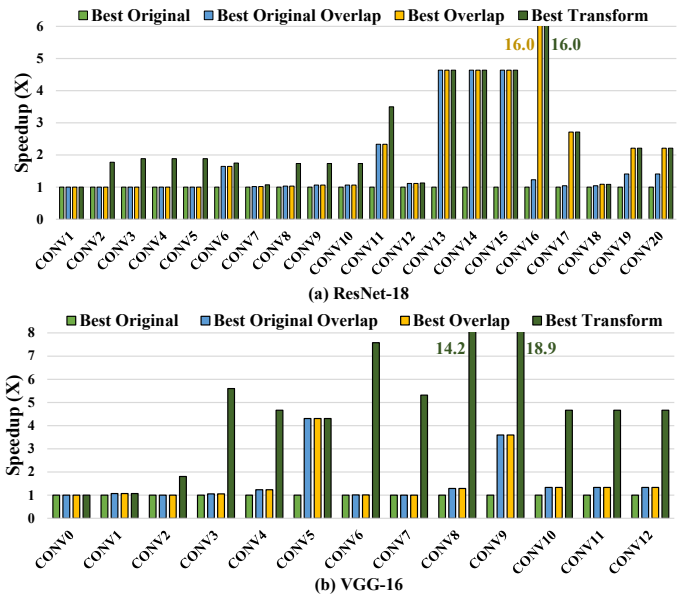

Fig. 7. The per-layer performance comparison on ResNet-18 and VGG-16. All results are normalized to "Best Original" for each layer.

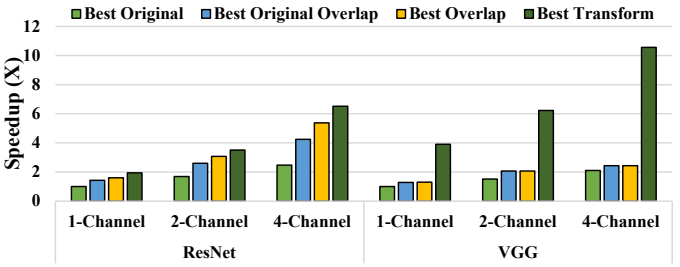

Fig. 8. Efficiency of OverlaPIM on different memory capacity. All results are normalized to the "Best Original" time for the 1-channel setting.

C. Per-layer Breakdown

Figure 7(a) shows the per-layer performance comparison over different mapping optimization algorithms on ResNet-18. As shown in the figure, the overlap-based algorithms, "Best Overlap" and "Best Transform", find better mappings than the existing method (Best Overlap) for 15 out of 20 layers. Among these layers, "Best Transform" is 2.30× better than "Best Original Overlap". The proposed optimization achieves small performance improvements when the existing mapping algorithm (without overlap consideration) coincidentally produces mappings with a high overlap ratio, as shown in Figure 2. Figure 7(b) is the per-layer performance comparison on VGG-16, showing a more significant benefit of transformation, which improves the performance for 10 out of 13 layers. As a comparison, the normal overlap-based optimization (Best Overlap) is not faster than "Best Original Overlap" in any layer.

D. Sensitivity Analysis of Memory Capacity

We adopt OverlaPIM on various architecture settings where we allocate different amounts of memory resources for each layer. We compare the performance of different optimization algorithms in various architecture settings, as shown in Figure 8. The results show that, in ResNet-18, "Best Transform" is 2.63×, 1.54×, and 1.21× faster than "Best Original", "Best Original Overlap", and "Best Overlap" for 4-channel setting (2× larger than the default 2-channel). Such performance improvements are slightly better than the results shown in Figure 6. The performance improvements on the 1-channel setting are similar to the 2-channel setting, which achieves 1.94×, 1.36×, and 1.21× speedup over three baseline algorithms. We observe a similar result in VGG-16 where the relative performance of the algorithm is similar across different settings. Such results prove that OverlaPIM is a scalable and general optimization for DNN mapping on PIM architectures.

V. Conclusion

This work proposes a novel DNN mapping framework, OverlaPIM, on PIM architectures that consider the computation overlapping. OverlaPIM integrates the overlap-based analysis in the DNN mapping optimization that searches for the best DNN mapping based on the overlapped latency, instead of sequential latency considered by existing methods. We propose several techniques to improve the effectiveness and efficiency of OverlaPIM, including a fine-grained operation space generation, an overlap-based performance analysis, and a transformation algorithm to quickly find overlap-friendly mappings. OverlaPIM can find mappings that achieve 2.10× to 4.11× better performance than mappings optimized by existing methods.

Acknowledgment

This work was funded by CRISP, one of six centers in JUMP (an SRC program sponsored by DARPA), SRC Global Research Collaboration (GRC) grant, and NSF grants #2112167, #2003279, #2100237, #2112665, and #2120019.

References

[1] M. F. Ali *et al.*, "In-memory low-cost bit-serial addition using commodity dram technology," *IEEE TCAS I: Regular Papers*, 2019.
[2] Y.-H. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, 2016.
[3] F. Gao *et al.*, "Computedram: In-memory compute using off-the-shelf drams," in *IEEE/ACM MICRO*, 2019.
[4] N. Hajinazar *et al.*, "Simdram: A framework for bit-serial simd processing using dram," in *ACM ASPLOS*, 2021.
[5] K. He *et al.*, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.
[6] Q. Huang *et al.*, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *ACM/IEEE ISCA*, 2021.
[7] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ACM/IEEE ISCA*, 2019.
[8] C. Oh *et al.*, "22.1 a 1.1v 16gb 640gb/s hbm2e dram with a data-bus window-extension technique and a synergetic on-die ecc scheme," in *IEEE ISSCC*, 2020.
[9] M. O'Connor *et al.*, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *IEEE/ACM MICRO*, 2017.
[10] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *IEEE ISPASS*, 2019.
[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[12] X. Yang *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *ACM ASPLOS*, 2020.
[13] M. Zhou *et al.*, "Pim-dl: Boosting dnn inference on digital processing in-memory architectures via data layout optimizations," in *PACT*, 2021.
[14] M. Zhou, W. Xu *et al.*, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *IEEE HPCA*, 2022.
[15] M. Zhou *et al.*, "Gram: Graph processing in a reram-based computational memory," in *ACM ASPDAC*, 2019.