Processing-in-Memory Using Optically-Addressed Phase Change Memory

Guowei Yang[*], Cansu Demirkiran[*], Zeynep Ece Kizilates[*], Carlos A. Ríos Ocampo[†], Ayse K. Coskun[*], Ajay Joshi[*]
[*]Boston University, [†]University of Maryland, College Park
{guoweiy, cansu, zecek, acoskun, joshi}@bu.edu, riosc@umd.edu

***Abstract*—Today's Deep Neural Network (DNN) inference systems contain hundreds of billions of parameters, resulting in significant latency and energy overheads during inference due to frequent data transfers between compute and memory units. Processing-in-Memory (PiM) has emerged as a viable solution to tackle this problem by avoiding the expensive data movement. PiM approaches based on electrical devices suffer from throughput and energy efficiency issues. In contrast, Optically-addressed Phase Change Memory (OPCM) operates with light and achieves much higher throughput and energy efficiency compared to its electrical counterparts.**

This paper introduces a system-level design that takes the OPCM programming overhead into consideration, and identifies that the programming cost dominates the DNN inference on OPCM-based PiM architectures. We explore the design space of this system and identify the most energy-efficient OPCM array size and batch size. We propose a novel thresholding and reordering technique on the weight blocks to further reduce the programming overhead. Combining these optimizations, our approach achieves up to 65.2× higher throughput than existing photonic accelerators for practical DNN workloads.

***Index Terms*—optical computing, phase change memory, processing-in-memory, deep neural networks**

I. Introduction

Deep Neural Networks (DNNs) are commonly used today for a variety of tasks such as image classification [1]–[3] and natural language processing [4]. The size of the DNNs has grown over the years, and current state-of-the-art DNNs contain hundreds of billions of parameters [5]. Moving DNN parameters as well as the DNN activation data between memory and compute causes large time and energy overheads. Moreover, the gap between computation and memory speed is continuously increasing, exacerbating the problem.

Processing-in-Memory (PiM) has emerged as a viable approach to mitigate this data movement cost. Existing PiM architectures incorporate various types of devices: DRAM [6], [7], ReRAM [8], [9], Spin-Transfer Torque RAM (STT-RAM) [10], and Phase Change Memory (PCM) [11]–[15]. DRAM-based PiM designs [6], [7] perform computation across the DRAM cells, thus, avoiding expensive data transfers between DRAM and compute. However, DRAM requires frequent refreshing to maintain the stored data, increasing power consumption. Both ReRAM [8], [9] and STT-RAM [10] are non-volatile memory. They do not consume power to maintain data storage, and offer higher throughput and consume lower power than DRAM-based devices. However, ReRAM suffers from process variation challenges and endurance issues, and STT-RAM has low storage density.

This research was partially funded by the NSF CCF 2131127 grant.

PCM-based PiM consumes lower power and provides higher throughput compared to PiM based on other technologies [12]. PCM is also a non-volatile memory that does not consume power to retain the stored data. Moreover, the multi-level capability of a PCM cell achieves higher throughput compared to a 1 bit/cell storage in DRAM; e.g., a PCM cell can store up to 6 bits/cell [16] and perform analog computation on multi-bit data [12]. We can access and perform computations in PCM cells using electrical or optical signals. Electrically-addressed PCMs (EPCMs) [11] offer higher storage density but have lower throughput [17] than Optically-addressed PCMs (OPCMs). OPCM-based PiM systems achieve high compute density (up to 162 TOPS/mm^2 [12]), making OPCM a promising solution for next-generation computing systems.

Recent works [12]–[15] have explored the idea of PiM using OPCM. These works, however, focus on small OPCM arrays such as 4×4 [13]. Moreover, these works evaluate their designs using small DNNs (such as four 2×2 kernels [12]), which can easily fit in small OPCM arrays. So, in those works, one needs to perform the expensive OPCM programming step only once, and so they do not consider programming cost (programming latency and energy are $2-3$ and $4-5$ orders of magnitude higher than compute latency and energy, respectively) in DNN inference. In contrast, state-of-the-art DNN models contain hundreds of billions of parameters and, therefore, cannot fit in a single OPCM array. In a practical scenario, we need to periodically program the OPCM array, and the cost of this reprogramming should be considered in the overall inference cost. In fact, the reprogramming latency and energy can easily dominate the total latency and energy and become bottlenecks. Therefore, to make OPCM-based PiM practical, we need to reduce OPCM's programming cost.

To this end, in this paper we make the following contributions:

- We provide a full system-level design of an OPCM-based PiM architecture that explicitly accounts for programming of the OPCM cells for performing DNN inference. To the best of our knowledge, we are the first to identify that the programming overhead dominates the DNN inference time and energy on OPCM, and the first to present a solution for this issue.
- We investigate the impact of OPCM array size and batch size on latency and energy efficiency in OPCM-based PiM and identify the optimal OPCM array size and inference batch size that achieve the highest energy efficiency in DNN inference, considering the programming cost.

- We present a novel thresholding and reordering technique to reduce the OPCM programming overhead further. Our method applies thresholding to limit the number of OPCM cells that we should reprogram. It also reorders the programming of the matrix blocks to the OPCM array in a way that minimizes the number of OPCM cells we need to reprogram.

We evaluate the proposed OPCM-based PiM system using practical DNN workloads, including VGG-11 [1], AlexNet [2], ResNet-50 [3], and BERT-Large [4]. Our solution achieves $4918\times$ higher IPS and 41.3% worse IPS/W than Eyeriss v2 [18], and $4.5\times$ higher IPS and $1.2\times$ higher IPS/W than TPU v3 [19]. Compared to photonic accelerators ADEPT [20] and Albireo-C [21], our solution has lower IPS/W. However, our system still achieves $2.3\times$ and $65.2\times$ higher IPS than ADEPT and Albireo-C, respectively.

II. OPCM Background

The basic building block of the OPCM design is an optical waveguide with embedded $Ge_2Sb_2Te_5$ (GST), as shown in Figure 1. The ratio between the amorphous and crystalline areas of the GST cell determines the device transmittance, which is encoded into multiple bits. Both the amorphous and the crystalline states are non-volatile. Therefore, the data is stored permanently, allowing its use as an optical multi-bit memory [22]. Experimental demonstrations have achieved up to 64 deterministic states in a GST cell, which allows storing up to 6 bits/cell [16]. Given these properties and developments, GST cells have been used as a platform for optical in-memory computing [23] by realizing scalar-scalar multiplications that map a multiplicand to the GST transmittance (i.e., stored in memory) and the multiplier to the amplitude of an input pulse.

To change the phase state of the GST cell and, thus, to modulate the transmission of the waveguide, electrical or optical pulses create the transition-triggering heat stimuli. Electrical switching uses a variety of waveguide-embedded microheaters to electro-thermally induce the reversible phase transitions. Researchers have demonstrated switching energies as low as 5.55 nJ and 860.71 nJ to amorphize and crystallize, respectively, using graphene microheaters [24]. Optical switching, on the other hand, uses the optical absorption of GST to trigger the amorphization or crystallization using pulses inside the waveguide. This method consumes less energy at the GST cell; 460 pJ for amorphization and 140 pJ for crystallization [22]. However, in a large architecture, the optical pulses must be routed through a switching network, which imposes the need for energy-hungry phase-shifters. Alternatively, dedicated couplers can be used to reach each GST cell, an optical solution that requires a large footprint and complicated experimental setups [12]. Even though electrical programming consumes more energy per switching event, the scalability, compatibility with microelectronics, and form factor make it a more suitable solution for setting the optical transmission of the GST array. Therefore, for optimized DNN inference operations, we assume electrical programming for efficiently writing the GST array weights and optical addressing for high-throughput General Matrix Multiply (GEMM) operations [13], [24].

III. Related Work

Moving data between memory and compute units causes significant overhead, and PiM is a potential solution to this problem.

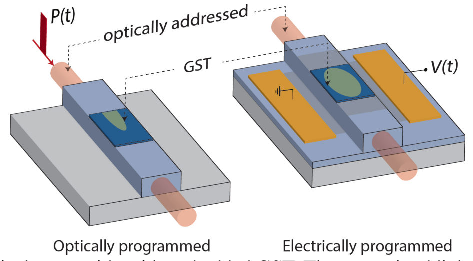

Fig. 1. Optical waveguide with embedded GST. The transmitted light is correlated to the phase state of GST, which can be controlled optically using pump pulses or electrically using fast electro-thermal heating in waveguide-integrated microheaters.

By performing computation inside the memory cells, PiM reduces the data movement overhead and improves the throughput and energy efficiency. PiM architectures based on DRAMs [6], [7] are promising, but they suffer from high latency due to charging/discharging of the capacitance [6], which also results in higher energy consumption. The latency of a single operation in DRAM-based PiM architectures can take almost 28 ns and requires at least 3 nJ of energy because of the activation, read/write, and precharge processes [7]. Furthermore, the technology scaling for DRAM is slowing down. Considering the increase in the amount of data to be processed, this scalability issue becomes more critical [25].

ReRAM-based PiM architectures are being actively explored, and their nonvolatile nature makes ReRAMs a favorable candidate for implementing PiM designs [8], [9]. STT-RAM [10] is also a non-volatile memory that shows great potential to accelerate DNN inference. Both ReRAM and STT-RAM achieve higher throughput and energy efficiency compared to DRAM-based PiM system. However, ReRAM suffers from process variation challenges and endurance issues, and STT-RAM has a low storage density. They are also fundamentally limited by the energy-throughput tradeoff.

PCM is another type of device that has gained attention as a suitable candidate for PiM. EPCM devices [11] are accessed with electrical signals, while OPCM devices are accessed with optical signals. When programming the PCM cells, existing designs use either electrical switching [15], or optical switching [12], [14].

In contrast to DRAM, both OPCM and EPCM are fully passive and do not need power to retain data. Their multi-level capability provides higher throughput, and their analog computing approach consumes less power than digital computing [15]. We argue that OPCM is a better choice to accelerate DNN inference. Compared to EPCM whose frequency is limited by energy, OPCM can operate at high frequency (up to 25 GHz) to provide extremely high throughput. Thus, in this paper, we focus on OPCM-based PiM architectures.

IV. OPCM-Based PiM Design and Operation

In this section, we first describe the architecture of our proposed OPCM-based PiM system and the dataflow for mapping DNN inference to that system. Then, we discuss the microarchitecture of the OPCM array and how it performs GEMM operations. Finally, we present the thresholding and reordering technique for the DNN weights to reduce the PCM programming overhead during inference.

A. Full System Architecture and Dataflow

1) System Architecture: Figure 2 shows our proposed 2.5D-integrated OPCM-based PiM system that consists of the host processor chiplet, DRAM chiplet, OPCM chiplet containing multiple crossbar arrays for GEMM operations, the CMOS chiplet for electrical-optical and optical-electrical (E-O-E) conversions and non-GEMM operations (referred to as "E-O-E + non-GEMM" chiplet henceforth), and the laser source chiplet. The non-GEMM operations include inter-block accumulation, non-linear activation functions, pooling, and batch normalization. In addition, the E-O-E + non-GEMM chiplet contains SRAM arrays for storing inter-block accumulation outputs and buffering the data received from/sent to the DRAM. The host processor chiplet is connected to the "E-O-E + non-GEMM" chiplet using electrical links embedded in the interposer. The DRAM chiplet and the laser source chiplet are connected to the "E-O-E + non-GEMM" chiplet using optical links embedded in the interposer. The laser source is also connected to the DRAM chiplet using optical links embedded in the interposer. Prior to performing the DNN inference operations, it is more efficient to program the weights in the OPCM array electrically than optically; however, during DNN inference, it is more efficient to perform the GEMM operations with OPCM than EPCM, which uses optical links [13], [24]. So the "E-O-E + non-GEMM" chiplet connects to the OPCM array through both electrical and optical links embedded inside the interposer.

2) Dataflow of DNN Inference: The host processor chiplet executes the DNN-based application. When executing the DNN inference part of the application, the processor uses Application Programming Interfaces (APIs) to transfer the control to the control logic in the "E-O-E + non-GEMM" chiplet. Given that an OPCM cell requires an area of $30 \times 30\,\mu\text{m}^2$ [12], and DNNs contain hundreds of billions of parameters, it is not practical to build an OPCM array that is large enough to fit an entire DNN at one time. For example, VGG-11 has 133 million parameters and requires an OPCM array with an area of 239,400 mm^2, which is impractical. To perform DNN inference on the OPCM chiplet, we process the DNN layer by layer. For each layer, we apply the standard blocking technique [12], which breaks down the large matrix into smaller blocks and processes one block at a time. We have multiple OPCM crossbar arrays in the OPCM chiplet that run in parallel. For every OPCM array, the control logic first reads a weight block of a layer from DRAM using optical links, saves it in the SRAM buffer in the "E-O-E + non-GEMM" chiplet, and programs it into the OPCM array using electrical links. The control logic then reads the inputs to this block from DRAM and feeds them to the OPCM array via optical links. Note that the inputs get converted from the electrical domain to the optical domain in the DRAM and are then routed directly to the OPCM array. The OPCM array performs GEMM operation in the optical domain. The output of the OPCM array is routed to the "E-O-E + non-GEMM" chiplet, where the data is converted into the electrical domain and fed to the digital logic performing the non-GEMM operations, including inter-block accumulation and non-linear operations. When the OPCM array is performing operations on one block, the control logic loads the next block to be processed into the SRAM buffer in a pipelined fashion. The above operations are repeated for all blocks in every layer. Once the operations for a layer finish, the results are transferred back to DRAM, and these results serve as the input for the following layer. After all the layers of the DNN are executed, the inference result is written back into the DRAM and is accessible to the host processor. The traditional communication between the host processor chiplet and the DRAM chiplet is through the "E-O-E + non-GEMM" chiplet.

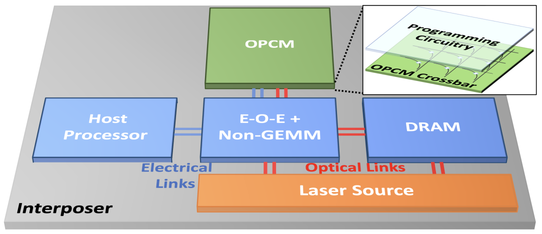

Fig. 2. Architecture of an OPCM-based computing system.

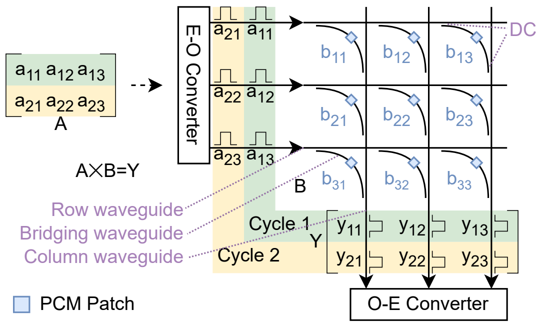

Fig. 3. GEMM example of 2×3 input matrix A multiplied with a 3×3 weight matrix B on OPCM array.

B. OPCM Array Architecture

The OPCM chiplet consists of multiple OPCM crossbar arrays for GEMM computation and the programming circuitry for setting the state of each OPCM cell. The OPCM crossbar array (see Figure 3) is based on the Photonic Tensor Core [12], which consists of row waveguides, column waveguides, bridging waveguides, and Directional Couplers (DCs). We deposit the phase change material (e.g., GST) on top of every bridging waveguide. The DCs connect the horizontal and vertical waveguides through the bridging waveguide. The split ratio of the DC is carefully designed such that the horizontal DCs split the light from a row evenly into every column, and the vertical DCs combine the attenuated light received from every row in a single column [12].

C. GEMM operations in OPCM

1) GEMM Operation: We follow the standard scheme to perform GEMM operations in OPCM [12]. Consider two matrices, matrix A that is $P \times M$ and matrix B that is $M \times N$. To multiply A and B, assume we have an OPCM array of M rows and N columns. We map matrix B to that OPCM array. To perform the matrix-matrix multiplication, we split the matrix A into P $1 \times M$ vectors and then perform P matrix-vector multiplications (MVMs) in the OPCM array. Below we describe the matrix multiplication with a concrete example.

To understand the matrix-matrix multiplication process, assume A is a 2×3 matrix and B is a 3×3 matrix. So $M = 3$, $N = 3$, $P = 2$ (see Figure 3). Assume we have a 3×3 OPCM array. Elements of the matrix B are electrically programmed into the PCM cells of the OPCM array. Then three elements of the first row of matrix A (i.e., a_{11}, a_{12}, and a_{13}) are converted into the optical domain (each element is mapped to a unique wavelength), and then these three optical signals are routed into the three rows of the OPCM array. Each optical signal is split equally across the three columns and routed into the bridging waveguides. As the optical signal passes through the bridging waveguide, it is attenuated by the PCM material on the bridging waveguide and then coupled into the column waveguide. This attenuation process performs the multiplication operations. At the output of the first column, we get three products in three different wavelengths–$a_{11} \times b_{11}$, $a_{12} \times b_{21}$ and $a_{13} \times b_{31}$. These three products are accumulated using a photodetector, i.e., $y_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$ to give us the first element of the first row of the product matrix. Similarly, the second column and the third column of the OPCM array give us the second element and third element of the first row of the product matrix. Then we similarly send the second row of matrix A through the OPCM array to get the second row of the product matrix.

2) Handling negative weights and inputs: DNN weights are usually signed; however, the intensity of light waves and the loss in PCM cells are always non-negative. To support negative weights, we need to decompose the weight matrix B into positive and negative component matrices B^p and B^n, where $b_{ij}^p = \max(0, b_{ij})$ and $b_{ij}^n = \max(0, -b_{ij})$. The final result of matrix multiplication then becomes $A \times B = A \times B^p - A \times B^n$ [14], where the subtraction happens in the electrical domain. With two 6-bit PCM cells [16] representing the positive and negative components, we effectively achieve 7-bit quantization. To support negative inputs, we need to offset the input and include an extra column of precomputed weight references [13].

3) Mapping DNN to OPCM: DNNs include a variety of layers, including convolution and fully-connected layers. The operations in these layers can be mapped to GEMM operations. We use a weight-stationary approach for performing these GEMM operations. The blocking approach requires us to program the weights within a block onto the OPCM array, perform multiplication using that block, then repeat this process for another block. Unfortunately, for large DNNs, the cumulative latency of programming all the blocks into the OPCM array is on the order of seconds, and cumulative energy is on the order of joules. This is because we need to re-program the OPCM array frequently due to its small capacity compared to large DNN weights. For example, a 64×64 array is programmed at least $30{,}000$ times during one inference of the four example DNNs. The programming time and energy are $2-3$ orders of magnitude and $4-5$ orders of magnitude larger than the time and energy for performing a single inference (more details in Section V-B, see Figure 4). Effectively, this reprogramming overhead cancels OPCM's performance and energy advantages for DNN inference. Therefore, we need to reduce the programming cost of OPCM.

D. OPCM Programming Cost Reduction

1) Choosing Batch Size and Array Size: One way to reduce the programming cost is to use large batch sizes during inference. After we program a block of a matrix into the OPCM array, the same block can operate on all the inputs in a batch. This way, we can amortize the time and energy overhead of programming the OPCM array across the large batch. However, the intermediate memory required to perform the accumulations between blocks increases proportionally with batch size, which poses an upper bound for the batch size.

For example, Figure 5 (a) shows the IPS/W for VGG-11 with various batch sizes and array sizes. It is clear that in terms of performance per unit power, array size $=64$ yields the highest IPS/W, and the larger the batch size, the better the IPS/W.

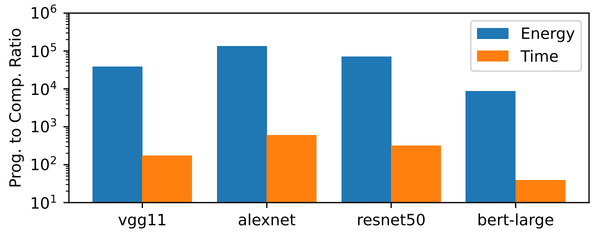

Fig. 4. Ratio of programming cost to computation cost for one inference, with 16 arrays each of size 64×64. The time and energy spent on programming the OPCM cells are $2-3$ and $4-5$ orders of magnitude higher than that on computation only, showing that programming cost dominates the inference performance and energy.

Figure 5 (b) shows the SRAM required for inter-block accumulation for different batch sizes and OPCM array size combinations. SRAM requirement increases with batch size and OPCM array size. For an array size $=64$, the SRAM requirement reaches 392 MB at batch size $=4{,}096$, taking up about 600 mm^2 on GF22FDX technology node. Further increasing the batch size would increase the IPS/W, but the SRAM area will become prohibitively large. Therefore, we need to choose the batch size and array size for each DNN to maximize energy efficiency under an area constraint.

2) Block reordering to minimize state changes: We propose a block reordering technique to reduce the energy required for programming the OPCM array. This technique exploits the observation that there exists a similarity between the elements across blocks. Essentially, during inference, when programming a new block into the OPCM array, not all OPCM cells need to be programmed. For example, suppose the difference between the value of an element at a specific position in the new block and the value of an element at that same position in the current block is less than a certain threshold, we don't need to program the OPCM cell corresponding to that element, i.e., we do not need to overwrite it. We can reuse the old value as DNNs are tolerant to minor weight variations.

Moreover, when performing DNN inference, the blocks within a layer can be mapped to the OPCM array in arbitrary order without degrading inference accuracy. This provides a chance to further reduce the programming cost as we can follow a block processing order that minimizes the programming latency and energy for a given layer. Figure 6 shows a toy example where the matrix is divided into four blocks. We construct an undirected graph where each vertex represents a block. Suppose block 1 is currently in the OPCM array, and we now program block 2 into that array; the cost of this programming, i.e., the number of cells to be overwritten, is denoted on the edge between block 1 and block 2. There are 24 possible orders in which the four blocks can be programmed into the OPCM array. Out of the 24 possible orders, the $3 \rightarrow 1 \rightarrow 4 \rightarrow 2$ order has the least latency and energy. We can use this order to minimize the programming cost. Note that one needs to determine the order of using blocks only once, and that can be done offline. The overhead of the control logic for processing the blocks in non-sequential order is minimal.

In today's DNNs, for a typical 64×64 OPCM array, we have up to $50{,}000$ blocks in a layer. The number of possible block processing orders is up to $50{,}000! = 10^{200{,}000}$. Performing an exhaustive search to determine the order of processing the blocks is impractical, even if the process is done offline. In fact, finding the minimum cost of traversing all the vertices in a graph is a well-studied problem called

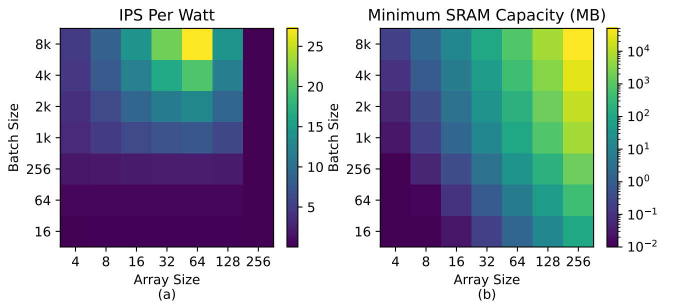

Fig. 5. (a) IPS/W and (b) Minimum SRAM capacity (for inter-block accumulation) for VGG-11. Plots for AlexNet, ResNet-50 and BERT-Large look similar.

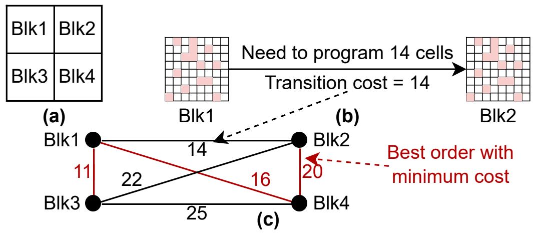

Fig. 6. Reordering the weight blocks to reduce programming cost. (a) Split the weight matrix into blocks of the same size as the OPCM array. (b) Compute the transition cost between each pair of blocks. (c) Construct an undirected graph whose vertices are weight blocks and edges are the transition cost between the two blocks. Solve Traveling Salesman Problem to find the order that minimizes cost.

Traveling Salesman Problem (TSP) [26], and there are various tools to solve it. We use Google OR-Tools [27], a widely used library for combinatorial optimization, to solve our TSP problem.

V. Evaluation

A. Methodology

In this section, we evaluate the OPCM-based PiM in terms of its performance and energy efficiency for performing DNN inference. We limit the total number of OPCM crossbar cells to 256×256, which is divided into multiple OPCM arrays. The OPCM array uses electrical signals to program the PCM cells and optical signals to perform GEMM operation. During GEMM operation, the power consumption is primarily in the laser source and the E-O-E conversion unit. The required laser power depends on the optical losses experienced by the optical signals as it passes through the OPCM array. We assume the losses of the GST cell, waveguide crossing, and DC are 0.6 dB, 0.0028 dB, and 0.01 dB, respectively, and the combined quantum efficiency of the laser and photodetector is assumed to be 10% [12]. The OPCM array is a passive device with passive microring resonators (MRRs) [28] and does not consume any thermal tuning power. The OPCM arrays perform MVM operations at 25 GHz. We assume the programming energy of each cell to be the average of amorphizing (5.55 nJ) and crystallizing (860.71 nJ), and it takes 400 ns to program the entire array [24]. Each crossbar cell occupies an area of 30×30 μm^2 [12], and the diameter of MRRs is 10 μm.

For the "E-O-E + non-GEMM" chiplet, we synthesize the SRAM and electrical non-linear units in GF22FDX technology node to get the power and area estimations. The ADC (which also performs the O-E conversion) power and E-O power are 194 mW/channel and 1 pJ/bit, respectively, at 25 GHz [12]. DRAM accesses are assumed to be 20 pJ/bit [29].

We choose the following four popular DNNs as workloads: VGG-11 [1], AlexNet [2], ResNet-50 [3], and BERT-Large [4]. The first three DNNs run image classification on the Imagenet dataset, and BERT-Large runs a question-answering task on the SQuAD 1.1 dataset. All models are pre-trained with FP32 and quantized to 7-bit precision (combining two 6-bit cells, as discussed in Section IV-C2). We use Google OR-Tools [27] to solve the reordering problem, apply thresholding to weights with in-house scripts, and then use PyTorch to test the inference accuracy after thresholding and reordering.

B. Results

1) OPCM Programming Cost: We first focus on the OPCM array's programming overhead during a DNN inference. We perform inference using four DNNs with the OPCM array size of 64×64 and observe the time and energy spent on programming versus computation. As shown in Figure 4, the time and energy spent on programming the OPCM cells for one inference operation are $2-3$ orders of magnitude and $4-5$ orders of magnitude higher than those for computing in the OPCM cells. This is because the capacity of the OPCM array is at least $30{,}000\times$ smaller than the weights in the four DNNs, so we must reprogram it frequently.

2) Choosing Batch Size and Array Size: Using an extremely large batch size is a straightforward way to amortize away the programming cost per inference. From Figure 5 (a), it is evident that array size $= 64$ yields the best energy efficiency, while the larger the batch size, the better. However, the batch size is limited by the SRAM capacity. Figure 5 (b) shows the minimum SRAM requirement. As discussed earlier in Section IV-D1, with array size $= 64$ and batch size $= 4{,}096$, the minimum SRAM required reaches 392 MB, which is a feasible design. We perform similar analyses for all DNNs evaluated and find that array size $= 64$ and batch size $= 4{,}096$ is the practical and most energy-efficient configuration.

3) Using Thresholding and Reordering: Next, we evaluate the effectiveness of the thresholding and reordering technique. Figure 7 shows the programming cost reduction for different thresholds. For each threshold value, we find the mapping order with the maximum programming cost saving. With only reordering (i.e., threshold $= 0$), we observe a $27.8\% - 29.7\%$ reduction in the programming cost across the four DNNs. As we increase the threshold, the savings increase. We observe up to $62.2\% - 77.6\%$ reduction in programming energy for threshold $= 16$.

The thresholding approach can, however, introduce errors in the weights and might cause accuracy degradation. Figure 8 shows the accuracy metrics for various thresholds. "f" means the accuracy of the float-point model, and "0" means only quantization and no thresholding applied. The figure shows that all DNNs can tolerate small thresholds without having a significant accuracy drop. With less than 5% accuracy loss, the thresholding and reordering approach can achieve 42.9%, 46.5%, 47.4%, and 45.2% programming cost reduction for AlexNet, VGG-11, ResNet-50, and BERT-Large, respectively.

4) Comparison with Related Work: In Table I, we compare our OPCM-based PiM solution against the results reported by previous works, including Eyeriss v2 [18], TPU v3 [19], ADEPT [20], and Albireo-C [21]. Our solution achieves $4.5\times$ higher IPS and $1.2\times$ higher IPS/W than TPU v3, and $4{,}918\times$ higher IPS but 41.3% worse IPS/W than Eyeriss v2. Compared with photonic accelerators, ADEPT, and Albireo-C, our solution has lower IPS/W. However,

TABLE I
PERFORMANCE AND ENERGY EFFICIENCY OF OPCM-BASED PIM ARCHITECTURE

	This work				Eyeriss v2 [18]	TPU v3 [19]	ADEPT [20]	Albireo-C [21]
Configuration	Array size 64×64, 16 arrays, 25 GHz, batch size 4096, 7-bit quantization				ASIC	ASIC	Photonic	Photonic
Model	VGG-11	BERT	ResNet-50	AlexNet	AlexNet	ResNet-50	AlexNet	AlexNet
Threshold*	6	7	4	5				
IPS	91,493	10,162	148,166	501,629	102	32,716	217,201	7,692
IPS/W	26.05	0.76	21.55	102.64	174.8	18.18	7476.78	344.17

* Thresholds are chosen to achieve maximum programming cost savings with less than 5% accuracy loss.

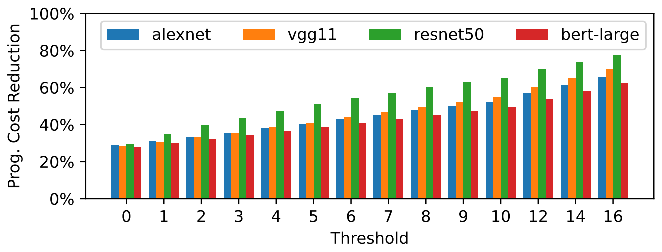

Fig. 7. Reduction in programming energy after thresholding and reordering for 64×64-sized blocks. A threshold of "0" means we quantize the model to 7-bit precision and do not use thresholding. With reordering alone and no thresholding (i.e., threshold = 0), we observe a $27.8\% - 29.7\%$ reduction in the programming cost. Using a larger threshold leads to even more reduction. The thresholding and reordering technique can reduce the programming cost by $42.9\% - 47.4\%$ with less than 5% accuracy loss (See Figure 8).

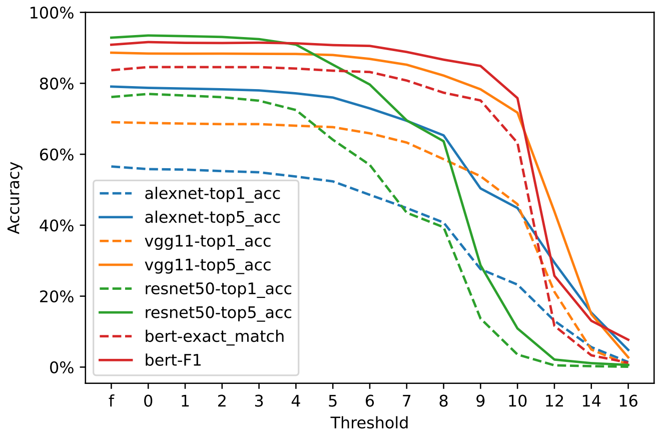

Fig. 8. Accuracy after thresholding and reordering for 64×64 blocks. "f" on the X-axis corresponds to the original FP32 model, and a threshold of "0" means we quantize the model to 7-bit precision and do not apply thresholding. All these four DNNs can tolerate a threshold of at least 4 without a significant accuracy drop.

it still achieves $2.3\times$ and $65.2\times$ higher throughput than ADEPT, and Albireo-C, respectively.

VI. CONCLUSION

The performance of DNN inference is limited by the data movement cost. To tackle this problem, we propose a complete OPCM-based PiM system architecture, which combines the OPCM array with photonic links. In the OPCM-based PiM system, the OPCM programming cost dominates. We propose to use three techniques - strategically choosing the batch size, reordering the blocks during matrix multiplication, and using thresholding when updating the OPCM array, to amortize the programming cost. Using our approach, we achieve 42.9%, 46.5%, 47.4%, and 45.2% programming energy reduction for AlexNet, VGG-11, ResNet-50, and BERT-Large, respectively.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2014.
[2] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
[3] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016.
[4] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019*, pp. 4171–4186, 10 2018.
[5] P. Villalobos *et al.*, "Machine learning model sizes and the parameter gap," *arXiv*, 7 2022. [Online]. Available: http://arxiv.org/abs/2207.02852
[6] X. Xin *et al.*, "Roc: Dram-based processing with reduced operation cycles," *DAC*, 2019.
[7] D. Quan *et al.*, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *DAC*, 2019.
[8] P. Chi *et al.*, "Prime:a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016.
[9] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.
[10] S. Jain *et al.*, "Computing in memory with spin-transfer torque magnetic ram," *TVLSI*, 2018.
[11] B. C. Lee *et al.*, "Architecting phase change memory as a scalable dram alternative," in *ISCA*, 2009.
[12] J. Feldmann *et al.*, "Parallel convolutional processing using an integrated photonic tensor core," *Nature*, 2021.
[13] F. Brückerhoff-Plückelmann *et al.*, "Broadband photonic tensor core with integrated ultra-low crosstalk wavelength multiplexers," *Nanophotonics*, 2022.
[14] P. Guo *et al.*, "Starlight: a photonic neural network accelerator featuring a hybrid mode-wavelength division multiplexing and photonic nonvolatile memory," *Optics Express*, p. 37051, 2022.
[15] H. Zhu *et al.*, "Elight: Towards efficient and aging-resilient photonic in-memory neurocomputing," *TCAD*, 2022.
[16] C. Wu *et al.*, "Programmable phase-change metasurfaces on waveguides for multimode photonic convolutional neural network," *Nat. Commun.*, 2021.
[17] A. Narayan *et al.*, "Architecting optically controlled phase change memory," *ACM Trans. Archit. Code Optim.*, vol. 19, pp. 1–26, 12 2022.
[18] Y. H. Chen *et al.*, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Power Electron.*, vol. 9, pp. 292–308, 7 2018.
[19] N. P. Jouppi *et al.*, "A domain-specific supercomputer for training deep neural networks," *Commun. ACM*, vol. 63, no. 7, p. 67–78, jun 2020.
[20] C. Demirkiran *et al.*, "An electro-photonic system for accelerating deep neural networks," *arXiv preprint arXiv:2109.01126*, 2021.
[21] K. Shiflett *et al.*, "Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics," in *2021 ISCA*, 2021.
[22] C. Ríos *et al.*, "Integrated all-photonic non-volatile multi-level memory," *Nature Photonics*, 2015.
[23] C. Ríos *et al.*, "In-memory computing on a photonic platform," *Science advances*, vol. 5, no. 2, p. eaau5759, 2019.
[24] Z. Fang *et al.*, "Ultra-low-energy programmable non-volatile silicon photonics based on phase-change materials with graphene heaters," *Nature Nanotechnology*, vol. 17, no. 8, pp. 842–848, 2022.
[25] S. K. Kim and M. Popovici, "Future of dynamic random-access memory as main memory," *MRS Bulletin*, 2018.
[26] M. Jünger *et al.*, "Chapter 4 the traveling salesman problem," in *Network Models*, ser. Handbooks in Operations Research and Management Science. Elsevier, 1995, vol. 7, pp. 225–330.
[27] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/
[28] A. A. Nikitin *et al.*, "Optical bistable soi micro-ring resonators for memory applications," *Optics Communications*, vol. 511, p. 127929, 5 2022.
[29] M. Horowitz, "Computing's energy problem (and what we can do about it)," *IEEE ISSCC*, vol. 57, pp. 10–14, 2014.