Received June 9, 2021, accepted June 16, 2021, date of publication July 1, 2021, date of current version July 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094043

PIMCaffe: Functional Evaluation of a Machine Learning Framework for In-Memory Neural Processing Unit

WON JEON[1], JIWON LEE[2], (Student Member, IEEE), DONGSEOK KANG[3], HONGJU KAL[2], (Graduate Student Member, IEEE), AND WON WOO RO[2], (Senior Member, IEEE)

[1]AI SoC Research Department, AI Processor Research Team, Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea
[2]Department of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea
[3]Memory Division, Design Verification Team, Samsung Electronics, Hwaseong 18448, South Korea

Corresponding author: Won Woo Ro (wro@yonsei.ac.kr)

This work was supported in part by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] (No. 2021-0-00853, Developing Software Platform for Programming of Processing-In-Memory (PIM), 50%), in part by Samsung Electronics Company Ltd., Hwaseong, South Korea, and in part by the Graduate School of Yonsei University Research Scholarship Grants, in 2020.

ABSTRACT The large amount of memory usage in recent machine learning applications imposes a significant system burden with respect to power and processing speed. To cope with such a problem, Processing-In-Memory (PIM) techniques can be applied as an alternative solution. Especially, the recommendation system, which is one of the major machine learning applications used in data centers, requires a large memory capacity and therefore represents a suitable candidate application that could be helped by the PIM technique. In this paper, we introduce a machine learning framework, PIMCaffe, designed for in-memory neural processing units and its evaluation environment. PIMCaffe consists of two components: a Caffe2-based deep learning framework that supports PIM acceleration and a PIM-emulating hardware platform. We develop a suite of functions, libraries, application programming interfaces, and a device driver to support the framework. In addition, we implement a prototype Neural Processing Unit (NPU) in PIMCaffe to evaluate the performance of our platform with machine learning applications. Our prototype NPU design includes a vector processor for parallel vector processing and a systolic array unit for matrix multiplication. Using the proposed software framework, we perform a detailed analysis of the in-memory neural processing unit. PIMCaffe supports evaluations of recommendation systems and various convolutional neural network models on the in-memory neural processing unit. PIMCaffe with the NPU shows up to 2.26×, 5.99×, and 1.71× speedup, compared to the ARM Cortex-A53 CPU, for the recommendation system, AlexNet, and ResNet-50, respectively.

INDEX TERMS Deep learning framework, recommendation system, processing-in-memory, neural processing unit, FPGA prototyping, functionality, system verification.

I. INTRODUCTION

Deep Neural Networks (DNNs) have been extensively researched in various areas and scale up very fast. Convolutional Neural Networks (CNNs) have been widely researched and have outperformed traditional techniques in vision processing, including for image recognition, classification, and synthesis. On the other hand, for voice processing tasks, Recurrent Neural Networks (RNNs) show superior performance compared to conventional approaches. More recently, Recommendation Systems (RS) using DNNs have been widely employed in data centers to provide better user experience and recommendations (for advertisements, social networks, movies, books, etc.) to numerous customers. In an

The associate editor coordinating the review of this manuscript and approving it for publication was Mehul S. Raval.

RS, deep learning models utilize large-sized data structures to estimate the likelihood of user preference or click-through rates for a particular item.

One of the primary characteristics of recent neural network applications is their significant memory usage [1]. CNN applications already require several tens of GBs of memory capacity, specifically for training with a large number of batched images. Moreover, RNN applications impose a certain level of burden on memory capacity and bandwidth. Furthermore, the memory capacity requirement for the recently proposed RS is expected to be over hundreds of GBs owing to the large embedding tables [1]–[5]. Such extremely high requirements for memory resources make the memory wall problem more critical in an RS than in conventional CNN or RNN applications. The Processing-In-Memory (PIM) technique, which integrates processing logic within memory, is a promising solution for the memory problems associated with RS applications. The processing elements in a PIM system can benefit from higher bandwidth and lower data movement overhead [2], [5]. Consequently, several previous studies have applied PIM techniques to neural network computations [2], [5]–[8].

Although using PIM helps resolve memory bottleneck problems, developing new DNN models in PIM is a time-consuming process. In fact, several software deep learning frameworks (e.g., Caffe2, PyTorch, TensorFlow, etc.) that support flexible and efficient deep neural networks have been widely used. To facilitate research on software models and hardware accelerators for RS in PIM systems, we propose PIMCaffe, a novel machine learning framework. With extensive modifications to the Linux system software and device driver, as well as supports of Application Programming Interfaces (APIs), PIMCaffe offers a Caffe2-based machine learning framework that is executable on Neural Processing Units (NPUs) with PIM functionality. We aim to provide an easy-to-use PIM-based deep learning framework to develop different neural network models and research with rapid implementation and evaluation in PIM.

To emulate the PIM-based NPU, we develop a multi-FPGA-based hardware platform that emulates PIM functionalities, such as near-memory and far-memory accesses. We compose our PIM evaluation platform by combining the ZCU102 and KCU1500 boards. As a processing element for PIMCaffe, we implement a prototype NPU composed of a Single Instruction Multiple Data (SIMD) processor and systolic array structure. PIMCaffe supports Facebook's Deep Learning Recommendation Model (DLRM) and 11 different presets of CNN models, including AlexNet, ResNet-50, VGGNet-19, DenseNet-121, etc. To the best of our knowledge, PIMCaffe is the first work that implements recommendation models for Field-Programmable Gate Arrays (FPGAs) with PIM functionality.

The main contributions of this paper are as follows:

- PIMCaffe framework, a Caffe2-based machine learning framework that is executable on NPUs with PIM functionality, is proposed. Core computations of neural networks, such as BLAS operations, are accelerated using the proposed PIM device. With PIMCaffe, both conventional CNN models and the latest recommendation models can be executed and analyzed on PIM-emulating platforms.
- Our proposed method has a faster evaluation speed compared to software-based PIM evaluation simulators. The evaluation speed of the proposed PIM evaluation platform is approximately 20× times faster than that of software-based simulations.
- We provide APIs for the PIM-NPU and PIM device driver in the Linux kernel. The PIM APIs allow users to easily accelerate common vector and matrix computations using the proposed PIM evaluation platform.
- A PIM-emulating FPGA evaluation platform is developed using our prototype NPU architecture. With SIMD and systolic array computing engines, our NPU processor performs vector and matrix computations in the PIM device. In our evaluations, PIMCaffe with the NPU achieves speedups of 2.26×, 5.99×, and 1.71× for a recommendation system, AlexNet, and ResNet-50, respectively, as compared to the performance of ARM Cortex-A53 CPU.

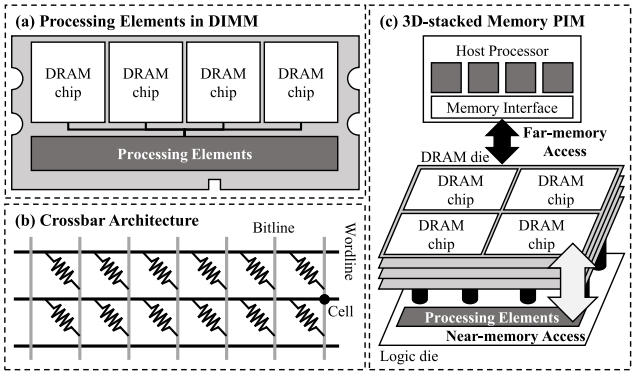


FIGURE 1. Methods for implementing processing elements in PIM devices.

II. BACKGROUND AND MOTIVATION

A. PROCESSING-IN-MEMORY TECHNIQUE

The primary feature of PIM systems is their different memory access performance between the host processor and processing element. Because the near-memory access between memory modules and processing elements in a PIM device can achieve higher bandwidth and lower data movement overhead than host processors, the PIM technique has been widely researched mainly for memory-intensive applications. Generally, processing elements of a PIM system can be implemented in three different ways; additional processing elements in DIMMs [2], [5], crossbar architectures in a non-volatile memory [8]–[10], and additional processing elements in logic dies of 3D-stacked memory [11]. Fig. 1 illustrates the core structures of the three different PIM implementations. Implementing PIM processing elements, as shown in Fig. 1(a) and (b), can achieve a lower data

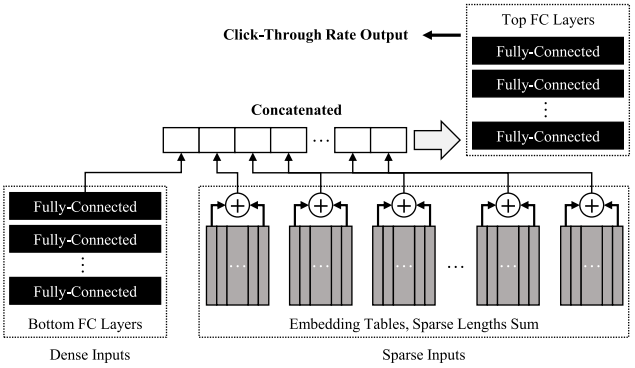


FIGURE 2. Architecture of Facebook's deep learning recommendation model [1].

movement penalty, as compared to (c), because the processing elements and memory devices are in the same die or package. However, due to CMOS technology limitations, implementing logic circuits in memory devices is difficult and poses hardware constraints on the processing elements. Alternatively, processing elements in the 3D-stacked memory PIM are implemented on separate dies and connected to memory dies using Through Silicon Via (TSV), as shown in Fig. 1(c). By separating the logic die and memory dies, the 3D-stacked memory PIM can provide a higher number of computing resources than other PIM models. In particular, a function-in-memory (FIM) DRAM implementation has been recently proposed for running machine learning applications using HBM systems [12]. With the advent of FIM DRAM, the machine learning development environment for PIM systems has become more critical.

As the importance of the PIM technique has become more apparent, there has been substantial research on PIM architectures and applications. Several cycle-accurate simulators [13], [14] and performance prediction frameworks [15] have been proposed to support the evaluation of PIM research. However, because of their slow evaluation speeds, these software-based approaches cannot run DNN applications with a realistically large-scale dataset; thus, they cannot effectively demonstrate the impact of PIM techniques.

B. RECOMMENDATION SYSTEM

RSs are widely used in the industry for better user experiences (e.g., YouTube, Fox, Facebook) and efficient advertising (e.g., Amazon, eBay, Google). To facilitate research on recommendation models, Facebook released a DLRM framework [16]. Fig. 2 presents the architecture of Facebook's deep learning recommendation model [1]. The recommendation model predicts Click-Through Rates (CTRs) and recommends items with high CTRs to a user. Unlike conventional neural network applications, the recommendation model has a unique data structure called an embedding table. Every user or item in the recommendation model requires an embedding vector in the embedding table. Although the size of the embedding table may differ according to the recommendation model's design, the embedding table is the primary cause

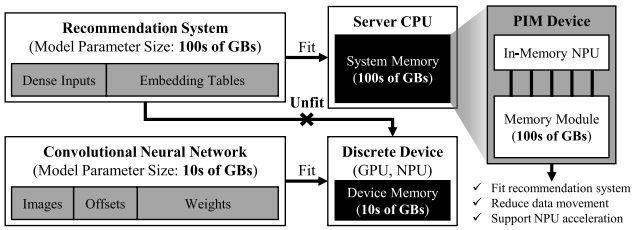


FIGURE 3. Memory capacity problem of recommendation model and PIM solution.

of the large memory capacity requirement of RSs. Consequently, the recommendation model requires more memory resources than conventional CNN applications.

Fig. 3 depicts the memory capacity and platform fitness of RS and CNN applications. For production-scale applications, a recommendation model inference requires hundreds of GBs of main memory, whereas a CNN application requires merely tens of GBs [1]–[5]. Therefore, although CNN applications can be effectively accelerated with discrete devices, such as GPU or NPU, recommendation models can only be handled by a server-side CPU containing massive main memory capacity. However, a general purpose CPU cannot fully utilize the potential parallelism in recommendation models. Furthermore, the massive data movement present in the recommendation model on the CPU imposes significant memory overhead on the server system. In this context, the PIM technique can be a promising solution for RS [2]. However, despite the increasing importance of applying the PIM technique to recommendation systems, insufficient suitable research environments has slowed the potential technological advances in those techniques.

C. MACHINE LEARNING FRAMEWORK FOR FPGA

For deep learning applications, the neural network model must vary based on the purpose of the application and target execution environment. Many deep learning frameworks, such as Caffe2 [17], PyTorch [18], MXNet [19], and TensorFlow [20], have been proposed to provide an efficient and flexible neural network development environment. With the support of deep learning frameworks, many novel neural network designs, including DLRM [1], VGGNet [21], ResNet [22], and DenseNet [23] have been developed.

Although many neural network applications operate on conventional platforms, such as CPUs or Graphic Processing Units (GPUs), the computational characteristics of the neural network do not fit within such conventional processors. To achieve higher performance and energy efficiency, NPUs require unconventional and domain-specific computing structures, such as systolic arrays and adder trees. However, unlike CPUs and GPUs, NPUs have not been standardized; thus, various unique NPU designs have been widely proposed. Therefore, FPGAs have been actively used as a development platform for various NPU designs [24]–[31].

To achieve flexibility in both hardware and software development for neural networks, several previous works

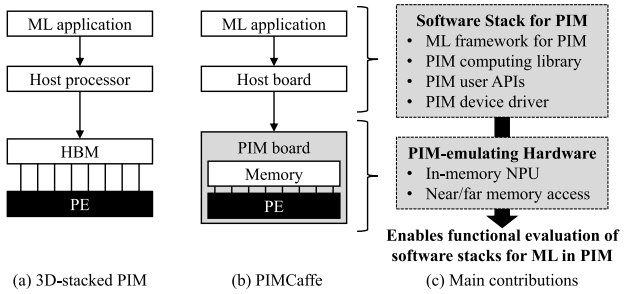


FIGURE 4. Conceptual hardware and software implementations of the (a) 3D-stacked memory-based PIM system, (b) PIMCaffe, and (c) main contributions of the proposal.

proposed deep learning frameworks that are executable on FPGAs [27]–[31]. These works provide toolflows for mapping Caffe or Torch-based deep learning applications on FPGAs. However, previous studies have focused on implementing only CNN or RNN models for FPGAs. Emerging deep learning applications, such as RS, cannot be mapped to FPGAs using the previously proposed toolflows. Furthermore, previous works do not support PIM functionality; thus, the impact of PIM on both DNN applications and NPUs cannot be studied. In summary, the FPGA-based PIM evaluation platform can support research on the impact of the PIM technique on previous FPGA-based NPU architectures. Moreover, FPGA-based evaluation can achieve a faster evaluation speed than software-based simulators.

III. PIMCAFFE ARCHITECTURE

Addressing the absence of a deep learning framework for PIM techniques, we propose PIMCaffe, which is a novel machine learning framework for NPUs with PIM functionality. PIMCaffe consists of a Caffe2-based deep learning framework to support PIM-NPU acceleration for various neural network models, including DLRM and CNNs, and a PIM-emulating hardware evaluation platform with NPU processors. Furthermore, PIMCaffe provides several different levels of abstraction to programmers for better programmability and optimization. The abstraction levels are the DNN framework, BLAS library, user APIs, and device driver interfaces. A programmer can simply use the PIMCaffe framework to accelerate DNN applications with the PIM device or conduct fine-grained optimizations for general applications using PimBLAS, PIM APIs, and PIM device driver interfaces. Fig. 4 presents the primary contributions of this study by summarizing the comparison between the 3D-stacked memory PIM system and PIMCaffe implementations. We focused on functional evaluation and validation across the machine learning application, software stack, and PIM hardware. Accordingly, we implement a representative NPU design with the SIMD and systolic array near the memory module to build the PIM hardware implementation. The implementation details of the PIMCaffe software stack and hardware evaluation platform are discussed in this section.

TABLE 1. List of APIs for PIM device.

API Name	Argument	Description
`pim_open`	-	Open PIM device.
`pim_sync`	-	Check if previous PIM operation is completed.
`pim_simd`	3 addresses, 1 vector size, 1 opcode	Perform general vector-vector arithmetic computations with SIMD unit.
`pim_simd_const`	2 addresses, 1 vector size, 1 constant, 1 opcode	Perform general vector-constant arithmetic computations with SIMD unit.
`pim_systolic`	3 addresses, 1 matrix size	Perform matrix multiplication with systolic array.

A. SOFTWARE STRUCTURES OF PIMCAFFE

We performed a detailed analysis of the software structures of the Caffe2 framework and modified core computation modules in the framework. As shown in Fig. 5, although the user-level deep learning applications and network models differ, they consist of common operations. In particular, most data structures in neural network applications are vectors or matrices; thus, most computations in neural networks are handled by the Basic Linear Algebra Subprograms (BLAS) library [32].

Linear algebraic computations in DNN frameworks are processed with standardized BLAS function interfaces (e.g., GEMV and GEMM). Insofar as the input interfaces and the output results of the BLAS functions are identical, various BLAS libraries such as Intel MKL [33], NVIDIA cuBLAS [34], and OpenBLAS [35] are exchangeable. By modifying the external BLAS library linking process, we added our novel BLAS library, PimBLAS, as a selectable option in PIMCaffe. With this modification, the core computations of neural network applications are automatically processed in the PIM device. Accordingly, deep learning programmers do not have to make complicated modifications to applications to use the PIM device. With PIMCaffe, any deep learning applications using the BLAS library can transparently benefit from in-memory computing and NPU acceleration. To achieve such transparency, we developed a device driver and APIs for the PIM device. In this work, we confirmed correct operations of the DLRM and 11 different CNN models (SqueezeNet, MobileNet, GoogLeNet, Inception, DenseNet, ResNet, R-CNN, CaffeNet, AlexNet, ZFNet, and VGGNet) with PIMCaffe. Although we integrated our work with only the Caffe2 framework, any deep learning framework that uses the BLAS library as a core computing module could be integrated with our work.

B. PIM APPLICATION PROGRAMMING INTERFACE

Using the proposed NPU processor inside the PIM device without proper software support can be challenging for deep learning programmers. Although most core computations in the DNN framework are already processed by the PIM device, its programmability is important because more custom optimizations and accelerations can be performed by

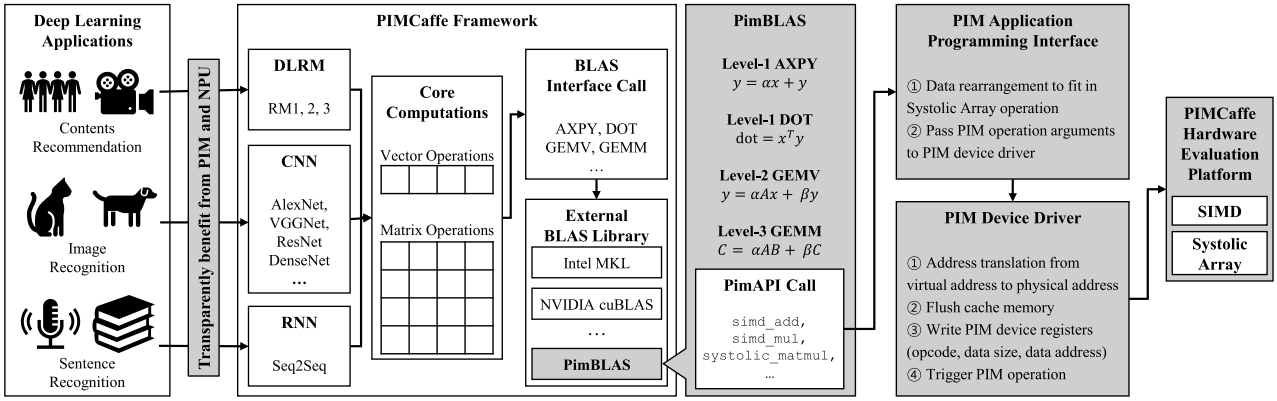


FIGURE 5. Execution flow of the PIMCaffe machine learning framework.

the programmers. However, because both the PIM-emulating evaluation platform and prototype NPU are non-conventional computing platforms, programmability is not guaranteed. Therefore, proper system software supports and interface abstractions are necessary to utilize the proposed PIM device without a complicated understanding of PIM operations. Table 1 lists a set of APIs for the NPU processor in the PIM device. The APIs consist of simple function interfaces, such as `pim_sync`, `pim_simd`, and `pim_systolic`. Each API requires certain arguments for a PIM-NPU processor, such as data addresses for the operand or result, operation code, constant values, vector sizes, or matrix sizes. The arguments of the APIs are passed to the PIM device hardware through the PIM device driver. In particular, the data addresses for APIs are virtual addresses, and virtual to physical translation is performed in the PIM device driver. The following subsections present a detailed implementation of the device driver.

When the `pim_open` API is called, it opens the PIM device driver using the `open()` function with the `O_RDWR` option in the `fcntl` library. If the PIM device is not opened properly, the API terminates the program. `pim_sync` API can be called after a PIM operation, and it checks the value of a certain register with an infinite loop. The register is set to zero after starting a PIM operation and is set to one after the operation completion. The infinite loop in `pim_sync` API can be finished when the register value is confirmed as one. A programmer can prevent the desynchronization problem of the result variables from the PIM device using the API. The `pim_simd` and `pim_simd_constant` APIs trigger the SIMD unit in the PIM device to perform general arithmetic computations, such as addition, subtraction, multiplication, and division on two vectors, or a vector and constant. Finally, the `pim_systolic` API triggers the systolic array unit in the PIM device to multiply two matrices. Before the API triggers hardware operations, it rearranges the input matrices to fit in the systolic array operation in the PIM device. The input matrices are tilted, and empty elements are filled with zeros. Once the pre-processing is completed, the API writes

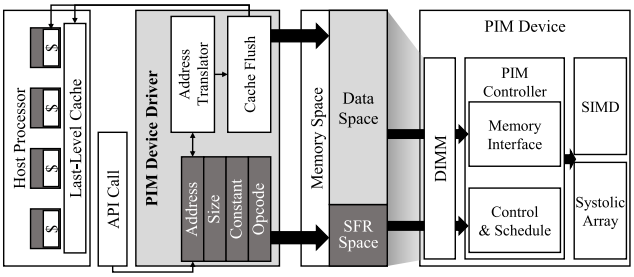


FIGURE 6. Operation of the PIM device driver.

the necessary arguments through the PIM device driver. With the support of the APIs, any general application can be accelerated with a PIM-NPU processor as well as the PIMCaffe machine learning framework.

C. PIM DEVICE DRIVER

To perform user-defined computations on the PIM device, several operating system (OS) level operations must be performed in the PIM device driver, as shown in Fig. 6. First, the PIM device driver translates the virtual addresses of the target data into physical addresses. The target data addresses in the user programs are virtual addresses. However, the PIM device can only understand physical addresses because the PIM device is located in a memory module and performs computations separately from the OS. Second, the target data in the cache memory of the host processor must be flushed to the main memory to guarantee data coherency. Target data in the main memory can become outdated if the data are modified in the cache memory. In addition, if the data are modified in the PIM device while it is still in the host cache memory, the data can become incoherent. The PIM device driver performs a cache flush on every cache line of the target data.

We implement the PIM device driver as a Linux device driver to perform OS-level operations. In addition to the address translation and cache flush operations, the PIM device driver passes the PIM instructions and arguments from

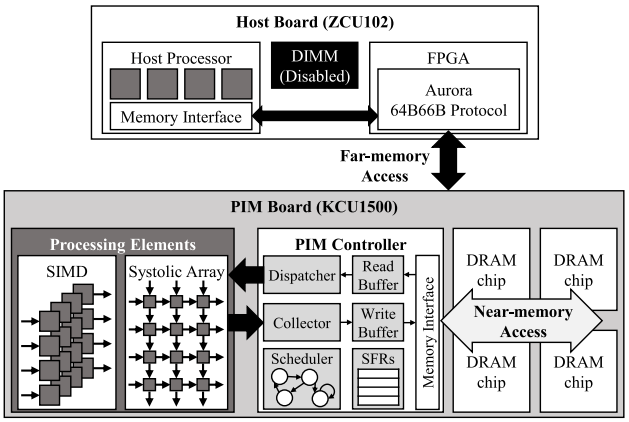


FIGURE 7. PIMCaffe hardware system architecture.

the host processor to the PIM device. The PIM device driver reserves a small area (few KBs) of the memory space for Special Function Registers (SFRs). Whenever data is written to the reserved memory region, the PIM device driver detects it and sends the data to the corresponding SFR in the PIM device. Information required for PIM operations, such as the physical addresses and size of the target data, constant operand value, and operation code, is transferred to the SFRs in the PIM device. The PIM controller holds the SFRs and utilizes them for direct near-memory access in the PIM device. After completing the preparation for the PIM operation, the PIM device driver sends a triggering signal to the PIM device, and the processing elements in the device begin operations using the information in the SFRs.

Because of the additional operations and preparation processes in the PIM device driver, such as address translations and cache flushing, the performance of the PIM device can be degraded if the size of the target data is extremely small. For small-sized data, computing them on the host processor can be faster than transferring them to the PIM device. We evaluate the effects of such overheads in Section IV-B.

D. IN-MEMORY NEURAL PROCESSING UNIT HARDWARE DESIGN

To emulate PIM functionalities, we focus on the presence of near-memory and far-memory accesses in the proposed PIM evaluation platform. Emulating the PIM behaviors, we propose a multi-FPGA-based PIM evaluation platform. The proposed system architecture consists of a host board, comprising a host processor, and a PIM board with the main memory and processing elements. Fig. 7 illustrates the proposed system architecture. We composed our evaluation platform with the Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit and the Xilinx Kintex UltraScale FPGA KCU1500 Acceleration Development Kit as the host and PIM boards, respectively. In the host board, we used an embedded CPU as a host processor, disabled DRAM chips, and implemented a board-to-board interface in the FPGA to communicate with the PIM board using the Aurora 64B66B protocol. The CPU in the host board, ARM Cortex-A53, runs an OS and the proposed PIMCaffe machine learning framework. In the PIM board, we used DRAM chips as the main memory of the system and implemented the processing elements and PIM controller in the FPGA. Because the host processor considers DRAM chips in the PIM board as the system main memory, the memory accesses of the host processor become board-to-board far-memory accesses. On the other hand, memory accesses from processing elements become on-board near-memory accesses. Whereas near-memory access requires only an on-board memory interface, far-memory access requires board-to-board communication overhead, in addition to the near-memory access process. In summary, the multi-FPGA-based PIMCaffe hardware evaluation platform possesses both far-memory access and near-memory access characteristics, which are the key features of a PIM system.

The PIM controller fetches the target data from the DRAM chips to the processing elements and schedules the operations of the processing elements. First, it receives some information from the PIM device driver and stores it in the SFRs, as explained in the previous section. Then, according to the physical address and size information, the target data are transferred from the DRAM to the *read buffer*. Once the data are ready, *Dispatcher* passes the operands to the processing elements, and the PIM controller triggers the arithmetic operations of the processing elements. The result values are written back to the DRAM by the *Collector* and *write buffer*. The memory read/write operations of the dispatcher/collector are pipelined with arithmetic operations of the processing elements and scheduled by the PIM controller.

In the PIMCaffe hardware evaluation platform, we modeled the prototype NPU processor with SIMD engines and systolic array structures to effectively accelerate the neural network computation. In Fig. 7, the processing elements in the PIM board depict the hardware designs of the SIMD and systolic array-based prototype NPU. The SIMD module performs general arithmetic vector operations. Vector computations in neural networks, such as vector element-wise addition and constant-vector multiplication, can be computed using the SIMD engine. In contrast, the systolic array module performs matrix multiplications with Multiplier-Accumulators (MACs). Data in a systolic array can be relayed from one MAC to adjacent MACs. Compared to conventional non-relaying computing structures, the systolic array structure reduces the interconnection overhead of the processor while efficiently computing matrix multiplication. Because a large portion of neural network computations involves matrix multiplication, the systolic array can effectively accelerate neural network applications. However, when the input matrix is too narrow, MACs are underutilized, deteriorating the relative performance of the systolic array. Detailed hardware specifications of the NPU are presented in Section IV.

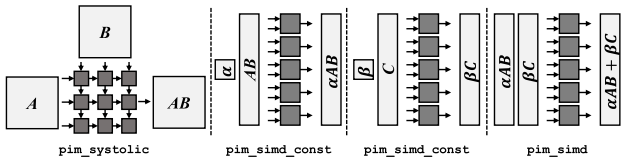


FIGURE 8. GEMM ($C = \alpha AB + \beta C$) operations in PimBLAS, where *A*, *B*, and *C* are matrices and α and β are constants.

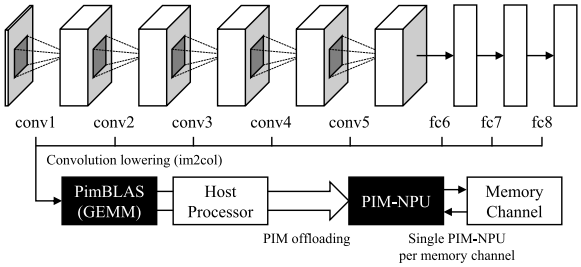


FIGURE 9. Neural network layer computation mapping of AlexNet in the case wherein there is a single PIM-NPU per memory channel.

E. PIMBLAS ACCELERATION

PimBLAS is a BLAS library that supports NPU acceleration in the proposed PIM device. We selected frequently used BLAS functions in the Caffe2 framework and performed fine-grained optimizations of the functions. As shown in Fig. 5, the selected BLAS functions are AXPY (alpha-vector product), DOT (vector-vector dot product), GEMV (generalized matrix-vector multiply), and GEMM (generalized matrix-matrix multiply). BLAS functions consist of various mathematical operations for scalars, vectors, and matrices. We applied different computing modules based on the characteristics of each operation. In the selected BLAS functions, the SIMD and systolic array modules perform any element-wise operations (e.g., scalar-vector multiplication, vector-vector addition, matrix-matrix addition) and matrix-matrix multiplication, respectively. In the case of DOT and GEMV functions, serialized accumulating operations are necessary, and we process the operations with the host CPU. Fig. 8 illustrates an example of a GEMM operation accelerated using the systolic array and SIMD units of the proposed PIM device.

For matrix multiplication in GEMM, the size of the matrix can be very narrow, especially in fully-connected layers (e.g., [1024 $\times$ 1] and [1 $\times$ 1024]). As explained in the previous subsection, the systolic array unit operates inefficiently for narrow matrices. For optimization, PimBLAS processes such matrix multiplications using the SIMD unit, not a systolic array. Because PimBLAS is a standalone BLAS library, it can be utilized in any application that can benefit from NPU acceleration and PIM functionality. In this work, we integrated PimBLAS with the Caffe2 framework to accelerate deep learning applications.

F. MAPPING NEURAL NETWORK TO PIM DEVICE

Because most computations in neural network layers consist of BLAS functions and we replace the BLAS library in the Caffe2 framework with PimBLAS as discussed in Section III-E, neural network layers are automatically mapped into the PIM device. In the proposed PIM system, only one PIM device exists in one memory channel in the entire memory system, making the neural network layer mapping technique of the current PIMCaffe system straightforward. If, in the future, we extend the PIMCaffe hardware evaluation platform to multi-channel PIM devices, we can further optimize the PIMCaffe software stack to support more efficient neural network layer mapping schemes. Accordingly, Jeong *et al.* [36] proposed studies on the data layout for a multi-channel PIM architecture with PIM device-to-device communication functionality. A future PIMCaffe system with multi-channel PIM devices would benefit from the multi-channel PIM data layout management methods proposed in the previous work.

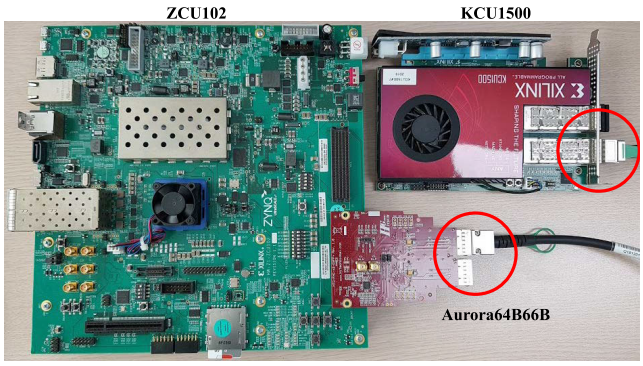


FIGURE 10. PIMCaffe hardware evaluation platform with ZCU102 and KCU1500.

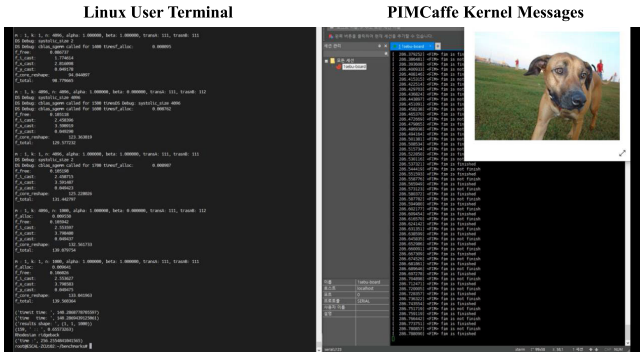


FIGURE 11. PIMCaffe execution example of an AlexNet inference.

IV. EVALUATION

A. EXPERIMENTAL SETUP

Fig. 10 shows the implementation of the PIMCaffe hardware evaluation platform, and Table 2 presents its hardware specification. We use the Xilinx Zynq ZCU102, Xilinx Kintex KCU1500 boards, and Xilinx Aurora 64B66B link as the host board, PIM board, and board-to-board connection interface, respectively. The host processor of the ZCU102 board, ARM Cortex-A53, runs an OS, PIM device driver, and performs general user programs or computations that do not use PIM-NPU functionality. Fig. 11 illustrates an example of an

TABLE 2. PIMCaffe hardware specification.

Parameter	Value
Host Board	Zynq ZCU102
Processor	ARM Cortex-A53
On-Chip Memory	32 KB Private L1 I/D Cache 1 MB Shared L2 Cache
Off-Chip Memory	Not available (Disabled)
PIM Board	Kintex KCU1500
Processing Element	64 SIMD Unit 32 × 32 Systolic Array Unit
On-Chip Memory	768 B R/W Buffer (PIM Controller) 1 MB Internal Buffer (Systolic Array)
Off-Chip Memory	4 GB DDR4 DRAM (Shared with CPU)

TABLE 3. Network parameters of recommendation models.

Parameter	RM1	RM2	RM3	RM4
Embedding Table Count	5	40	5	5
Embedding Table Size	$5 * 10^4$	$1 * 10^5$	$5 * 10^4$	$5 * 10^4$
Sparse Feature Size	32	32	128	128
Indices per Lookup	80	80	80	80
Bottom MLP	1000-128-64-32	1000-128-64-32	4096-1536-512-128	10000-2560-256-128
Top MLP	128-32-1	128-32-1	128-32-1	128-32-1

AlexNet inference operation using the PIMCaffe evaluation platform. The left side of the figure shows the user terminal outputs. Users can execute Caffe2 commands on the terminal, and the output is *Rhodesian ridgeback*, as displayed in the figure. The right side of the figure depicts that users can obtain information about the PIM device using kernel messages, such as status, opcode, target memory address, etc.

The NPU design of a PIM device can vary based on the design goals and resource constraints. In this study, we implemented a prototype NPU with a 64-way SIMD computing engine and a 32 × 32 systolic-array-structured MAC. Computation units in both the SIMD and systolic array process half-precision floating-point data. For our prototype NPU design, we used approximately 30% of the hardware resources in the PIM board (e.g., LUT, Flip-Flop, BRAM, and DSP). Although resources in the host board are used for board-to-board communication, they only require less than 2% of the available resources. The software used are Ubuntu 16.04.6 LTS, Linux kernel version 4.9.0, Python 2.7.12, and Caffe2 version 1.2.0 (with PyTorch). We evaluated the performance of the DLRM and CNN inference on different network models using the PIMCaffe hardware evaluation platform. We compared the number of execution cycles of PIMCaffe with the host processor. For evaluation speed comparison, we modeled several hardware/software components of PIMCaffe in a gem5 full-system simulator [37]. We compared the evaluation speed between PIMCaffe and the simulator which is executed on a real server system with an AMD EPYC 7551 CPU.

B. PIMBLAS PERFORMANCE

As mentioned in Section III-E, PimBLAS is executable as a standalone BLAS library separated from PIMCaffe. Fig. 12 shows the proportion of execution time of PimBLAS and its speedup as compared to BLAS execution on the host board CPU. When the size of the input data is too small, the performance of PimBLAS is degraded due to preparation overhead for the PIM device, as discussed in Section III-C. As the input data size is increased, each PimBLAS function reaches a maximum speedup of 3.79×, 0.92×, 1.21×, and 16.27× for AXPY, DOT, GEMV, and GEMM, respectively. As explained in Section III-E, the PIM speedups of DOT and GEMV functions are limited because they require heavily serialized CPU operations. The GEMM function shows superior speedup among the PimBLAS functions because the matrix multiplication can be effectively accelerated using the systolic array in the PIM device.

C. PIMCAFFE EVALUATION WITH DLRM

We modeled four different Recommendation Models (RMs) in DLRM with the PIMCaffe framework and evaluated them on the PIMCaffe hardware platform, as shown in Table 3. Fig. 13 shows the breakdown of the parameter memory size and the number of Floating point Operations (FLOPs) for each RM when the batch size is 128. The total model parameter size of each RM ranges from 33.19 MB to 514.02 MB, and the total number of FLOPs ranges from 0.116 to 20.293 billion. Overall, *SparseLengthsSum* (embedding table) requires the most memory capacity, whereas *FC* and *FCGradient* are dominant with respect to computations. The characteristics of the RMs are determined by the size of the embedding tables and fully-connected layers. RM-1 and RM-2 possess a large embedding table and a small fully-connected layer, whereas RM-3 and RM-4 exhibit the reverse characteristics (i.e., relatively small embedding tables and large fully-connected layer).

Fig. 14 shows the normalized execution cycles of each RM for different batch sizes. When the batch size

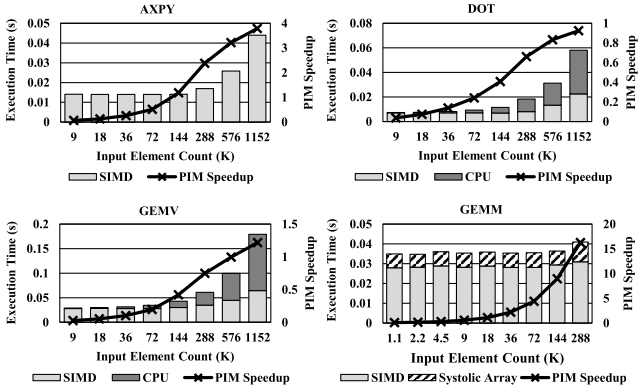


FIGURE 12. PimBLAS execution time and speedup compared to the host processor.

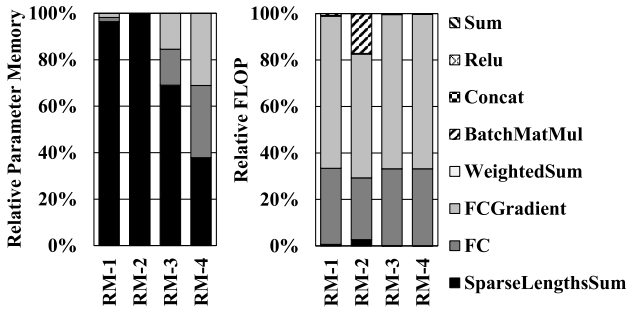


FIGURE 13. Relative parameter memory size (left) and relative number of FLOP (right) of operators in DLRM applications.

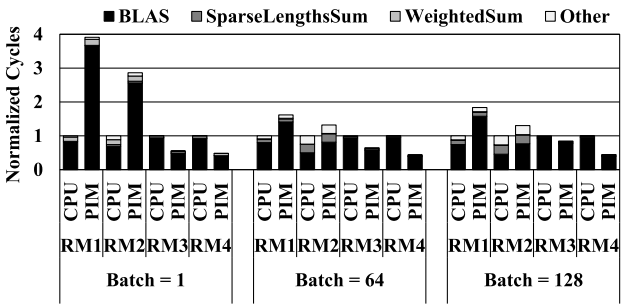


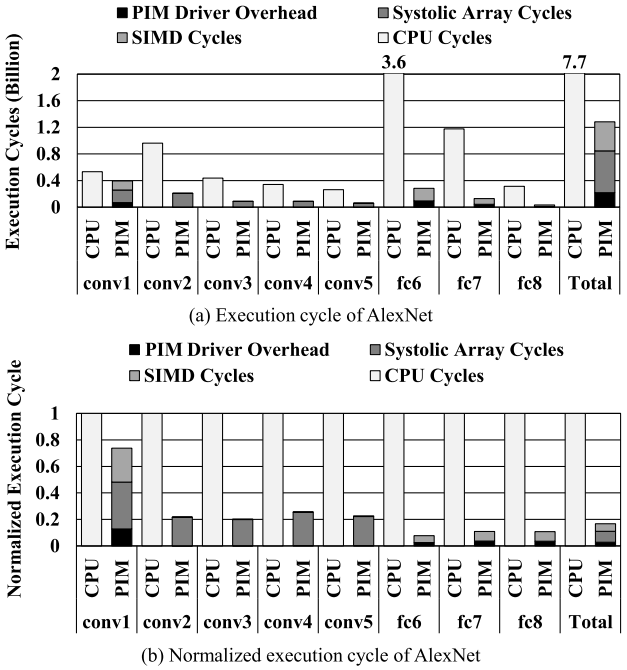
FIGURE 14. DLRM performance for different recommendation models and batch sizes.

is 1, the systolic array is underutilized, as explained in Section III-D; thus, PIMCaffe is slower than the CPU. In addition, the small sizes of the fully-connected layers of RM-1 and RM-2 degrade the performance of PIMCaffe. Meanwhile, RM-3 and RM-4 show improvements in performance from the BLAS acceleration of PIMCaffe. For 128 batched RM-4, PIMCaffe shows 2.26× speedup, compared to the host processor.

D. PIMCAFFE EVALUATION WITH CNN

Fig. 15(a) shows the execution cycle breakdown of an AlexNet [38] inference by layer in PIMCaffe with a batch size of 1. Fig. 15(b) shows the execution cycle breakdown of PIMCaffe normalized to the host processor. *PIM Driver Overhead* contains the elapsed cycles of the PIM device driver, such as address translation, cache flushing, and SFR writing. Detailed operations of the PIM device driver were introduced in Section III-C. *Systolic Array Cycles* and *SIMD Cycles* contain elapsed cycles of hardware operation time for the systolic array unit and SIMD unit, respectively. In our evaluation, PIMCaffe with the prototype NPU shows 5.99× speedup for AlexNet as compared to the host processor. In PIMCaffe, 17.06% of the total cycle is used for the PIM device driver; 48.82% and 34.12% are used for the systolic array and SIMD units, respectively.

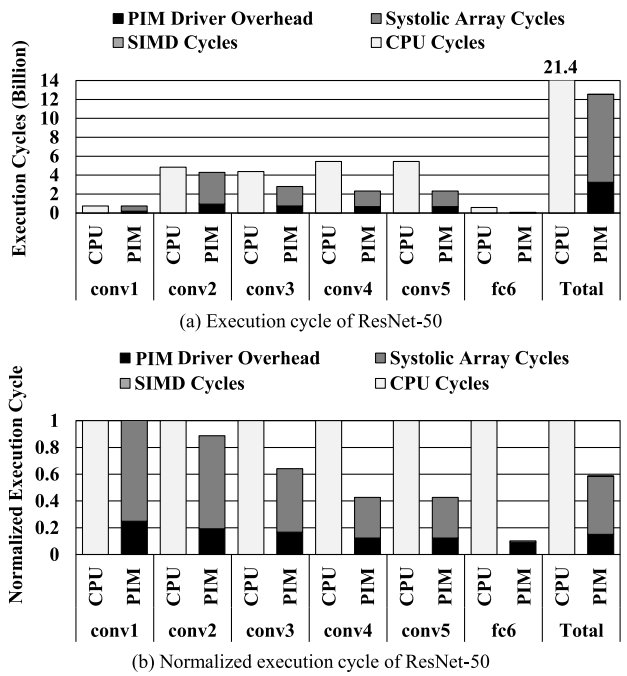
Fig. 16 demonstrates the execution cycle breakdown and normalized execution cycle breakdown of a ResNet-50 [22] inference by layer in PIMCaffe with a batch size of 1. PIMCaffe shows 1.71× speedup for ResNet-50 as compared



(a) Execution cycle of AlexNet

(b) Normalized execution cycle of AlexNet

FIGURE 15. AlexNet inference performance.



(a) Execution cycle of ResNet-50

(b) Normalized execution cycle of ResNet-50

FIGURE 16. ResNet-50 inference performance.

to the host processor. In PIMCaffe, 25.84% and 74.11% of the total cycle are used for the PIM device driver and systolic array unit, respectively. The SIMD unit takes only 0.05% of the total cycle because most BLAS computations are processed in the systolic array unit. In our analysis, ResNet-50 is composed of more and smaller-sized workloads compared to AlexNet. Frequent accesses degrade PIMCaffe performance

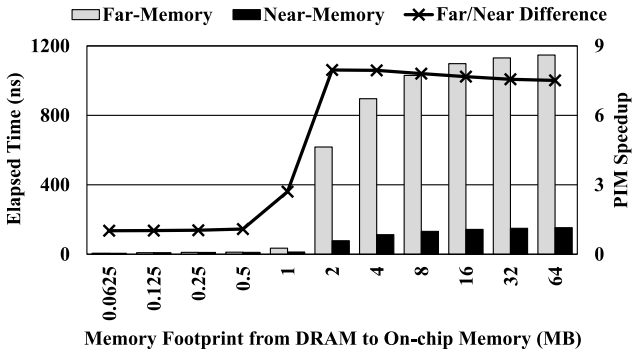


FIGURE 17. Far-memory and near-memory performance comparison on the PIMCaffe hardware evaluation platform.

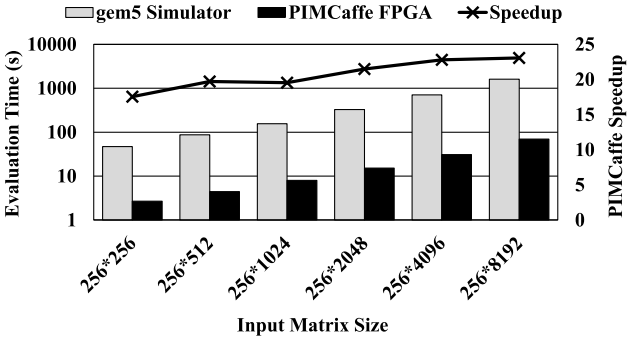


FIGURE 18. Evaluation speed of PIMCaffe and gem5 simulator on same matrix multiplication workloads.

due to the PIM device driver, whereas small-sized workloads degrade the performance of the systolic array unit for reasons explained in Section III-D. Therefore, ResNet-50 shows lower speedup compared to AlexNet.

E. IMPACT ON MEMORY PERFORMANCE

As shown in Fig. 7, the PIMCaffe hardware evaluation platform emulates the PIM functionality using multiple FPGAs. To quantitatively compare the performances of far-memory and near-memory accesses, we performed a memory benchmark analysis on both the baseline and proposed platforms [39]. The baseline platform consists of only a single ZCU102 board without a KCU1500 board. In the baseline platform, the DDR4 DRAM in the ZCU102 board conducts memory requests from the processor to the system main memory. In contrast, in the proposed evaluation platform, the host processor accesses the DRAM on the KCU1500 board. In summary, Fig. 17 presents the memory performance of the off-board and on-board memory access for the far-memory and near-memory, respectively. The figure shows the elapsed time of accessing various sized data; thus, a lower elapsed time represents a higher memory bandwidth. For 2 MB or larger-sized accessing data, near-memory access outperforms far-memory access by approximately eight times. The processing elements in the PIM device can access data with near-memory performance, whereas memory access from the host processor suffers from degraded far-memory performance.

F. EVALUATION DELAY OF PIMCAFFE

We modeled the PIM API, PIM device driver, SIMD, and systolic array units of the PIM device using the gem5 full-system simulator [37]. The modified simulator emulates PIMCaffe hardware design and system software. We evaluate PIMCaffe with the full-system mode of gem5 to analyze both hardware modifications (e.g., PIM functionality and NPU acceleration) and software implementations (e.g., API and device driver). The simulator for PIMCaffe is not feasible for running the entire execution of the machine learning frameworks because they comprise complicated software structures and require heavy operations. Therefore, we performed matrix multiplication micro-benchmarks using the systolic array in both the FPGA-based hardware evaluation platform and gem5 simulator, and compared the evaluation times. Fig. 18 shows the elapsed times in log scale for the same workloads on both the PIMCaffe FPGA and the gem5 simulator. To explicitly compare the workload evaluation delay, irrelevant evaluation parts, such as system booting time, are excluded from the elapsed time. In our evaluation, the FPGA-based PIMCaffe evaluation platform shows 17.56–23.06 times faster evaluation speed when compared to the software-based gem5 full-system simulator.

V. RELATED WORK

To the best of our knowledge, PIMCaffe is the first work that proposes a machine learning framework for PIM devices, specifically designed for hardware platforms. In particular, PIMCaffe supports the evaluation of RS that require a large memory capacity. We introduce previous studies that proposed evaluation techniques for PIM systems.

Xu *et al.* [13] proposed a flexible and full-system-based PIM simulator called *PIMSim*. PIMSim integrates DRAMSim2 [40], HMCSim [41], and NVMain [42] to support various memory models. In addition, the processing elements in PIMSim can be configured with gem5 [37] core models. In PIMSim, a user can select three different simulation modes, namely a full-system mode, an instrumentation-driven mode, and a fast mode, with a trade-off between speed and accuracy.

Singh *et al.* [15] proposed an ensemble learning-based performance prediction framework for near-memory computing applications called *NAPEL*. Based on microarchitectural parameters and application characteristics, NAPEL trains simulation models using the ensemble learning method. With pre-trained models, NAPEL provides a much faster evaluation speed for early design space exploration with acceptable error rates compared to microarchitectural simulators.

Previous studies have presented software-based PIM evaluation platforms. However, DNN applications require numerous hardware resources for software-based simulators, compared to conventional applications. Compared to previous works, PIMCaffe provides an FPGA-based PIM evaluation platform, thus supporting evaluations of DNN applications with a realistic large-scale dataset, including RS.

Furthermore, the existing FPGA-based NPU designs can be easily implemented as processing elements in PIMCaffe.

Recently, Kwon *et al.* [12] proposed the real product deployment of an HBM-based PIM device, called function-in-memory (FIM) DRAM. Compared to PIMCaffe, the processing elements (programmable computing unit, PCU) of FIMDRAM are placed inside the DRAM bank using Samsung's HBM2 fabrication technology. A PCU consists of ALU arrays that support add, multiply, multiply-accumulate, and multiply-and-add. On the other hand, the processing elements (PIM-NPU) of PIMCaffe are located on the logic die of the HBM package, and the PIM-NPU consists of SIMD and systolic array units. In particular, the FIMDRAM paper focused on chip-level DRAM architecture and control, whereas PIMCaffe focuses on the functional verification of software stacks for PIM-accelerated neural network applications.

VI. CONCLUSION

We observe that the absence of a rapid PIM evaluation platform for neural network applications is limiting essential research, specifically for memory-intensive models, such as RS. Addressing this point, in this study, we propose PIMCaffe, which consists of a Caffe2-based deep learning framework that supports PIM acceleration, and a PIM-emulating hardware platform. With PIMCaffe, any neural network model using the BLAS library can transparently benefit from the PIM functionality and NPU acceleration without degrading programmability. To the best of our knowledge, PIMCaffe is the first work that supports a machine learning framework for PIM devices. We design a hardware evaluation platform that emulates PIM functionalities and validates the execution of 11 CNN models and DLRM applications using the PIMCaffe framework on the evaluation platform. In our evaluation with the prototype NPU, PIMCaffe shows up to 2.26×, 5.99×, and 1.71× speedup for RS, AlexNet, and ResNet-50, respectively, when compared to the ARM Cortex-A53 CPU. In addition, the PIMCaffe evaluation platform shows approximately 20× times faster evaluation speed for the same workload than a software-based PIM simulation.

REFERENCES

[1] U. Gupta, C. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, and H. Lee, "The architectural implications of Facebook's DNN-based personalized recommendation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 488–501.

[2] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Apr. 2019, pp. 740–753.

[3] J. Park *et al.*, "Deep learning inference in Facebook data centers: Characterization, performance optimizations and hardware implications," 2018, *arXiv:1811.09886*. [Online]. Available: http://arxiv.org/abs/1811.09886

[4] J. Hestness, N. Ardalani, and G. Diamos, "Beyond human-level accuracy: Computational challenges in deep learning," in *Proc. 24th Symp. Princ. Pract. Parallel Program.*, 2019, pp. 1–14.

[5] L. Ke, U. Gupta, B. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H. Lee, and M. Li, "RecNMP: Accelerating personalized recommendation with near-memory processing," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2020, pp. 790–803.

[6] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, May 2018, pp. 655–668.

[7] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "CMP-PIM: An energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proc. 55th Annu. Design Autom. Conf.*, Apr. 2018, pp. 1–6.

[8] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, Jun. 2016

[9] G. W. Burr, R. M. Shelby, S. Sidler, C. Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Trans. Electron Devices*, vol. 62, no. 11, pp. 3498–3507, Nov. 2015.

[10] A. F. Vincent, J. Larroque, N. Locatelli, N. Ben Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J.-O. Klein, S. Galdin-Retailleau, and D. Querlioz, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 2, pp. 166–174, Apr. 2015.

[11] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCaulex, P. Morrow, D. Nelson, D. Pantuso, and P. Reed, "Die stacking (3D) microarchitecture," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 469–479.

[12] Y. C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, and H.-S. Yu, "25.4 A 20 nm 6GB function-in-memory DRAM, based on HBM2 with a 1.2TFLOPS programmable computing unit using bank-level parallelism, for machine learning applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 350–352.

[13] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li, "PIMSim: A flexible and detailed processing-in-memory simulator," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 6–9, Jun. 2019.

[14] J. P. C. D. Lima, "PIM-gem5: A system simulator for processing-in-memory design space exploration," M.S. thesis, Instituto de Informática, Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2019.

[15] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "Napel: Near-memory computing application performance prediction via ensemble learning," in *2019 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[16] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," 2019, *arXiv:1906.00091*. [Online]. Available: http://arxiv.org/abs/1906.00091

[17] F. Research. (2016). *Caffe2: A New Lightweight, Modular, Scalable Deep Learning Framework*. [Online]. Available: https://caffe2.ai/

[18] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," *31st Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1–4.

[19] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*. [Online]. Available: http://arxiv.org/abs/1512.01274

[20] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

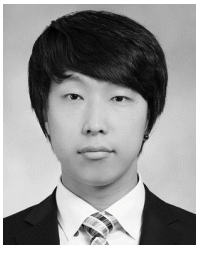
[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Oct. 2016, pp. 770–778.

[23] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Apr. 2017, pp. 4700–4708.

[24] S. Sahin, Y. Becerikli, and S. Yazici, "Neural network implementation in hardware using fpgas," in *Proc. Int. Conf. Neural Inf. Process.* Berlin, Germany: Springer, 2006, pp. 1105–1112.

[25] L. Lu and Y. Liang, "SPWA: An efficient sparse Winograd convolutional neural networks accelerator on FPGAs," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[26] D. Wang, K. Xu, Q. Jia, and S. Ghiasi, "ABM-SpConv: A novel approach to FPGA-based acceleration of convolutional neural network inference," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.
[27] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," 2018, *arXiv:1803.05900*. [Online]. Available: http://arxiv.org/abs/1803.05900
[28] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *Proc. 53nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Feb. 2016, pp. 1–6.
[29] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.
[30] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.
[31] S. I. Venieris and C.-S. Bouganis, "FPGAConvNet: Mapping regular and irregular convolutional neural networks on FPGAs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 326–342, Feb. 2019.
[32] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "CuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*. [Online]. Available: http://arxiv.org/abs/1410.0759
[33] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," in *High-Performance Computing on the Intel Xeon PhiT*. Cham, Switzerland: Springer, 2014, pp. 167–188.
[34] C. Nvidia, *Cublas library*, vol. 15. Santa Clara, CA, USA: NVIDIA Corp., 2008, p. 31.
[35] Z. Xianyi, W. Qian, and Z. Chothia. (2012). *Openblas*. [Online]. Available: http://xianyi.github.io/OpenBLAS
[36] T. Jeong, D. Choi, S. Han, and E.-Y. Chung, "A study of data layout in multi-channel processing-in-memory architecture," in *Proc. 7th Int. Conf. Softw. Comput. Appl.*, Feb. 2018, pp. 134–138.
[37] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011.
[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[39] S. Siamashka. (2017). *Tinymembench: Simple Benchmark for Memory Throughput Latency V0.4.9*. [Online]. Available: https://github.com/ssvb/tinymembench
[40] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.
[41] J. D. Leidel and Y. Chen, "HMC-Sim-2.0: A simulation platform for exploring custom memory cube operations," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 621–630.
[42] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A user-friendly memory simulator to model (Non-)Volatile memory systems," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 140–143, Jul. 2015.



WON JEON received the B.S. and Ph.D. degrees in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2014 and 2021, respectively. He currently works as a Researcher with the AI Processor Research Team, AI SoC Research Department, Electronics and Telecommunications Research Institute. His current research interests include GPU memory systems, processing-in-memory architecture designs, approximate computing for neural network applications, and neural processor architecture designs.

JIWON LEE (Student Member, IEEE) received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2018, where he is currently pursuing the Ph.D. degree with the Embedded Systems and Computer Architecture Laboratory, School of Electrical and Electronic Engineering. His current research interests include virtual memory, GPU memory systems, and heterogeneous computing.

DONGSEOK KANG received the B.S. and M.S. degrees in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2018 and 2021, respectively. He is currently working as an Engineer with the Memory Division, Samsung Electronics. His current research interests include DRAM verification, processing-in-memory, neural processing units, and computer architectures.

HONGJU KAL (Graduate Student Member, IEEE) received the B.S. degree in electronic and IT media engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2018. He is currently pursuing the Ph.D. degree with the Embedded Systems and Computer Architecture Laboratory, School of Electrical and Electronic Engineering, Yonsei University, Seoul. His current research interests include memory architectures, memory hierarchies, near-memory processing, and neural networks.

WON WOO RO (Senior Member, IEEE) received the B.S. degree in electrical engineering from Yonsei University, Seoul, South Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California, in 1999 and 2004, respectively. He worked as a Research Scientist with the Electrical Engineering and Computer Science Department, University of California, Irvine. He currently works as a Professor with the School of Electrical and Electronic Engineering, Yonsei University. Prior to joining Yonsei University, he worked as an Assistant Professor with the Department of Electrical and Computer Engineering, California State University, Northridge. His industry experience includes a college internship with Apple Computer, Inc., and a contract software engineer with ARM, Inc. His current research interests include high-performance microprocessor design, GPU microarchitectures, neural network accelerators, and memory hierarchy design.