Received December 31, 2021, accepted January 17, 2022, date of publication January 26, 2022, date of current version February 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3146413

Spatial Data Dependence Graph Based Pre-RTL Simulator for Convolutional Neural Network Dataflows

JOOHO WANG[1], SUNGKYUNG PARK[2], (Senior Member, IEEE), AND CHESTER SUNGCHUNG PARK[1], (Senior Member, IEEE)

[1]Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, South Korea
[2]Department of Electronics Engineering, Pusan National University, Pusan 46241, South Korea

Corresponding author: Chester Sungchung Park (chester@konkuk.ac.kr)

This work was supported by Samsung Research Funding and Incubation Center of Samsung Electronics under Project SRFC–IT1802–10.

ABSTRACT In this paper, a new pre-RTL simulator is proposed to predict the power, performance, and area of convolutional neural network (CNN) dataflows prior to register-transfer-level (RTL) design. In the simulator, a novel approach is adopted to implement a spatial data dependence graph (SDDG), which enables us to model a specific dataflow alongside inter-instruction dependencies by tracking the status of each processing element (PE). In addition, the proposed pre-RTL simulator makes it possible to evaluate the impact of memory constraints such as latency and bandwidth. The latency-insensitive and bandwidth-insensitive PE controllers assumed in the proposed pre-RTL simulator guarantee both functional correctness and maximum performance, regardless of memory constraints. In particular, it is shown that the optimal distribution method of local memory bandwidth can reduce the accelerator execution time by up to 37.6% compared with the equal distribution method. For weight stationary (WS) and row stationary (RS) dataflows, the accelerator performance closely depends on memory constraints. The simulation results also show that the relative performances of dataflows depend on the layer shape of the convolutional layer. For example, for an identical hardware area in a standard convolutional layer of AlexNet, WS dataflows do not provide any performance gain over RS dataflows when the memory latency is sufficiently high. In addition, WS dataflows cannot fully reuse the input activation, thereby increasing local memory accesses, since the number of weights loaded at a specific time is limited. Moreover, in a depth-wise convolutional layer of MobileNet, WS dataflows tend to outperform RS dataflows even in the presence of large memory latency. The source code is available on the GitHub repository: https://github.com/SDL-KU/SDDGSim.

INDEX TERMS Convolutional neural networks (CNNs), data dependence graph, design space exploration (DSE), hardware accelerators, latency-insensitive controller, pre-RTL simulator, spatial data dependence graph (SDDG).

I. INTRODUCTION

Recently, convolutional neural networks (CNNs) have been widely implemented in a variety of research areas, including image processing, computer vision, and voice recognition [1]–[5]. CNNs are composed of activation functions (e.g., ReLU, sigmoid, or SoftMax) as well as fully connected, convolutional, and pooling layers. More specifically, convolutional layers require tens to hundreds of megabytes of parameters over billions of operations in a single inference process; this requires considerable on-chip and off-chip data movements to support the computation [6], [7]. As shown in [8], [9], the energy consumption of data movement can exceed that of data calculation, and it requires the optimization of hardware accelerator movements to achieve high energy efficiency. When processing these convolutional layers, the use of general-purpose processors (GPPs) and graphic processing units (GPU) is inefficient in terms of the computational speed, and frequent access to off-chip memory involves considerable power consumption [10]–[14]. Recently developed hardware accelerators have been implemented with more than one multiply-and-accumulate (MAC) unit; these

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

parallel operations improve the performance and energy consumption [15]–[17]. However, state-of-the-art studies show that the communication time overhead between the memory and hardware accelerators may affect performance [18]–[21]. Therefore, to overcome the overhead, recent studies have considered several dataflow types for the hardware accelerators [22]–[28]. These studies also reduced the number of accesses to off-chip memory by introducing additional local memories inside the hardware accelerators, as well as additional storage locations (e.g., registers) inside the processing elements (PEs). Meanwhile, recently developed hardware accelerators exhibit different bandwidths according to the number of banks distributed to local memories [29], [30], and different latencies may arise when the implemented local memory blocks are configured to multiple banks or emerging memories [29, [31]. Therefore, to improve the performance, power consumption, and hardware area (PPA) of hardware accelerators, the aforementioned factors (i.e., dataflows, memory latency, and multiple memory banks) must be considered. However, the hardware accelerators that selectively implement these design factors require substantial design time when evaluating the PPAs at the register-transfer-level (RTL) phase. Thus, recently developed pre-RTL simulators may reduce the design time using a framework that estimates the PPA of the hardware accelerators [32]–[38]. However, these pre-RTL simulators only determine the performance and power consumption via the dataflows applied to the hardware accelerator, and it is impossible to estimate the impact of memory constraints (e.g., latency or multiple memory banks).

For the first time in the reported literature, we propose the use of a spatial data dependence graph (SDDG) in a pre-RTL simulator, to model not only inter-instruction dependence but also dataflow-specific operations under memory constraints. More specifically, the generated SDDG contains spatial information, such as which arithmetic/logic unit (ALU) or register is involved by each of the instructions (e.g., which register location the incoming pixel is loaded into). Such spatial information makes it possible to model the dataflow. In addition, the use of an SDDG facilitates the modeling of memory constraints such as memory latency and bandwidth. The latency-insensitive approach in [39] is applied to each PE controller, to ensure operational correctness and maximal performance, regardless of memory constraints. The PPA estimation of the proposed pre-RTL simulator is shown to be consistent with the previously published implementation results [24] and [28], with errors of less than 7.1% and 7.3%, respectively. In addition, design space exploration (DSE) using the proposed pre-RTL simulator can provide useful insights into PPA improvement in the early design phases. For the well-known weight stationary (WS) and row stationary (RS) dataflows, the simulation results show a close dependence of accelerator performance on memory constraints. In particular, it is shown that the optimal distribution of memory bandwidth can reduce the accelerator execution time under RS dataflows by up to 37.6%. In addition, the two dataflows are compared in terms of performance, power consumption, and area for a constant PE number or hardware area. The simulation results show that the superiority of a dataflow depends on the layer shape of the convolutional layer. For example, for the same hardware area, RS dataflows tend to be less sensitive to memory constraints than WS dataflows since the RS dataflows reuse registers more aggressively and therefore require fewer memory accesses. In addition, it is shown that RS dataflows have a significant power-saving effect because they involve fewer memory accesses. This implies that the proposed pre-RTL simulator can be used to optimize dataflows and memory constraints according to the layer shape.

The novelty of this study can be summarized as follows:

- The proposed pre-RTL simulator makes it possible to evaluate the impact of the layer shape (tile size) and memory constraints, such as the memory bandwidth, as opposed to the conventional CNN hardware accelerator simulators [35]–[38]. The latency- and bandwidth-insensitive PE controllers assumed in the proposed pre-RTL simulator guarantee the functional correctness and maximum performance, regardless of memory constraints.
- The proposed pre-RTL simulator makes it possible to evaluate the bank distribution for the local memory for each data type is varied according to the tile size and computation schedule in order to reduce the execution time of the CNN hardware accelerator. The proposed pre-RTL simulator may provide a local memory bank allocation method for each data type that can effectively reduce the execution time when hardware accelerators utilize a limited number of local memory banks.

The remainder of this paper is organized as follows. Section II describes the related work. Section III introduces CNN basics and describes the hardware accelerator structure used in the proposed pre-RTL simulator; in addition, we introduce the RS and WS dataflows and describe the framework of the dynamic data dependence graph (DDDG) simulator in Aladdin [32]. Section IV introduces the latency- and bandwidth-insensitive controller [40]. Section V describes the characteristic framework of the proposed pre-RTL simulator in this study, and contrasts it with Aladdin. Section VI verifies the estimated results obtained for the SDDG simulator, using the previously implemented hardware accelerator results. In addition, we estimate the PPA results for the hardware accelerators with respect to the dataflows and local memory constraints, using the DSE results of the proposed pre-RTL simulator; in particular, this section presents the extended simulation results given in previous studies [41]. Finally, Section VII describes the conclusions and future work.

II. RELATED WORK

Several studies have used pre-RTL simulators to estimate the PPA of hardware accelerators. For example, Aladdin [32] is a pre-RTL simulator for PPA estimation frameworks that targets the rapid prototyping of data-parallel hardware accelerators. Aladdin uses high-level descriptions (i.e., C program) of algorithms as inputs and applies DDDGs to represent the hardware accelerator without having to generate RTL designs. Aladdin starts with a DDDG without constraints, which corresponds to a primary representation of a hardware accelerator; one framework of Aladdin applies optimizations as well as constraints to the graph, to create a realistic model of the hardware accelerator. ESP [33] introduces a research platform for hardware accelerators into a complete system-on-a-chip (SoC) architecture. The platform combines a modular tile-based architecture (e.g., a memory tile) with an accelerator tile. In particular, ESP provides accelerator design flows that include automated steps, using high-level synthesis (HLS) tools such as Xilinx Vivado HLS and Catapult HLS. The generated accelerator tile can be combined with other tiles to estimate the performance of the SoC architecture. In [34], a platform for accelerator-rich architectural design and exploration (PARADE) was proposed to model full-system SoC architectures via a cycle-accurate approach. This approach ensures that the whole accelerator-rich architecture (ARA) system is accurately modeled, including processor cores, specified hardware accelerators, shared memory and network-on-a-chip (NoC). PARADE uses a fully-automated process to generate the specified hardware accelerator modules, by implementing HLS tools such as AutoPilot HLS. In addition, PARADE can facilitate DSE in the ARA system. However, none of these pre-RTL simulator models provides spatial information for a hardware accelerator; thus, they cannot be used to model a specific dataflow. Recalling that most state-of-the-art hardware accelerators for CNNs are based on a dataflow-specific spatial architecture [22]–[28], the aforementioned pre-RTL simulators are unsuitable for estimating the PPA of CNN hardware accelerators.

Several existing pre-RTL simulators are capable of modeling a dataflow-specific architecture (e.g., [35]–[38]). These pre-RTL simulators contain spatial information, such as which ALU/register each of the instructions invokes (i.e., which register location the incoming pixel is loaded into). In most pre-RTL simulators, a set of pragmas/directives (e.g., for loop unrolling or tiling) is used to specify a dataflow. For example, Interstellar [35] presents a systematic approach to concisely describe the design space of a hardware accelerator for a CNN, using schedules of loop transformations. In addition, this suggests that dataflow mappings for existing hardware accelerators in CNNs can be represented as a schedule in a Halide program, and it extends the Halide schedule language to generate different hardware designs in the space of CNN hardware accelerators. In [36], MAESTRO uses a set of data-centric directives (e.g., TemporalMap or SpatialMap) to concisely specify the CNN dataflow space in a compiler-friendly form. In addition, it describes how these directives can be analyzed to infer diverse forms of reuse, as well as how to exploit these using hardware accelerator capabilities. Finally, it codifies this analysis into an analytic cost model referred to as ''modeling accelerator efficiency via spatio-temporal reuse and occupancy (MAESTRO),'' which estimates many dataflow-specific cost-benefit tradeoffs, including the performance and power consumption for a deep neural network (DNN) model and hardware accelerator. Timeloop [37] provides a concise and unified method of describing the key attributes of a diverse class of DNN architectures (and their implementation features) as the input to an analytical model. Moreover, infrastructure of Timeloop [37] effectively combines the exploration of a large design space with a mapper that identifies the optimal mapping for the dataflow of any workload on the targeted hardware accelerator. [38] introduces a systolic array CNN accelerator simulator (SCALE-Sim), a configurable systolic array-based cycle-accurate hardware accelerator for CNNs. The proposed simulator offers diverse micro-architectural characteristics (e.g., mapping strategy) and system integration parameters (e.g., bandwidth requirement) to the designer, to facilitate comprehensive design space exploration (DSE). However, these pre-RTL simulators lack the modeling details (e.g., memory constraints such as memory latency and bandwidth).

To improve energy efficiency, most CNN hardware accelerators exploit a memory hierarchy [42]–[44]; this typically consists of PE registers, first-in-first-outs (FIFOs), local memories and off-chip memories. Different memory blocks in the memory hierarchy tend to exhibit order-of-magnitude difference in their latencies and bandwidths; thus, each PE tends to experience similar differences, depending on which memory block (e.g., local memories) the pixel is transferred from/to. Several reports analyze the impact of local memory constraints on the performances of hardware accelerators. In Caffeine [19], it is mentioned that the local memory banks should be carefully designed to process on-chip data. In addition, Caffeine interleaves the data for different local memory banks, to reduce bank read/write conflicts. In order to build on-chip caches with large capacities, the authors [29], [30] proposed to split large block random access memory (BRAM) structures into multiple banks, offering additional latency for the maintenance of high operating frequencies. In [31], it is mentioned that the absolute latency of different memory technologies (i.e., emerging memory) is experienced during data fetching to the hardware accelerators. In addition, they presents a DSE flow for benchmark hardware accelerators with incumbent and emerging memories, emphasizing practical technological characteristics. Thus, the performance and power consumption of hardware accelerators (taking into account dataflows and memory constraints) need to be estimated in the pre-design phase.

The contributions of this study can be summarized as follows:

- The proposed pre-RTL simulator can estimate the PPA of a spatial hardware accelerator more accurately than the Aladdin pre-RTL simulator [32]. The proposed pre-RTL simulator takes into account both dataflows (e.g., RS or WS) and memory constraints (latency and bandwidth) using spatial information, to evaluate the spatial characteristics of the hardware accelerator, including the register location where the incoming pixel is loaded to. This spatial information makes it possible to model dataflows. The experimental results for the proposed pre-RTL simulator are similar to the power consumption and performance measured in the conventional hardware accelerator [24], [28]. The normalized energy and execution time of the Eyeriss hardware architecture [24] modeled using the proposed SDDG pre-RTL simulator can be predicted with errors of less than 7.1% and 6.3%, respectively, compared with the actual measured results of the Eyeriss hardware architecture. In addition, the normalized energy and execution time of the deep-learning specific instruction-set processor (DSIP) hardware architecture [28] modeled with the proposed SDDG pre-RTL simulator shows estimation errors of up to 7.8% and 5.6%, respectively, compared with the actual measurement results of the DSIP hardware architecture.

- Compared with the Aladdin pre-RTL simulator approach, the proposed pre-RTL simulator makes it possible to explore the design space (e.g., the optimal memory interface) in the early design phases, without RTL implementation. The latency- and bandwidth-insensitive controllers assumed in the proposed pre-RTL simulator ensure functional correctness and maximal performance, regardless of memory constraints. As indicated in Section VI, the simulation results show a close dependence of hardware accelerator performance on memory constraints (latency and bandwidth). For example, given a total bandwidth of six memory banks, the optimal distribution of these memory banks can reduce the execution time of the hardware accelerator by up to 37.6% compared with the equal distribution method [29], [30]. Assuming the use of an identical hardware area in a standard convolutional layer of AlexNet, WS dataflows do not offer any performance advantage over RS dataflows when the memory latency is sufficiently high. In a depth-wise convolutional layer of MobileNet, the performances of WS dataflows exceed those of RS dataflows, even in the case of high-memory latency. Thus, it can be concluded that the proposed pre-RTL simulator can explore the performance of the hardware accelerator under memory constraints (latency and bandwidth).

III. BACKGROUND

This section describes the operations of convolutional layers (which are computationally sensitive and characterized by repetitive patterns) among the different layers required by CNNs to perform classification. In addition, we briefly illustrate the structure of the hardware accelerator in the proposed pre-RTL simulator. Finally, we introduce the two dataflows considered in the proposed pre-RTL simulator.

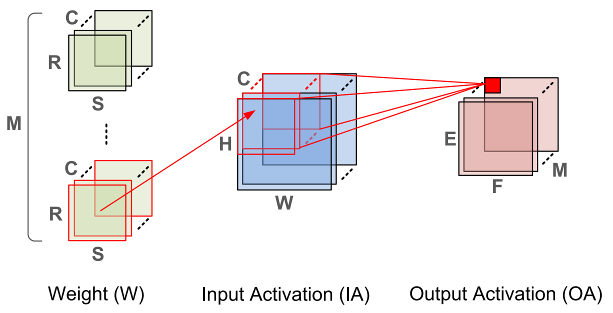


FIGURE 1. Structure of a convolutional layer.

TABLE 1. Layer shape parameters of a convolutional layer.

Params.	Definition
H/W	Height and width of input activation
R/S	Height and width of weight
E/F	Height and width of output activation
M	Number of output channels
C	Number of input channels

A. BASICS OF CONVOLUTIONAL LAYERS

Convolutional layers perform the main operations of CNNs, and they involve a number of MACs. Figure 1 and Table 1 show the parameters that represent the convolutional layers. First, input activations (IAs) are expressed by height (H), width (W) and number of input channels (C). Next, weights are expressed by height (R), width (S), number of input channels (C) and number of output channels (M). Finally, output activations (OAs) are expressed by height (E), width (F) and number of output channels (M). Algorithm 1 illustrates the pseudocode of a convolutional layer. The loop order of Algorithm 1 is arbitrarily arranged; if it is changed, the same value is generated.

B. TILE PARAMETERS OF CNN ACCELERATORS

The hardware accelerator loads activations and filters from the off-chip memory to each local memory block. A PE array executes MACs through the activations and filters loaded to local memory blocks. Partial sums are returned to the local memory blocks or off-chip dynamic random access memory (DRAM). This procedure is defined as a processing pass in Eyeriss [24]. A single convolutional layer is divided into multiple processing passes through the tile parameters described in Table 2. The tile parameters determine the amount of computation and communication for each processing pass. To describe this in further detail, the values of p and q represent the number of output and input channels handled within a single processing pass, respectively. In addition, r and t indicate that the PE array is divided into multiple PE sets; that is, the PE array is a parallel configuration of r by t PE sets that simultaneously run r different input channels for t different output channels.

Figure 2 depicts the processing passes of a convolutional layer with 12 input channels $(C = 12)$ and 8 output channels $(M = 8)$. The tile parameters are set as $p = 2$, $q = 1$, $r = 3$ and $t = 2$. As shown in the figure, a convolutional layer

TABLE 2. Tile parameters of a CNN hardware accelerator.

Params.	Definition
p	Number of output channels processed by a PE set
q	Number of input channels processed by a PE set
r	Number of PE sets that process different input channels in the PE array
t	Number of PE sets that process different output channels in the PE array

Pass		0	1	2	3	4	5	6	7
IA	C	0-2	0-2	3-5	3-5	6-8	6-8	9-11	9-11
W	C	0-2	0-2	3-5	3-5	6-8	6-8	9-11	9-11
	M	0-3	4-7	0-3	4-7	0-3	4-7	0-3	4-7
OA	M	0-3	4-7	0-3	4-7	0-3	4-7	0-3	4-7

FIGURE 2. Processing passes of a convolutional layer ($C = 12$, $M = 8$, $p = 2$, $q = 1$, $r = 3$, and $t = 2$).

consists of a total of eight processing passes. Figure 3 is a block diagram depicting a PE array for r is 3 and t is = 2. Because $r \times t$ is 6, the PE array consists of six PE sets in the hardware accelerator. The hardware accelerator processes different output channels (M) in a parallel manner according to parameter t. Additionally, the hardware accelerator simultaneously processes different input channels (C) using parameter r. The OAs with different input channels (r) processed during one processing pass are accumulated in the same output channel (t). As shown in Figure 3, OAs with the same t index are stored in a local memory according to the corresponding t index.

Algorithm 1 Pseudocode of a Convolutional Layer

```
1: for (m = 0; m < M; m++) {
2:   for (c = 0; c < C; c++) {
3:     for (e = 0; e < E; e++) {
4:       for (f = 0; f < F; f++) {
5:         for (r = 0; r < R; r++) {
6:           for (s = 0; s < S; s++) {
7:             OA[m][e][f] + = IA[c][e+r][f+s] ∗
                   W[m][c][r][s];
8: }}}}}}
9: post_processing(OA[0:M-1][0:E-1][0:F-1])
```

C. SYSTEM UNDER CONSIDERATION

Figure 3 illustrates the overall system using the CNN hardware accelerator described in [45], which consists of a hardware accelerator, processor core, DRAM controller and on-chip buses. The processor core is responsible for synchronizing the direct memory access controllers (DMACs) and configuring them by setting the transfer addresses and sizes. The hardware accelerator is assumed to be equipped with multiple direct memory access controllers (DMACs), as shown in [6]–[8]. Each of the DMACs serves as a bus master and accesses the DRAM subsystem through the on-chip bus. The hardware accelerator is assumed to be equipped

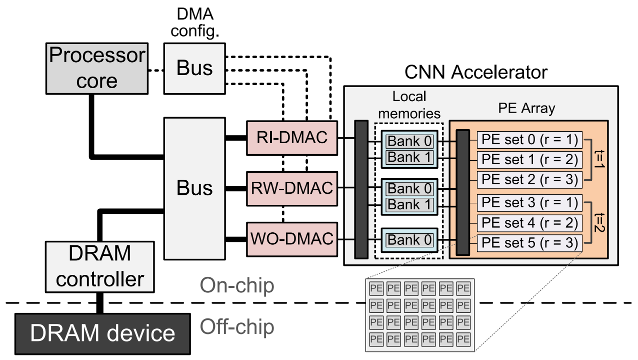


FIGURE 3. Overall system with a CNN hardware accelerator.

with either one (e.g., [46]–[48]) or multiple (e.g., [19], [26] and [49], [50]) DMACs and each of the DMACs accesses the DRAM subsystem as a bus master. For example, Figure 2 shows a CNN accelerator that consists of three DMACs, wherein each of the DMACs is dedicated to one of the three data types of the CNN accelerator: IAs, filters (W), or OAs. A read DMAC for the IA (RI-DMAC), a read DMAC for filter weights (RW-DMAC) and a write DMAC for the OA (WO-DMAC) are used. In addition, the processor core is responsible for synchronizing the DMACs in the hardware accelerator (i.e., for the starting and stopping of the DMAC execution [51]–[53]). Finally, the processor core makes it possible to reconfigure the hardware accelerator according to the number of DMACs. Given that a hardware accelerator is typically designed as a standalone IP block, a standardized interface may ease integration in the system (e.g., [19], [26] and [46]–[48]). The DRAM controller assists the hardware accelerator with its access to off-chip DRAM. The DMACs are assumed to access several local memory blocks through the NoC inside the hardware accelerator. The figure shows the three local memory blocks included in the hardware accelerator, which implies that each data type of the CNN is distributed. The figure shows that the number of ports (i.e., number of banks) distributed to each local memory block differs. Therefore, the communication bandwidth of each local memory that communicates with the PE array may differ [29], [30]. In addition, each local memory block can be implemented through one of several emerging memory technologies [31]. This characteristic can be manifested differently as latency when the PE array accesses each local memory block.

The proposed pre-RTL simulator in this study focuses on communication between the PE array and local memory blocks inside the hardware accelerator depicted in Figure 3. In addition, it processes spatial information inside the PE array. Finally, the pre-RTL simulator estimates the PPA of the hardware accelerator by analyzing spatial information and the impact of local memory constraints.

D. MODELING OF DATAFLOWS

Several dataflows have been proposed to efficiently compute a single convolutional layer of a CNN [22]–[28]. This

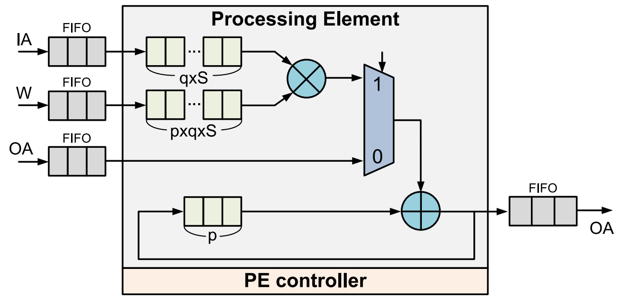


FIGURE 4. Block diagram of a single PE for RS dataflow.

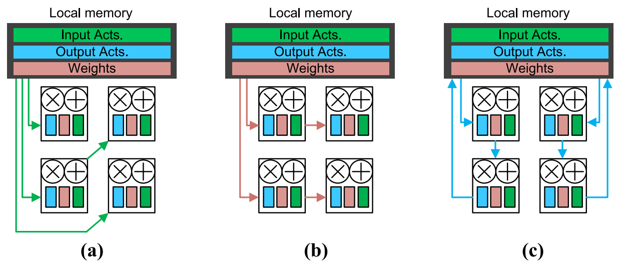


FIGURE 5. Communication manner for each data type in RS dataflow: (a) input activations are reused across PEs diagonally, (b) weights are reused across PEs horizontally, (c) output activations are accumulated across PEs vertically.

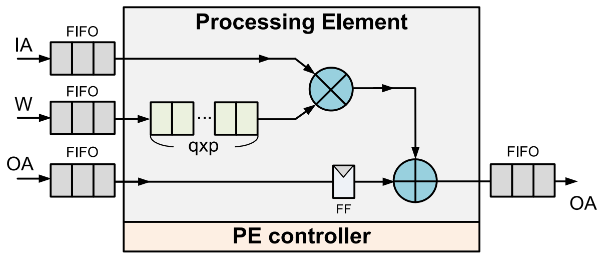


FIGURE 6. Block diagram of a single PE for WS dataflow.

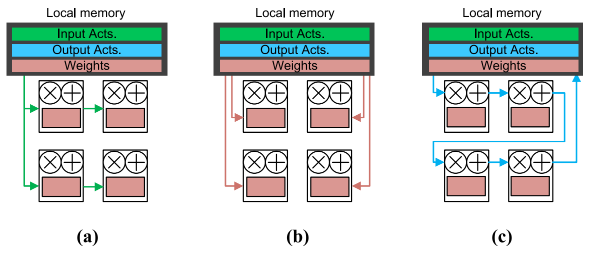


FIGURE 7. Communication manner for each data type in WS dataflow: (a) input activations are reused across all PEs, (b) weights are unicasted to each PE, and (c) output activations are relayed after the MAC operation on each PE.

subsection describes the hardware accelerator structure of the RS [24] and WS [28] dataflows considered in the proposed pre-RTL simulator, as well as the communication procedure for each data type.

Figure 4 shows the block diagram of a single PE in the RS dataflow [24]. More specifically, the PE assumed in RS dataflow stores the IAs, weights and OAs in each register inside the single PE, and it reuses those activations and filters. In addition, in [24], different communication techniques are

```
for (e = 0; e < E; e++) {
 for (f = 0; f < F; f++) {
  for (r = 0; r < R; r++) {
   for (s = 0; s < S; s++) {
     OA(F×e+f) += IA(H×(e+r)+f+s) × W(S×r+s);
}}}}
```

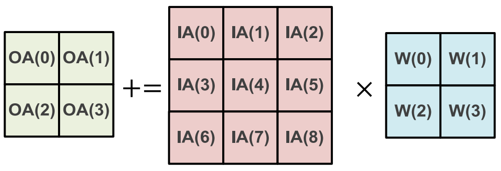


FIGURE 8. Example of a 2-dimensional convolutional layer.

applied to access PEs for each data type from the local memory. For example, in Figure 5 (a), the IAs are multicast to PEs that are arranged diagonally in a PE set. Next, in Figure 5 (b), the weights are multicast to PEs arranged horizontally in a PE set. Lastly, in Figure 5 (c), the OA is sequentially unicast to a PE in the first row of a PE set. Additionally, each PE relays the partial sum (psum) of all the MACs of the weight row for q input channels to the adjacent PE in the next row. OAs computed from the PEs in the last row of a PE set are stored in the off-chip DRAM or local memory blocks.

Figure 6 shows the block diagram of a single PE in the WS dataflow [28]; it stores the weights in a register inside the single PE. In addition, [28] presents different communication techniques for accessing PEs of each data type from the local memory. First, Figure 7 (a) shows that the IAs are reused across all PEs. Next, Figure 7 (b) indicates that the weights are unicast to each of the designated PEs in a PE set and stored in the register. In addition, Figure 7 (c) shows that the OAs are relayed following the MAC operation on each PE. OAs that have been integrated through relays between PEs are stored in off-chip DRAM or local memory blocks.

Before leaving this subsection, it should be mentioned that the proposed pre-RTL simulator in this study is modeled for dataflows via the controller of each PE [40]. The proposed approach not only models the dataflows of a PE array in the conventional pre-RTL simulators [35]–[38]: it is also latency- and bandwidth-insensitive to local memory blocks. Moreover, the latency- and bandwidth-insensitive approach [39] motivates this study because it can be easily extended by combining the DSE of the dataflows under memory constraints. For example, estimation results in the proposed pre-RTL simulator show that the hardware accelerator's performance is affected by local memory constraints related to the dataflow computation schedule (as will be shown in Section VI).

E. DYNAMIC DATA DEPENDENCE GRAPH

In this subsection, we describe Aladdin [32], a pre-RTL simulator that forms the basis of the proposed pre-RTL simulator. The dynamic data dependence graph (DDDG) proposed in Aladdin is a directed, acyclic graph in which nodes represent computations and edges represent dynamic data dependencies between nodes. Graph-based representations are widely applied in the early design stages to model the behavior of hardware [54], [55]; thus, the generated graph provides information regarding the hardware operations required for implementation, expressed in a cycle-accurate manner. Figure 8 shows the processing procedure for simple two-dimensional convolutional layers (H = W = 3, E = F = 2 and R = S = 2), where OA, IA and W denote an output activation, an input activation and a filter weights, respectively. As shown in the figure, we must multiply and accumulate the IAs to obtain OAs, because the filter weights comprises four weights. The accumulation process is repeated when W moves from left to right with a stride size of one. According to the pseudocode of Figure 8, the OAs are calculated as

$$OA(0) = IA(0)W(0) + IA(1)W(1) + IA(3)W(2) + IA(4)W(3) \quad (1)$$

$$OA(1) = IA(1)W(0) + IA(2)W(1) + IA(4)W(2) + IA(5)W(3) \quad (2)$$

$$OA(2) = IA(3)W(0) + IA(4)W(1) + IA(6)W(2) + IA(7)W(3) \quad (3)$$

$$OA(3) = IA(4)W(0) + IA(5)W(1) + IA(7)W(2) + IA(8)W(3) \quad (4)$$

In this section, the two-dimensional convolutional layer shown in Figure 8 is used as an example (input) of an Aladdin pre-RTL simulator [32] and the proposed SDDG pre-RTL simulator.

Figure 9 illustrates the hardware accelerator assumed in the Aladdin pre-RTL simulator [32]. The hardware accelerator in the figure exhibits a structure in which six MACs are available, and the number of local memory block banks for the IAs, weights, and OAs are 3, 2, and 3, respectively. The communication bandwidth is determined by the number of banks distributed to each local memory block. For example, the number of local memory block banks for an IA value of 3 indicates that three IAs can be loaded from the local memory block simultaneously. Furthermore, six MACs are simultaneously available in the hardware accelerator. Finally, a shared register reduces the number of accesses to the local memory blocks.

Figure 10 shows the Aladdin framework [32]. First, the intermediate representation (IR) phase of the framework translates the workload of the hardware accelerator (written in a C program) into the IR level. The IR converted in this phase contains workloads for memory accesses (e.g., load and store) and computations (e.g., MACs).

Subsequently, in the optimization phase of the framework, the IR generated in the previous phase is converted to a DDDG. Figure 11 shows an idealized DDDG generated by the converted IR. As shown in the figure, the DDDG generated in this phase only takes into account fundamental dependencies. For example, the figure shows the characteristics of the memory access dependence that loads/stores each IA, OA, and W through the generated memory address. In addition, it shows the computation dependencies of MACs with data types loaded from the local memory blocks. Finally, the store-load forwarding dependence is characterized, to remove unnecessary local memory block access from the operation

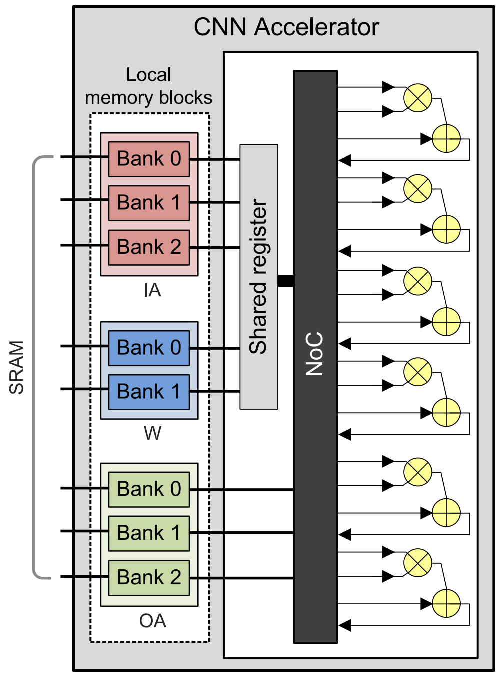


FIGURE 9. Hardware accelerator assumed in Aladdin pre-RTL simulator [32].

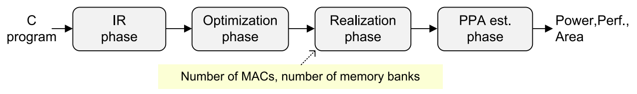
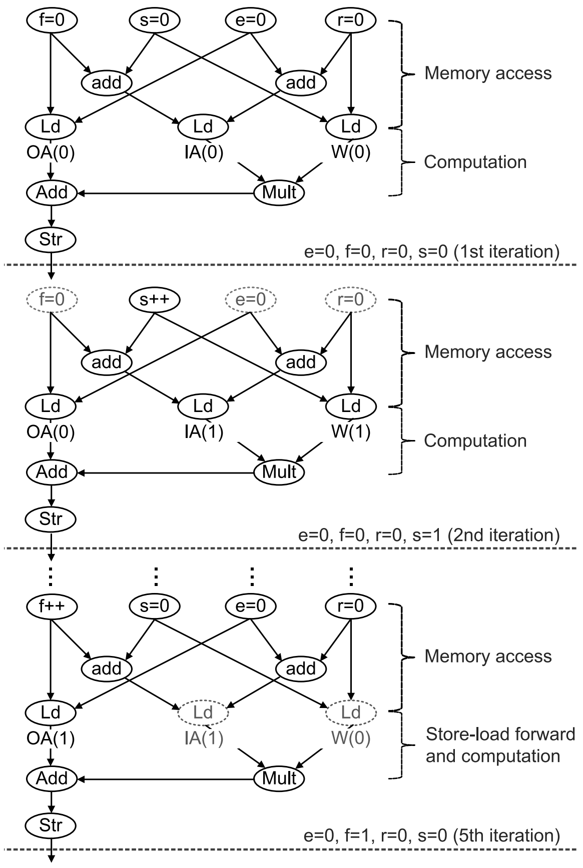


FIGURE 10. Framework of Aladdin pre-RTL simulator [32].

with the shared register, as depicted in Figure 9. For example, it is assumed that IA(1), which is reused in the fifth loop iteration in Figure 11, is stored as a shared register in the memory access (i.e., load) operation of the first loop iteration. However, in the case of OAs, redundant load and store operations are required in all MACs because of the C program assumed in Figure 8. Aladdin defines the DDDG generated through this phase as an idealized DDDG.

In the realization phase, resource constraints with directives (e.g., unroll) are used to constrain the idealized DDDG generated in the optimization phase. The resource constraints are determined by the directives described, which determine the number of MAC units in the hardware accelerator and the number of banks distributed to the local memory block for each data type. More specifically, the number of MAC units determines how many MAC operations can be simultaneously performed, and the number of memory banks determines the number of activations or weights that can be simultaneously accessed. Figure 12 shows a constrained DDDG under hardware resource constraints. For example, the figure shows that the number of local memory banks for the IA, W, and OA are 1, 1, and 4, respectively, through a constrained DDDG. In addition, two MAC units are allocated to the hardware accelerator. This data dependence graph is generated by writing unroll directives in the outermost two



Memory access
Computation
e=0, f=0, r=0, s=0 (1st iteration)
e=0, f=0, r=0, s=1 (2nd iteration)
Store-load forward and computation
e=0, f=1, r=0, s=0 (5th iteration)

FIGURE 11. Idealized DDDG in optimization phase.

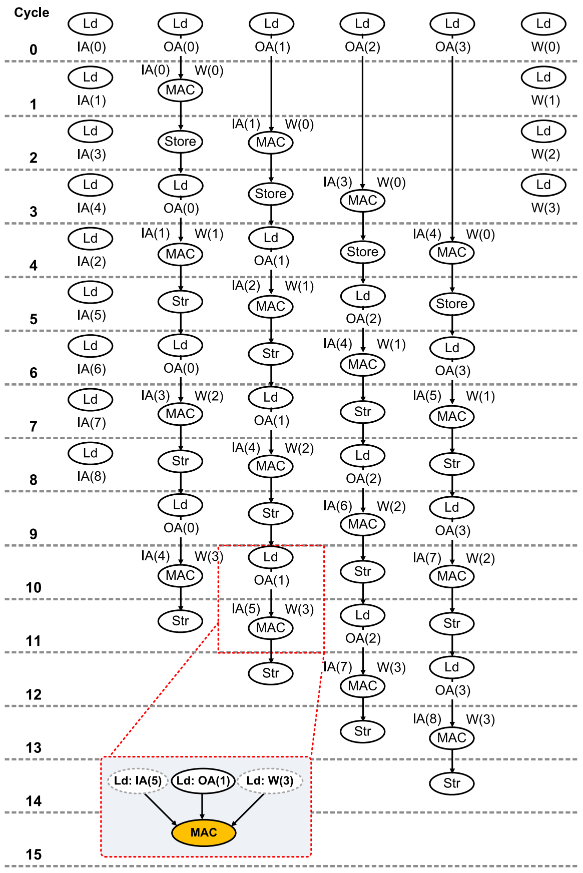


FIGURE 12. Constrained DDDG in realization phase.

loops of the algorithm in Figure 8. In addition, the constrained DDDG shows that although two MAC units are assumed in the resource constraints, they are rarely activated simultaneously, because this does not satisfy the data dependence. For example, only a single MAC from the two MACs in cycle 1 operates, because IA(0), OA(0), and W(0) are insufficient in terms of data dependence to perform other MACs. The pre-RTL simulator proposed in [32] focuses on modeling the computations and memory accesses of the hardware accelerator in a cycle-accurate manner according to the given hardware resource constraints. However, this pre-RTL simulator does not take into account the spatial information of hardware accelerators using dataflows.

Finally, the PPA estimation phase of the framework calculates the performance and power consumption using the activity of each cycle of the generated functional unit and memory unit of the DDDG. To model the power consumption of hardware accelerators, the Aladdin pre-RTL simulator must capture the operation nodes of each accelerator per cycle-level resource activity. More specifically, given the DDDG in Figure 12, it is necessary to know which resources (e.g., multipliers, adders, or memory) are either activated or deactivated in each cycle. For example, the constrained DDDG in Figure 12 shows that the local memory resources are activated in cycle 0. In this case, the power consumption in cycle 0 can be calculated using the weighted sum of the premeasured switching power and the internal power of the memory resources. In addition, in cycle 0, the power consumption of the MAC resources is determined as the weighted sum of the premeasured leakage power. As a result, power consumption can be calculated by activated or deactivated resources at each DDDG cycle.

IV. LATENCY- AND BANDWIDTH-INSENSITIVE PE CONTROLLER

Before explaining the framework of the proposed pre-RTL simulator, a latency- and bandwidth-insensitive controller [40] is explained in this section. As shown in Figure 13, the assumed controller of a PE provides multiple control signals (summarized in Table 3) for datapath consisting of multipliers, adders, and storage elements (registers and FIFOs). Depending on these control signals, a PE may perform different operations. For example, the PE takes three pixels from the registers (pointed by *PTR_R_I*, *PTR_R_F*, and *PTR_R_O*), performs a MAC, and then stores the result to the register (pointed by *PTR_W_O*) when the status of *EN_MAC* and *WE_REG_O* are 1.

In order to guarantee the operation correctness regardless of memory latency and bandwidth, the assumed controller of each PE should satisfy all the inter-operation dependence constraints. For example, each MAC should be preceded by its relevant loads (i.e., MAC after-load dependence).

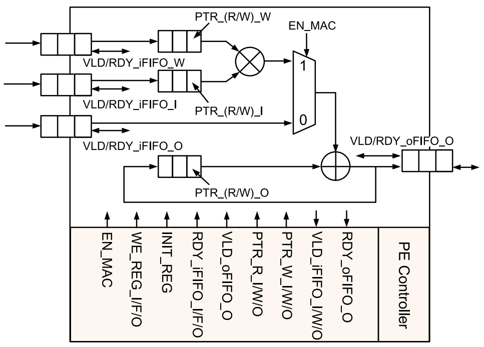


FIGURE 13. Latency - and bandwidth-insensitive PE controller assumed in the proposed pre-RTL simulator.

In addition, in order to maximize the performance for the given memory latency and bandwidth, the assumed controller of PE should schedule each operation as early as possible. To maximize the performance in latency-insensitive and bandwidth-insensitive, the PE controller needs to keep track of the status by updating the number of loads, stores, and MACs to the internal counters, which are described in Table 3 and Figure 13. Each layer consists of multiple processing passes as shown in Eyeriss [24]. Each processing pass is divided into multiple row-MACs. A row-MAC is defined as a set of MACs corresponding to the convolution of an input row with a filter row. Following the notation of the layer shape parameters (R, S, H, W, E, F) and architectural parameters (p, q, r, t) assumed in [24], each processing pass is divided into F row-MACs, each of them consisted of a total of pqS MACs, as shown in Figure 14. Within a row-MAC, the same loop ordering is assumed in [24], i.e., the output channel (p), the input channel (q), and the filter column (S). Figure 14 also illustrates how the counters are updated within a processing pass. For simplicity, the memory bandwidth of the IA, W, and OA is determined to be 3, 2, and 2, respectively. It is clearly shown that the counters progress independently of one another. For example, in the 1st row-MAC, the counter for the input activation load (*CNT_iFIFO_I*) reaches beyond the minimum achievable level (qS), whereas the counter for the filter load (*CNT_iFIFO_F*) gets stuck to the minimum required level (pqS). In addition, the figure also shows how the counters affect some of the control signals. For example, the MAC enable (*EN_MAC*) is set to one only if the minimum between *pCNT_iFIFO_I* and *CNT_iFIFO_F* is larger than *CNT_MAC*, i.e., there are some MACs remaining to be completed for the given input and filter pixels which are loaded into the registers. Finally, by comparing the values of *CNT_MAC* and *CNT_iFIFO_O*, it is determined whether to store the output activation to the outside of the PE (*VLD_oFIFO*). When *CNT_oFIFO_O* reached p, the next row-MAC could be started. The counters described in Table 3 are updated by the ready/valid handshaking protocol of FIFO and register with latency-insensitive and bandwidth-insensitive manners. The PE internal registers being assigned to each data type. When the required value is reached, operations (e.g., MAC, local communication) are performed.

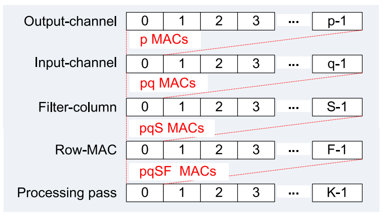
TABLE 3. Control signals of the PE controller.

Signals	Definition	I/O
VLD_oFIFO_O	Valid of output FIFO for OA	O
RDY_oFIFO_O	Ready for output FIFO for OA	I
VLD_iFIFO_I/W/O	Valid of input FIFO for IA/W/OA	I
RDY_iFIFO_I/W/O	Ready of input FIFO for IA/W/OA	O
PTR_R/W_I/W/O	Register read/write pointer for IA/W/OA	O
WE_REG_I/W/O	Write enable of IA/W/OA	O
INIT_REG	Register initialization	O
EN_MAC	MAC enable	O
CNT_iFIFO_I/W/O	Number of loads IA/W/OA per processing pass (from input FIFO to register)	-
CNT_oFIFO_O	Number of stores OA per processing pass (from register to input FIFO)	-
CNT MAC	Number of MACs per processing pass	-

In summary, the proposed pre-RTL simulator utilizes an SDDG, which enables us to model a specific dataflow and the inter-instruction dependence by keeping track of the status of each PE. In addition, the proposed pre-RTL simulator facilitates the evaluation of the impact of memory constraints, such as memory latency and bandwidth. The latency-insensitive and bandwidth-insensitive PE controllers [40], assumed in the proposed SDDG pre-RTL simulator, guarantee the operation correctness and maximum performance, regardless of memory constraints. In other words, the approach proposed in [39] can be applied to the pre-RTL simulator to estimate the performance and power consumption for dataflows and memory constraints of the hardware accelerator.

V. SPATIAL DATA DEPENDENCE GRAPH

Figure 15 exemplifies the CNN accelerator assumed in the proposed pre-RTL simulator. As shown in the figure, the CNN accelerator contains multiple PEs, each of which consists of a single MAC unit and a set of registers. In addition, the CNN accelerator is assumed to exploit a dedicated local memory, as depicted in [42]–[44]. This paper assumes the use of a heterogeneous memory hierarchy in which different memory blocks are based on different memory technologies such as static RAM (SRAM), magnetic RAM (MRAM) [56], or resistive RAM (ReRAM) [57]. Therefore, the latency of local memory varies significantly between memory blocks. For example, the latency of MRAM tends to be substantially larger than that of SRAM latency [58]. Moreover, the bandwidth of local memory also varies between memory blocks, depending on the number of banks allocated to those blocks (e.g., [19] and [59], [60]). Thus, each PE tends to experience an order-of-magnitude difference in its latency and bandwidth, depending on which memory block the activations (or filter weights) are transferred from/to. We denote the memory latencies [cycles/pixel] and bandwidths [ports/pixel or ports/weight] for the IA, weigths, and OA by Li, Lw, and Lo and Bi, Bw, and Bo, respectively. The DSE results in Section VI.B compare the execution times of the hardware accelerator under memory constraints. For example, the DSE results show a comparison of the execution time for the hardware accelerator implemented using SRAM with a latency of (1, 1, 1) or MRAM with a latency of (3, 3, 3) for each local memory. In addition, it is possible to examine



	Row-MAC 0											Row-MAC 1										Row-MAC 2									
CNT_iFIFO_I	0	1	2	3	4	5	6	6	6	6	6	4	4	4	4	5	5	6	6	6	6	4	4	4	4	4	4	4	4	4	4
CNT_iFIFO_F	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
CNT_iFIFO_O	0	0	0	1	1	1	1	1	1	1	1	0	1	2	2	2	2	2	2	2	2	0	1	1	1	1	1	1	2	2	2
CNT_oFIFO_O	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
CNT_MAC	0	0	1	1	2	3	4	5	6	7	8	0	0	0	1	2	3	4	5	6	7	0	0	1	2	3	4	4	5	6	7
VLD_oFIFO_O	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1
EN_MAC	0	1	0	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1

Name	Definition
CNT_iFIFO_I/F/O	Number of loads of input activation/filter/ output activation from input FIFO per row MAC.
CNT_oFIFO_O	Number of stores output activation to output FIFO per row MAC.
CNT_MAC	Number of MACs per row MAC.

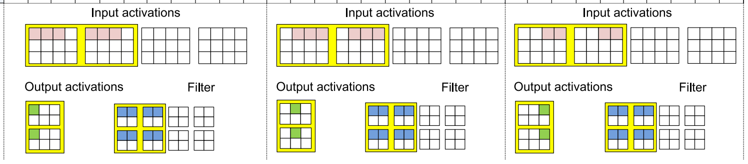


FIGURE 14. Latency- and bandwidth-insensitive PE controller ($p = q = 2$, $R = S = 2$, $E = 2$, $F = 3$).

the differences in execution time according to the bandwidth, by modifying the number of memory banks. Therefore, the latency and the bandwidth (which are assumed as the memory constraints) indicate the characteristics of an emerging memory. Figure 15 illustrates the memory latency of (2, 3, 1) and the bandwidth of (3, 2, 3). The latency- and bandwidth-insensitive PE controllers [40] assumed in the proposed pre-RTL simulator ensure functional correctness, regardless of memory constraints; meanwhile, they also maximize the performance. Lastly, the internal NoC connects multiple PEs with the local memory dedicated to the CNN accelerator. The detailed topology and protocol depend on the dataflows of the CNN accelerator [22]–[28].

The proposed pre-RTL simulator consists of three major steps derived from the Aladdin pre-RTL simulator [32], as shown in Figure 16. The figure shows that the proposed pre-RTL simulator utilizes the frontend (IR phase) and backend (PPA estimation phase) of the Aladdin pre-RTL simulator [32], and it adds a new spatial architecture phase that consists of the dataflow and memory phases. Given the intermediate representation from the Aladdin pre-RTL simulator, the spatial architecture phase generates the corresponding SDDG, taking into account the dataflow and memory constraints (i.e., bandwidth and latency). The dataflow phase tracks the statuses of ALUs by registering and modeling the behavior of the controllers. The memory phase is assumed to be operated by a bandwidth- and latency-insensitive PE controller [40], which makes it easier to explore the optimum memory interfaces and thereby improves the hardware accelerator's environment. Finally, the generated SDDG is used as the input to the PPA estimator, which is the backend of Aladdin.

A. DATAFLOW PHASE

The proposed pre-RTL simulator introduces the dataflow characteristics by adding PE information to all nodes written in the generated graph. For example, Aladdin pre-RTL simulator [32] expresses data reusage via a shared register, whereas the proposed pre-RTL simulator expresses data reuse as a characteristic of the assumed dataflow PE. More specifically, the proposed pre-RTL simulator is characterized by

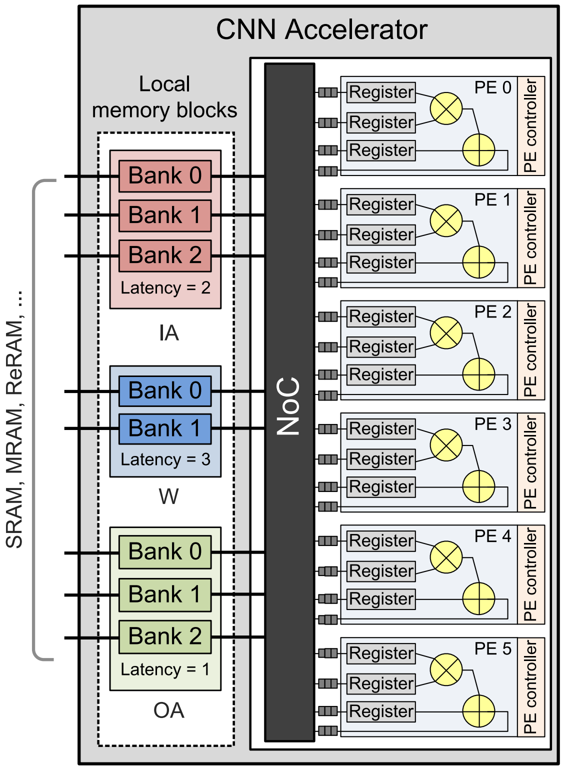


FIGURE 15. CNN hardware accelerator assumed in the proposed pre-RTL simulator.

the tailored-sized registers inside the PEs (i.e., which register location the incoming activations or weights are loaded into); moreover, it is color-coded to indicate the spatial information conveyed by the SDDG, which enables us to model the operations (e.g., MAC, load/store) of each PE in a cycle-accurate manner during this phase. In Figure 17, the color-coded nodes express spatial information relevant to communication. In general, these communication methods are designed to model specific dataflows (RS [24] and WS [28]) inside the hardware accelerator. In this study, we propose to use an SDDG pre-RTL simulator that allows us to model a specific dataflow which exploits these communication methods, in contrast to those used in the Aladdin pre-RTL simulator [32]. For example, Figure 17 (a) shows that the PE loads

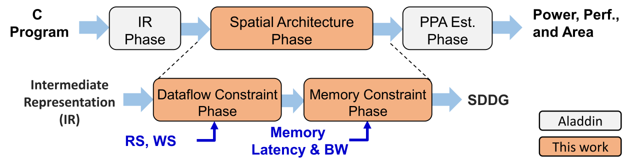


FIGURE 16. Framework of the proposed pre-RTL simulator.

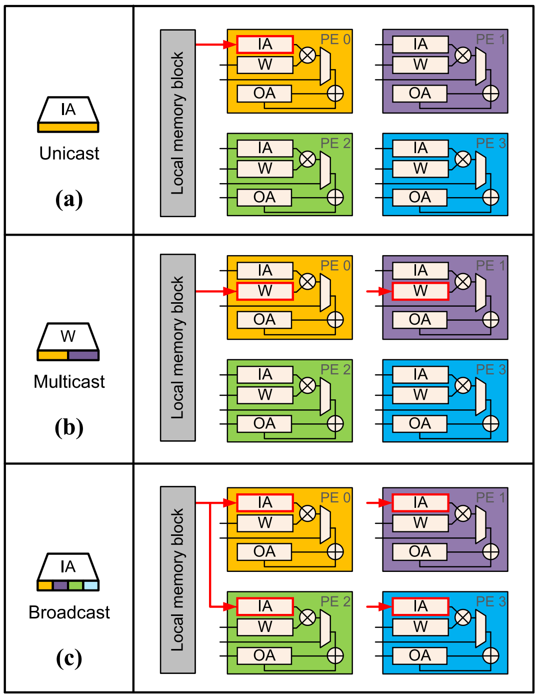


FIGURE 17. Color-coded nodes to express spatial information with respect to communication manner: (a) unicast, (b) multicast, (c) broadcast.

activations from local memory blocks in a unicast manner. Next, in the case of a multicast approach, two or more PEs loaded from the local memory blocks are depicted in Figure 17 (b). Lastly, Figure 17 (c) shows that all PEs load activations from local memory blocks in a broadcast manner. Furthermore, the SDDG generated in this phase includes the local communication between PEs. Thus, such spatial information makes it possible to model a dataflow.

Assuming the RS dataflow proposed in [24], Figure 18 shows the SDDG for the single 2-dimensional convolutional layer of Figure 8, which consists of such instructions as load, store, and MAC. In this graph, the assumed hardware accelerator consists of R by E PEs, and each PE has a tailored-sized register for activations and filters, as shown in Figure 4. Figure 18 shows the memory latency for (1, 1, 1) and the bandwidth for (1, 1, 1). As shown in the figure, in cycle 0, it can be seen that IA(0) and OA(0) are loaded from local memory blocks to PE 0. Likewise, it can be observed that W(0) is loaded with PE 0 and PE 1 using the multicast approach in cycle 0. In cycle 1, IA(3) is loaded into PE 1 and 2 using the multicast approach, and OA(2) is loaded into PE 1 using the unicast one. In addition, W(2) is

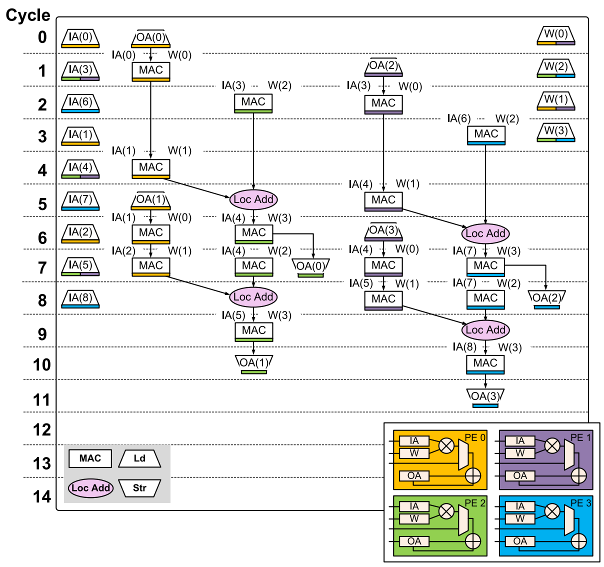


FIGURE 18. SDDG for RS with respect to memory latency (1, 1, 1) and bandwidth (1, 1, 1).

loaded into PE 2 and 3 using the multicast approach. In the same cycle, PE 0 performs a MAC upon the IA(0), OA(0), and W(0) loaded in cycle 0. Because of the characteristics of RS dataflows, each PE relays the psum to the PE of the adjacent row after performing a MAC for q input channels. That is, local communication is required between the PEs in adjacent rows. As shown in cycles 5, 6, 8, and 9 in the figure, the characteristics of the RS dataflow are expressed by a new node defined as local addition (Loc. Add). Finally, the memory access order assumed in the SDDG differs from the algorithm in Figure 8. For example, IA(0) is loaded into PE 0 in cycle 0, and IA(3) is loaded into PE 1 and 2 in cycle 1 before IA(1) is loaded into PE 0 cycle 3. The graph shows that the computation scheduling proposed by RS dataflow affects the memory access order. Figure 19 shows in more detail the status of each PE during the initial eight cycles of the SDDG in Figure 18. For example, in cycle 6, PE 0 shows that the register space of IA(0) is overwritten by IA(2). It also shows the relay of psums to local communications between PEs. For example, in cycle 4, the figure shows that PE 0 relays the psum for OA(0) to PE 2.

Assuming the WS dataflow proposed in [28], Figure 20 shows the SDDG for a single 2-dimensional convolutional layer from Figure 8. In this graph, the assumed hardware accelerator consists of R by S PEs, each of which has a tailored-size register for weights, as shown in Figure 6. Figure 20 shows that an IA is broadcast to all PEs in each cycle. Weights are unicast to each of the specified PEs. In addition, an OA is relayed from each PE to the PE of the adjacent column after a MAC. In particular, the OA(0) loaded in PE 0 in cycle 0 relays to PE 1 after a MAC, and it relays to PE 2 after a MAC in cycle 1. Note that OA is always loaded as the first PE and stored in local memory or off-chip DRAM after MACs are performed through all PEs.

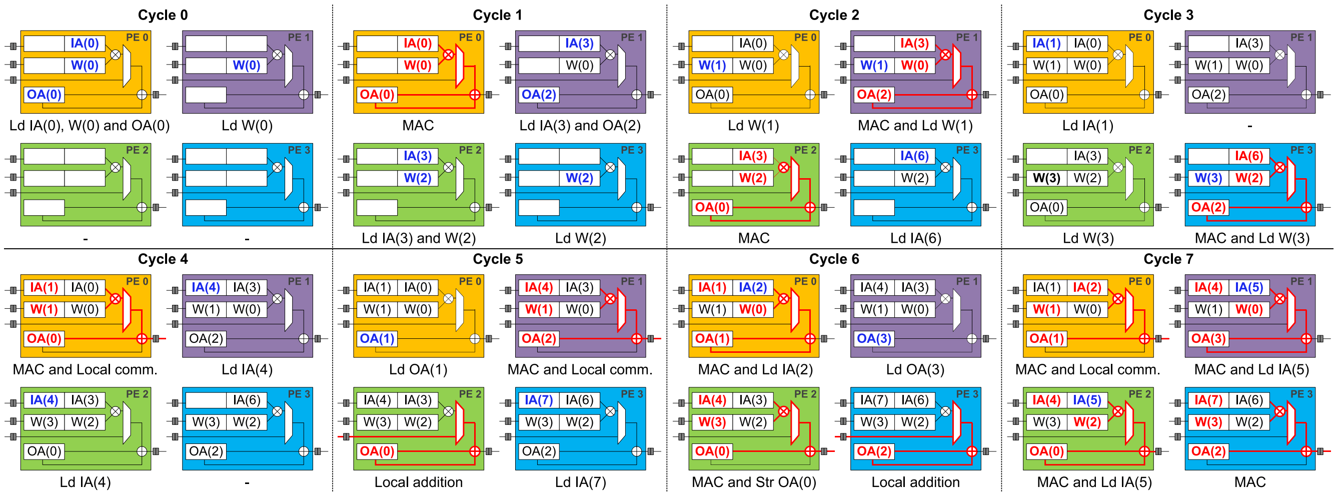


FIGURE 19. Status of each PE during the initial eight cycles of the SDDG in Figure 18.

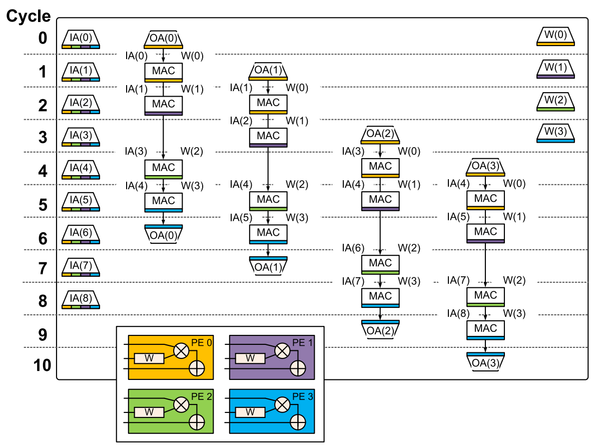


FIGURE 20. SDDG for WS with respect to memory latency (1, 1, 1) and bandwidth (1, 1, 1).

To summarize, the simulation results from the proposed pre-RTL simulator show the spatial information under the generated SDDG, to observe detailed dataflow characteristics. For example, as mentioned earlier, the SDDG in Figure 18 assumes four PEs, which means that up to four MAC units can be activated in every cycle. However, the figure shows that all MAC units are simultaneously activated only in cycle 7. The generated SDDG indicates that this occurs because the activations or weights required by the MAC are not loaded into the PE registers. Though quite natural for the aforementioned characteristics of the data dependence, this underutilization of PEs can be considered the bottleneck of the hardware accelerator. Assuming that the bandwidth of the local memory block allocated (i.e., the number of local memory banks) in the hardware accelerator increases, it is possible for the activation or weight required for the MAC to be delivered earlier, thereby improving PE utilization. The effect of increased bandwidth on the improvement of PE utilization is explained in detail in the following subsection.

B. MEMORY PHASE

The memory phase introduces two additional constraints to the SDDG generated in the dataflow phase: the number of banks distributed to each local memory block and the local memory block latency. First, the number of banks distributed to the local memory blocks determines the number of activations or weights that can be accessed simultaneously (i.e., memory bandwidth). Second, this phase assumes that each local memory block is implemented with a different memory latency. This indicates the number of cycles required by the PEs to access local memory blocks.

The proposed SDDG pre-RTL simulator can model the latency attributable to conflicts during the processes of reading from and writing to the same memory bank. For example, Figure 21 (a) shows that the memory constraints of the SDDG are assumed to be (1, 2, 1) and (2, 2, 1) for memory latency and bandwidth, respectively. The figure shows that OA(0) and OA(2) are written to the same memory bank at the same time; however, the store operation of OA(2) is delayed by one cycle in cycle 6 owing to memory conflicts. Likewise, in cycle 9, OA(3) accesses the same memory bank at the same time as OA(1); hence, it is delayed by one cycle. In Figure 21 (b), the execution time is two cycles longer than in Figure 21 (a) when the IA-allocated local memory block has a large latency, despite the same local memory bandwidth. The SDDG in the figure shows that the latency of the local memory block for the IAs, which requires a larger amount of communication compared to the weights and OAs, has a greater impact on the execution time of the hardware accelerator. These results imply that the local memory block distribution of each data type must be carefully considered according to the communication amount determined by predefined parameters, such as those in Tables 1 and 2. However, it is not generally applicable to the case with a specific computation scheduling for dataflows (as will be shown in Sections VI-B and C).

Figure 22 shows an SDDG of the hardware accelerator with a local memory latency of (2, 1, 1) and bandwidth of

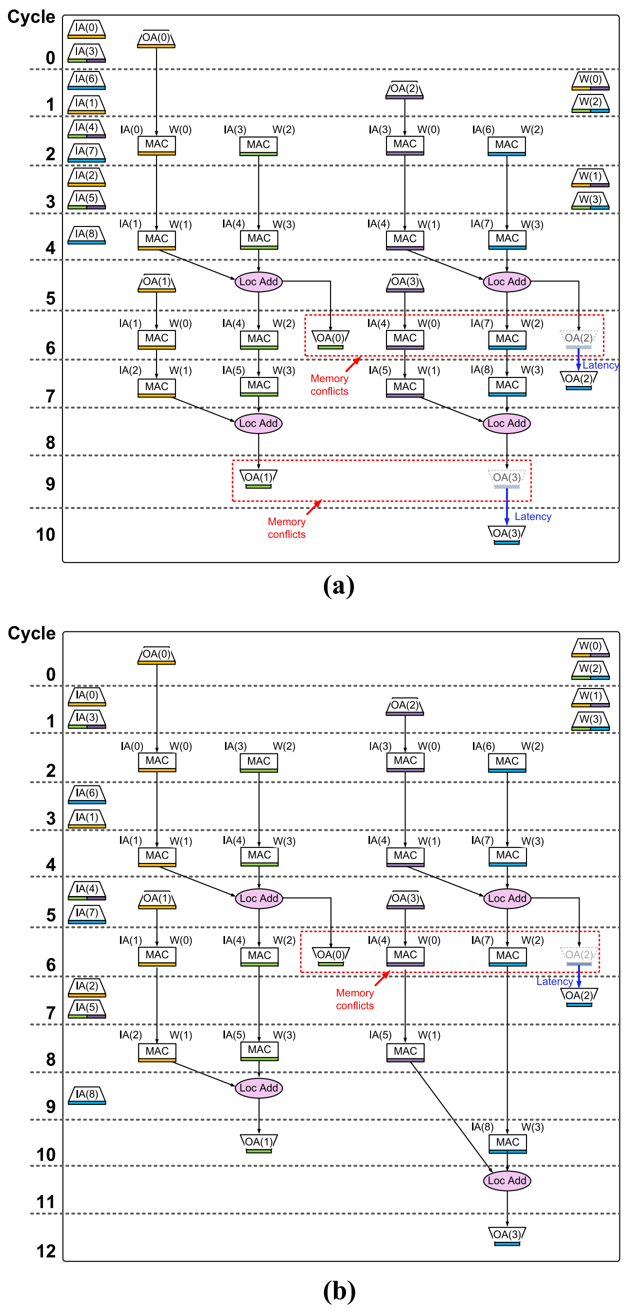


FIGURE 21. SDDG for RS with respect to memory constraints: (a) latency (1, 2, 1) and bandwidth (2, 2, 1), (b) latency (2, 1, 1) and bandwidth (2, 2, 1).

(1, 4, 1). As shown in the figure, the execution time increases by 9 cycles compared with that in Figure 20. This occurs because the amount of communication increases the amount of time for which large IAs are broadcast to PE, owing to the latency of the local memory block. It appears that the execution time of a hardware accelerator can be determined by the communication amount of activations or weights. However, the computation scheduling of the dataflow may have a more dominant effect on the execution time of the hardware accelerator according to our experimental results (as will be discussed in Section VI.B).

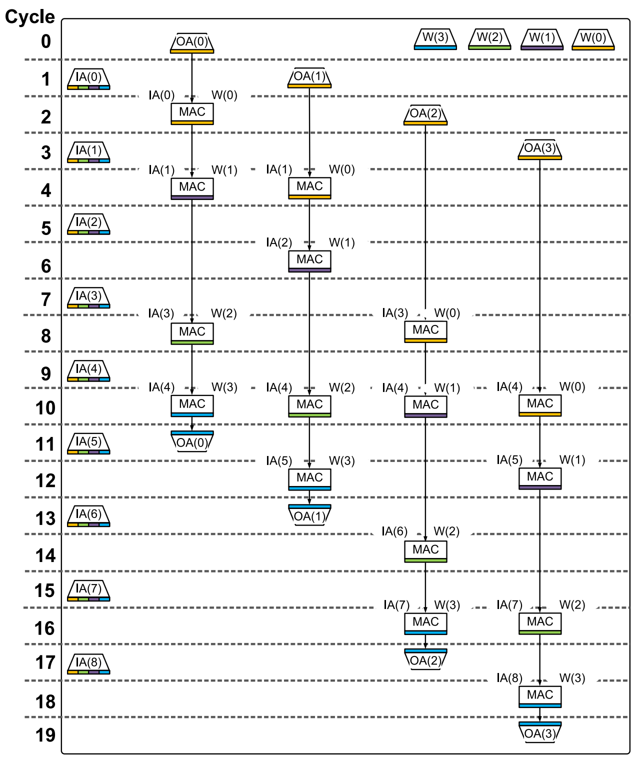


FIGURE 22. SDDG for WS with respect to memory constraints: latency (2, 1, 1) and bandwidth (1, 4, 1).

VI. SIMULATION RESULTS

In this section, the PPA results of the hardware accelerator are evaluated using the proposed pre-RTL simulator. Although the proposed SDDG pre-RTL simulator is generally applicable to any CNN applications, the third and fourth convolutional layers of AlexNet [61] and the fourteenth convolutional layer of MobileNet [62] are taken as the example applications in this section. For the convolutional layers, the tile size is set according to the three different tile sizes given in Table 4: tAlex1, tAlex2, and tMobile. As mentioned earlier, once the tile parameters are determined, the convolutional layer is divided into several processing passes. The number of computations and the amount of transferred activations and filters for each processing pass are almost identical; hence, a single processing pass is sufficient to represent the performance and power consumption of the hardware accelerator for a convolutional layer. Therefore, the estimation results in this section evaluate the performance and power consumption of a single processing pass. Table 5 shows the normalized energy cost of each operation with respect to that of multiplication energy cost. The table lists the measured energy costs for a 16-bit adder, 16-bit multiplication, 16-bit register operation, and 16-bit memory operation subject to the operating clock frequency of the hardware accelerator at 200 MHz, using a design compiler logic synthesis tool with a CMOS 45-nm cell library [32]. This section explains how, assuming the well-known dataflows, RS [24] and WS [28] dataflows, the simulation results exhibit a close dependence of accelerator performance upon memory

TABLE 4. Layer shapes and tile sizes of the convolutional layers considered in this paper.

Convolutional layer	Layer shape parameters						Tile parameters			
	M	C	H/W	E/F	R	S	p	q	r	t
Layer 3 of AlexNet (tAlex1)	384	256	15	13	3	3	16	4	1	1
Layer 4 of AlexNet (tAlex2)	256	384	15	13	3	3	4	16	1	1
Layer 14 of MobileNet (tMobile)	512	1	14	12	3	3	16	1	1	1

TABLE 5. Normalized power consumption relative to a multiplier extracted from Aladdin pre-RTL simulator [32].

	Memory	Register	Adder	Multiplier
Normalized energy	3.7x	0.2x	0.3x	1x

constraints. In addition, the simulation results also show that the relative performances of dataflows depend on the layer shape of the convolutional layer. It is worth mentioning that the proposed pre-RTL simulator can provide insights into local memory constraints, unlike the conventional pre-RTL simulators, which are focused on dataflows.

A. FRAMEWORK VALIDATION

In order to validate the proposed pre-RTL simulator, we present the energy comparison between our simulation results and the post-synthesis results. The hardware accelerators assumed for comparison are RS [24] and WS [28] dataflows, and the latency and bandwidth of the local memory blocks are (1, 1, 1) and (1, 1, 1), respectively. In addition, the proposed pre-RTL simulator measures the energy of the hardware accelerator via the energy of each operation in the CMOS 45-nm standard cell library, as assumed in [19]; in contrast, Eyeriss [24] and DSIP [28] measure the energy of the hardware accelerator in the CMOS 65-nm standard cell library. Thus, the energy of both hardware accelerators is normalized via a ratio to the summed power consumption of all convolutional layers, measured with each standard cell library.

Figure 23 shows the results for the comparative validation experiment against the Eyeriss [24] and DSIP [28] hardware accelerator architectures, with AlexNet [61] as the workload. The figure shows that the estimation results of the proposed SDDG simulator fairly accurately follow a function of the energy reported from prior studies. Figure 22 (a) shows that the normalized energy and execution time of the Eyeriss hardware architecture modeled with the SDDG pre-RTL simulator can be predicted with errors of less than 7.1% and 6.3%, respectively, when compared with the actual measurement results of the Eyeriss hardware architecture. Moreover, in Figure 23 (b), the normalized energy and execution time of the DSIP hardware architecture modeled with the proposed SDDG pre-RTL simulator can be predicted with errors of less than 7.8% and 5.6%, respectively, when compared with the actual measured results. These estimation errors may arise because the proposed pre-RTL simulator does not take into account components other than PE arrays and local memory blocks (e.g., run-length compression in Eyeriss [24]). In addition, because the controller implemented inside the PE for computation scheduling of the proposed pre-RTL simulator

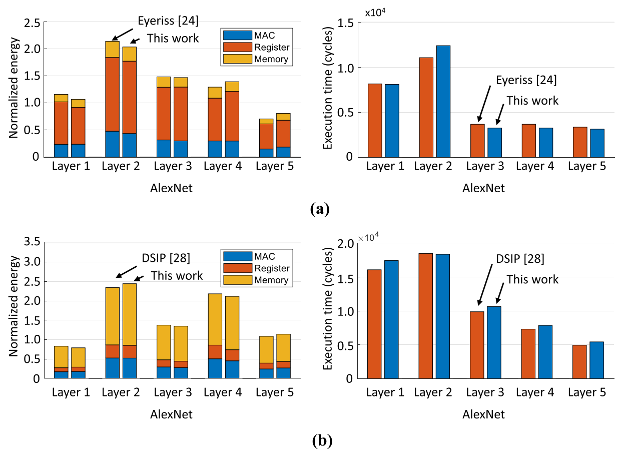


FIGURE 23. Validation of the proposed SDDG pre-RTL simulation results against measured results: (a) Eyeriss [24], (b) DSIP [28].

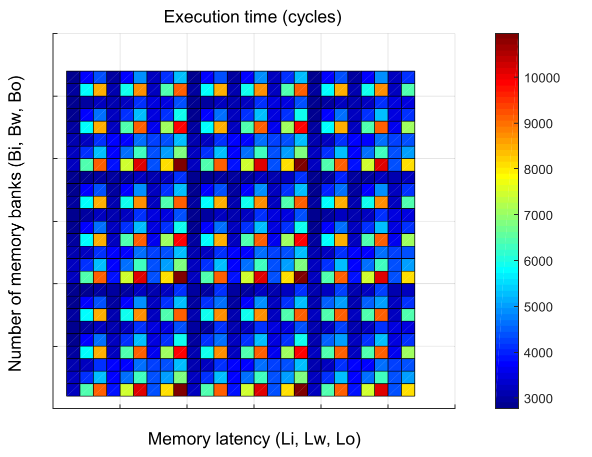


FIGURE 24. Impact of memory constraints for CNN accelerator with RS.

is assumed to be that presented in [40], it may differ from that assumed in Eyeriss and DSIP. Lastly, since the standard cell library adopted in our experiment and the standard cell library used in Eyeriss and DSIP are different, it is expected that the power consumption will be affected differently.

B. SENSITIVITY ANALYSIS

In this subsection, we show the impact of tile parameters and local memory constraints on the performance and power consumption of hardware accelerators. Figure 24 shows the impact of the latency ($1 \leq Li \leq 3$, $1 \leq Lw \leq 3$, $1 \leq Lo \leq 3$) and the number of banks ($1 \leq Bi \leq 3$, $1 \leq Bw \leq 3$, $1 \leq Bo \leq 3$) of the local memory blocks on the execution time of the hardware accelerator under RS dataflows. The layer shape is set as tAlex1 in Table 4. Figure 25 (a) shows the loops pertaining to the index of the x-axis in Figure 23. Lo indicates the inner-most loop of Figure 25 (a), followed by Lw and Li as the outer-most loop. Likewise, Figure 25 (b) presents loops pertaining to the y-axis of Figure 24, where the inner-most loop is Bo, followed by Bw, and Bi as the outermost loop. As shown in Figure 24, the latency of the

```
index = 0;
for Li = 1:3 {
   for Lw = 1:3 {
      for Lo = 1:3 {
         index++;
         x-axis[index] = (Li, Lw, Lo);
}}}
```

(a)

```
index = 0;
for Bi = 1:3 {
   for Bw = 1:3 {
      for Bo = 1:3 {
         index++;
         y-axis[index] = (Bi, Bw, Bo);
}}}
```

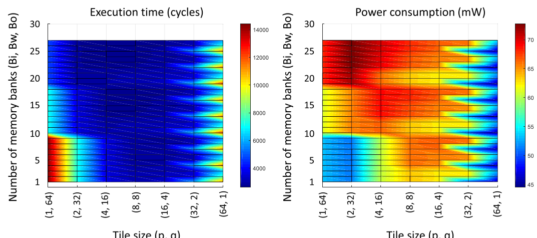
(b)

FIGURE 25. Memory constraints index: (a) memory latency (x-axis in Figure 24), (b) memory bandwidth (y-axis in Figure 24).

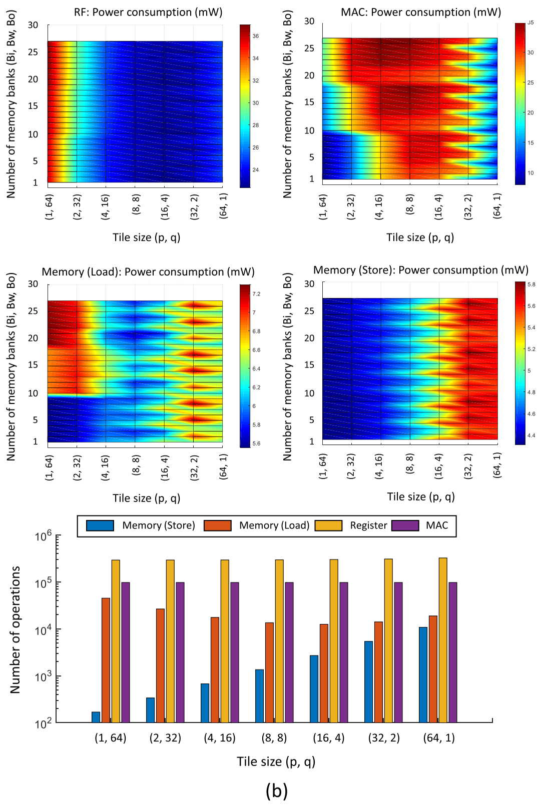
local memory and the number of local memory banks for each data type are set to (3, 3, 3) and (1, 1, 1), respectively, as a baseline; when the local memory latency is reduced to (1, 1, 1), the execution time of the hardware accelerator is reduced by 65.3%. In addition, when the number of local memory banks for each data type of the hardware accelerator is increased to (3, 3, 3), the execution time of the hardware accelerator is reduced by 70.1%.

Figure 26 represents the power consumption and execution time of the hardware accelerator according to the number of local memory banks for each data type under various tile sizes for the hardware accelerator. In this case, the accelerator is assumed to operate according to the RS dataflow computation scheduling [40], with the third convolutional layer of AlexNet as the workload. As shown in Table 6, it is assumed that the number of local memory banks for each data type can be allocated from a minimum of 1 to a maximum of 3 (i.e., $1 \leq Bi \leq 3$, $1 \leq Bw \leq 3$, $1 \leq Bo \leq 3$). In the experimental results, an increase in the index of the y-axis indicates that the local memories for each data type and PE between the bandwidth (i.e., number of transmitted activations or weights per cycle) increases. The experimental results show the considerable impact of the tile size (x-axis) and local memory banks (y-axis), which are the parameters of the hardware accelerator that influence the execution time and power consumption of a processing pass.

Figure 26 (a) shows the changes in execution time and power consumption of each data type according to its number of local memory banks. First, the figure shows a decreasing trend relating to the execution time of the hardware accelerator when the number of local memory banks allocated to each data type increases. For example, the execution time is reduced by 67% when the tile size is (1, 64) and the maximum number of local memory banks is allocated for each data type (i.e., index = 27), in contrast to the hardware accelerator



(a)



(b)

FIGURE 26. Sensitivity analysis according to the tile size and the number of local memory banks for each data type: (a) execution time and power consumption, (b) power breakdown and number of operations.

in which the distribution of local memory banks is set to the minimum (i.e., index = 1). These results show that the performance is improved because the PEs of the accelerator can access the local memory simultaneously when the number of local memory banks increases. The experimental results of Figure 26 (a) show the power consumption of the hardware accelerator, which varies according to the various tile sizes, and the number of banks distributed in the local memory for each data type. As shown in the figure, the power consumption of the hardware accelerators is inversely proportional to the decreasing execution time when the number of local memory banks for each data type increases. For example, when the given tile size of the hardware accelerator is (2, 32) and the number of local memory banks increases from (1, 1, 1) to (3, 3, 3), the execution time of the hardware

TABLE 6. Y-axis in Figure 26.

Index	Number of memory banks (Bi, Bw, Bo)	Index	Number of memory banks (Bi, Bw, Bo)	Index	Number of memory banks (Bi, Bw, Bo)
1	(1, 1, 1)	10	(2, 1, 1)	19	(3, 1, 1)
2	(1, 1, 2)	11	(2, 1, 2)	20	(3, 1, 2)
3	(1, 1, 3)	12	(2, 1, 3)	21	(3, 1, 3)
4	(1, 2, 1)	13	(2, 2, 1)	22	(3, 2, 1)
5	(1, 2, 2)	14	(2, 2, 2)	23	(3, 2, 2)
6	(1, 2, 3)	15	(2, 2, 3)	24	(3, 2, 3)
7	(1, 3, 1)	16	(2, 3, 1)	25	(3, 3, 1)
8	(1, 3, 2)	17	(2, 3, 2)	26	(3, 3, 2)
9	(1, 3, 3)	18	(2, 3, 3)	27	(3, 3, 3)

accelerator decreases by 61%, whilst the power consumption increases by 30%. It can be observed that a higher hardware accelerator power consumption is observed when the same number of operations is performed within a shorter time.

As shown in Figure 26, the experimental results for the power consumption breakdown of each component in the hardware accelerator show that MAC operations have a dominant impact on the power consumption of the hardware accelerator, and such operations account for the largest portion among the operations of all components. For example, Figure 26 (a) shows that a power consumption of 24% more is necessary for the hardware accelerator with a tile size of (8, 8) compared with one with a tile size of (1, 64) given that the number of local memory banks is (1, 1, 1). This is attributed to the fact that the execution time for processing MAC operations differs according to the tile size determined by the hardware accelerator. As shown in Figure 26 (a), the execution times of a hardware accelerator whose tile sizes are determined as (1, 64) and (8, 8) span 14404 and 3075 cycles, respectively. Figure 26 (b) shows that when the tile size of the hardware accelerator is determined as (8, 8), most of the execution time is spent for MAC operations. Conversely, when the tile size of the hardware accelerator is determined as (1, 64), most of the execution time is spent upon memory operations.

Figure 26 (a) shows that the distribution of the number of local memory banks for each data type affects the trend of the execution time, which varies according to the characteristics of the tile size determined for each hardware accelerator. For example, in the case of a hardware accelerator whose tile size is determined as (1, 64), the execution time tends to decrease when the number of local memory banks for IAs increases, as shown in Figure 26 (a). This is attributed to the fact that the input channel tile size (q) is larger than the output channel tile size (p). Thus, the number of memory accesses for the IAs exceeds the number of memory accesses for weights or OAs. As a result, the method that allocates more banks to the local memory to which the IAs are allocated will reduce the execution time of the hardware accelerator compared to other data type distributions. Conversely, the hardware accelerators whose tile size is determined as (64, 1) show a decreasing execution time with respect to the number of local memory banks for OA. This occurs because the output channel tile size (p) is larger than the input channel tile size (q). Thus, the number of memory accesses of the OA is higher than the number of memory accesses of the weights or input activation. As a result, the execution time of the hardware

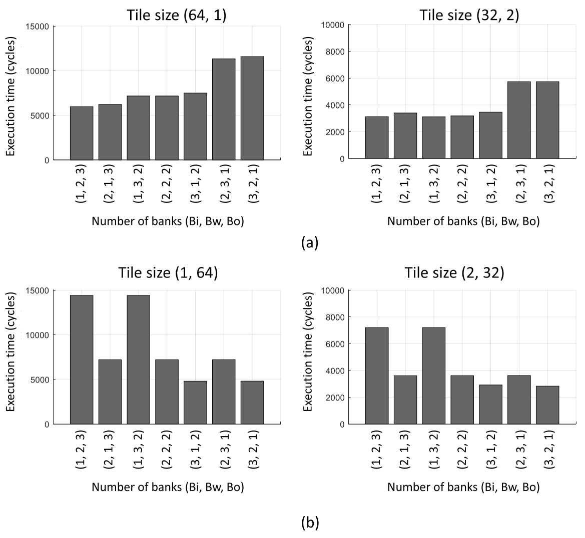


FIGURE 27. Execution time of hardware accelerator according to local memory banks distribution for each data type according to various tile sizes: (a) (64, 1) and (32, 2), (b) (1, 64) and (2, 32).

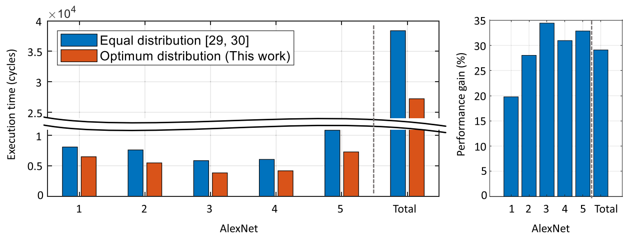


FIGURE 28. Performance gain of optimum distribution against equal distribution in five convolutional layers of AlexNet.

accelerator can be reduced by distributing more banks to the local memory to which the OA is allocated, when compared with the local memory allocation under distributions of other data types.

Figure 26 (a) shows that the distribution of the number of local memory banks (with respect to data type) when obtaining the optimal execution time differs according to the tile size determined by the hardware accelerator. Figure 27 shows the method for distributing the number of local memory banks (which are limited to six) to obtain the shortest execution time for the tile size specified in the hardware accelerator. Figure 27 (a) shows that when the tile size of the hardware accelerator is (64, 1) and (32, 2), the tile size (p) of the output channel exceeds the tile size (q) of the input channel. Thus, the number of memory accesses for the OAs inside the hardware accelerator is greater than that of the number of memory accesses for weights or IAs. This tile size can help reduce the execution time of the hardware accelerator through the distribution of more local memory banks that have been allocated with OAs, when compared with the number of local memory banks of other data types. As shown in Figure 27 (a), when the tile size is (32,2), the execution time is reduced by up to 13.6% by distributing more number banks to the local memory of OAs, compared with the case in which the

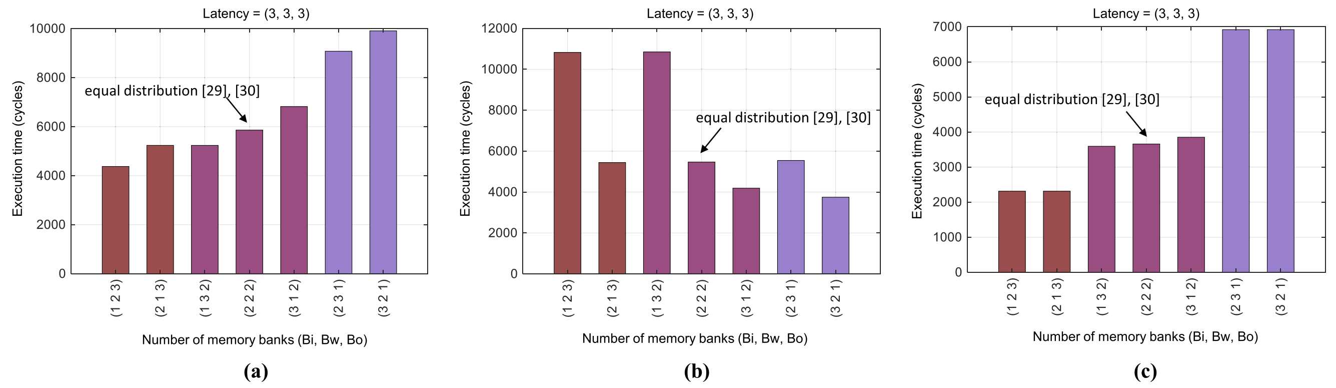


FIGURE 29. **Performance of hardware accelerators assuming RS in terms of local memory bandwidth: (a) Third layer of AlexNet (tAlex1), (b) Fourth layer of AlexNet (tAlex2), (c) Fourteenth layer of MobileNet (tMobile).**

local memory is equally distributed (i.e., (Bi, Bw, Bo) = (2, 2, 2)). In the RS dataflow computation scheduling [40] assumed by the proposed pre-RTL simulator, this is given priority such that one IA calculates the weights of several output channels (p). Thus, it can be observed that expanding the number of banks in the local memory for weights allows the MAC operations of the output channel (p) to be conducted at a fast rate. This helps reduce the execution time further. For example, when the tile size of the hardware accelerator is (64, 1), its execution time is reduced by 7% when distributed as (3, 2, 1) compared to its distribution under (3, 1, 2), as shown in Figure 27 (a).

Figure 27 (b) shows that when the tile sizes of the hardware accelerator are (1, 64) and (2, 32), the tile size (q) of the input channel is larger than the tile size (p) of the output channel; thus, the number of memory accesses for the IA inside the hardware accelerator is greater than those of the weights or OA. The computing scheduling [40] assumed by the SDDG pre-RTL simulator proposed in this study prioritizes the accumulation rule of several input channels (q), based on the rule according to which one IA is calculated using the weights of several output channels (p). Therefore, by adding the number of banks to local memory for IAs, PEs receive IAs from local memory earlier, time to perform MAC operations as quickly as possible. This can help improve the execution time of the hardware accelerator. As shown in Figure 27 (b), when the tile size is (2, 32), the execution time is reduced by up to 16.8%, by distributing more number banks to the local memory of IAs, when compared with the case in which the local memory is equally distributed (i.e., (Bi, Bw, Bo) = (2, 2, 2)).

C. IMPACT OF MEMORY BANDWIDTHS IN CNN HARDWARE ACCELERATORS

The SDDG pre-RTL simulator proposed in this study can estimate the execution times of hardware accelerators when the number of memory banks is differentially (as opposed to equal distribution, as seen in [29], [30]) distributed to local memory for each data type. For example, when a total of six banks in the local memory of the hardware accelerator is given, the baseline in [29] and [30] distributes two banks to the local memory for each data type. On the other hand, the proposed SDDG pre-RTL simulator can reduce the execution time of the hardware accelerator by distributing the number of local memory banks for each data type according to the communication amount for each data type and the computation scheduling characteristics. Figure 28 shows that regarding the five convolutional layers of AlexNet [61], the execution time was reduced by up to 34.8% compared to equal distribution [29], [30] when the number of banks was optimally distributed based on the communication amount and computational scheduling. In addition, optimal distribution reduces the AlexNet [61] execution time of hardware accelerators by 29% compared to equal distribution.

Figure 29 evaluates the performances of hardware accelerators under RS dataflows in terms of the local memory bandwidth (i.e., number of local memory banks per data type). Figure 29 (a) shows the effect of the number of banks distributed to each local memory block on the performance when a local memory latency of (3, 3, 3) is included in the results of Figure 24. It is interesting to note that such a DSE helps to optimize the bandwidth distribution across different data types. For example, as shown in Figure 29 (a), when the total number of banks is 6, the optimum bandwidth distribution for each data type that the hardware accelerator can obtain in a minimum execution time is (1, 2, 3). The execution time of the hardware accelerator in which this manner is adopted is reduced by 28.1% compared to that of the hardware accelerator implementing an equal distribution manner ((i.e., (Bi, Bw, Bo) = (2, 2, 2)). In addition, the execution time of the hardware accelerator under the optimal bandwidth distribution is reduced by 58.4% compared to that of the accelerator with the worst bandwidth distribution (i.e., (Bi, Bw, Bo) = (3, 2, 1)). This is because the amount of communication required for the data types is determined by the tile parameters. According to the tile parameters defined in Table 4, the amount of communication for each data type between the PEs and local memory blocks is 2704 pixels for OAs, 900 pixels for IAs, and 576 filters for weights. Therefore, it is straightforward to allocate a large number of banks to the local memory block of OAs to improve execution time. However, although the communication amount for the IAs exceeds that of the communication amount for the weights, the execution time is improved by distributing more banks to the local memory block of weights. This can be explained by the computation scheduling of the PEs in Eyeriss [24], which prioritizes the convolutional layers such that a single IA pixel is reused for multiple output channels of weights. Furthermore, a single IA pixel generates psums with the weights of p output channels. This means that a single IA pixel must wait for all p weights to be loaded into the PE, which may be considered a bottleneck of the hardware accelerator. That is, increasing the number of banks in the local memory block for weights means that a single IA pixel can be reused quickly. This reduces the execution time of the hardware accelerator. In addition, this observation tends to be replicated for different layer shapes and tile parameters. As shown in Figure 29 (b), in the case of the fourth convolutional layer of AlexNet in Table 4 (tAlex2), the amount of communication is large for IAs, OAs, and weights. As shown in Figure 29 (b), when the total number of banks is 6, the optimum bandwidth distribution (for each data type) with which the hardware accelerator can obtain the minimum execution time is (3, 2, 1). The execution time of the hardware accelerator in which this manner is adopted is reduced by 34.6% compared to the execution time of one applying the equal distribution method (i.e., (Bi, Bw, Bo) = (2, 2, 2)). In addition, the execution time of the hardware accelerator under the optimal bandwidth distribution is reduced by 64.3% compared to that of the hardware accelerator with the worst bandwidth distribution (i.e., (Bi, Bw, Bo) = (1, 2, 3). Because PEs can load IAs more often, the space of the register for the IAs allocated to PEs is quickly filled. As mentioned earlier, a single IA pixel is quickly reused by expanding the number of banks of the local memory block used for weights. That is, extra banks should be distributed to the local memory block of weights rather than OAs. As shown in Figure 29 (c), when the total number of banks is 6, the optimum bandwidth distribution (for each data type) with which the hardware accelerator can obtain the minimum execution time is (1, 2, 3). The execution time of the hardware accelerator in which this manner is adopted is reduced by 37.6% compared to that of the hardware accelerator applying an equal distribution (i.e., (Bi, Bw, Bo) = (2, 2, 2)). In addition, the execution time of the hardware accelerator applying the optimal bandwidth distribution is reduced by 69.4% compared to that of the hardware accelerator with the worst bandwidth distribution (i.e., (Bi, Bw, Bo) = (3, 2, 1)). This improvement can be realized by distributing a large number of banks to the local memory blocks for the OAs that require a large amount of communication, instead of according to the communication amounts of IAs and weights. The number of banks of local memory blocks for IAs and weights has a minimal effect on the hardware accelerator execution time, because there is little difference in the amount of communication between the two data types (i.e., 169 pixels for IAs, 144 filters for weights). These simulation results provide a novel insight, in contrast with the equal distribution method [29], which distributes the number of banks in local memory blocks equally for each data type. In other words, when the number of available banks is determined, distributing the number of differentiated banks for each data type according to the computation scheduling of dataflows helps to improve the performance of the hardware accelerator.

TABLE 7. Memory constraints assumed in Section VI.C.

Case 0	(Li, Lw, Lo) = (1, 1, 1), (Bi, Bw, Bo) = (1, 1, 1)
Case 1	(Li, Lw, Lo) = (1, 4, 1), (Bi, Bw, Bo) = (1, 4, 1)
Case 2	(Li, Lw, Lo) = (3, 1, 2), (Bi, Bw, Bo) = (1, 3, 1)
Case 3	(Li, Lw, Lo) = (2, 2, 3), (Bi, Bw, Bo) = (1, 1, 1)

To summarize, the proposed pre-RTL simulator provides the design space of hardware accelerators for dataflows under memory constraints. In other words, the proposed pre-RTL simulator evaluates the performances of hardware accelerators in the design space, unlike the conventional pre-RTL simulators. Thus, it can be concluded that the optimum memory bandwidth of the hardware accelerator depends on the amount of communication of each data type and the computation scheduling characteristics of the dataflows.

D. IMPACT OF MEMORY LATENCY IN CNN HARDWARE ACCELERATORS

This subsection estimates the PPA results of hardware accelerators modeled with several memory constraints (as shown in Table 7) and several dataflows, such as RS [24] and WS [28] dataflows. The difference in the characteristics of the hardware accelerators in the RS and WS dataflows means that the following assumptions are needed to facilitate a comparison. First, WS0 indicates that the hardware accelerator is configured with the same number of PEs as RS0. Second, WS1 means that hardware accelerators are configured with approximately the same area size as the RS0. The simulation results in this subsection show that the superiority of a dataflow depends on the layer shape of the convolutional layer and the latency of the memory blocks.

Figure 30 shows the simulation results of hardware accelerators under RS [24] and WS [28] dataflows, respectively, for the third convolutional layer of AlexNet (tAlex1). Figure 30 (a) shows the size of the hardware accelerator area under two dataflows. The RS dataflows approach assigns registers for each data type to all PEs, in contrast with the WS dataflows. That is, the hardware accelerator area of the RS0 case is 3.7x larger than that of the hardware accelerator area of the WS0 case, which is assumed to have the same number of PEs. In addition, WS1 adds PEs to the hardware accelerator, to show that it almost matches the hardware accelerator area under the RS dataflow. The simulation results of WS0 and RS0 in Figure 30 (b) show that the execution time between the two dataflows differs by up to 27% under increasing memory latency. For example, the difference between the execution time of WS0 and the execution time of RS0 is only 6% in Case 0. However, in Case 3, this difference increases when the latency of the local memory block increases. This is because no registers are allocated inside the PEs for the

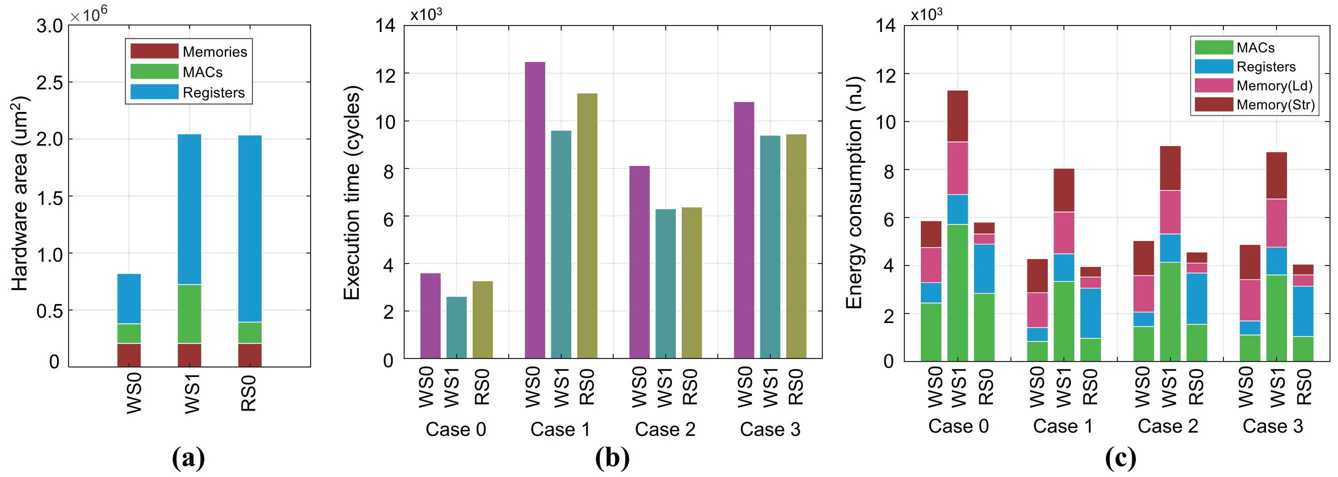


FIGURE 30. Simulation results of hardware accelerators assuming RS and WS, respectively, in a third convolutional layer of AlexNet (tAlex1): (a) hardware area, (b) execution time, (c) energy consumption.

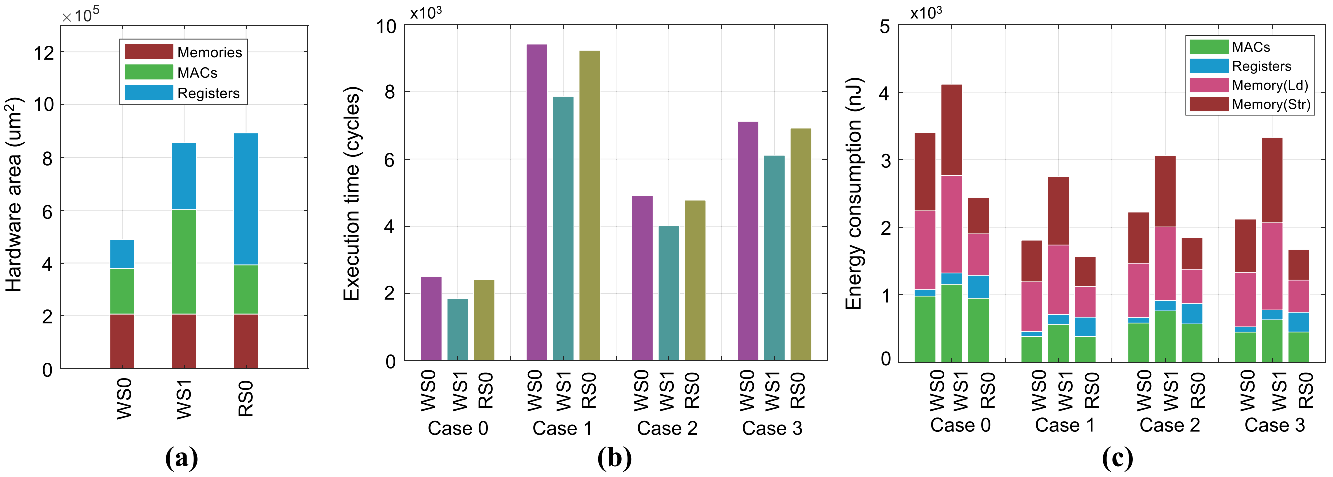


FIGURE 31. Simulation results of hardware accelerators assuming RS and WS, respectively, in a depthwise convolutional layer of MobileNet (tMobile): (a) hardware area, (b) execution time, (c) power consumption.

reuse of activations (IAs and OAs) in WS dataflow. That is, in WS dataflow, PEs require access to local memory blocks for activation loads, and an increase in the hardware accelerator execution time is inevitable, owing to the latency with which PEs are experienced. Owing to these characteristics, the execution time of WS1 may not be significantly better than that of RS0 execution time. For example, in Case 0, there is a minimum latency in the local memory blocks of IAs and OAs in Figure 30 (b). Therefore, the execution time of WS1 is 20% smaller than that of RS0 execution time in Case 0. However, WS1 in Cases 2 and 3 has a higher latency than RS0 when accessing local memory for activations, increasing the execution time. These simulation results show that RS dataflow tends to be less sensitive to memory constraints than WS dataflow, because they utilize register reuse more aggressively and therefore end up with fewer memory accesses. In Figure 30 (c), the energy consumption of RS0 has a slight advantage over the energy consumption of WS0. More specifically, it can be observed that the energy consumption in RS0 registers is at least 2.4x more than those of WS0. The reason for this is that, in RS dataflow, registers are allocated to all data types for the calculation of the psums of a row in all PEs, whereas in the case of the WS dataflow, registers are only used to store weights. That is, in the case of hardware accelerators where the RS dataflow case is assumed, the number of redundant accesses to the local memory block is reduced, owing to the reuse of data inside each PE, and the energy consumption attributable to the local memory block access shows that the WS dataflow is more dominant than the RS dataflow. As shown in Table 5, the energy consumed by the local memory block access is 18.5x higher than that of the register energy consumption, and the energy consumed by the additional registers under the RS dataflow is amortized by the local memory block access. Thus, it is shown that RS dataflow achieves a significant energy saving because it can lead to fewer memory accesses. In addition, WS1 shows that the total energy consumption of the hardware accelerator is increased by up to 2.2x compared with that under the RS0.

Figure 31 shows the simulation results of hardware accelerators assuming RS [24] and WS [28] dataflows, respectively,

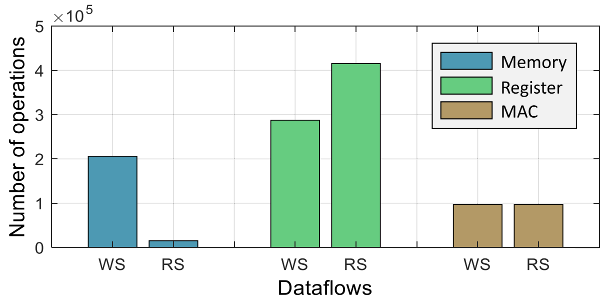


FIGURE 32. Comparison of the number of operations of RS and WS dataflows.

in a depth-wise convolutional layer of MobileNet (tMobile). Figure 31 (a) shows the size of the hardware accelerator area under two dataflows. As shown in the figure, most of the RS0 area consists of registers allocated to each PE, and an area size 2.7x area size larger than that of the register area of WS0 is required. In particular, this shows that the difference in the size of the hardware accelerators to which the two dataflows are applied is smaller than the result of Figure 30 (a). This can be considered a feature of the depth-wise convolutional layer shape of MobileNet [i.e., weights per input channel (input depth)]. In the figure, WS1 uses fewer registers than RS0, though it shows that the MAC units allocated in the hardware area of WS1 have increased. The difference between the execution time of RS0 and that of WS0 execution time in Figure 31 (b) is reduced by up to 3% in Case 1, unlike the difference between the execution times of RS0 and WS0 in Figure 30 (b). These simulation results are attributed to the reduced number of times PEs access local memory blocks when reusing IAs and OAs, owing to the layer shape of the depth-wise convolutional layer in WS. In addition, the simulation results show that the performance of WS1 exceeds that of RS0 performance in all cases. For example, WS1 provides a performance advantage over RS0 for Cases 2 and 3 in Figure 31 (b). This result contrasts with WS1, which provides no performance advantage over RS0 in Cases 2 and 3 in Figure 30 (b). To summarize, in a depth-wise convolutional layer of MobileNet, the performance of WS is better than that of RS performance, even in the case of high memory latency. This implies that the proposed pre-RTL simulator can be used to optimize the dataflows and memory constraints according to the convolutional layer shape. The simulation results show that the difference between the energy consumptions of WS0 and RS0 in Figure 31 (c) is up to 38% larger than in Figure 30 (c). This is because the register access required for a single PE of the RS dataflow is reduced, owing to the layer shape of the depth-wise convolutional layer of MobileNet. As shown in Figure 31 (c), in the case of WS1 (which is assumed to be the same hardware area as the RS0), the energy consumption is increased by a factor of up to 2.2x compared with the hardware accelerator implemented in the RS dataflow.

It is difficult to estimate PPA results for the hardware accelerator, owing to the characteristics of the dataflows, local memory blocks constraints, tile parameters, and convolutional layer shapes. In particular, recalling that the performance and energy consumption of the hardware accelerator estimated by conventional pre-RTL simulators can only explore the effects of dataflows, the design space for local memory constraints is not taken into account. The proposed pre-RTL simulator estimates the hardware accelerator performance and energy consumption according to local memory constraints as well as dataflows in the early design phases for latency- and bandwidth-insensitive controllers. For example, when the latency of the local memory block increases, the fact that the execution time of the WS dataflow increases more than the RS dataflow can be explained by the dataflow characteristics (e.g., number of local memory operations). Figure 32 shows the number of operations for each component of two specific dataflows (e.g., RS [24] and WS [28]). The figure shows that RS dataflow has considerably fewer memory accesses than WS dataflow, which means the RS dataflow achieves more data reuse than the WS dataflow by employing the register within the hardware accelerator. This result implies that it is possible for the execution time of the hardware accelerator implemented under WS dataflow to be heavily affected by the memory latency. Thus, it can be concluded that the effect of the local memory latency pertaining to the layer shape characteristics is reduced through the simulation results.

VII. CONCLUSION

The proposed pre-RTL simulator helps to explore the design space of CNN hardware accelerators. The use of an SDDG makes it possible to accurately model a specific dataflow under given memory constraints. More specifically, an SDDG contains spatial information, such as which ALU/register each of the instructions involves (e.g., which register location the incoming pixel is loaded into). In addition, it is possible to evaluate the impact of the memory constraints imposed by memory block constraints in the memory hierarchy. In order to maximize performance as well as functional correctness, the pre-RTL simulator assumes a latency- and bandwidth-insensitive controller for each PE.

According to the DSE, it is shown that the optimal distribution method of local memory banks (considering communication amount and computation scheduling) can reduce the execution time of the hardware accelerator by up to 37.6% (compared with the equal distribution). In addition, the two well-known dataflows, WS and RS dataflows, are compared in terms of performance, power consumption, and area, assuming either the same number of PEs or the same hardware area. The simulation results show that the superiority of a dataflow depends on the layer shape of the convolutional layer. For example, under identical hardware areas, RS dataflow tends to be less sensitive to memory constraints than WS dataflow, because it undertakes register reuse more aggressively and therefore ends up with fewer memory accesses. In addition, it is shown that RS dataflow offers significant power savings, because it can lead to fewer memory accesses. This implies that the proposed pre-RTL simulator can be used to optimize the dataflows and memory constraints according to the layer shape, thereby facilitating more efficient CNN designs.

The extension of the proposed pre-RTL simulator into the case with dynamic (i.e., state-dependent) memory latency appears promising as future work. Such an extension makes the simulation results more realistic by taking into account the impact of the row buffer status (e.g., row buffer hit/miss) [63]-[65] and on-chip bus contention.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[3] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, May 2014, pp. 4087–4091.

[4] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.

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

[6] Y. Shen, M. Ferdman, and P. Milder, "Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2017, pp. 93–100.

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

[8] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2010, pp. 37–47.

[9] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[10] NVDIA. (Nov. 2015). *GPU-Based Deep Learning Inference: A Performance and Power Analysis*. [Online]. Available: http://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf

[11] A. Coates, B. Huval, T. Wang, D. J. Wu, and A. Y. Ng, "Deep learning with COTS HPC systems," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2013, pp. 1337–1345.

[12] K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin, "Cost-effective active learning for deep image classification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 12, pp. 2591–2600, Dec. 2017.

[13] J. Hestness, S. W. Keckler, and D. A. Wood, "GPU computing pipeline inefficiencies and optimization opportunities in heterogeneous CPU-GPU processors," in *Proc. IEEE Int. Symp. Workload Characterization*, Oct. 2015, pp. 87–97.

[14] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.

[15] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93 TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 238–239.

[16] L. Cavigelli and L. Benini, "Origami: A 803 GOp/s/W convolutional network accelerator," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Nov. 2017, pp. 2461–2475.

[17] D. Kim, J. Ahn, and S. Yoo, "A novel zero weight/activation-aware hardware architecture of convolutional neural network," in *Proc. IEEE Design Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1462–1467.

[18] J. Wang, S. Park, and C. S. Park, "Optimization of communication schemes for DMA-controlled accelerators," *IEEE Access*, vol. 9, pp. 139228–139247, 2021.

[19] 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.

[20] R. V. W. Putra, M. A. Hanif, and M. Shafique, "DRMap: A generic DRAM data mapping policy for energy-efficient processing of convolutional neural networks," 2020, *arXiv:2004.10341*.

[21] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[22] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2011, pp. 109–116.

[23] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2014, pp. 682–687.

[24] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[25] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2014, pp. 609–622.

[26] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, Feb. 2015, pp. 161–170.

[27] J. Jo, S. Kim, and I.-C. Park, "Energy-efficient convolution architecture based on rescheduled dataflow," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4196–4207, Dec. 2018.

[28] J. Jo, S. Cha, D. Rho, and I.-C. Park, "DSIP: A scalable inference accelerator for convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 605–618, Feb. 2018.

[29] S. Dave, A. Shrivastava, Y. Kim, S. Avancha, and K. Lee, "DMazeRunner: Optimizing convolutions on dataflow accelerators," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 1544–1548.

[30] H.-J. Yang, K. Fleming, M. Adler, F. Winterstein, and J. Emer, "Scavenger: Automating the construction of application-optimized memory hierarchies," in *Proc. Field Program. Log. Appl. (FPL)*, 2015, pp. 1–8.

[31] H. Li, M. Bhargava, P. N. Whatmough, and H.-S.-P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[32] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 97–108.

[33] P. Mantovani, D. Giri, G. D. Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile SoC development with open ESP," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[34] J. Cong, Z. Fang, M. Gill, and G. Reinman, "PARADE: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 380–387.

[35] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using Halide's scheduling language to analyze DNN accelerators," in *Proc. Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, Mar. 2020, pp. 369–383.

[36] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May 2020.

[37] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "TimeLoop: A systematic approach to DNN accelerator evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2019, pp. 304–315.

[38] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN accelerator simulator," 2018, *arXiv:1811.02883*.

[39] L. P. Carloni, K. L. McMillan, and A. L. Sangio-ni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.

[40] Y. Seo, S. Lee, S. Kim, J. Wang, S. Park, and C. S. Park, "Latency-insensitive controller for convolutional neural network accelerators," in *Proc. Int. SoC Design Conf. (ISOCC)*, Oct. 2019, pp. 249–250.

[41] J. Wang, J. Kim, S. Moon, S. Kim, S. Park, and C. S. Park, "Spatial data dependence graph simulator for convolutional neural network accelerators," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Mar. 2019, pp. 309–310.

[42] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1160–1174, Aug. 2021.

[43] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1253–1268, Aug. 2021.

[44] S. Liang, Y. Wang, C. Liu, L. He, H. Li, D. Xu, and X. Li, "EnGN: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Trans. Comput.*, vol. 70, no. 9, pp. 1511–1525, Sep. 2021.

[45] S. Kim, J. Wang, Y. Seo, S. Lee, Y. Park, S. Park, and C. S. Park, "Transaction-level model simulator for communication-limited accelerators," 2020, *arXiv:2007.14897*.

[46] B. Khabbazan and S. Mirzakuchaki, "Design and implementation of a low-power, embedded CNN accelerator on a low-end FPGA," in *Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2019, pp. 647–650.

[47] M. Carreras, G. Deriu, L. Raffo, L. Benini, and P. Meloni, "Optimizing temporal convolutional network inference on FPGA-based accelerators," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 348–361, Sep. 2020.

[48] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "A high-efficiency runtime reconfigurable IP for CNN acceleration on a mid-range all-programmable SoC," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.

[49] F. Spagnolo, S. Perri, F. Frustaci, and P. Corsonello, "Designing fast convolutional engines for deep learning applications," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 753–756.

[50] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.

[51] N. Neves, N. Sebastião, D. Matos, P. Tomás, P. Flores, and N. Roma, "Multicore SIMD ASIP for next-generation sequencing and alignment biochip platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1287–1300, Jul. 2015.

[52] R. F. Molanes, L. Costas, J. J. Rodriguez-Andiña, and J. Farina, "Comparative analysis of processor-FPGA communication performance in low-cost FPSoCs," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 3826–3835, Jun. 2021.

[53] B. Drozdenko, M. Zimmermann, T. Dao, K. Chowdhury, and M. Leeser, "Hardware-software codesign of wireless transceivers on Zynq heterogeneous systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 4, pp. 566–578, Oct. 2018.

[54] V. Dadu, J. Weng, S. Liu, and T. Nowatzki, "Towards general purpose acceleration by exploiting common data-dependence forms," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2019, pp. 924–939.

[55] S. Yin, X. Yao, T. Lu, D. Liu, J. Gu, L. Liu, and S. Wei, "Conflict-free loop mapping for coarse-grained reconfigurable architecture with multi-bank memory," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2471–2485, Sep. 2017.

[56] K. Asifuzzaman, R. S. Verdejo, and P. Radojković, "Enabling a reliable STT-MRAM main memory simulation," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*, Oct. 2017, pp. 283–292.

[57] M. Jagasivamani, C. Walden, D. Singh, L. Kang, S. Li, M. Asnaashari, S. Dubois, D. Yeung, and B. Jacob, "Design for ReRAM-based main-memory architectures," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*, Sep. 2019, pp. 342–350.

[58] S. Ikegawa, F. B. Mancoff, J. Janesky, and S. Aggarwal, "Magnetoresistive random access memory: Present and future," *IEEE Trans. Electron Devices*, vol. 67, no. 4, pp. 1407–1419, Apr. 2020.

[59] A. Azizimazreah and L. Chen, "Polymorphic accelerators for deep neural networks," *IEEE Trans. Comput.*, early access, Jan. 1, 2021, doi: 10.1109/TC.2020.3048624.

[60] Y. Liang, L. Lu, and J. Xie, "OMNI: A framework for integrating hardware and software optimizations for sparse CNNs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1648–1661, Aug. 2021.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, 2012, pp. 1097–1105.

[62] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

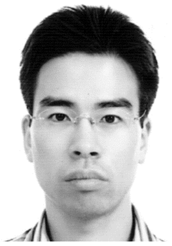
[63] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[64] Y. Kim, W. Yan, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.* vol. 15, no. 1, pp. 45–49, Jan. 2016.

[65] M. Poremba and Y. Xie, "NVMain: An architectural-level main memory simulator for emerging non-volatile memories," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Aug. 2012, pp. 392–397.

JOOHO WANG received the B.S. degree in electronics engineering from Korea Polytechnic University (KPU), Siheung, South Korea, in 2014. He is currently pursuing the M.S./Ph.D. degree in electronics engineering with Konkuk University, Seoul, South Korea. His research interests include hardware/software co-design of programmable accelerators and simulation for SoC architecture.

SUNGKYUNG PARK (Senior Member, IEEE) received the Ph.D. degree in electronics engineering from Seoul National University, South Korea, in 2002. From 2002 to 2004, he was with Samsung Electronics, as a Senior Engineer, where he worked on the development of system-level simulators for cellular standards. From 2004 to 2006, he was with the Electronics and Telecommunications Research Institute (ETRI), as a Senior Member of Research Staff, where he worked on fiber-optic front-end IC design. From 2006 to 2009, he was with Ericsson Inc., as a Senior Staff Hardware Designer, where he worked on the design and modeling of multi-standard RF transceivers and clocking circuits. In 2009, he joined the Faculty of the Department of Electronics Engineering, Pusan National University, South Korea, where he is currently a Professor. His research interests include design and modeling of SoC, hardware accelerators, and virtual platforms for neural networks and 5G.



CHESTER SUNGCHUNG PARK (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 2006. From 2006 to 2007, he was with Samsung Electronics, Giheung, South Korea. From 2007 to 2013, he was with Ericsson Research, USA, as a Senior Engineer. Since 2013, he has been with the Department of Electronics Engineering, Konkuk University, South Korea, as an Associate Professor, where he is working on the design and modeling of SoC, hardware accelerators, and virtual platforms for neural networks and 5G. His research interests include SoC architecture design for artificial intelligence, processing in memory, and wireless communication.