Roofline-Model-Based Design Space Exploration for Dataflow Techniques of CNN Accelerators

CHAN PARK¹, (Member, IEEE), SUNGKYUNG PARK¹, (Senior Member, IEEE), AND CHESTER SUNGCHUNG PARK², (Senior Member, IEEE)

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

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

This work was supported by the Samsung Research Funding and Incubation Center of Samsung Electronics under Project SRFC-IT1802-10. The EDA tool was supported by the IC Design Education Center (IDEC), South Korea.

ABSTRACT To effectively compute convolutional layers, a complex design space must exist (e.g., the dataflow techniques associated with the layer parameters, loop transformation techniques, and hardware parameters). For efficient design space exploration (DSE) of various dataflow techniques, namely, the weight-stationary (WS), output-stationary (OS), row-stationary (RS), and no local reuse (NLR) techniques, the processing element (PE) structure and computational pattern of each dataflow technique are analyzed. Various performance metrics are calculated, namely, the throughput (in giga-operations per second, GOPS), computation-to-communication ratio (CCR), on-chip memory usage, and off-chip memory bandwidth, as closed-form expressions of the layer and hardware parameters. In addition, loop interchange and loop unrolling techniques with a double-buffer architecture are assumed. Many roofline model-based simulations are performed to explore relevant dataflow techniques for a wide variety of convolutional layers of typical neural networks. Through simulation, this paper provides insights into the trends in accelerator performance as the layer parameters change. For convolutional layers with large input and output feature map (ifmap and ofmap) widths and heights, the GOPS of the NLR dataflow technique tends to be higher than that of the techniques. For convolutional layers with low weight and ofmap widths and heights, the RS dataflow technique achieves optimal GOPS and on-chip memory usage. In the case of convolutional layers with small weight widths and heights, the GOPS of the WS dataflow technique tends to be high. In the case of convolutional layers with small ofmap widths and heights, the OS dataflow technique achieves optimal GOPS and on-chip memory usage.

INDEX TERMS Accelerator, convolutional neural networks (CNNs), dataflow techniques, roofline, simulation, processing element (PE), design space exploration (DSE), field-programmable gate array (FPGA).

I. INTRODUCTION Convolutional neural networks (CNNs) have been widely adopted in a variety of deep learning applications, including image and signal processing, object recognition, and computer vision [1]–[5]. However, modern CNNs require significant data movement and computational complexity, which poses great challenges to the power efficiency and performance [23]. The convolutional layer, part of the CNN, is a computationally intensive layer consisting of a large number of multiply-accumulate (MAC) operations. When processing these convolutional layers, the use of general-purpose processors (GPPs) is inefficient in terms of the computational speed, and frequent access to off-chip mem-ory leads to considerable energy consumption. To overcome this challenge, the design of CNN accelerators considering the efficient utilization of memory levels is currently a main topic [25]. Field-programmable gate array (FPGA) based accelerators for reconfigurable systems can be a cost-effective and viable solution to achieve both high energy efficiency and large throughput in CNN implementations [3], [6], [7], [24]. Therefore, this paper selects FPGA-based accelerators covering the broadest possible space of CNN-specific accelerators to the simulation target.
To achieve high performance on CNN-specific accelerators, recently proposed specialized hardware CNN accelerators have exploited parallel processing elements (PEs) to increase throughput. However, state-of-the-art CNNs require significant data movement on-chip and off-chip, which involves more extravagant overhead than computations, so a limited number of PEs is used when designing bandwidth-limited accelerators targeting edge devices. Therefore, minimizing data movement for any CNN is the key to high throughput and meeting the real-time constraints of edge devices. The area and energy efficiency of most CNN accelerators are heavily affected by memory utilization, and the performance is limited by the memory bandwidth [25]. Every MAC operation requires at least three register file (RF) accesses, namely, input, output and weight accesses. To address these challenges, the management of parallel computations and the organization of data storage and access across different levels of memory should be the focus of hardware accelerators to achieve optimal performance under resource constraints. It is essential to consider dataflow techniques that can support highly parallel compute schemes by optimizing the number of data movements from both on-chip and off-chip [12], [26]. CNN dataflow techniques were exploited to reuse data in various ways to improve inefficient energy consumption resulting from the transfer of data between the on-chip buffer and off-chip memory generated to process high-dimensional convolutional layers. In many previous studies, convolutional accelerators targeting edge devices with different dataflows have been introduced [12], [16], [30], [32]. These architectures can successfully reduce the number of memory accesses to the off-chip memory. Among them, representative dataflow techniques are the weight-stationary (WS) [8], [9], output-stationary (OS) [10], [11], row-stationary (RS) [12], [13] and no local reuse (NLR) [14]–[16] dataflow techniques. However, they fail to employ the full potential performance of their architectures due to the limited data bandwidth of devices [31]. The bandwidth bottleneck prevent their architectures from providing the required parallelism for their PEs immediately after each access to the off-chip memory. Additionally, references [16], [19] and [26] use loop transformation techniques to reorganize computations and memory accesses, increasing the performance of CNN accelerators. These applications need to be highly tuned to balance the performance of CNN accelerators with their memory system.

The operational intensity reminds us that data reuse is critical to managing bandwidth use. The simulation we present achieves these goals by proposing the optimal processing dataflow for each convolutional layer. The computational mapping of a given hardware, which optimizes the performance efficiency by maximally reusing data locally to reduce expensive data movement, such as dynamic RAM (DRAM) accesses. To effectively compute convolutional layers within limited hardware resources, detailed design space exploration (DSE) resulting from various combinations of design techniques, such as the proper selection of dataflow techniques, loop tiling or unrolling for parallelism, and interchange of the loop order, is considered. However, evaluating every point of possible combinations is time consuming and expensive. Moreover, the design space of CNN accelerators is not clearly defined, and hence, directly determining key factors of an optimal design space is difficult. There is a need for extensive simulation to quickly and easily search the large design space of representative CNN architectures [25]. As the memory dominates both the performance and power of accelerators, a reasonable first estimate for these can be made by considering mainly the memory. By effectively exploring the complex design space, insights and design guidelines can be obtained for the optimal dataflow technique and architecture of CNN accelerators when certain layer parameters are given. The roofline model applies bottleneck analysis to improve computational performance by modeling the hardware with peak performance and peak off-chip memory bandwidth [17], [27].

This paper presents extensive DSE and closed-form expressions to evaluate the performance of various dataflow techniques based on the roofline model. Through various simulations, the performances of feasible systems are analyzed when the computation resources, sizes of the on-chip buffers block RAMs (BRAMs) in FPGAs, and maximum memory bandwidth constraints between the on-chip buffer and off-chip memory vary in different combinations. In this paper, the following aspects will be depicted in depth as primary works. This paper quantitatively compares and analyzes several dataflow techniques and presents pseudocodes that apply loop transformations, such as changing the loop order, loop unrolling and tiling, to minimize the number of DRAM accesses for each dataflow technique. Based on these analyses, several design parameters, such as the number of PEs, required memory bandwidth, and the number digital signal processing (DSP) slices used in a given FPGA, are used to formulate the performance as a generalized closed-form expression. Based on these equations and pseudocodes, the accelerator for each dataflow technique is modeled. Additionally, this paper adopts some assumptions and runs various simulations to find the optimal dataflow technique for each convolutional layer. The simulation is depicted in Fig. 6. Four primary dataflow techniques, namely, the WS, OS, RS and NLR dataflow techniques, are modeled under the same design frame (Tables 1 and 2) of parameters describing the convolutional layer and the hardware of the accelerators. We envision this simulation where the engineer specifies the optimization techniques and a mapping to hardware in space and time. This simulation will facilitate the rapid exploration of the accelerator design space and provide intuition for analyzing the tendencies of each optimization technique.

The rest of this paper is organized as follows: Section II covers related work. Section III introduces the basics of CNNs, parameters related to convolution operations and parameters to represent the structure of the accelerator. Section IV proposes roofline-based simulations for exploring the design space for CNN accelerators and dataflow techniques and details the modeling methods of each dataflow
technique for simulation. Section V presents simulation experiments and results for exploring the design space of CNN accelerators for processing AlexNet and VGG-11 on the specifications of a Virtex-7 VX485T FPGA board. Finally, Section VI describes the conclusions and future work.

II. RELATED WORK

This section introduces some references that quantitatively analyze computing throughput and required memory bandwidth using loop transformation techniques such as loop tiling and unrolling, various optimization techniques such as dataflow techniques [13], [28], [29] and the roofline model [7], [16], [22], [28]. Tu et al. [29] presented a deep neural architecture (DNA), which can reconfigure its data paths to support dataflow techniques for different layer sizes. To find the optimal dataflow pattern, they simulate the analysis of buffer access and DRAM access. This paper performs a simulation by applying the roofline model to analyze the tendencies and provide an intuitive solution for optimal parameters in terms of the number of giga-operations per second (GOPS) and the computation-to-communication ratio (CCR). Chen et al. [13] proposed the RS dataflow technique to minimize energy consumption due to data movement on a spatial architecture exploiting local data reuse. To evaluate the energy efficiency of the different dataflows, Yu-Hsin Chen et al. proposed an analysis framework to estimate the number of PEs and data accesses to memory in a fixed hardware. In addition, dataflow techniques are incorporated into calculations by quantifying the overall data movement energy cost. As a result, reference [13] shows that the RS dataflow technique is the optimal dataflow under a given architecture. However, the simulation we propose is a much more generalized model, which adopts the roofline model to consider GOPS and the CCR in exploring design spaces. It has wide applicability to a range of architectures. In fact, different optimization results could be obtained for each network and layer, not just with the RS dataflow. Hu et al. [28] proposed a hybrid stationary storage pattern for a CNN accelerator. To identify the optimal dataflow technique for the best performance and minimal bandwidth at the same time, a roofline model is applied to explore the design space of the proposed accelerator. Hu X et al. explored legal combinations of three dataflow patterns, namely, the OS, IS, and WS dataflow patterns, and tiling parameters. However, since the IS dataflow is not used often, this paper models the OS, WS, RS, and NLR dataflow techniques; the RS dataflow technique is suggested in reference [13], and the NLR dataflow technique is more commonly used than the other patterns. To consider the idle time, the “pass” of each dataflow was modeled in detail. In addition, the expected size of BRAMs and SPADs is presented through simulation, and all the equations necessary for hardware DSE are summarized in Table 5 in a closed form. Not only is the memory bandwidth considered, but also, various interpretations of the tendency are presented for each layer. Reference [22] focuses on the number of active PEs constrained by dataflow techniques and PE utilization. Chen et al. [22] used roofline modeling to determine whether a network-on-chip (NoC) has enough bandwidth to transfer data from a global buffer (GLB, on-chip buffer) to PEs. They also focus on analyzing the bottlenecks of NoCs for each type of data, using roofline models of the input feature map (ifmap), weight and output feature map (ofmap), rather than comparing performances of the system according to the dataflow techniques. Reference [7] analyzes loop optimization techniques (loop unrolling, loop tiling, and loop interchanging), numerically characterizes the accelerator architecture using design variables, and quantitatively estimates the performances of accelerators to be implemented with those design variables. Ma et al. [7] estimated the performance indicators of accelerators for various dataflow techniques but did not consider idle cycles, which significantly impact computation cycles. They assumed that the data needed for MAC operations are continuously supplied to the accelerator, so the characteristics of the dataflow techniques are less reflected than the computation unit modeled in this paper. As in reference [16], each PE in the accelerator based on the NLR dataflow technique completes MAC operations for one ifmap pixel and computes convolution operations for adjacent ofmap pixels; it is assumed that there is no data communication between PEs. The computation cycles of this accelerator can be easily formulated by multiplying the number of iterations of the loop iterators without considering PE utilization. However, for other dataflow techniques, unlike the NLR dataflow technique, ofmap pixels processed in parallel in different PE sets must be accumulated in the input channel. Additionally, to model the dataflow technique precisely, it is necessary to consider idle cycles, which is the time it takes for the pixels required in MAC operations to relay and reach each PE. This paper analyzes the computation pattern for each dataflow technique in depth and elaborately formulates the number of operations and computational cycles required to process a specific convolutional layer differently from previous studies.

III. CNN BASICS AND PARAMETERS

CNNs consist of feature extraction and classification. Feature extraction is achieved through convolutional layers, activation functions, and pooling layers, and classification is achieved via a fully connected layer. This paper focuses on exploring the design space of the convolutional layer for feature extraction because this layer is the main target of hardware acceleration due to computationally intensive and repetitive patterns.

A. BASICS OF CONVOLUTIONAL LAYERS

The convolutional layer extracts features from the input image through convolution operations. The convolution operation means generating an ofmap by performing MAC operations while sliding the weight across the image (ifmap). Fig. 1 and Table 1 describe the parameters representing the convolutional layer. The ifmap is composed of three dimensions and is expressed by height (H), width (W) and the number of input channels (C). Weight is composed of
TABLE 1. Parameters of the convolutional layer.

| Parameter | Description                        |
|-----------|------------------------------------|
| C         | Number of ifmap channels (= weight channels) |
| M         | Number of 3D weights (= ofmap channels) |
| H/W       | Height/width of the 2D ifmap         |
| R/S       | Height/width of the 2D weight        |
| E/F       | Height/width of the 2D ofmap         |

Algorithm 1 Basic Pseudocode of the Convolutional Layer

```plaintext
for (e = 0; e < E; e++) { // row of ofmap
    for (f = 0; f < F; f++) { // column of ofmap
        for (m = 0; m < M; m++) { // channel of ofmap
            for (c = 0; c < C; c++) { // channel of ifmap
                for (r = 0; r < R; r++) { // row of weight
                    for (s = 0; s < S; s++) { // column of weight
                        ofmap[m][c][e][f] +=
                                ifmap[c][e + r][f + s]*
                                weight[m][c][r][s];
                    }
                }
            }
        }
    }
}
```

B. DESCRIPTION OF VARIOUS PARAMETERS

Table 1 shows the parameters indicating the shape of the convolutional layer. The practical CNN model is composed of several convolutional layers, and the shape of each layer is different, so the values of the convolutional layer parameters may be different for each layer. Table 2 describes the parameters related to the structure of the accelerator for processing the convolutional layer. The notations of the various parameters in Tables 1 and 2 are from reference [12]. When the accelerators perform MAC operations in the convolutional layer, it loads pixels from off-chip memory (DRAM) to on-chip buffer (GLB) and stores the completed ofmap pixels into DRAM. The timeline for the behavior of the CNN accelerator processing the convolutional layer is divided into the “pass”, as assumed in reference [12].

Within one pass, ifmap and weight pixels stored in the GLB are cast to the PE array in the amount defined by the parameters, and after convolution operations are performed on the PE array, the accumulated ofmap pixels are stored in the GLB. As p, q, r and t change, the number of input channels and the number of output channels processed by the accelerator during one pass vary. As the values of p and q (mapping of parameters in the same PE set) increase, the scratchpad (register) memory size in the PEs and the GLB size increase, and the number of 2D ifmaps and weights processed during one pass increases. As parameters in the same PE set increase, the time required for one pass increases, the number of pixels that load and store between DRAM and the GLB during one pass increases, and the total number of passes to process one layer decreases. As the parameters r and t (unrolling and tiling parameters, respectively) values increase, additional PE sets are used to process input and output channels in parallel. Additional BRAMs are required to simultaneously supply the ifmap, the weight pixels of different input channels and the weight pixels for different output channels to the PE array. Unlike p and q, if the r and t values increase, the parallelism of the accelerator is increased by using additional DSP slices, so the number of cycles for MAC operations to be processed during one pass does not increase. As r and t increase, the total number of passes to process a convolutional layer decreases. The constraints for the hardware parameters p, q, r, and t are given in (1). The total number of passes for processing a convolutional layer is represented by Equation (2). The parameters r and t are related to the number of PE sets and the number of GLBs (BRAMs).

![Diagram](image-url)

**FIGURE 1. Structure of a convolutional layer.**

**TABLE 2. Hardwae parameters of the accelerator.**

| Parameter | Description                        |
|-----------|------------------------------------|
| p         | Number of 3D weights 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 3D weights in the PE array |

\[
\begin{align*}
0 < q \times r & \leq C \\
0 < p \times t & \leq M \\
\text{Totalpasses} & = \left\lfloor \frac{C}{q \times r} \right\rfloor \times \left\lfloor \frac{M}{p \times t} \right\rfloor 
\end{align*}
\]
FIGURE 2. Example of a PE array block diagram.

FIGURE 3. Example of an adder tree block diagram.

FIGURE 4. Block diagram of an FPGA-based accelerator system.

FIGURE 5. Basic roofline model.

GLBs to supply data to the accelerator for parallel processing. Ofmap pixels with different input channels processed during one pass are accumulated in the same output channels. As shown in Fig. 2, ofmap pixels with the same t index are stored in an ofmap GLB with the corresponding t index through the adder tree. The structure of the adder tree is described in Fig. 3. This module accumulates ofmap pixels from different input channels generated from PE sets for the same output channel. The number of cycles it takes for the r ofmap pixels to accumulate through the adder tree is \( \lceil \log_2 r \rceil \) cycles. Fig. 4 shows a block diagram of a system with an accelerator. In this paper, the PE array structure and whether the adder tree is included in the system differ according to the dataflow technique. However, the same system architecture as in Fig. 4 is assumed for the other parts regardless of the dataflow technique.

IV. ROOFLINE-BASED DATAFLOW SIMULATION

A. INTRODUCTION TO THE SIMULATION

The roofline model is an analysis model that can intuitively explore the performance of a system in terms of the hardware and software design [17]. The roofline model can allow engineers to intuitively search for the optimal point by exploring the ratio of the CR to the performance. References [17] and [27] apply bottleneck analysis to computational performance by modeling the hardware with peak performance and peak off-chip memory bandwidth. This paper evaluates the performance of the system on a chip (SoC) including a CNN accelerator through a roofline model. The performances of FPGA-based systems, including CNN accelerators, are limited by two major constraints: the FPGA resources (DSP slices) for the accelerator to process MAC operations in parallel and the memory bandwidth between the external memory and accelerator. Based on these constraints, a roofline model representing the performance of the CNN accelerator can be plotted on a 2D graph, as shown in Fig. 5. The x-axis represents the ratio of the total amount of computations processed by the accelerator to the amount of data load and stored between off-chip DRAM and GLBs (on-chip buffers). The x-axis is the CCR. The unit of the CCR is OP/byte, and the larger this value is, the higher the reuse of data from external memory. The CCR can be expressed as in Equation (3).

\[
\text{CCR} = \frac{\text{total number of multiplication and addition operations}}{\text{bytes of DRAM access}}
\]  

The y-axis represents the number of operations per cycle processed by the accelerator. The y-axis value represents the performance, and the unit is GOPS. The performance is represented by Equation (4).

\[
\text{Performance} = \frac{\text{total number of multiply, add operations}}{\text{computation cycles for total passes}}
\]  

The maximum value of performance is computationally bound. The CCR and performance for each dataflow technique can be expressed by equations, which are described in
section IV B. The DRAM bandwidth (BW) required for the system to be implemented can be obtained as in Equation (5) through the ratio of CCR and performance.

\[
\text{required BW} = \frac{\text{Performance}}{\text{CCR}} = \frac{\text{GOP/s}}{\text{OP/byte}} = \frac{\text{GB/s}}{} \tag{5}
\]

The maximum memory bandwidth between the off-chip and on-chip memory that can be provided by the system is defined as a memory bound. In the roofline model, if the slope between the origin and the plotted point is higher than the memory bound, that point cannot actually be implemented. The memory bound can be represented by Equation (6). FP is the clock frequency of the port (i.e., an I/O port for communicating between the on-chip and off-chip memory). BP means the bit width of the port, and NP means number of ports.

\[
\text{memory bound} = \text{FP} \times \text{BP} \times \text{NP} \tag{6}
\]

Fig. 6 shows the simulation flow for the dataflow techniques. The inputs for the simulation are the hardware specifications, parameters related to the convolutional layer, and features of the dataflow technique. The hardware specifications include the resources of the FPGA to implement the CNN accelerator, the total number of FP, BP, NP, and DSP slices, and the total number of BRAMs. In section IV B, the features of the dataflow techniques are modeled, and how to obtain various performance metrics of the dataflow techniques is described.

B. MODELING OF THE DATAFLOW TECHNIQUES

In this section, the features of each dataflow technique are numerically modeled on the required form of roofline-based simulation. The characteristics of the four representative dataflow techniques (WS, OS, RS, and NLR) are described. For each dataflow technique, we generalize the number of PEs required for the accelerator, the execution cycles of the accelerator required to process the convolutional layer, the total number of operations, and the total number of DRAM access bytes.

1) WEIGHT-STATIONARY (WS) DATAFLOW TECHNIQUE

The WS dataflow technique increases the reuse of the weight pixels by storing the weight pixels in scratchpad (SPAD) memory when performing MAC operations. Ifmap pixels and weight pixels used for operation during one pass are stored in the corresponding GLBs. In each cycle, ifmap pixels are broadcast from the GLBs to the PEs that process the MAC operations in that cycle. In the PE array, partial sums (psum) of the ofmap for t output channels are accumulated by relaying in each PE set. Ofmap pixels that reach the last relay point of each PE set are accumulated for different input channels through the adder tree and then stored in the ofmap GLB of the corresponding t index. Algorithm 2 is the pseudocode for the WS dataflow technique. The loop iterators m and c increase when the behavior of the accelerator for that pass is complete. m and c are increased by tileM and tileC is \( q \times r \). The product of m and c iterations is the total number of passes. The data access is minimized to store from ofmap GLB to DRAM by relocating iterator c to the inner loop rather than iterator m to prioritize the accumulation of the input channels. Loop iterators tm and tc are repeated p and q times, respectively, and as p and q increase, the pass time increases, and the number of input channels and output channels processed in 1 pass increase. Loop iterators h and w indicate that ifmap is broadcasted from the ifmap GLB to the PE array by 1 pixel. The four unrolled loops are for accelerator architectures that increase the parallelism by using more MAC units. The loop iterators rr and ss are unrolled for one PE set consisting of R \( \times S \) concurrently running PEs, which is the size of the 2D plane weight. The PE array is composed of these \( r \times t \) PE sets through unrolled loop iterators mm and cc. Thus, the WS dataflow accelerator consists of a total of \( R \times S \times r \times t \) PEs. The PEs that actually perform computations are determined through functions (compute_data_type_index) that specify the index of the PEs that perform MAC operations in the corresponding cycle. Fig. 7 shows the

Algorithm 2 Pseudocode of the WS Dataflow Technique

\[
\text{for} (m = 0; m < M; m + = \text{tileM}) \{
\text{for} (c = 0; c < C; c + = \text{tileC}) \{
\text{for} (tm = m; tm < m + tileM; tm + = t) \{
\text{for} (tc = c; tc < c + tileC; tc + = r) \{
\text{for} (h = 0; h < H; h + +) \{
\text{for} (w = 0; w < W; w + +) \{
\text{compute_weights_index}(r_en\^*; s_en\^*);
\text{compute_ofmap_index}(e_en\^*; f_en\^*);
\text{for} (mm = tm; mm < tm + t; mm + +) \{
\text{ LOOP UNROLLING }
\text{for} (cc = tc; cc < tc + r; cc + +) \{
\text{ LOOP UNROLLING }
\text{for} (rr = 0; rr < R; rr + +) \{
\text{ LOOP UNROLLING }
\text{ofmap}[mm][e_en][f_en] + = \text{ifmap}[cc][h][w]^* \text{weight}[mm][cc][r_en][s_en];
\} \} \} \} \} \}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\]

FIGURE 6. Flow of the dataflow technique simulation based on a roofline model.

set. Ofmap pixels for different input channels computed from \( r \) PE sets are accumulated through the adder tree. Since there are \( t \) of these PE sets for different output channels, the input channel accumulations are calculated as in Equation (7).

number of multiply, add operations for 1 pass
\[
= 2 \times \text{MAC operations} + \text{input channel accumulations}
\]

where

\[
\text{MAC operations} = (E \times F \times p \times t) \times (R \times S \times q \times r)
\]

input channel accumulations
\[
= E \times F \times r \times t \times p
\]

computation cycles for 1 pass
\[
= H \times W \times q \times r + \lceil \log_2 r \rceil
\]

Equation (8) is formulated with the assumption that \( H \times W \) is greater than \( R \times S \). As the loop iterator \( tc \) increases, the time that ofmap pixels of different input channels accumulate in the adder tree can be pipelined, so only the cycles \( \lceil \log_2 r \rceil \) needed to accumulate the last input channel of 1 pass must be considered. The loop iterator \( tc \) takes \( H \times W \) cycles each iteration, and \( tc \) iterates \( p \times q \) times within 1 pass.

bytes of DRAM access = \( \text{ifmap}_{\text{load}} + \text{weight}_{\text{load}} + \text{ofmap}_{\text{store}} \)

where

\[
\text{ifmap}_{\text{load}} = H \times W \times q \times r \times \text{total passes}
\]

\[
\text{weight}_{\text{load}} = R \times S \times q \times r \times p \times t \times \text{total passes}
\]

\[
\text{ofmap}_{\text{store}} = E \times F \times M
\]

To process a convolutional layer with Equations (9)-(12), the total access is expressed as load and store data between DRAM and the GLB. As shown in the pseudocode of each dataflow technique, the loop order is rearranged to minimize DRAM access, so the bytes of DRAM access are the same regardless of the dataflow technique.

2) OUTPUT-STATIONARY (OS) DATAFLOW TECHNIQUE

The OS dataflow technique stores ofmap pixels in the SPAD memory inside each PE as performing the convolution operation so that ofmap pixels stay in each PE and increase the reuse of the partial sum. The ifmap pixels transmitted from \( r \) ifmap GLBs are multicast to the PE at the first index of the \( t \) PE sets that process different output channels, as shown in Fig. 2. Ifmap pixels are computed in the PE for different \( p \) output channels and relayed through adjacent PEs. Weight pixels are broadcast to PEs when the ifmap pixels that need to be convolved with the corresponding weight pixel reach the appropriate PE inside the PE set, as shown in Fig. 10. When all the weight pixels to be processed during 1 pass are broadcast to PEs, the convolution operations of all the PE sets for the \( t \) output channels are completed simultaneously.

In the case of a pass when accumulation is completed for all input channels, the PEs of each PE set transmit the
Algorithm 3 Pseudocode of the OS Dataflow Technique

```plaintext
for (m = 0; m < M; m += tileM) {
    for (c = 0; c < C; c += tileC) {
        for (tc = c; tc < c + tileC; tc += r) {
            for (rr = 0; rr < R; rr++) {
                for (ss = 0; ss < S; ss++) {
                    wait_ifmap_alignment();
                    for (tm = m; tm < m + tileM; tm += t) {
                        for(mm = m; mm < tm + t; mm++) {
                            LOOP UNROLLING
                            for (cc = tc; cc < tc + r; cc++) {
                                LOOP UNROLLING
                                for (e = 0; e < E; e++) {
                                    LOOP UNROLLING
                                    for (f = 0; f < F; f++) {
                                        LOOP UNROLLING
                                        ofmap[mm][cc][e][f] +=
                                        ifmap[cc][e + rr][f + ss]*
                                        weight[mm][cc][rr][ss];
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

completed p ofmap pixels by 1 pixel to the adder tree. Algorithm 3 is the pseudocode of the OS dataflow technique. The loop iterators m, c, tm, and tc have the same meaning as in the WS dataflow technique. In contrast to the other dataflow techniques, the loop iterator tm is placed in the inner loop. The reason is that when the ifmap pixels reach the appropriate PEs, the ifmap’s relay is temporarily stopped for reuse of the ifmap pixels, and the weight pixels for the p output channels are broadcast to the PEs. The loop iterators rr and ss indicate that the weight is broadcast from the weight GLB to the PE array by 1 pixel. The wait_ifmap_alignment function makes the weight broadcast wait until the ifmap pixels are arranged in the proper PEs for the convolution operation. The unrolled loop iterators e and f indicate that the PE set composed of E × F PEs operates concurrently, and it can be seen through loop iterators mm and cc that the PE array consists of r × t PE sets. Fig. 9 shows the PE architecture for the OS dataflow technique. The PE has one adder and one multiplier. Each PE has a SPAD for the ofmap pixels, and the size of the SPAD is p. In a PE, there is a register in which the ifmap pixels received from adjacent PEs are stored. Similar to the WS dataflow technique, a FIFO for the delay is required for each row of the PE set so that ifmap pixels to be relayed are accurately MAC-operated, and the size of this FIFO is W − F. The OS dataflow accelerator consists of a total of E × F × r × t PEs. Fig. 10 shows an example of the structure of a PE set for the OS dataflow technique.

![FIGURE 9. PE for the OS dataflow technique.](image)

![FIGURE 10. Example of a PE set for the OS dataflow technique.](image)

The computation cycles for 1 pass

\[
= (H \times W + (p - 1) \times R \times S) \times q
\]

(13)

where \((p - 1) \times R \times S\) is the time it takes to broadcast the weight pixels for p output channels to reuse the ifmap pixels. Each time loop iterator m increases, it takes an additional \(E \times F \times (p - 1) + \lceil \log_2 r \rceil\) cycles, which is the time it takes for the completed ofmap pixels to be stored in the ofmap GLBs through the adder tree.

3) ROW-STATIONARY (RS) DATAFLOW TECHNIQUE

The RS dataflow technique stores the ifmap pixels, weight pixels, and ofmap pixels in each scratchpad memory inside the PE and reuses those pixels. Ifmap pixels are multicast to PEs arranged diagonally in the PE set. Weight pixels are multicast to PEs in the row direction of the PE set. Ofmap pixels are entered into PEs in the first row of each PE set. Each PE transmits the psum that has computed all the MAC operations of the weight row for q input channels to the adjacent PE in the next row. Ofmap pixels computed from the last row PEs of each PE set are stored in the ofmap GLB through the adder tree; p ofmap rows are processed for 1 pass from the PEs located in the PE set column.
Algorithm 4 Pseudocode of the RS Dataflow Technique

```
for (m = 0; m < M; m += tileM) {
    for (c = 0; c < C; c += tileC) {
        for (tc = c; tc < c + tileC; tc += r) {
            for (tm = m; tm < m + tileM; tm += t) {
                for (w = 0; w < W; w++) {
                    for (h = 0; h < H; h++) {
                        compute_ifmap_index(h_en*, w_en*);
                        compute_weights_index(r_en*, s_en*);
                        compute_ofmap_index(e_en*, f_en*);
                        for (mm = tm; mm < tm + t; mm++) {
                            LOOP UNROLLING
                            for (cc = tc; cc < tc + r; cc++) {
                                LOOP UNROLLING
                                for (rr = 0; rr < R; rr++) {
                                    LOOP UNROLLING
                                    for (ee = 0; ee < E; ee++) {
                                        ofmap[mm][e][f] = ifmap[cc][h][w]*weight[mm][cc][r][s];
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

**FIGURE 11. PE for the RS dataflow technique.**

Algorithm 4 is the pseudocode of the RS dataflow technique. Loop iterators $h$ and $w$ indicate that the `ifmap` and `weight`, respectively, are multicast from the GLBs to the PE array. The function (compute_data_type_index) determines the PEs to be multicast to `ifmap` and `weight` pixels and calculates the index of the PE that performs a MAC operation at the corresponding cycle. Unlike the WS and OS dataflow techniques, the `ifmap` and `weight` pixels of the RS dataflow technique are cast in the column direction before being cast in the row direction.

The unrolled loop iterators `rr` and `ee` indicate that the PE set composed of $R \times E$ PEs operates concurrently, and the PE array consists of $r \times t$ PEs sets through loop iterators `mm` and `cc`. Fig. 11 shows the PE structure of the RS dataflow technique. The PE has 1 adder, 1 multiplier, and 1 multiplex (MUX) operator. Each PE has SPADs for the `ifmap`, `weight`, and `ofmap` pixels, and the size the SPADs are $S \times q$, $S \times q \times p$, and $p$, respectively. Additionally, similar to the PE of the OS dataflow technique, there is a register in which the psum is temporarily stored before transferring it to the `ofmap` SPAD. The RS dataflow technique does not require additional FIFOs for delay. The RS dataflow accelerator consists of a total of $R \times E \times r \times t$ PEs. Fig. 12 shows the PE set in the case of a $3 \times 3$ `ifmap`, a $2 \times 2$ `weight`, and a $2 \times 2$ `ofmap`. `Ifmap` rows 0-2 are assigned to the PE group in the diagonal direction of the PE set, and `weight` rows 0-1 are assigned to the PE group in the horizontal (row) direction of the PE set. `Ofmap` rows 0-1 are processed in the PE group in the vertical (column) direction of the PE set. As with other dataflow techniques, the RS dataflow technique assumes the same loop order (iterators $m$ and $c$) associated with the `pass` to minimize DRAM data access and uses an adder tree for the efficient accumulation of the input channels. Therefore, the number of multiplication and addition operations and bytes of DRAM access are the same as those of the WS and OS dataflow techniques. The computation cycles for 1 pass of the RS dataflow technique can be divided into cases (14) and (15) as the parameters change. To distribute pixels evenly to the PE set, the `ifmap` pixels are cast in column direction and then cast in the direction of the input channels. Likewise, the `weight` pixels are cast in the column direction, then in the order of the output channels, then in the order of the input channels.

```
if ($H \times S \times q < R \times S \times p \times q$)
    computation cycles for 1 pass
    $= R \times S \times p \times q + S \times (W-S) \times p \times q$
    $+ E \times p + \lceil \log_2 r \rceil$ (14)
if ($H \times S \times q \geq R \times S \times p \times q$)
    computation cycles for 1 pass
    $= H \times W \times q + E \times p + \lceil \log_2 r \rceil$ (15)
```

$H \times S \times q$ of cases (14) and (15) is the number of cycles it takes to multicast the `ifmap` pixels before the `ofmap` pixels are first transferred to the adder tree from the PE set within 1 pass. Similarly, $R \times S \times p \times q$ is the number of cycles it takes to multicast the `weight` pixels required for MAC computation.
before the ofmap pixels are sent to the adder tree. For \( q \) ifmap columns, among the cycles required to process the MAC operation and the cycles required to transfer the pixels required for the MAC operation to the PE set, the former is the bottleneck in case (14), and the latter is the bottleneck in case (15). \( S \times (W - S) \times p \times q \) of case (14) is the number of cycles it takes to compute the remaining \( q \times (W - S) \) ifmap columns other than the \( S \) ifmap columns cast simultaneously while the weight pixels are multicast. The PE of the RS dataflow technique processes the MAC operation for \( p \times q \) weight rows, which is the product of temporal parameters \( p \) and \( q \), to accumulate 1 ofmap pixel during 1 pass. The MAC computation for \( q \) ifmap columns and communication for transmitting accumulated ofmap pixels from the column group of the PE set to the adder tree can be pipelined. Additionally, the PEs in the column group concurrently compute the MAC operations for the psum of the same ofmap index. Therefore, processing the MAC operation of \( q \) ifmap columns takes \( S \times p \times q \) cycles. As the cycles of sending ofmap pixels to the GLB are pipelined with the MAC computational time, only the equation \( E \times p + \lceil \log_2 r \rceil \) is considered as the cycles of transferring the pixels of the last ofmap column to the adder tree. \( E \times p \) is the number of cycles needed for each of the \( E \) columns of the PE set to transfer the accumulated \( p \) psums of the ofmap SPAD to the adder tree. \( \lceil \log_2 r \rceil \), which, as described previously, is the number of cycles it takes for input channel accumulations. In case (15), more cycles are required to cast the ifmap to the PE set than to compute the MAC operations. Thus, when modeling computational cycles of the RS dataflow technique, Equation (15) can be simplified by using \( H \times W \times q \), which is the number of ifmap pixels processed during 1 pass.

4) NO LOCAL REUSE (NLR) DATAFLOW TECHNIQUE

Unlike the WS, OS, and RS dataflow techniques, the NLR dataflow technique does not use scratchpad memory for the local reuse of each data type. Instead of using SPAD, NLR dataflow-based accelerators use GLBs to store the ifmap, weight, and ofmap pixels. Every cycle, the weight pixels are unicast to the designated multiplier, and the ifmap pixels are multicast to multipliers that compute the same input channel. Algorithm 5 is the pseudocode of the NLR dataflow technique. Unlike the other dataflow techniques, there are two unrolled loops, and the NLR dataflow accelerator consists of one PE set. Loop iterators \( mm \) and \( cc \) indicate that the size of the PE set, which is computed concurrently, is \( r \times t \). The loop iterators \( m \) and \( c \) indicate the progress of the pass, and the inner loops rather than the \( c \) loop indicate the operation of the accelerator within one pass. As with reference [16], the loop order is rearranged within 1 pass to minimize DRAM access. Fig. 13 shows an example of a PE set for the NLR dataflow technique when \( r = 2 \) and \( t = 3 \). The PE set consists of \( r \times t \) MAC units. The PE set receives data for the convolution operation from the GLB instead of the SPAD. Unlike the other dataflow accelerator, it is independent of the parameters of the convolutional layer. The hardware parameters \( r \) and \( t \) determine the number of PEs (MAC units) and the GLBs of the NLR dataflow accelerator. The convolutional layer parameters and temporal parameters \( p \) and \( q \) determine the size of the GLBs. In the ifmap GLB, ifmap pixels are multicast; in the weight GLB, weight pixels are unicast. Unlike the other dataflow techniques, the accelerator based on the NLR dataflow technique consists of one PE set, so there is no need for additional cycles for input channel accumulations, as in Equation (16). The computation cycles for 1 pass of the NLR dataflow technique can be derived from Algorithm 5. The number of computational cycles is the product of the number of iterations of the remaining loops, excluding the unrolled \( mm \) and \( cc \) loops, and the \( m \) and \( c \) loops, which indicate the progress of the pass, as in Equation (17).

\[
\text{number of multiplication and addition operations for 1 pass} = 2 \times \text{MAC operations} \\
= 2 \times (E \times F \times p \times t) \times (R \times S \times q \times r) \\
= E \times F \times R \times S \times p \times q
\]  

Table 5 summarizes the equations for the modeling performance indicators of the dataflow techniques.

V. EXPERIMENT

A. EXPERIMENTAL ASSUMPTIONS

This paper uses the specifications of the Virtex-7 VX485T FPGA as the input of the roofline-based simulation. The maximum memory bandwidth of the Virtex-7 VX485T is assumed to be 4.5 GB/s, as mentioned in reference [16]. The maximum number of DSP slices is 2800, and the number of 18-KB BRAM slices is 2080. The accelerator uses GLBs (BRAMs) for each datatype to perform \( r \times t \) PE sets concurrently, and it is assumed that these GLBs are large enough for double buffering. The number of DSP slices used to implement one PE depends on the bit-width of the pixel that the PE computes and whether it is a floating point or a fixed point. In this experiment, ifmap, weight, and ofmap pixels for the CNN are assumed to be 32-bit fixed points. The number of DSP slices needed for one PE to process a 32-bit fixed point is four, as shown in Table 3. The clock
frequency of the accelerator is 100 MHz. The double-buffer structure for overlapping computation and communication from reference [16] is used. During the computation of the nth pass, IP such as that with a direct memory access (DMA) controller loads ifmap pixels and weight pixels from off-chip memory to the GLBs to be used for the MAC operations in the (n + 1)th pass. This paper focuses on exploring the design space of the accelerator. Therefore, the computation is limited when it takes longer to compute MAC operations than to transfer pixels between DRAM and the GLBs. Because the system has a double-buffer structure and is computationally limited, the cycles required to process the convolution operation for a convolutional layer can be obtained by multiplying the total number of passes by the computation cycles of 1 pass. The experiment is carried out by Conv1 and Conv2 of the AlexNet model.

### B. RESULTS OF SIMULATION

#### 1) AlexNet

For the WS dataflow in AlexNet convolutional layer 1 (Conv1), the height and width of ifmap, weight, and ofmap are larger than those of Conv2-5. The size of the PE set of other dataflow techniques, except for the NLR dataflow technique, is proportional to the layer parameters. When implementing the WS, OS, and RS dataflow accelerators in FPGAs with limited DSP slices, the number of PE sets for parallel processing is limited compared to the NLR dataflow. Therefore, for Conv1, the NLR dataflow technique has higher GOPS than the other dataflows and has an optimal implementation point. In the case Conv2 of AlexNet, the height and weight of the weights are relatively smaller than the height and width of the input and output feature maps. Therefore, the WS dataflow technique can use more PE sets for parallel processing than the OS and RS dataflows and can achieve high GOPS. As the PE set size of the WS dataflow technique is dependent on \( R \times S, r \times t \) is smaller than the NLR dataflow technique, and the number of required BRAMs is less than that of the NLR dataflow technique. If the main purpose of the system is to increase the GOPS, the NLR dataflow technique has an optimal design point, and if it is important to use less on-chip buffer memory, the WS dataflow technique has an appropriate design point. Fig. 14(a)-(d) depicts all of the possible solutions of dataflow techniques and parameters (p, q, r, t) for Conv2 of the AlexNet model. The x-axis denotes the CCR (OP/byte), and the y-axis denotes the attainable performance (GOPS). The slope of the blue line between any point and the origin point (0, 0) is 4.5 GB/s as the maximum memory bandwidth of the Virtex-7 VX485T board or denotes the minimal bandwidth required to achieve a specific performance point. The black dots are the results of sweeping the p, q, r, and t layer parameters. The red circle is a design point with a higher required bandwidth instead of using fewer on-chip buffers, and the green circle is a design point with a high CCR and less bandwidth instead of using additional on-chip buffers. By examining the red circle and green circle, designers can identify where the performance of the CNN operation is limited. For AlexNet Conv3 and 4, the height and width of the input and output feature maps are reduced through Conv1 and Conv2, so the RS dataflow technique can have an optimal design point. The RS dataflow technique requires a larger register size and more BRAM than the WS dataflow technique, but the GOPS of the RS dataflow technique is higher than that of the WS dataflow method. Although the GOPS of the NLR dataflow technique is still higher than that of the other dataflows, the number of on-chip buffers (BRAMs) is insufficient, and the RS dataflow technique is an alternative. Fig. 15 depicts all of the possible points of the AlexNet Conv1-5 DSE. In the case of AlexNet Conv5, E and F are larger than the R and S of weight, so the accelerator of the OS dataflow technique uses fewer PE sets than the WS data flow technique, but the OS dataflow-based accelerator achieves proper performance and requires low memory bandwidth. Because the ofmap pixels of the OS dataflow technique reside inside the PE and input channel accumulations are processed without psum relaying, the number of MAC operations per cycle are higher than that of the WS and RS dataflow techniques. Thus, when implementing an accelerator that uses fewer hardware resources (DSP slices, BRAMs) as a design goal, the OS dataflow technique can have an optimal design point.

#### C. VGGNET-11

The weight of VGG-11 is \( 3 \times 3 \). Similar to Conv1 of AlexNet, for Conv1 and Conv2 of VGG-11, the height and width of the input and output feature maps are large. Therefore, a large number of BRAMs to implement the GLB is required, and there is a limit on the number of PE sets, so the NLR dataflow-based accelerator has a low GOPS. Similarly, the size of the PE set for the OS and RS dataflow techniques is related to the
TABLE 5. Equations of various dataflow techniques.

| Layer Parameter | WS dataflow technique | OS dataflow technique | RS dataflow technique | NLR dataflow technique |
|-----------------|-----------------------|-----------------------|-----------------------|------------------------|
| No. of PEs | $R \times S \times r \times t$ | $E \times F \times r \times t$ | $R \times E \times r \times t$ | $r \times t$ |
| No. of BRAMs (18 KB) | | | | | 
| Register size (pixels) | | | | | 
| SPADs (ifmap) = 0 | SPADs (weight) = $R \times (S-1) \times r \times t$ | SPADs (ofmap) = $(W-S) \times (R-1) \times r \times t$ | SPADs (local link) = $(W-F) \times (E-1) \times r \times t$ | |
| SPADs (weight) = $R \times S \times p \times q \times r \times t$ | SPADs (ofmap) = $E \times F \times p \times r \times t$ | SPADs (weight) = $S \times q \times R \times E \times r \times t$ | SPADs (ofmap) = $p \times R \times E \times r \times t$ | |
| SPADs (ofmap) = $R \times (S-1) \times r \times t$ | SPADs (local link) = $(W-S) \times (R-1) \times r \times t$ | SPADs (weight) = $S \times q \times R \times E \times r \times t$ | SPADs (ofmap) = $p \times R \times E \times r \times t$ | |
| SPADs (local link) = $(W-S) \times (R-1) \times r \times t$ | |

FIGURE 14. Roofline graphs of Conv2 of AlexNet.

layer parameter $E$, so the number of PE sets of the accelerator is limited. For the above reasons, in the case of Conv1 and Conv2 VGG-11, the WS dataflow-based accelerator, where the size of the PE set depends on the height and width of the weight, achieves a high GOPS due to the high parallelism. In the case of Conv3 and Conv4, the height and width of the ifmap and weight are shorter than those of layers 1 and 2, and more MAC units for parallel processing of the NLR dataflow technique can be used. The GOPS of the NLR dataflow technique, which is independent of the number of PE sets and layer parameters, is the highest compared to the other dataflow techniques. Fig. 16 depicts all of the possible points of the VGGNET-11 Conv1-5 DSE. In Conv5 and Conv6, the height and width of ifmap and ofmap are reduced, and RS dataflow can be a candidate for optimal accelerator implementation. The RS dataflow technique requires more
TABLE 6. Results of simulation (AlexNet, VGG-11).

| Layer            | Optimal dataflow technique | Hardware parameters \([r, t, p, q]\) | DSP | BRAM 18 KB | Register Size (KB) | CCR (OP/DRAM byte access) | GOPS | Required Bandwidth (GB/s) | Cycles \((\times 1000)\) |
|------------------|----------------------------|---------------------------------------|-----|------------|-------------------|-----------------------------|------|---------------------------|-------------------------|
| AlexNet Conv1    | NLR                        | 3, 48, 1, 1                           | 144 | 1326       | -                 | 83.08                       | 140  | 0.346                     | 366                     |
| AlexNet Conv2    | NLR                        | 35, 20, 1, 1                          | 700 | 920        | -                 | 95.65                       | 140  | 1.465                     | 109                     |
| AlexNet Conv3    | RS                         | 6, 3, 5, 38                           | 702 | 360        | 1892.1            | 30.79                       | 120.89 | 3.925                     | 112                     |
| AlexNet Conv4    | RS                         | 9, 2, 6, 12                           | 702 | 188        | 710.2             | 26.77                       | 119.2 | 4.45                      | 106                     |
| AlexNet Conv5    | NLR                        | 28, 25, 5, 4                          | 700 | 912        | -                 | 65.72                       | 140  | 2.13                      | 61                      |
| VGG-11 Conv1     | WS                         | 3, 7, 1, 1                            | 189 | 1967       | 37.49             | 9.81                        | 39.19 | 3.99                      | 452                     |
| VGG-11 Conv2     | WS                         | 19, 4, 2, 1                           | 684 | 1402       | 71.25             | 32.11                       | 139.22 | 4.34                      | 1183                    |
| VGG-11 Conv3     | WS                         | 77, 1, 8, 1                           | 693 | 1087       | 54.14             | 32.84                       | 135.62 | 4.13                      | 1493                    |
| VGG-11 Conv4     | NLR                        | 70, 10, 1, 1                          | 700 | 1640       | -                 | 39.04                       | 140  | 3.58                      | 2340                    |
| VGG-11 Conv5     | NLR                        | 140, 5, 2, 1                          | 700 | 1290       | -                 | 32.43                       | 140  | 4.32                      | 971                     |
| VGG-11 Conv6     | NLR                        | 140, 5, 2, 1                          | 700 | 1280       | -                 | 31.66                       | 140  | 4.42                      | 1619                    |
| VGG-11 Conv7     | NLR                        | 11, 63, 8, 3                          | 693 | 841        | -                 | 30.89                       | 138.6 | 4.49                      | 221                     |
| VGG-11 Conv8     | OS                         | 1, 13, 34, 1                          | 468 | 105        | 64.49             | 18.53                       | 83.21 | 4.48                      | 186                     |

Fig. 16 depicts all of the possible points of the VGG-11 Conv1-8 DSE. Due to the reduction in E and F in Conv7-8, an accelerator based on the OS dataflow technique can also use the PE set for parallel processing. Therefore, the OS dataflow technique with higher PE utilization than the WS and RS dataflow techniques can have an optimal design point. The proposed design point of the OS dataflow technique for Conv7 has a lower GOPS than the NLR dataflow technique, but the OS dataflow technique uses fewer BRAMs and requires less memory bandwidth. However, as \( m \) and \( c \) increase, \( CCR \) decreases and the required memory bandwidth tends to increase, so the NLR dataflow-based accelerator cannot utilize DSP slices efficiently. Therefore, for Conv8, the OS dataflow technique that requires a relatively low memory bandwidth has a higher GOPS than the NLR dataflow technique and has an optimal design point.

Table 6 describes the dataflow technique with optimal design points for each convolutional layer and the corresponding hardware parameters and simulation results. Through the results of this table, different combinations of hardware parameters and types of dataflow techniques for the optimal performance of the accelerator are confirmed on each convolutional layer. For convolutional layers when the height and width of the \( ifmap \) and \( ofmap \) are relatively large, the NLR dataflow technique for which the size of the PE set is independent of the layer parameters has an optimal design point. For convolutional layers where the \( R \) and \( S \) of the \( weight \) are significantly less than the height and width of the \( ifmap \) and \( ofmap \), the WS and NLR dataflow techniques are candidates for the optimal dataflow technique. The NLR dataflow technique has a high GOPS, but the WS dataflow technique achieves a proper GOPS and has low BRAM requirements. Through the convolutional layers, the height and width of the \( ifmap \) and \( ofmap \) are gradually shortened, and the RS dataflow technique becomes a balanced dataflow technique with appropriate GOPS and BRAM requirements. As the end of the convolutional layer gets closer, the \( E \) and \( F \) of the \( ofmap \) are significantly reduced, and the OS dataflow technique with higher MACs per cycle and lower required
memory bandwidth is the optimal dataflow technique for accelerator implementation.

VI. CONCLUSION

To efficiently process the computationally intensive convolutional layers of CNNs, various customized accelerators, dataflow techniques, and loop transformation techniques are proposed. However, exploring a complex design space for the optimal structure of CNN accelerators brings new challenges to engineers. Consequently, a generalizable simulation to rapidly search for the large design space of complex CNN accelerators is required [25]. As the area and power of most accelerators are largely dominated by the memory bandwidth, applying the roofline model to a simulation can be a reasonable method to search the design space in terms of memory access. This paper proposed a roofline-based simulation considering the sizes of the on-chip buffers (BRAMs), the maximum memory bandwidth between the on-chip buffers and off-chip memory, and dataflow techniques. Furthermore, the “pass” of each dataflow is modeled in detail to consider the timeline of CNN accelerator processing. This simulation can provide insights not only into the tendency of accelerator performances as the layer parameters change but also into the intuition about the design guidelines on the optimal hardware parameters and dataflow techniques under limited hardware resources. In addition, this paper quantitatively analyzed each dataflow technique and modeled the performance indicators needed for roofline-based simulations as a closed-form equation. Detailed interpretations for each figure were presented. The simulation was run on a Virtex-7 VX485T board, and the design space of the accelerators processing AlexNet and VGG-11 was explored. It is confirmed that the NLR and RS dataflow techniques are suitable for AlexNet, and the WS and NLR dataflow techniques are optimal for VGG-11.

Using this simulation, when trying to design a hardware accelerator suitable for each CNN layer parameter, engineers can rapidly make a first estimate. Additionally, the closed-form equation considering the “pass” about the dataflows can be adopted in another FPGA board and a similar dataflow analysis. However, as the roofline model-based simulation graphically offers insights, when a hardware is unbalanced in the operation processor and memory hierarchy, this simulation is powerful, but if it is already well balanced, detailed correction is slightly difficult. The proposed simulation focuses on exploring the structure of the accelerator and dataflow techniques. Therefore, the situation in which the data supplied to the accelerator are limited (i.e., the communication-limited situation) is not considered. In future work, a communication-limited system can be considered due to the collision of data transactions on the on-chip bus, and combining other techniques conjugating PE utilization and solving the sparsity issue are needed. In this paper, only convolutional layers are considered due to computationally intensive and repetitive patterns, but in a future work, a simulation considering not only convolutional layers but also fully connected layers will be modeled.

REFERENCES

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556
[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[3] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, “CNN: An FPGA-based processor for convolutional networks,” in Proc. Int. Conf. Field Program. Log. Appl., Aug. 2009, p. 32.
[4] Google. (Jun. 12, 2013), Improving Photo Search: A Step Across the Semantic Gap. [Online]. Available: http://googleresearch.blogspot.com/2013/06/improving-photo-search-step-across.html
[5] 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.
[6] C. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, “A programmable parallel accelerator for learning and classification,” in Proc. 19th Int. Conf. Parallel Archit. Compilation Techn. PACT, 2010, pp. 273–284.
[7] 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 Integ. (VLSI) Syst., vol. 26, no. 7, pp. 1354–1367, Jul. 2018.
[8] L. Cavigelli and L. Benini, “Origami: A 803-GOp/s/W convolutional network accelerator,” IEEE Trans. Circuits Syst. Video Technol., vol. 27, no. 11, pp. 2461–2475, 2017.
[9] V. Gokhale, J. Jin, A. Dundar, B. Martin, and E. Culurciello, “A 240 G-op/s mobile coprocessor for deep neural networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops, Jun. 2014, pp. 682–687.
[10] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, “Shidiannao: Shifting vision processing closer to the sensor,” in Proc. 42nd Annu. Int. Symp. Comput. Archit., 2015, pp. 92–104.
[11] D.-F. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Temam, X. Feng, X. Zhou, and Y. Chen, “PudianNao: A polyvalent machine learning accelerator,” in Proc. 26th Int. Conf. Architectural Support Program Lang. Operating Syst. ASPLOS, 2015, pp. 369–381.
[12] Y.-H. Chen, T. Krishna, J. S. 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.
[13] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA), Jun. 2016, pp. 367–379.
[14] 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, Dec. 2014, pp. 609–622.
[15] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in Proc. 19th Int. Conf. Architectural Support Program Lang. Operating Syst. ASPLOS, 2014, pp. 269–284.
[16] 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, 2015, pp. 161–170.
[17] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for floating-point programs and multicore architectures,” Office Sci. Tech. Inf. Tech. Rep., vol. 52, no. 4, pp. 65–76, 2009.
[18] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, “A survey of FPGA-based neural network accelerators,” 2017, arXiv:1712.08934. [Online]. Available: http://arxiv.org/abs/1712.08934
[19] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, “Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks,” in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays FPGA, 2017, pp. 45–54.

[20] 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, Dec. 2012, pp. 1097–1105.

[21] K. Simonyan and A. Zisserman, “Very deep convolutional networks for largescale image recognition,” 2014, arXiv:1409.1556. [Online]. Available: https://arxiv.org/abs/1409.1556

[22] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices,” 2018, arXiv:1807.07928. [Online]. Available: http://arxiv.org/abs/1807.07928

[23] W. J. Dally, Y. Turakhia, and S. Han, “Domain-specific hardware accelerators,” Commun. ACM, vol. 63, no. 7, pp. 48–57, Jun. 2020.

[24] G. Desoli, N. Chawla, T. Boesch, S.-P. Singh, E. Guidetti, F. De Ambroggi, A. Ardakani, C. Condo, and W. J. Gross, “Fast and efficient convolutional accelerator for edge computing,” IEEE Trans. Comput., vol. 69, no. 1, pp. 138–152, Jan. 2020.

[25] M. Hill and V. Janapa Reddi, “Gables: A roofline model for mobile SoCs,” in Proc. IEEE 31st Int. Conf. Comput. Design (ICCD), Oct. 2013, pp. 13–19.

[26] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, “Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks,” in Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays FPGA, 2017, pp. 45–54.

[27] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, “14.5 TOPS/W deep convolutional neural network SoC in FD-SOI 28nm.”

[28] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, “Deep convolutional neural network architecture with reconfigurable computation patterns,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[29] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, “14.5 TOPS/W deep convolutional neural network processor in 28nm FD-SOI,” in IEEE ISSCC Dig. Tech. Papers, Feb. 2017, pp. 246–247.

[30] A. Ardağan, C. Condo, and W. J. Gross, “Fast and efficient convolutional accelerator for edge computing,” IEEE Trans. Comput., vol. 69, no. 1, pp. 138–152, Jan. 2020.

[31] M. Hill and V. Janapa Reddi, “Gables: A roofline model for mobile SoCs,” in Proc. IEEE 31st Int. Conf. Comput. Design (ICCD), Oct. 2013, pp. 13–19.