Row-Streaming Dataflow Using a Chaining Buffer and Systolic Array+ Structure

Hweesoo Kim, Sunjung Lee, Jaewan Choi, and Jung Ho Ahn, *Senior Member, IEEE*

Abstract—Convolutional Neural Networks (CNNs) are widely used to solve complex problems in various fields, such as image recognition, image classification, and video analysis. Convolutional (CONV) layers are the most computational part of the CNN inference; various architectures have been proposed to process it efficiently. Among those, a systolic array consists of a 2D array of processing elements, which handle GEneral Matrix Multiplication (GEMM) with high efficiency. However, to process a CONV layer as a GEMM type, image-to-column (im2col) processing, which is also called lowering, is required per layer, necessitating a larger on-chip memory and a considerable amount of repetitive on-chip memory access. In this letter, we propose a systolic array+ (SysAr+) structure augmented with a chaining buffer and a row-streaming dataflow that can maximize data reuse without the im2col pre-process in the CONV layer and the repetitive access from the large on-chip memory. By applying the proposed method to the 3×3 CONV layers, we reduce the energy consumption by up to 19.7 percent in ResNet and 37.4 percent in DenseNet with an area overhead of 1.54 percent in SysAr+, and we improve the performance by up to 32.4 percent in ResNet and 12.1 percent in DenseNet.

Index Terms—Convolutional neural network, convolutional layer, general matrix multiplication, systolic array, image-to-column, lowering, ResNet, DenseNet

1 Introduction to CNN Accelerators

VARIOUS CNN accelerators [1], [5], [8] proposed as Convolutional Neural Networks (CNNs) are widely used for understanding visual information. Among those, Google's TPU adopts a systolic-array (SysAr) structure [5]. The SysAr [**x**×**y**] structure can be largely divided into three parts: a 2D [**x**×**y**] processing element (PE) array, a unified buffer (UB), and accumulators (ACC). The PE array is the unit for MAC operation, in which the size of **x** and **y** determines the number of MAC operations that can be processed at a time. UB serves to store input and output for data reuse. ACC stores partial sums calculated in the PE array and performs accumulation operations necessary to generate the final output.

ResNet [2] and DenseNet [4] are popular CNNs. These networks are composed of several layer types, such as convolutional (CONV), fully-connected, pooling, and batch-normalization layers. In inference, 98 percent of ResNet-50 and 97 percent of DenseNet-121 are attributed to the CONV layers in the total number of operations. A CONV layer consists of repetitive MAC operations and has the characteristic that the possibility of data reuse is very high. The input is in a 3D data format consisting of width (IW) × height (IH) × channel (IC), and the output format consists of width (OW) × height (OH) × channel (OC). The kernel consists of the width (KW) × height (KH) × IC × OC in a 4D data format. When processing the CONV layer in SysAr, repeated access is required to the same data in the on-chip memory, or a larger on-chip memory is required to duplicate the input data by KW×KH. Both approaches increase the energy consumed in the on-chip memory. This paper proposes a row-streaming (RS) dataflow and systolic array+ (SysAr+) to increase the energy efficiency by removing duplicated data and eliminating repetitive on-chip memory access through reuse by exploiting an internal chaining buffer.

• *The authors are with the Department of Intelligence and Information & Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea. E-mail: {hs5006.kim, jwchoi}@scale.snu.ac.kr, {shiish, gajh}@snu.ac.kr.*

Manuscript received 11 Dec. 2020; revised 12 Jan. 2021; accepted 14 Jan. 2021. Date of publication 26 Jan. 2021; date of current version 8 Feb. 2021.
(Corresponding author: Jung Ho Ahn.)
Digital Object Identifier no. 10.1109/LCA.2021.3054371

2 Energy-Efficiency Challenges of Processing CONV Layers in a SysAr

SysAr generally processes a CONV layer by converting 3D data into 2D data through im2col and then processing it with GEneral Matrix Multiplication (GEMM). When the CONV layer is processed as a GEMM operation, the mapping types of the input, kernel (weight), and output can be expressed as in Fig. 1. In a SysAr [**k**×**n**] using the weight stationary [9] method, after pre-loading the **k**×**n** weights, the **m**×**k** inputs are transferred to the left-most column of SysAr at most **k** per cycle. This method reuses the transmitted input while moving from left to right in every cycle in the 2D PE array. The partial sum, for which the MAC operation is completed in each PE, is accumulated while moving from top to bottom in the PE array. When the accumulation operation finishes, a maximum of **n** outputs is generated per cycle at the PE array output stage, and **m**×**n** outputs are generated after all operations are complete.

When SysAr processes the CONV layers, the aforementioned conventional strategy has the following limitations. First, to process the CONV layer in the form of GEMM, an im2col preprocess (also called lowering) is required that converts 3D data into 2D data. Second, when im2col is performed, the input is duplicated as much as the kernel size (e.g., 9 times for a 3 × 3 kernel), requiring a large UB. Third, regardless of whether performing im2col, a considerable amount of repetitive on-chip memory access is inevitable to process a KW×KH CONV, and the energy consumed by the on-chip memory (UB) takes a significant portion of the total energy.

For CNN accelerators, various methods have been proposed to handle CNNs by reusing inputs without explicitly using GEMM. Eyeriss [1] utilizes local register files per PE to reuse inputs by multicasting them from a global buffer. Eyeriss does not need to process im2col, but the inputs are duplicated in the local register files as much as the size of duplicated im2col inputs. Additionally, Eyeriss has a large area overhead to use the row-stationary dataflow as it requires register files for all three data types: input, weight, and partial sum. Thus, Eyeriss is more than 2× as large as SysAr, assuming an equal number of PEs [7]. NVDLA [8] does not duplicate inputs in the global buffer for convolution operations. However, it accesses the global buffer repeatedly as much as KH×KW times per input. Moreover, the size of a global buffer is generally about several hundred KBs to MBs, which is larger than the other buffer types. Thus, NVDLA dissipates significant on-chip memory access energy. Duplo [6] was proposed to address the limitation of using GEMM for CONV layers, albeit targeting GPUs.

Mapping for the Weights of the CONV Layers in SysAr. When CONV is calculated in SysAr, *ic* stands for partial input channel and indicates the number of input channels mapped to SysAr at one time. The range of *ic* is between 1 and IC.

When preloading weights in SysAr [**k**×**n**] with a weight stationary method, there are two types of mapping methods for **k**. The first method allocates one weight per input channel to each row of SysAr (**k** = *ic*1). Whenever the weight mapped in SysAr is changed to the next weight, the input used in the previous weight is read again. Here, because the reuse of the inputs is possible only after the weight is changed, the reuse distance is long. To reuse data on a chip, we must retain a significant amount of data for a long period.

The second method allocates all the weights per input channel to the adjacent rows of SysAr (**k** = KW × KH × *ic*2). In this case, the input takes a form in which the data of the adjacent locations are duplicated. As the transmitted input values overlap each other

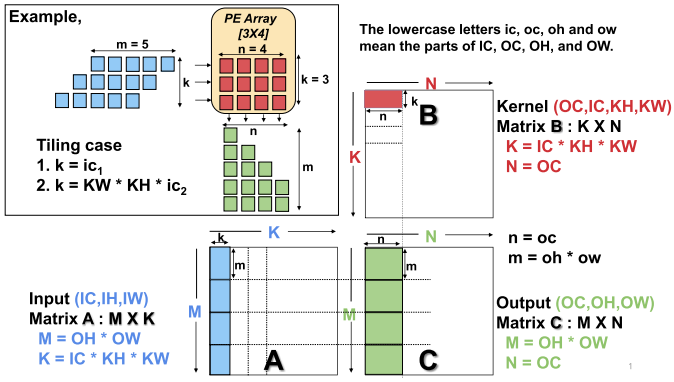

Fig. 1. Duplicated inputs by im2col and kernels for GEMM to process a CONV layer in a SysAr.

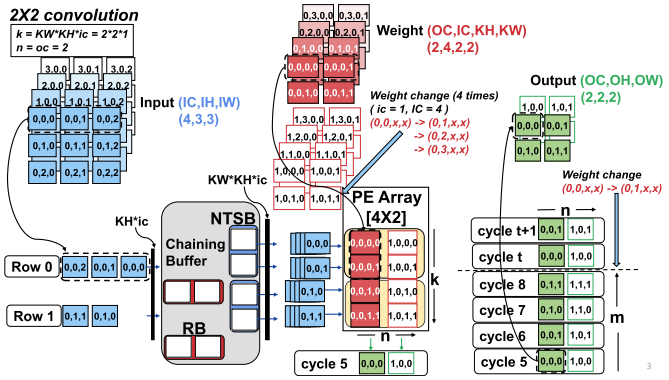

Fig. 3. Data mapping in SysAr+ for Row-streaming dataflow.

continuously, the input data is duplicated every cycle. Because the reuse of the input data occurs every cycle, the data reuse distance is short. To reuse data, we can keep a small amount of the data for a short time. Therefore, the second method reuses the input data more efficiently compared to the first one.

Besides, in the KW×KH CONV operation with stride 1, the input data excluding the edge part always require MAC operations with KH kernel rows. Therefore, data in one input row calculated with one kernel row can be reused when calculating the other (KH-1) kernel rows. This paper proposes a weight mapping method that requires a short data reuse distance and reduces on-chip memory access by adding a small size buffer. Furthermore, we propose to maximize the reusability of the input data by adding a buffer for the input row data that are reused (KH-1) times in a KW×KH CONV.

3 SysAr+ to Support Row-Streaming Dataflow

SysAr+ Microarchitecture. Our proposed SysAr+ adds a chaining buffer between UB and the 2D PE array of the baseline SysAr (see Fig. 2). The 2D PE array of SysAr+ is a structure in which a multi-input adder is combined for reduction by as much as KW. Each PE, equipped with a MAC in SysAr, has a multiplier instead; three PEs (KW=3) are grouped to have a common multi-input adder. SysAr+ operates by reconfiguring the input data transferred through UB into the form required for the PE array by internally shifting or storing it using the chaining buffer. This method eliminates the im2col process required for the CONV operation in SysAr so that the size of UB does not need to be increased and UB access is minimized.

The chaining buffer consists of two types of buffers: N-tap Shift register Buffers (NTSBs) and Row Buffers (RBs). NTSB is a buffer for input data reuse while operating each kernel row consisting of a

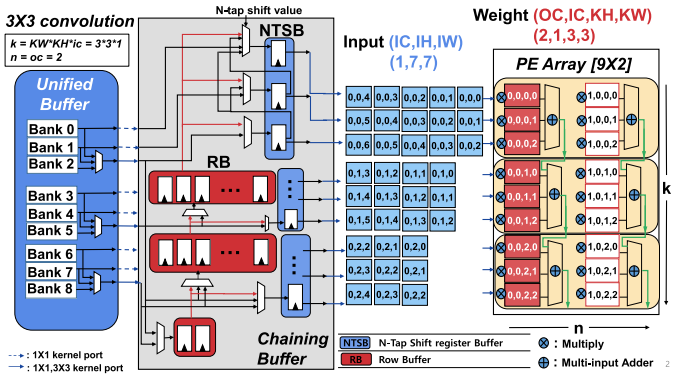

Fig. 2. SysAr+ Structure.

shift register and operates to reuse data with a short data reuse distance. NTSB can support stride 1 and stride 2, the dominant cases in CONV, using the N-tap shift value as a control signal. RB is a buffer for reusing input data in other kernel rows, storing data with a relatively long data reuse distance. The multi-input adder calculates the partial sum accumulation in parallel as much as the kernel width, meaning that the adder has KW or KW+1 input ports.

Row-Streaming (RS) Dataflow for SysAr+. RS dataflow refers to accessing the input data in the row units and delivering it in the form of continuous streaming data. In a CONV layer with IC=4, IH=3, IW=3, KH=2, KW=2, OC=2, and stride=1, when using the RS dataflow in SysAr+[4×2], its data mapping pattern is shown in Fig. 3.

Because KH=2 and *ic*=1, two lines of input row data are read from UB to the chaining buffer through two ports. The RS dataflow delivers the input row data from each port to the chaining buffer in a streaming format while increasing the index one by one in the IW direction.

When the input data is expressed as i[IC,IH,IW], the data in input row 0 is read from port 0 one by one in the order of i[0,0,0], i[0,0,1], and i[0,0,2]. Port 1 is delayed by 1 cycle compared to port 0, and the data in input row 1 is read in the order of i[0,1,0], i[0,1,1], and i[0,1,2]. The maximum RS input width size that can be processed through RS is determined by the RB size of the chaining buffer, which is expressed as *m_w*. If the input row data is read as much as *m_w*, it can no longer be useful in the input width direction, so the next new row (row 2) of data is read.

In other words, the RS dataflow reads the input data while increasing the width one by one to *m_w*; then it moves to the next required input row to read the data in the same width direction. When all IH rows are executed in this way, RS is completed. If IW is larger than *m_w*, some input widths are not processed by RS, necessitating additional RS for the remaining input to complete the operation.

Chaining Buffer for the RS Dataflow. An example of the operation according to the cycle in the chaining buffer is shown in Fig. 4. To reuse the input data read through each port from UB as much as possible, the chaining buffer operates in three ways as follows. First, the read input is stored in NTSB, such as i[0,0,0] data of cycle 1. Second, the read input is stored in both NTSB and RB, such as i[0,1,0] data of cycle 2. Third, the read input is stored only in RB, such as i[0,2,0] data of cycle 4.

Data stored in NTSB can be used for MUL operations with KW kernels through up-shift as much as N-tap every cycle. When N-tap is 1, data stored in NTSB is shifted up by 1 every cycle, and when all NTSBs are filled, the data of NTSBs are sent to the input of the PE array in the next cycle. If the value of the N-tap to be supported increases, the critical path of the N-tap shift register becomes longer, and the operating frequency may decrease, which needs to be considered. Because the value of N-tap may be different for each layer, a process of initially setting the N-tap shift value is required before

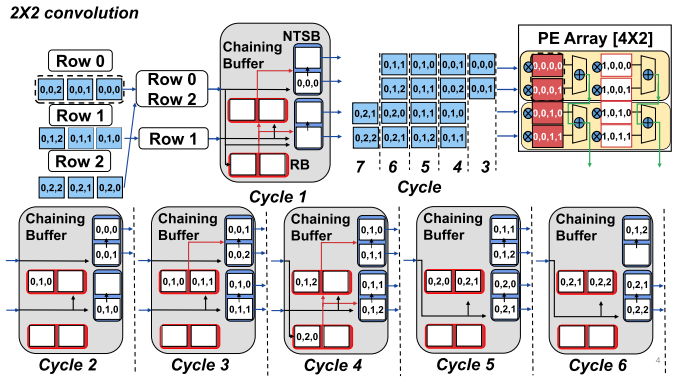

Fig. 4. Chaining buffer behavior over clock cycles.

processing each layer. These shift and output actions are always performed constantly in each layer. If CONV the layer has KH=3, KW=3, and stride=2, two out of three ports per row are used, and each data is stored in the bottom and middle of NTSB. The bottom data of NTSB is shifted by stride 2 and shifted to the top of NTSB. This approach supports stride 2, and the stride value is set before the processing of each layer by the value of the N-Tap shift register, and the N-tap shift value is used as a control signal.

RB reuses input data for calculations with other kernel rows; when a row change in the input data occurs in the RS dataflow, KW data are transferred from RB to NTSB at once. This can be seen in the operation of writing i[0,1,0] and i[0,1,1], two data stored in RB, to NTSB for the row change from row 0 to row 1 in cycle 3. When row change does not occur, one data is generally transmitted per cycle from RB to NTSB. This can be seen in the operation of writing i[0,1,2] data stored in RB to NTSB in cycle 4. When transmitting RB data to NTSB, if RB exists in the upper row, the data transmitted to the NTSB is also transmitted to the RB of the upper row. When i[0,2,0] data stored in RB is transmitted to NTSB in cycle 4, it is also transmitted to RB of the upper row. When using the chaining buffer, 16 UB accesses (KH×KW×OH×OW=2×2 × 2×2) required by the PE array are reduced to 9 (IH×IW=3×3).

Chaining Buffer Formula. To process a KW×KH CONV layer, NTSB with a size of KW×KH×*ic* and RB with a size of ((*m_w*-1)×(KH-1)+(KW-1))×*ic* are required inside the chaining buffer. For example, for *ic*=1, KW=3, KH=3, and *m_w* = 16, the size of NTSB is 3×3 = 9, and the size of RB is (16-1)×(3-1)+(3-1) = 32. If *m_w* is increased, more data can be processed with RS at one time, but the size of the RB also increases. Because the size of RB increases, it is inevitable that the area and energy overheads increase. If IW is greater than *m_w*, the input data must be processed by dividing it by *m_w*, and in this case, an overlap part occurs in the input. *ov_w* denotes the width overlapped between RS processing, which can be expressed as KW-stride. The whole part that is overlapped in the input is IC×IH×*ov_w*×(INT(IW/(*m_w*-*ov_w*)-1)). For example, in a SysAr+ with *m_w* set to 9 when processing a 3×3 CONV layer with stride 1, if the input data with an IW greater than 9 is processed, the input width (0∼8) is processed in the first RS. In the second RS, an overlap of *ov_w* (3-1 = 2) occurs, so the input width (7∼15) is processed. That is, an overlap width (7∼8) occurs between the first and second RS, and repeated access occurs for input data of IC×IH×2.

4 Evaluation

Experimental Setup. We synthesized logic components and registers such as multipliers, MACs, NTSB, and RB based on a 40 nm technology and evaluated the SRAM components such as UB and ACC using CACTI [10]. We set the clock frequency to 500 MHz and the data precision to 8 bits. Execution cycle counts and memory access counts according to each network were measured through our in-house simulator.

TABLE 1
SysAr+ 144x128 [*m_w*=16] Components

Component	Energy (pJ)	No. Units	Area (mm^2)
MUL (8bit)	0.120 pJ/op	144x128	4.497
3-input adder	0.048 pJ/op	16x128	0.238
4-input adder	0.193 pJ/op	32x128	1.573
UB (2MB)	2.569 pJ/b	1	17.850
ACC (1MB)	0.780 pJ/b	1	11.186
NTSB (9B)	0.035 pJ/b	16	0.005
RB (32B)	0.140 pJ/b	16	0.032
DRAM [3]	20 pJ/b	-	-
Total			35.381

We used the layers of ResNet-50 and DenseNet-121, especially 3×3 CONV, as the target layers. The evaluation was performed for all the 3×3 CONV layers for inference because SysAr+ is effective for CONV in which the kernel size is larger than 1×1. In ResNet-50 and DenseNet-121, the 3×3 CONV type accounts for the largest percentage of operations excluding the 1×1 CONV (42 to 47 percent of the total), and the other CONV layers (e.g., 5×5 and 7×7) can be replaced with 3×3 CONV layers through convolution factorization.

We set the baseline as SysAr with a PE array of 128×128, a 2 MB UB with a block size of 128B, and a 1MB ACC with a block size of 512B. The UB size is set to hold the largest input and output in the target networks of batch 1, and the ACC size is a scaled value of the TPU configuration. We set SysAr+ to have a 144×128 PE array with 3-input/4-input adders, NTSB to 9B, and RB to 32B, respectively. This configuration can handle 3×3 CONV most efficiently, and the experiments on the optimality of this configuration will be shown later in this session. Table 1 shows the energy and area for each component of SysAr+ depending on the configuration.

Architectures for Comparison. SysAr [128×128] uses the data mapping method of parallelizing one weight per input channel in the form of **k** = *ic*1. In this method, the input data is not duplicated, and the necessary input data is repeatedly accessed and read from the UB. This configuration is denoted as SA_128. By using data mapping that duplicates inputs in the form of **k** = KW×KH×*ic*2, the energy per access increases as the size of the UB becomes larger, and hence, the total energy further increases. Therefore, the case of **k** = *ic*1, which has a smaller energy consumption among the two cases, was set as the baseline of SysAr. In the case of SysAr+ proposed in this study, we use the data mapping method of parallelizing all the weights per input channel in the form of **k** = KW×KH×*ic*2. Here, the main target was set to 3×3 CONV, and the size of ic2 was set to 14 or 16. Therefore, SysAr+ [126×128] and SysAr+ [144×128] were selected as the targets for comparison and are denoted as SA+_126 and SA+_144. Moreover, because the trend varies according to the chaining buffer size in SysAr+, various sizes of *m_w* were compared and analyzed. The results of 16 and 58 are presented, and the size according to *m_w* is expressed as a postfix of [*m_w*=16] and [*m_w*=58].

Area. In SysAr+, the area overhead due to the chaining buffer is insignificant compared to UB, PE array, and ACC. Compared to SA_128, there exists a 1.2∼1.5 percent area decreases in SA+_126, whereas there is a 1.4∼1.8 percent area increases in SA+_144. The increase in the number of PE units like SA+_144 has less impact on the area because the PE arrays account for around 20 percent of the total area, and MUL is used instead of MAC in SysAr+.

Energy Efficiency. In SA_128, the total energy of ResNet-50 is 7.7mJ, and the total energy of DenseNet-121 is 4.6mJ. The energy of 3×3 CONV occupies more than 49 percent of the total energy in both models. As shown in Fig. 5, on ResNet-50, compared to SA_128, in SA+_126, 16.1∼16.3 percent of the energy is reduced, and in SA+_144, 19.4∼19.7 percent of the energy is reduced. In the case of DenseNet-121, the proportion of UB to the total energy was

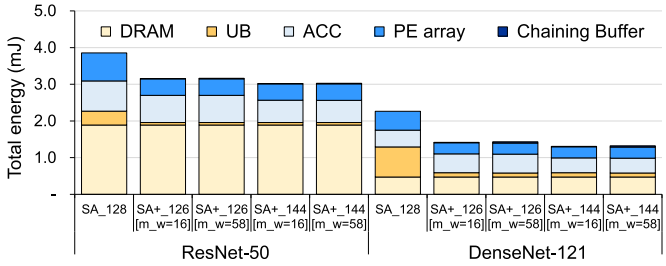

Fig. 5. Energy consumption comparison in 3×3 CONV layers.

relatively higher than that of ResNet-50, showing a greater energy reduction. For DenseNet-121, compared to SA_128, in SA+_126, 31.5∼32.4 percent of the energy is reduced and in SA+_144, 36.5∼37.4 percent of the energy is reduced. Thus, as m_w increases in SysAr+, the energy consumed in the chaining buffer increases, and the total energy increases. Our energy efficiency analysis shows that the size of m_w of SysAr+ was 16 which which led to the best energy efficiency for ResNet-50 and DenseNet-121.

Performance. In SA_128, the cycle count of 3×3 CONV in the total cycles occupies 55 percent in ResNet-50 and 72 percent in DenseNet-121. As shown in Fig. 6, in ResNet-50, SA+_126 performs better than SA_128. When the input channel is smaller than 128, the $\mathbf{k} = ic1$ weight mapping method used in SysAr lacks the input data that can be parallelized, so PE utilization decreases. In contrast, the $\mathbf{k} = \mathrm{KW}\times\mathrm{KH}\times ic2$ weight mapping method used in SysAr+ does not cause such a problem. SysAr requires data duplication to use the weight mapping method such as $\mathbf{k} = \mathrm{KW}\times\mathrm{KH}\times ic2$, which increases the energy consumption and hence cannot be used. In contrast, SysAr+ can use the method without data duplication. The performance was further improved for SA+_144 by the increased number of PEs.

Besides the impact of the weight mapping method and the number of PEs in SysAr+, the performance improvement due to the RS dataflow itself is small, leading to little performance change according to m_w. For example, when starting a new layer, in the case of SysAr, 128 cycles are added to fill the input data in the PE array. In contrast, in the case of SysAr+, the PE array is filled with just 1263 or 1443 cycles, not 126 or 144 cycles. In other words, SysAr+ reaches the steady-state period sooner. Therefore, in the case of SysAr+, the performance improves due to the characteristics of faster filling of a PE array with relatively smaller cycle counts compared to SysAr.

Due to these reasons, the performance is improved by 18.2 percent in SA+_126 and 32.4 percent in SA+_144 compared to SA_128 in ResNet-50. As opposed to ResNet-50, none of the 3×3 CONV layers in DenseNet-121 have fewer than 128 input channels, and hence, it does not take advantage of the weight mapping method in SysAr+. Therefore, the cycle counts increase by 9 percent becuase of the fewer PEs in SA+_126 compared to SA_128, and a 12.1 percent performance gain due to more PEs in SA+_144. In summary, the performance is largely affected by the applicability of the SysAr+ weight mapping method and the number of PEs.

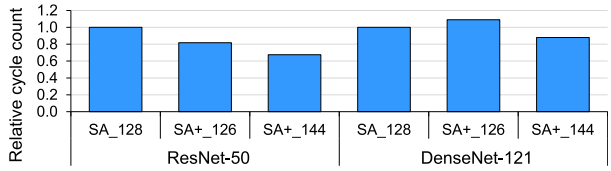

Fig. 6. Performance comparison in 3×3 CONV layers.

5 Conclusion

We have proposed a SysAr+ structure and a row streaming dataflow to accelerate the KW×KH CONV computation in CNNs. SysAr+ prevents the on-chip memory from duplicating input feature maps by removing the im2col preprocess that was required by the original systolic-array architecture for the CONV layer. Moreover, it minimizes repetitive on-chip memory reads in accessing the input by maximizing the data reuse through the proposed chaining buffer. Considering the energy efficiency gain at a given area cost among the SysAr+ configurations, SysAr+ with a PE size of 144×128 and a maximum RS input width size of 16 showed the best results. Compared to SysAr with a PE size of 128×128, the corresponding structure requires 1.54 percent more area while the total energy consumed by the 3×3 CONV layers decreases by 19.7 percent in ResNet-50 and 37.4 percent in DenseNet-121, and the total cycle counts also decrease by 32.4 percent in ResNet-50 and 12.1 percent in DenseNet-121.

Acknowledgments

This work was supported by Samsung Advanced Institute of Technology and the Engineering Research Center Program through the National Research Foundation of Korea funded by the Korean Government MSIT (NRF-2018R1A5A1059921). The EDA tool was supported by the IC Design Education Center.

References

[1] 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.

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

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

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.

[5] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[6] H. Kim *et al.*, "Duplo: Lifting redundant memory accesses of deep neural networks for GPU tensor cores," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 725–737.

[7] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2018, pp. 461–475.

[8] F. Sijstermans, "The NVIDIA deep learning accelerator," in *Hot Chips*, vol. 30, 2018.

[9] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[10] S. Thoziyoor *et al.*, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. Int. Symp. Comput. Archit.*, 2008, pp. 51–62.