A Comprehensive Evaluation of Convolutional Hardware Accelerators

Leonardo R. Juracy, Alexandre M. Amory, and Fernando G. Moraes, *Senior Member, IEEE*

***Abstract*—Industry increasingly adopts Convolutional Neural Networks (CNNs) in applications ranging from IoT to autonomous driving. Convolutional hardware accelerators for the inference phase are an option for CPUs and GPUs due to the smaller power consumption and improved performance. The literature presents hardware accelerators using different 2D architectures, including weight stationary (WS), input stationary (IS), and output stationary (OS). The main differentiation between these architectures is how accelerators access data (input feature map and weight tensors) and compute the output (output feature map tensor). There is a gap in the literature related to a comprehensive evaluation of such architectures. This brief aims to answer the following question: "which accelerator type should I use according to my design constraints and memory type (SRAM or DRAM)". Experiments show that when using SRAM as external memory to the accelerator, WS presents the smallest area and energy consumption, while IS presents the best performance. On the other hand, the IS accelerator stands out when using DRAM because it has a reduced performance sensitivity to memory latency.**

***Index Terms*—CNN, convolutional hardware accelerator, PPA, SRAM, DRAM.**

I. Introduction and Related Work

Convolutional Neural Network (CNN) is a class of machine learning technique used to solve problems such as classification and pattern recognition [1]. Industrial companies increasingly adopt machine learning on their products [2], [3]. The process of performing a convolution and classifying or predicting an output based on an external input is called inference. Classically, CPUs have been a common approach to execute the inference, but they are inefficient in terms of performance and energy (e.g., AlexNet from 2012 [4] requires billions of operations to process a single input). GPUs can be used as a solution to mitigate the performance problem. However, GPU presents a considerable energy consumption, a problem for energy-constrained applications, such as IoT and autonomous vehicles (e.g., drones).

Convolutional accelerators emerged as a solution to reduce energy consumption and, at the same time, improve performance. An accelerator can be implemented according to the way data enters on it: (*i*) weight stationary (WS); (*ii*) input stationary (IS); (*iii*) output stationary (OS) [5]. WS is a dataflow where the weight values are stored in internal buffers, and the input feature map (IFMAP) values are constantly fetched from memory. Weight values change when the computation of the output finishes. IS is a dataflow where IFMAP values are stored in internal buffers, and the weight values are constantly fetched from memory. The IFMAP values change when the computation of the output finishes. OS is a dataflow where partial outputs are stored in internal buffers, and the weight and IFMAP values are constantly fetched from memory.

Recent literature presents accelerators based on: WS [6], [7], IS [8], OS [9], [10]. Some works allow reconfiguring the accelerator to support the three dataflows [11], [12]. Other works use a custom dataflow, like Eyeriss [13] with a Row Stationary (RS) approach. Moolchandani et al. [5] survey CNN accelerators (including No Local Reuse (NLR) dataflow) and their design and optimizations, as reduction of the computation time, memory access time, and memory footprint.

We summarize in Table I the main features of the reviewed CNN accelerators. The second column presents the used dataflows, which seek to optimize a given design parameter (third column). There is no comprehensive comparison of dataflows themselves in the accelerator literature. An exception is the Chen et al. [13] proposal, which compares RS with WS and OS. Thus, this brief differs from the others as it seeks to evaluate the design space of convolutional hardware accelerators rather than proposing a new accelerator. The fourth and fifth columns present the CNN model and the adopted dataset. We adopted CIFAR-10, not synthetic values, to accurately estimate the power, i.e., considering an actual switching activity.

All accelerators generate intense data communication with external memories. Depending on the memory type, SRAM or DRAM, and its latency, the energy cost related to access the memories can be of the same order as the accelerator or even higher [13]. Therefore, when considering an architecture to speed up the convolution, it is essential to assess the cost related to the memory.

The *goal* of this brief is to execute a comprehensive evaluation of convolutional accelerator architectures by designing different dataflows with different buffering strategies. The evaluation considers physical post-synthesis performance data (performance, power, and area - PPA), the presence or absence of buffers, and the impact of the memory type (SRAM and DRAM) on the PPA. The comprehensive comparison includes the number of memory accesses, power, energy, area, and performance. Our *original* contribution is a guideline to which convolutional accelerator to adopt according to design constraints, considering the external memory technology.

Manuscript received 16 August 2022; revised 19 September 2022; accepted 19 November 2022. Date of publication 23 November 2022; date of current version 6 March 2023. This work was supported in part by the Conselho Nacional de Desenvolvimento Cientfico e Tecnolgico (CNPq) under Grant 309605/2020-2, and in part by the Coordenao de Aperfeioamento de Pessoal de Nvel Superior (CAPES)—Finance Code 001. This brief was recommended by Associate Editor Y. Wang. *(Corresponding author: Fernando G. Moraes.)*

Leonardo R. Juracy and Fernando G. Moraes are with the School of Technology, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre 90619-900, Brazil (e-mail: leonardo.juracy@acad.pucrs.br; fernando.moraes@pucrs.br).

Alexandre M. Amory is with Scuola Superiore SantAnna, 56127 Pisa, Italy (e-mail: alexandre.amory@santannapisa.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2022.3223925.

Digital Object Identifier 10.1109/TCSII.2022.3223925

TABLE I
RELATED WORKS ON CONVOLUTIONAL HARDWARE ACCELERATORS

Referece	Dataflow	Target	CNN model	Dataset
Tavakoli 2020 [8]	WS / IS	Optimize throughput	Custom, first layer: 256x256x1	Not Addressed
Das 2020 [9]	OS	Optimize energy	Inception V3, Resnet 50, Inception-Resnet	Not Addressed
Bai 2020 [6]	WS	Memory accesses reduction	SegNet-Basic	Not Addressed
Udupa 2020 [10]	OS	Optimize power and throughput	MnasNet	Not Addressed
Hsiao 2020 [12]	WS / IS / OS (reconfigurable)	Optimize power, memory accesses and arithmetic operations	VGG-16, MobileNet, Unet	ImageNet
Chen 2016 [13]	RS	Optimize energy	AlexNet	ImageNet
Lu 2017 [11]	WS / IS / OS (reconfigurable)	Optimize power and performance	PV, FR, LeNet-5, HG, AlexNet, VGG-11	ImageNet and others
Liu 2020 [7]	WS	Energy efficiency	Alexnet, Googlenet, Resnet, Zfnet, Gmean	ImageNet
Korol 2021 [14]	Not defined - Xilinx HLS	Optimize performance and energy	Alexnet, VGG-16	CIFAR-10, GTSRB, and Caltech-256
This Work	WS / IS / OS - 5 architectures	Accelerator design space exploration	Custom, first layer 32x32x3	CIFAR-10

II. CNNs AND THE CONVOLUTIONAL OPERATION

CNN contains four main layers: (*i*) convolutional layer; (*ii*) activation function; (*iii*) pooling layer; (*iv*) fully connected layer. The more complex layer is the convolutional layer, which computes the synapses by multiplying and accumulating weights and IFMAP. A training process obtains the weight values, and IFMAP values come from external inputs, like RGB images [15].

Convolution is based on filters, which limit the multiplications and sums to matrix windows. Multiplications are executed between the weights and the IFMAP values that come from an RGB image and accumulate the generated partial value of each operation to generate a complete convolution value. A bias value is added to the sum of the accumulated value, and it is applied to the activation function to generate the output feature map (OFMAP).

Equation (1) formally describes the process to obtain one OFMAP result. The convolution receives the IFMAP tensor (each map may also be called a channel) and another tensor of filters as inputs. Then, each filter window is convolved with (and slides across) its respective input channel forming a new set of feature maps. Next, a bias is added to the feature map, generating the final OFMAP tensor (**O**).

$$\mathbf{O}[f][x][y] = \mathbf{B}[f] + \sum_{k=0}^{C-1}\sum_{i=0}^{W-1}\sum_{j=0}^{H-1}(\mathbf{I}[k][S_x+i][S_y+j] * \mathbf{WF}[f][k][i][j]) \quad (1)$$

where: f, x and y are the current output channel, the horizontal and the vertical position, respectively; C is the total number of input and filter channels, W and H corresponds to the filter size; S is the stride (number of positions that the filter slides over to the next window), and **O** is the output, **I** the input, and **WF** the filter tensors and **B** the bias vector.

III. CONVOLUTIONAL HARDWARE ACCELERATORS

Figure 1 details the three modules required to build the convolutional accelerators: (*i*) input memory, stores the IFMAP, filter weights, and bias values; (*ii*) convolutional core, executes the convolution; (*iii*) OFMAP memory stores the partial and complete convolution values. The convolutional core contains:

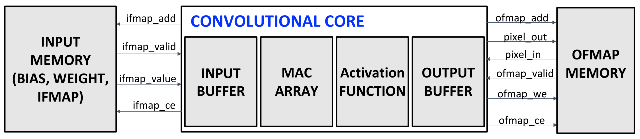

Fig. 1. Modules required to build the convolutional accelerators.

- input buffer: reduces the number of input memory readings. According to the accelerator type, this buffer may store, e.g., an input channel, a set of rows of the input channel, or a set of weights;
- MAC array: a matrix with multipliers, adders, and accumulators, responsible for executing the convolution, i. e., compute Equation (1);
- activation function: a non-linear function applied to the OFMAP results. Examples are sigmoid, ReLU, leaky ReLU [16]. This works adopts the ReLU function ($max(0, x)$) due to its simpler hardware implementation;
- output buffer: reduces the number of output memory readings and writings. As Equation (1) shows, to compute one output result, it is necessary to generate partial results, implying in OFMAP memory readings and writings.

In [17], we presented a framework for the design space exploration of CNNs, providing PPA estimation. This design space exploration focused on integrating the high-level (TensorFlow) with the low-level modeling using the WS dataflow. Juracy et al. [17] did not include the evaluation of other dataflows (IS and OS) nor the impact of the memory on hardware acceleration.

In this brief, we evaluate five convolutional cores (2 WSs, 2 ISs, and 1 OS), all having the same interface with the memories. The input memory is accessed by *ifmap_add* (address) and *ifmap_ce* (enable). Data is available in the *ifmap_value*, once *ifmap_valid* is high. The *valid* signal enables to simulate the memory latency and its impact on the performance and energy of the convolutional core under evaluation. The OFMAP memory is similar, with a *ofmap_valid* signal to simulate its latency. We selected these dataflows because they are the most used in the literature [5]. Such structure enables a fair comparison among accelerators since the same accelerator surrounding structure is applied to each one. Accelerators are synthesized with the same technology and constraints, which is not possible when comparing accelerators from different Authors.

A. *Weight Stationary – WS*

The WS dataflow stores the weights in the input buffer, aiming their reuse. Thus, each weight value is read once from the input memory, and the convolution is performed using stationary values for weight with values read from memory for the IFMAP window.

Figure 2 illustrates the WS-buffer accelerator architecture, connected to the input and output memories. The input buffer has three partitions: bias, weight (stationary values), and IFMAP values. An FSM (Finite State Machine) controls the input buffer loading. The arithmetic core contains a 3x3 matrix with 3 multipliers, 6 MACs, 3 adders, and 12 registers. The accelerator presents a double buffer approach for the input feature reading, making it possible to read the memory values and execute the arithmetic process in parallel. The output buffer is only used in the WS-buffer accelerator. It stores intermediate output values after the activation function, reducing the output memory accesses. The IS and OS present similar architectures, varying the control FSM and the location of the buffers that feed the arithmetic core.

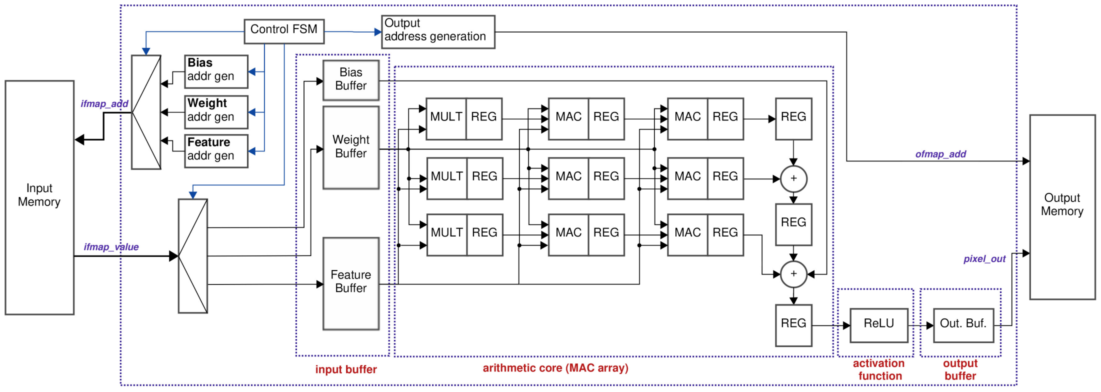

Fig. 2. WS-buffer accelerator architecture (adapted from [17]), with a simplified set of signals with the memories. Red labels correspond to the modules presented in Figure 1.

Algorithm 1: WS Pseudo-Code

```
   Input: C input channels, F output channels
   Output: O
1  foreach f in F do
2      foreach c in C do
3          Read weight filter set w(f,c) from input memory
4          Store filter set in the input buffer // weight stationary
5          foreach $I(i)(j)$ in IFMAP(c) do
6              Read a window $I(i)(j)$ from IFMAP
7              p ← convolution($I(i)(j)$, $w(f,c)$) // arithmetic core
8              O[f][x][y] ← O[f][x][y] + p
9          end
10     end
11 end
```

Algorithm 2: IS Pseudo-Code

```
   Input: C input channels, F output channels
   Output: O
1  Read weights and bias, storing in the input buffer // IS optimization
2  foreach c in C do
3      foreach $I(i)(j)$ in IFMAP(c) do
4          Read a window $I(i)(j)$ from IFMAP
5          Store window $I(i)(j)$ in the input buffer // input stationary
6          foreach f in F do
7              foreach cw in C do
8                  foreach w in w(f,cw) do
                       // stored in the input buffer
9                      p ← convolution($I(i)(j)$, $w(f,c)$)
10                     O[f][x][y] ← O[f][x][y] + p
11                 end
12             end
13         end
14     end
15 end
```

Algorithm 1 presents the pseudo-code describing the WS hardware behavior. The core of the algorithm comprises lines 3 to 9. Lines 3-4 fetch a filter set $w(f, c)$ from memory, storing it in the input buffer. The loop between lines 5-9 reads windows from the IFMAP, executes the convolution, and produces a partial result. Our WS accelerator reduces IFMAP accesses by reusing read values. For example, with a stride equal to 2, the last column of the currently loaded window matrix is reused in the next loaded window. This column is reused in the next partial value computation by using the double buffer.

To obtain a convolution value in the OFMAP memory (output memory), it is necessary $|F-1|$ reads (for partial convolution values) and $|F|$ writes (line 8). This implies many memory accesses, increasing the total energy consumption.

There are two WS implementations: with and without output buffer. Storing partial results in the output buffer reduces the OFMAP memory accesses. The output buffer size is equal to one output channel size, resulting in 1 writing operation instead of $|F-1|$ readings and $|F|$ writings. However, this solution increases the accelerator area and energy.

B. Input Stationary – IS

The Input Stationary (IS) dataflow registers an IFMAP window in the input buffer to provide its reuse. The window size is equal to the filter size. Algorithm 2 shows pseudo-code describing the IS hardware behavior. The core of the algorithm comprises lines 3 to 10. Lines 4-5 fetch an IFMAP window $I(i)(j)$ from memory, storing it in the input buffer. The loop between lines 7-12 reads filter sets from the input buffer, executes the convolution, and produces a partial result.

Differently from a standard IS, the implemented IS stores weights and bias values in the input buffer to reduce the memory accesses while introducing a penalty in area (line 1 of Algorithm 2). We adopted this approach once the data required for weights and bias can be smaller than an IFMAP channel as a function of the CNN-specific model. To circumvent the area penalty, it is possible to reduce the storage requirement using a prefetch method. Thus, each IFMAP window is read once, and the convolution is performed using stationary values for IFMAP and the stored weight values.

The IS also has two versions, with and without output buffer. The IS access the OFMAP in the same way of WS (line 9 of Algorithm 2). The IS output buffer size is equal to $x \times F$, while the WS output buffer is $x \times y$ (x and y: OFMAP size, F: number of output channels). For small OFMAPs, such as 15x15x16, both output buffers have similar sizes (WS: 15×15 and IS: 15×16). For a larger OFMAP size, there is a advantage for the IS output buffer (e.g., 128x128x16: WS: 128×128, IS: 128×16).

C. Output Stationary

The Output Stationary (OS) dataflow is based on registering the partial values generated on the convolution. The OS does not present buffers to store the inputs; each convolution fetches the IFMAPs and weight values in the memory. Algorithm 3 shows the pseudo-code describing the OS hardware behavior. The core of the algorithm comprises lines 4 to 9. Lines 5 and 6 fetch a IFMAP window $I(i)(j)$ and a filter set $w(f, c)$ from memory. Lines 7 and 8 perform the convolution and

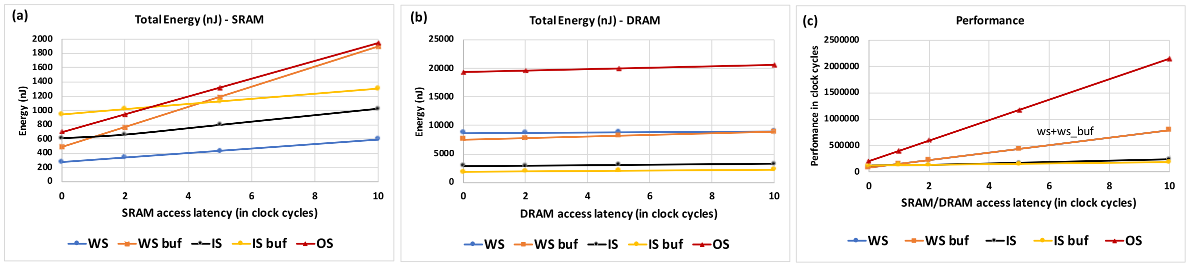

Fig. 3. Energy and performance varying the memory type (SRAM or DRAM) and the access latency.

Algorithm 3: OS Pseudo-Code

```
   Input: C input channels, F output channels
   Output: O
1  foreach $f$ in F do
2      foreach $c$ in C do
3          internal_p ← 0
4          foreach $I(i)(j)$ in IFMAP(c) do
5              Read a window $I(i)(j)$ from IFMAP Input Memory
6              Read a set of filters $w(f)(c)$ from Input Memory
7              p ← convolution($I(i)(j)$, $w(f, c)$)
8              internal_p ← internal_p + p
9          end
10     end
11     O[f][x][y] ← internal_p // output stationary
12 end
```

accumulate the partial result in the output buffer *internal_p*. Line 11 stores a complete convolution value.

OS uses as output buffer only a register to accumulate the partial values, which contributes to reduce its area. However, the OS is the dataflow that consumes more energy due to many memory accesses to fetch IFMAP and weights.

IV. Experimental Setup

Cadence Genus and Innovus tools were used for logic and physical synthesis, with 28nm technology and a frequency of 500MHz. The logic synthesis uses clock-gating to reduce the accelerator energy consumption. The power dissipation is obtained with a value change dump (VCD) file generated with a post-P&R netlist simulation and Cadence Voltus tool. The netlist simulation input is the first CNN layer with a 32x32x3 IFMAP (CIFAR-10), 16 3x3 filters, stride 2, generating a 15x15x16 output. Thus, power values come from a real dataset and not synthetic values. All inputs are quantized to 8-bit integers and results to 20-bit integers, decreasing the accuracy in the worst-case in 5% [17]. The total energy is computed by multiplying the average power by the number of clock cycles required to execute a complete convolution.

The external memories modeling, SRAM and DRAM, adopts the Cacti-IO tool [18]. For a 28nm 4KB SRAM, Cacti-IO reports 260 fJ/bit for reading and 180 fJ/bit for writing. For a 16 kB DRAM, 12 pJ/bit for reading, and 11.7 pJ/bit for writing. For comparison purposes, the literature reports energy values between 67 fJ/bit [19] and 20 fJ/bit [20] for 28nm SRAM, and 20 pJ/bit [21] and 15 pJ/bit [22] for DRAM. The SRAM consumption is larger than the one reported in the literature for two reasons: (*i*) we considered a 500MHz frequency, while the literature considers 200 MHz; (*ii*) the literature considers low-energy SRAM. The DRAM values are close to the ones reported in the literature.

We adopted a simple dataset, CIFAR-10, instead a more complex one, such as Imagenet, because increasing the IFMAP size affects the convolution performance and energy, not the accelerator PPA. Thus, the selected dataset does not compromise the comparison method. We executed experiments with 128x128 IFMAPs and observed the same trends in performance and energy depicted in Figure 3. Korol et al. [14] also observed this trend when increasing the IFMAP size.

V. Results

Figure 3(a) a 3(b) evaluate the energy consumption to obtain a 15x15x16 OFMAP, according to the memory type and its access latency (x-axis). Accelerators that do not adopt output buffers have a smaller energy consumption than accelerators using output buffers when using SRAM. This result is because SRAM and buffers use the same static implementation, resulting in a similar consumption. *Thus, buffering brings no advantage when using SRAM memories*. The observed result is inverse when using DRAMs as external memories, with buffers acting as cache memories. The buffered IS accelerator presents a significant energy reduction (IS buf) compared to the other implementations. The OS accelerator is expensive in terms of energy because it constantly fetches data from the input memory, regardless of the memory type.

Figure 3(c) evaluates the performance of the accelerators. The IS accelerators present a performance slightly affected by the memory latency because the IFMAP is read once, that is, input stationary. Also, the bufferization of weights and bias reduces memory access, decreasing the memory impact on performance (line 1 of Algorithm 2). The memory latency affects the WS performance because the number of IFMAP readings is higher than the IS architecture. The OS architecture has a small buffer in the output, requiring frequent IFMAP and weight readings, resulting in a heavy performance penalty due to memory latency.

Figure 4 details the results, considering an SRAM with access latency of 2 clock cycles. Required buffer size (internal buffers), area, and performance (cycles) are normalized by the worst result among accelerators. The energy (memory and accelerator) is normalized by the worst total energy, as well as the memory accesses (reads and writes). Figure 4 shows that:

- the convolutional core consumes the largest parcel of the total energy consumption (energy core in Figure 4).
- buffering at the output effectively reduces memory writings (from 10,800 to 3,600). The normalization masks these results due to the higher number of readings (71,008). Buffering brings a slight reduction in memory energy consumption at the cost of a larger area and consumption (WS versus WS_buf and IS versus IS_buf).
- IS is 66.2% faster than WS (225,078 and 135,450 clock cycles for WS and IS, respectively).
- OS is penalized by memory readings, resulting in the worst performance and highest memory and energy

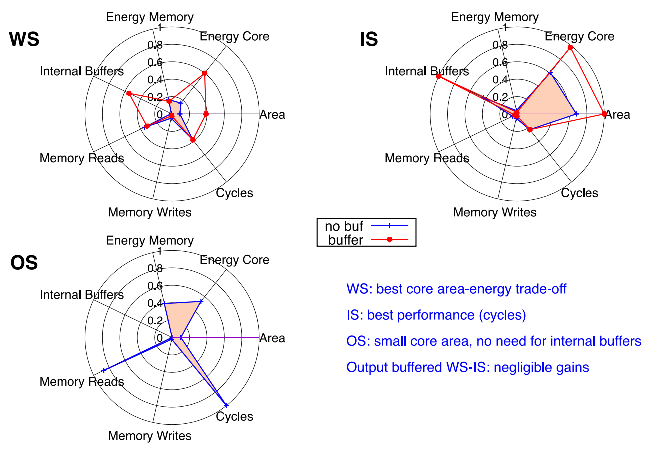

Fig. 4. Performance for the convolutional accelerators, considering a 32x32x3 IFMAP, 15x15x16 OFMAP, stride=2, and a 2 clock cycle SRAM latency.

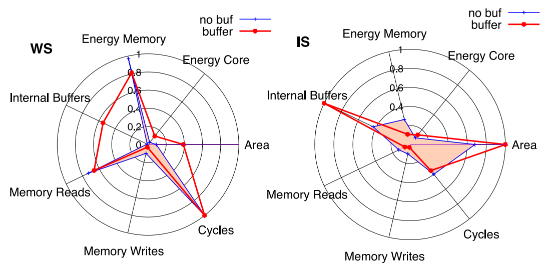

Fig. 5. Performance for the convolutional accelerators, considering a 32x32x3 IFMAP, 15x15x16 OFMAP, stride=2, and a 5 clock cycle DRAM latency.

TABLE II
STRENGTHS AND WEAKNESSES OF THE ACCELERATORS

	WS	IS	OS
Pros	Trade-off area-energy-performance	Performance	Area, there is no need of buffers
Cons	Performance, when the memory latency increases	Area, due to the input buffers	Performance, due to the memory accesses

consumption. Despite these disadvantages, it presents a small area footprint, similar to the WS.

When using SRAM as external memory, WS is the accelerator with the smallest area and energy consumption, while IS presents the best performance (axis cycles in Figure 4). Note that the IS area increases with the IFMAP size since it requires larger input buffers.

Figure 5 presents results when using DRAM as external memory, with a latency of 5 clock cycles (OS is omitted due to its worst performance - see Figure 3(b) and (c)). The IS architecture is 2.5 times faster, with energy consumption up to 3.97 times smaller than WS. Buffering the IS accelerator reduces its total energy by 33% and improves the performance by 12%, at the cost of 1.46x the area footprint.

The IS accelerator presents the best energy-performance trade-off when using DRAM as external memory. It is possible to reduce energy and improve performance by using an output buffer at a significant area overhead cost.

Table II summarizes the findings observed in evaluating the accelerators.

VI. Conclusion

This brief presented a guideline for designers to select the accelerator architecture according to their constraints, considering the external memory technology. Using SRAM as external memory, the WS without output buffers presents the smallest area and energy consumption, while IS without output buffers is recommended when performance is the goal. With DRAM, IS without output buffer is the architecture to use because it presents 2.5x better performance and 3.97x lower energy than WS. It is possible to reduce the IS total energy by 33% and improve performance by 12% using output buffers at the cost of extra area.

Future work includes 3 directions: (*i*) reduce the input buffer on IS, including a prefetch mechanism for weights and bias; (*ii*) include other accelerator architectures in the evaluation environment; (*iii*) integrate the evaluation results into the TensorFlow framework for an early and fast design space exploration.

References

[1] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[2] Google. "Google Assistant, Your Own Personal Google." 2020. [Online]. Available: https://assistant.google.com

[3] Tesla. "Autopilot." 2020. [Online]. Available: https://www.tesla.com

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.

[5] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, Feb. 2021, Art. no. 101887.

[6] L. Bai, Y. Lyu, and X. Huang, "A unified hardware architecture for convolutions and deconvolutions in CNN," in *Proc. ISCAS*, 2020, pp. 1–5.

[7] B. Liu et al., "Search-free accelerator for sparse convolutional neural networks," in *Proc. ASP-DAC*, 2020, pp. 524–529.

[8] M. R. Tavakoli, S. M. Sayedi, and M. J. Khaleghi, "A high throughput hardware CNN accelerator using a novel multi-layer convolution processor," in *Proc. ICEE*, 2020, pp. 1–6.

[9] S. Das, A. Roy, K. K. Chandrasekharan, A. Deshwal, and S. Lee, "A systolic dataflow based accelerator for CNNs," in *Proc. ISCAS*, 2020, pp. 1–5.

[10] P. P. Udupa, G. Mahale, K. K. Chandrasekharan, and S. Lee, "Accelerating depthwise convolution and pooling operations on Z-first storage CNN architectures," in *Proc. ISCAS*, 2020, pp. 1–5.

[11] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. HPCA*, 2017, pp. 553–564.

[12] S.-F. Hsiao, K.-C. Chen, C.-C. Lin, H.-J. Chang, and B.-C. Tsai, "Design of a sparsity-aware reconfigurable deep learning accelerator supporting various types of operations," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 376–387, Sep. 2020.

[13] Y.-H. Chen, J. S. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, 2016.

[14] G. Korol, M. G. Jordan, M. B. Rutzig, and A. C. S. Beck, "Synergistically exploiting CNN pruning and HLS versioning for adaptive inference on multi-FPGAs at the edge," *ACM Trans. Embedd. Comput. Syst.*, vol. 20, no. 5s, pp. 1–26, 2021.

[15] S. S. Haykin, *Neural Networks and Learning Machines*. London, U.K.: Pearson Educ., 2009.

[16] Keras. "PReLU Layer." 2021. [Online]. Available: https://keras.io/api/layers/activations/

[17] L. R. Juracy, M. T. Moreira, A. de Morais Amory, A. F. Hampel, and F. G. Moraes, "A high-level modeling framework for estimating hardware metrics of CNN accelerators," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4783–4795, Nov. 2021.

[18] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "Cacti-IO: Cacti with off-chip power-area-timing models," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1254–1267, Jul. 2015.

[19] H. Fujiwara et al., "A 20nm 0.6 V 2.1 μW/MHz 128kb SRAM with no half select issue by interleave wordline and hierarchical bitline scheme," in *Proc. VTS*, 2013, pp. C118–C119.

[20] T. Haine, Q.-K. Nguyen, F. Stas, L. Moreau, D. Flandre, and D. Bol, "An 80-MHz 0.4 V ULV SRAM macro in 28nm FDSOI achieving 28-fJ/bit access energy with a ULP bitcell and on-chip adaptive back bias generation," in *Proc. ESSCIRC*, 2017, pp. 312–315.

[21] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing memory access latency with asymmetric DRAM bank organizations," in *Proc. ISCA*, 2013, pp. 380–391.

[22] H. Li, M. Bhargava, P. N. Whatmough, and H.-S. P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *Proc. DAC*, 2019, pp. 1–6.