High Performance CNN Accelerators based on Hardware and Algorithm Co-Optimization

[1]Kothapally Kanchana, [2]B. Srilekha, *[3]Madipalli Sumalatha
[1,2,3]Department of ECE,
[1,2,3]Siddhartha Institute of Technology & Sciences, Narapally, Ghatkesar, Medchal-Malkajgiri, Telangana, India
*[3]sitsmtechms@gmail.com

Abstract: The efficacy of convolutional neural networks (CNNs) has led to their extensive application in picture recognition and categorization. Nevertheless, embedded systems face challenges when it comes to storing the massive volumes of weight data required by CNNs in their on-chip memory. Pruning a convolutional neural network (CNN) model can reduce its size with no impact on accuracy; however, when used with a parallel architecture, the resulting model will be slower to execute. A CNN compression method that is hardware-centric is introduced in this paper. A model of a deep neural network (DNN) will include "pruning layers (P-layers)" and "no-pruning layers (NP-layers)" attached to it. For effective and parallel processing, an NP-layer employs a regular distribution for its weights. Pruning causes a P-layer to have an irregular form, yet the high compression ratio it yields makes the shape undesirable. Using uniform and gradual quantization methods, we can achieve a processing efficiency/compression ratio trade-off with a small loss of precision. Integrating multiple parallel finite impulse response (FIR) filters into an NP-layer distributed convolutional architecture, we further improve the regular model. The P-layer irregular sparse model is suggested to use an ADF-based processing element. The suggested compression strategy and hardware architecture can be utilised by a hardware/algorithm co-optimization (HACO) method to run an NP→P hybrid compressed CNN model on FPGAs. The VGG-16 network achieves a compression ratio of 27.5× with a top-5 accuracy loss of 0.44% by using a hardware accelerator on a single FPGA chip without off-chip memory. A result of 83.0 FPS, or 1.8 times quicker, was achieved when the compressed VGG-16 model was implemented on a Xilinx VCU118 evaluation board for image applications.

Index Terms **- CNN, Accelerators, Co-Optimization.**

I. INTRODUCTION

Classification of images [1,2] and recognition [3,4] are two common applications of convolutional neural networks (CNNs). Computational resources must be allocated to CNNs in order to achieve higher accuracy. Typically, GPUs and other parallel processors are used to speed up CNNs for real-time processing [4]. Even while graphics processing units (GPUs) speed up computing, embedded systems can't use them because of the significant power increase. Accelerators for convolutional neural networks (CNNs) have lately attained low power consumption and great performance in digital systems through the use of field programmable gate arrays (FPGAs) and application specific integrated circuits (ASICs).

Unfortunately, storing a large-scale CNN model entirely would be beyond the capabilities of contemporary FPGAs' on-chip memory capacities. In FPGA implementations of CNNs, off-chip memory is typically used, which limits the speed and bandwidth. Consequently, there has been a lot of research on model compression techniques. One of the most popular compression techniques is network pruning [5]. The pruning process improves compression ratio, but at the expense of parallel computing efficiency due to the irregularities it introduces. In addition to necessitating decoding, a compressed sparse model produces an unbalanced strain on the weights and makes activation reading challenging. As seen in [6], processing a sparse layer requires a significant amount of memory and takes many milliseconds.

By utilising hardware-oriented compression and hybrid quantization techniques, this work demonstrates a more memory-efficient tradeoff between the size of the model and the performance of big CNNs. When considering a CNN's processing feature, the feature map size reduces, despite the fact that the model size grows with the number of layers. Less processing is needed for the decreased feature maps, but more memory is needed for the extended models. Based on the aforementioned feature, all layers are classified as either "no-pruning layers (NP-layers)" or "pruning layers (P-layers)". A high-performance NP-layer uses parallel computation with a regular weight distribution. In spite of its high compression ratio, a P-layer's uneven shape is a result of pruning.

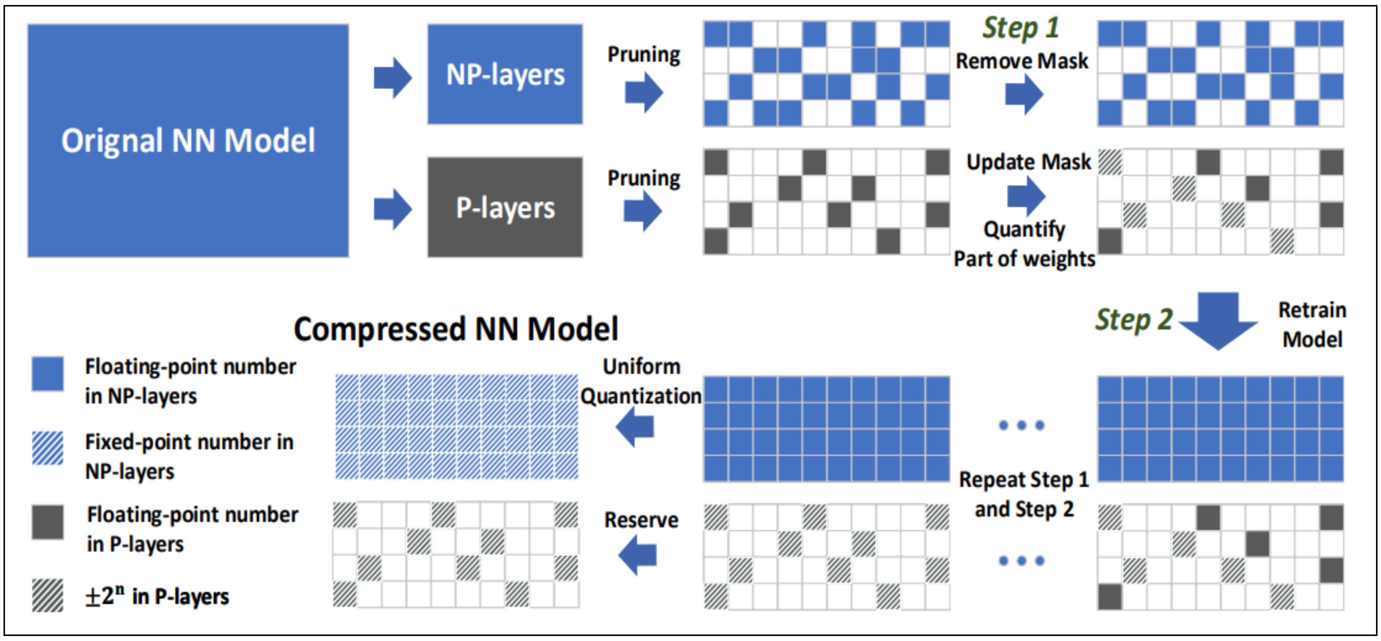

Fig. 1. An outline of the compression approach focused on hardware.

II. PROBLEM DESCRIPTION AND FORMULATION

The literature frequently discusses the design of sparse CNN accelerators. Convolution uses the sparsity of the inter-layer data—a result of activation in CNNs—to eliminate operations that provide zero-valued results in multiplication and addition. To avoid multiply-and-accumulate (MAC) operations with zero values, the Cnvlutin accelerator does not copy zero data from on-chip eDRAM to internal buffers.

The goal of [6] is to decrease the amount of computations needed by eliminating ineffective operations associated with the trimmed weights. The SCNN accelerator takes advantage of the internal dataflow architecture based on Cartesian products to leverage both weight and activation sparsity. A novel PE architecture is presented in [7] to avoid doing operations using zero-valued multiplications and additions. To efficiently find and match NZEs (also called inner-join), Sparten presents an accelerator architecture for vector-vector multiplication with sparse weights and inputs. The aforementioned research, however, primarily address accelerator design and pay little attention to data transfer reduction using compressed data.

In order to take advantage of the sparsity in feature maps, [8] introduced CNN accelerators that are based on FPGAs and ASICs. While prior methods depended only on hardware optimisations for acceleration, our proposed approach combines a hardware accelerator with a software-based compression methodology, even if the CNN accelerators demonstrated in achieved GOPs in particular CNN models.

The Winograd-based CNN accelerator concept on FPGA was first presented by [9]. Nevertheless, our suggested approach utilises HW/SW co-design methodology to expedite sparsity-aware CNN inference while obviating RAM data transfer, in contrast, which fails to account for sparsity in IFMs. Other CNN acceleration designs have also made use of FPGAs [10-12].

III. METHODOLOGY

It proposes a compression method that is hardware-targeted, which would enable large-scale CNNs to store their weights on a chip. The model in the front layers is smaller, but they use a lot of computing power. The front layers also aren't secure: has pruned the filters in the front layers of AlexNet and VGG (though at a slower rate), which drastically lowers accuracy. Parallel computing performance is also impacted by an irregular sparse model. Therefore, it is inefficient to prune the top layers. To make the network front-regular yet back-irregular, the suggested method relies on applying various compression algorithms to different levels.

There are N P-layers and P-layers. It all starts with a thorough pruning of all layers. Quantifying the P-layers is then accomplished using INQ. N non-sparse P-layers have their weights updated using the incremental quantization process to correct the pruning and quantization errors in the P-layers. Consequently, convergence is made easier with the proposed compressed model because it is more error-proof. Equipped with fixed-point numbers, the N P-layers are prepared for calculation once the quantization in the P-layers is complete.

In order to achieve a larger compression ratio, the P-layers are frequently compressed to a very high degree, whereas the N P-layers offer superior parallel performance due to their regular structure. In most cases, the N P-layers will be Conv layers, but they can also be FC layers. Figure 1 provides an overview of the proposed method.

The feature map displays data in a row format. The next stage in achieving parallel performance is to read the F data from various addresses at the same processing speed. Figures 2 and 3 show the feature map after we incorporate the F-adjacent data. The FPPB, which is made up of two F×F blocks called BLOCK0 and BLOCK1, then receives the integrated data (A, B, C…) in a specific order. Data can be input or outputted by these two blocks in a clockwise or anticlockwise fashion. The data is sent to the other block in columns after one block is filled.

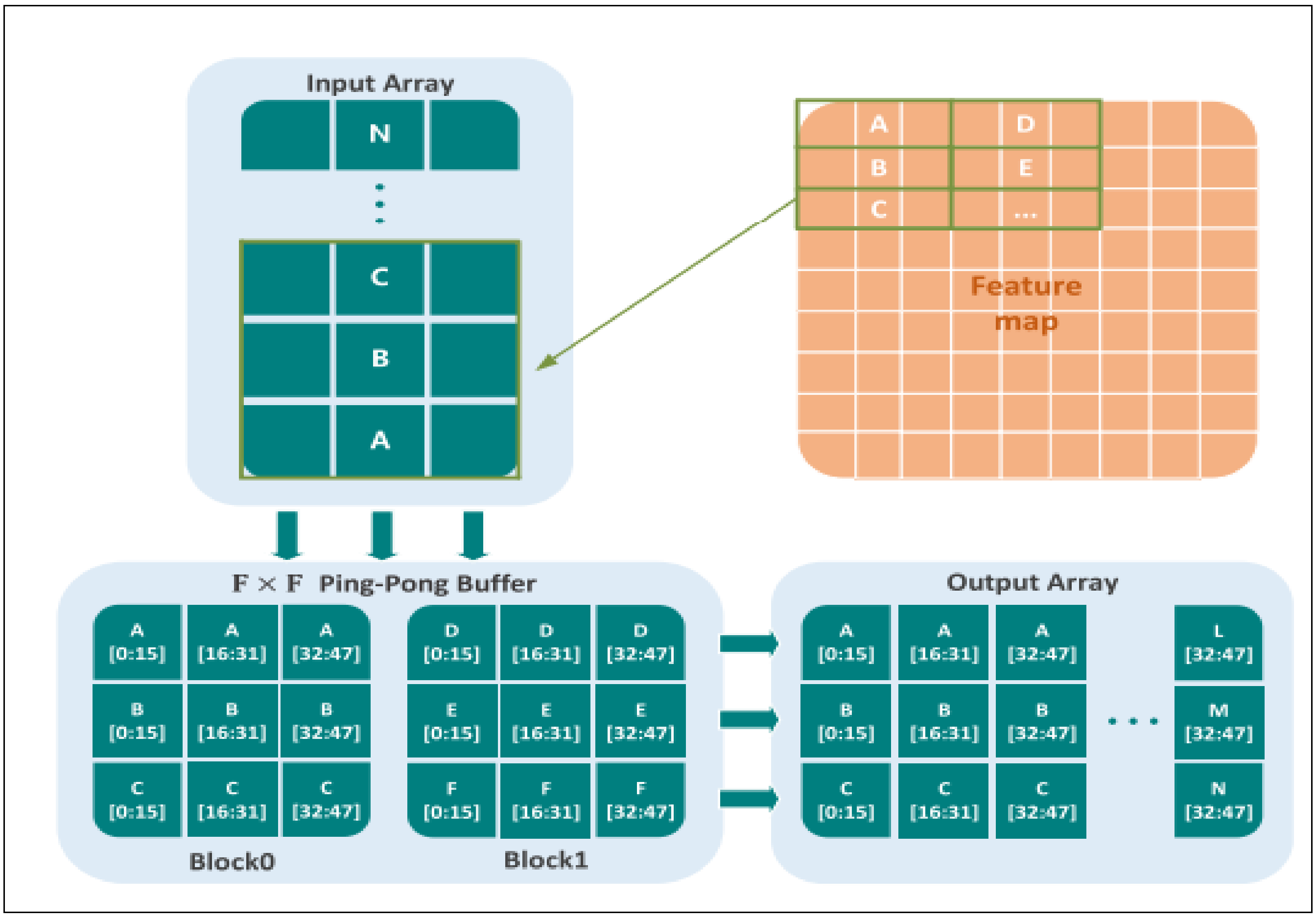

Fig. 2. Parallel FIRs with a F×F ping-pong buffer.

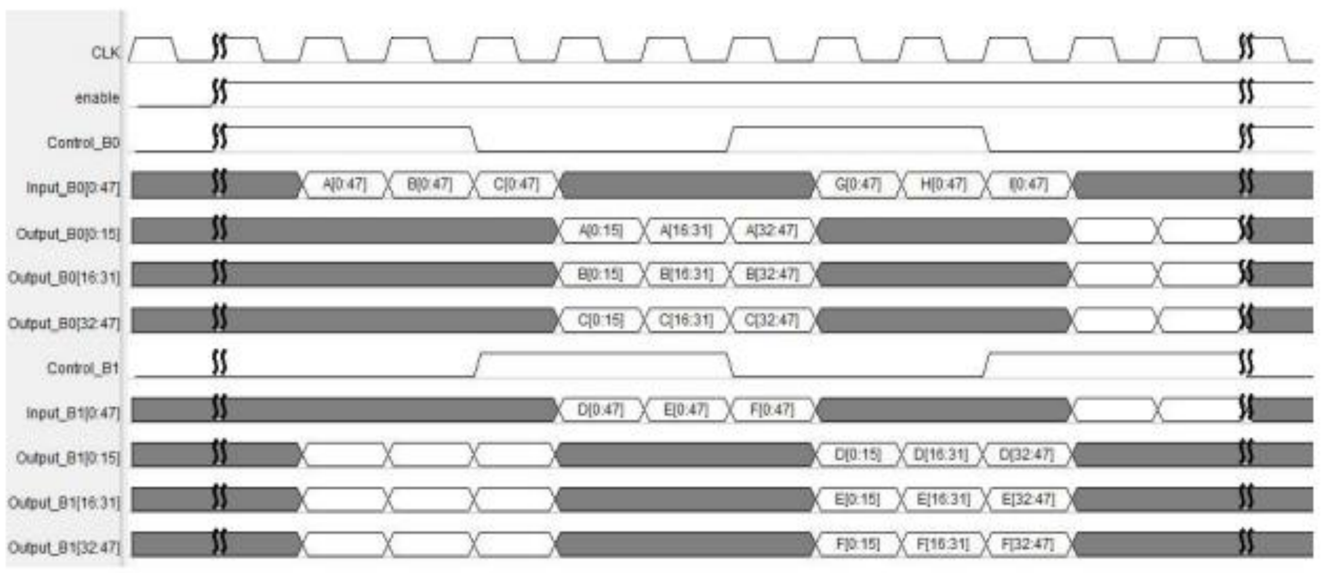

Fig. 3. The proposed FPPB's timing diagram.

IV. RESULTS AND DISCUSSION

The comparison of performance across different levels of IFM sparsity is summarised in Figure 4. The performance has been fine-tuned to accommodate CPU-based CNN inference without IFM compression, so please be advised of that. In order to ensure a fair comparison, our architecture incorporates the time required for compression and DMA data transmission (both read and write) into the total execution time.

Because the proposed acceleration technique drastically cuts down on the time it takes for CNN inferences to run, it also helps to lower energy consumption. We can calculate the power usage using the Vivado tool's anticipated average power results (Table

1). while a system is turned on, static power consumption happens constantly, unlike dynamic power consumption that happens only while a component is in use.

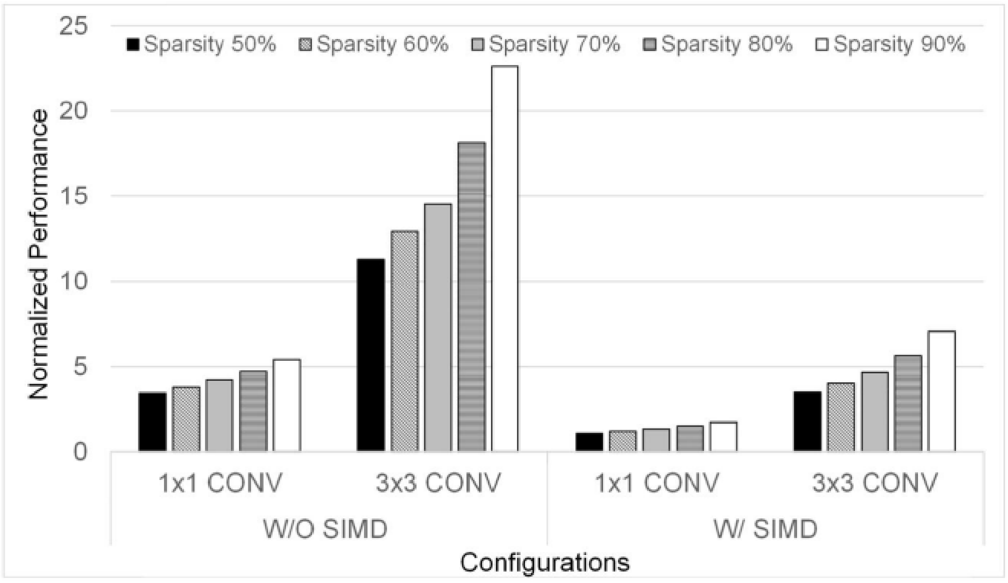

Fig 4 The proposed acceleration method's performance is compared with both CPU-based execution without SIMD and with it.

Table 1. Xilinx Vivado average power results

	Processor system		Programmable logic	
	Static	Dynamic	Static	Dynamic
Power (W)	0.100	2.701	0.602	1.133

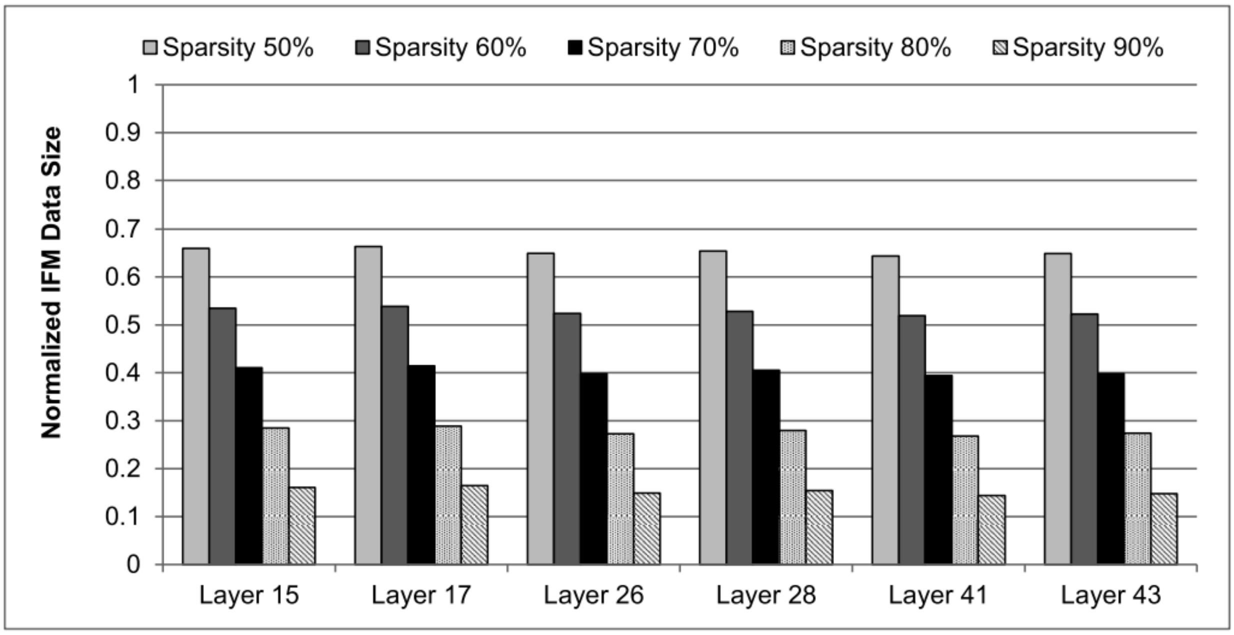

Fig 5. Evaluation of the proposed acceleration technique using normalised compressed data size (metadata included) across various CONV layers

The size of compressed data is significantly affected by the quantity of sparsity in IFM, as seen in Figure 5. Minimising the size of IFM data helps alleviate latency, memory bandwidth pressure, and energy consumption.

V. CONCLUSION

This study initially proposes a compression approach that is based on hardware. This method allows VGG-16 to attain a compression ratio of 27.5× and excellent performance. Using the ILSVRC2012 dataset and the Caffe framework, it was demonstrated that the proposed method resulted in a slight decrease in accuracy when contrasted with the single-precision floating-point implementation. For this case study, we'll be using the Xilinx FPGA VCU118 platform and the proposed compacted VGG-16 architecture. There will be no use of off-chip memory. Researchers found that when compared to alternative approaches that reach the same degree of accuracy in image processing, the design executed using the proposed HACO tool had the maximum performance at 83.0 FPS. The suggested universal Conv-PE structure is very efficient at handling huge Conv kernels, which allows the overall architecture to handle several CNN models with very little supplementary hardware.

REFERENCES

[1] Networks for Large-Scale Image Recognition," arXiv preprint arXiv:1409.1556, 2014.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[3] R. Girshick, "Fast R-CNN," in Proc. IEEE International Conference on Computer Vision, 2015, pp. 1440–1448.
[4] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," arXiv preprint arXiv:1410.0759, 2014.
[5 Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An EnergyEfficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127–138, 2016.
[6] Albericio, J., et al.: Cnvlutin: ineffectual - neuron - free deep neural network computing. In: Proceedings of the 43rd International Symposium on Computer Architecture, pp. 1 – 13 (2016)
[7] Zhang, S., et al.: Cambricon - X: an accelerator for sparse neural networks. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1 – 12 (2016).
[8] Kim, D., Ahn, J., Yoo, S.: A novel zero weight/activation - aware hardware architecture of convolutional neural network. In: Proceedings of the Conference on Design, Automation & Test in Europe, pp. 1466 – 1471 (2017)
[9] Aimar, A., et al.: A flexible convolutional neural network accelerator based on sparse representations of feature maps. IEEE Trans. Neural Netw. Learn. Syst. 30(3), 644–656 (2019)
[10] Kala, S., et al.: High - performance cnn accelerator on FPGA using unified winograd - gemm architecture. IEEE Trans. Very Large Scale Integrat. Syst. 27(12), 2816 – 2828 (2019)
[11] Liu, B., et al.: An FPGA - based cnn accelerator integrating depthwise separable convolution. MDPI Electron. 8(3), 1 – 18 (2019)
[12] Shen, J., et al.: Towards a multi - array architecture for accelerating large - scale matrix multiplication on FPGAs. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1 – 5 (2018)