TF2FPGA: A Framework for Projecting and Accelerating Tensorflow CNNs on FPGA Platforms

Spyridon Mouselinos, Vasileios Leon, Sotirios Xydis, Dimitrios Soudris, Kiamal Pekmestzi
School of Electrical & Computer Engineering, National Technical University of Athens, Greece
mouselinos.spur.kw@gmail.com, {vleon, sxydis, dsoudris, pekmes}@microlab.ntua.gr

***Abstract*—FPGA-based accelerators of Convolutional Neural Networks (CNNs) deliver significant performance, while leaving much room for optimizations. In this paper, we present a CNN accelerator based on the state-of-the-art Dataflow Hardware Mapping (DHM) methodology, optimized with the use of 8-bit unsigned integer quantization, 1-bit input mapping and deep-to-shallow conversion techniques. Interestingly, we introduce a Tensorflow-to-VHDL framework to generate high-performance accelerators for inferencing on FPGAs, adopting a holistic view of CNN problems. The proposed framework uses a VHDL library of our pre-optimized layers and modules, and creates the accelerator by extracting the model definition and weights from the respective Tensorflow files. The 8-bit quantized weights are stored in an on-board ROM memory, and thus, the need for external-to-FPGA "glue logic" modules and increased bandwidth is eliminated. Additionally, we include a Matrix Multiplication-to-FIR conversion method that manages to include the fully connected part of the CNN inside the FPGA. Finally, we achieve 75% less memory footprint and 53% less resource utilization compared to the state-of-the-art based architectures, and up to $\times$10 speedup vs different GPU, CPU combinations.**

***Index Terms*—Machine Learning, Convolutional Neural Networks, FPGA Accelerators, Dataflow Hardware Mapping**

I. Introduction

During the last decade a considerable amount of research effort has been made towards the development and optimization of Convolutional Neural Networks (CNNs), that excel in computer vision applications, such as object recognition [1] and classification [2]. However, such applications belong to the real-time constraint spectrum, meaning that speed and scalability are of high importance, while general-purpose processors (i.e., CPUs, low-end GPUs) tend to become unfit for these tasks. Thus, FPGA-based accelerators [3]–[5] have started to take their place as viable and promising solutions.

In this paper, we propose a framework that enables automatic and transparent generation of high throughput CNN accelerators implemented on FPGA. The proposed framework extends the well known TensorFlow [6] system with automatic FPGA acceleration capabilities. It accepts as input a CNN model description and parameters, i.e., image size, kernel filter window size, number of input and output feature maps, quantization mode, and the synthesis directives, i.e., 1-bit AND-Gate mapping, data bit-width etc., and automatically generates the CNN accelerator architecture for the target device in VHDL. The proposed framework is presented in Fig. 1.

We surpass the obstacles of the high convolutional complexity by transforming our Deep Convolutional Model into a Wide-and-Shallow (W&S) one, using the results of [7].

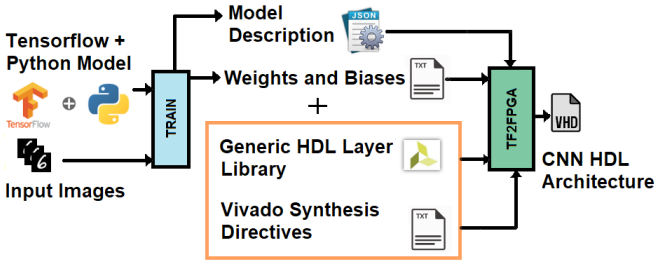

Fig. 1: TF2FPGA Framework Design.

This transformation caused the accumulation of the major computational load into the first convolutional layer, that got heavily pruned by replacing its Multiply-Accumulator (MAC) units with AND gates with our proposed 1-bit Input Mapping method. Consulting the LUT/DSP mapping techniques of [5], [8] we managed to maintain a restricted resource utilization scheme, enabling the mapping of more complex models even on small target FPGAs. Furthermore, we decided to take advantage of the newly added 8-bit Integer Quantization utility of Tensorflow Lite in order to train our own 8-bit network with as less as 0.8% drop, maintaining a balance between the performance/accuracy trade-off. Finally, data reuse is achieved through the transformation of cascading matrix multiplications in 2D convolutions into parallel reverse-FIR filters, surpassing "external-memory" latency and throughput capacity barriers.

II. Background

Optimized Direct Hardware Mapping of CNNs

The establishments of dataflow Models of Computation (MoC) were formalized by [9] and led to the concept of Direct Hardware Mapping (DHM), where a computational data-driven graph is projected through channels and actors on hardware. DHM of CNNs completely removes the need for an external memory to store intermediate results or parameters. Moreover, thanks to the fully pipelined execution model, the global throughput is only limited by the maximum clock frequency [8]. Taking this into consideration, approaches of DHM mapping directives and design strategies have been proposed in the literature [5], [8].

Quantization in Image Classification Tasks

There has been also significant effort to develop a principled approach that abandons floating point architectures and converts a model to its more efficient fixed point equivalent. Inspired by the interesting results of [10] proving that fixed point

978-1-7281-1184-1/19/$31.00 ©2019 IEEE

CIFAR-10 networks can achieve more than 20% reduction in model size, while maintaining a highly accurate performance, we pushed the boundaries of the accuracy-memory trade-off by applying an 8-bit Uint Quantization to our model weights.

Wide-and-Shallow CNNs

The Wide-and-Shallow design pattern targets to transform an Artificial Neural Network (ANN), that is small in node width but large in layer depth, into a large in width but shallow in depth one. With this method, the overall training time is reduced heavily, as well as the memory size of the models weights and biases. According to the results of [7], [11], while this method may not work out of the box, a careful grid search of sizes, activations and learning rates can reproduce the same generalization performance on regression or classification tasks, despite their obvious difference in the shape of layers and information capacity. On CIFAR-10 [7], these techniques lead to a better generalization capabilities of that of a deep model, giving better results. Even in other types of ANNs, i.e., the Residual Neural Networks [11], this design concept gave promising results in terms of performance. For these reasons, we decided to innovate by transferring the Wide-and-Shallow technique into the FPGA design spectrum. Taking advantage of its reduced memory footprint and simplicity of the model architecture, our framework leads to designs that are easier to test and prototype. Finally, by condensing all the CNN computation volume into the first layer, we introduce further optimizations such as our proposed 1-bit Input Mapping.

III. CNN Mapping Optimizations

The proposed framework applies a series of optimization steps for delivering the accelerated CNN designs. The remainder of this section introduces the proposed optimizations.

A. Deep to Wide-and-Shallow Conversion

We begin by performing a Pareto grid search on the deep to Wide-and-Shallow CNN conversion of the LeNet model [12], similar to the exploration strategy followed in [13], considering the following parameters:

- number of convolutional layers
- number of kernels per layer
- size of kernel matrix
- number of fully connected layers
- size of fully connected layer neurons

We opt for an accuracy vs model complexity trade-off, thus we search for convolutionally heavy, fully connected light model. Taking into account this trade-off, the resulting optimal model consists of one convolution layer of two 5x5 filters, one 2x2 max pooling layer, ReLu activation, one flatten layer and a final 10-neuron fully connected layer. The model was trained for 100 epochs of batch size 256 and intermediate dropout layer of 65% keep probability.

B. 1-Bit Input Mapping

After taking a closer look to our task, one can easily derive its edge-detection based nature. This nature is proven to be highly robust to noise and artifacts as well as easily captured within 1-2 layers of filtering and non-linearity [14]. In addition, the used images are in single-channeled grayscale 8-bit format, i.e., the pixels are in the range [0-255]. Thus, we propose a pre-processing method that maps the pixels under a threshold of 125 to 0, and the rest to 1. Therefore, the input pixels are driven in the convolution layer as illustrated in Fig. 2. The proposed mapping distorts slightly the input images, without altering their local information correlations.

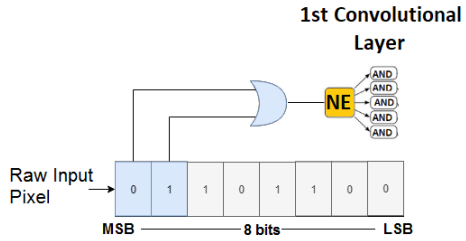

Fig. 2: Thresholding for 1-bit Input Mapping ($T = 125$).

TABLE I: Performance Impact of Structural Transformations

Structural Transformations	CPU Exec. Time*
Deep	$0.420s$
W&S + 8-bit Uint Quantization + 1-bit Input Mapping	$0.015s$

**full script execution for batch size 1.*

Finally, it affects slightly our final model's performance (i.e., 0.08% accuracy loss as shown in Table II), while enabling the MAC-to-AND conversion technique. With this technique, in the high-resource consuming convolutional layer the respective MAC modules (either LUT-based or DSP-based) are replaced by AND-Gates that use the input pixel as an enable signal (0/1) to forward the ROM stored weights to the next layer. As a result, the required resource utilization is significantly reduced, and the timing constraints imposed by the LUT/DSP methods are relaxed.

C. 8-Bit Uint Quantization

The next step was to perform the 8-bit Uint Quantization with the use of Tensorflow Lite quantization methods. These methods provide a pseudo-training node insertion and retraining of the final CNN graph. Throughout the procedure, we tried to minimize the loss function and maintain the original evaluation accuracy of the model, while constraining its weights to the desired bit-width. That led to a significant 75% memory footprint reduction while delivering easy-to-handle positive integer values.

The Deep to Wide-and-Shallow conversion, the 1-bit Input Mapping and the 8-bit Uint Quantization are structural transformations concerning the CNN inference engine, being implementation-independent. Thus, we can evaluate the impact of the aforementioned transformations at SW level, as shown in Table I that reports a $\times 28$ speedup. We note that the presented execution results refer to the full script execution for batch size 1.

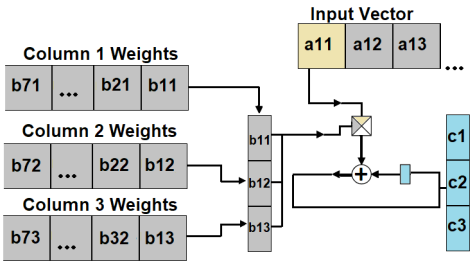

Fig. 3: Matrix Multiplication-to-FIR conversion.

D. Streaming Optimizations

Finally, targeting a high-throughput accelerator as a standalone system, we transformed matrix multiplications in fully connected layers. The matrix multiplication does not comply with the continuous operation of the pipelined design. It requires all operands to be available, while the results of the previous layer are coming in a streaming fashion. Here we take advantage of the property that swapping the columns of the coefficient table will not disturb the final result, and thus, we transform the act of multiplying a matrix with a vector into a parallel FIR-like operation.

In Fig. 3, we present the multiplication of an 1x7 input vector with a 7x3 matrix using the proposed architecture. Each of the FIFO streamed input items are used in a MAC with their respective matrix coefficients, while partially creating the correct output vector. By the time the last item is served, the correct result will be forwarded to the next layer and the processing of the next batch will start with no delay.

IV. Experimental Results

For the design generation, we employed Xilinx Vivado 2017.2, with the post-implementation reports providing the required information to evaluate our work. The primary target device was a Zybo Zynq-7000 (XC7Z010) with 28K logic cells, 240KB Block RAM and 80 DSP slices. For the speed test vs the CPU/GPU combinations, a 6-core AMD RYZEN 1600 and a 6.1 CUDA capability GPU were used. The model development framework was Tensorflow 1.10, compiled with and without the AVX2 optimized instruction set.

A. Accuracy Evaluation

We evaluate the accuracy drop of the Wide-and-Shallow converted models vs the originally generated models of the MNIST dataset. More specifically, we perform a comparison between the proposed 8-bit Integer Quantization method and a simple post-training rescaling & rounding to the nearest integer of the model weights. Moreover, we apply the 1-bit Input Mapping to both examined techniques.

As shown in Table II, the 8-bit Unit Quantization model delivers only 0.81% accuracy loss. On the contrary, the application of the post-training rescaling & rounding method gives poor results, as the accuracy drops over 2%. It is also proven that the 1-bit Input Mapping, if combined with the proposed Integer Quantization method, provides negligible extra accuracy drop, i.e., 0.08%, in exchange for the upcoming memory and utilization optimizations.

TABLE II: Accuracy Loss w.r.t. Quantization

Quantization Method	Accuracy Loss (%)
8-bit Uint Quantization	0.81
8-bit Uint Quantization + 1-bit Input Mapping	0.89
Post-Rescaling/Rounding	2.80
Post-Rescaling/Rounding + 1-bit Input Mapping	3.20

TABLE III: Performance w.r.t. Mapping

Mapping Configuration	Latency (CC)	Throughput (CC/res)
Exclusive Use of DSPs	827	224
1-bit Input Mapping w/ LUTs	824	223
1-bit Input Mapping w/ LUTs + DSPs for FCLs	825	222

TABLE IV: Memory Footprint w.r.t. Quantization

Quantization Method	Memory Size (KB)
Deep Model Weight Size	50.64
W&S Model Weight Size	13.28
W&S + 8-bit Uint Quantization	3.32
W&S + Post-Rescaling/Rounding	4.56

B. Performance Evaluation

We also evaluate the initial latency (clock cycles) and the throughput (clock cycles per result) of our design. We chose clock cycles (CCs) as a unit of measurement, so that the performance results would be independent of the target platform and system clock. Interestingly, we examined various design options w.r.t. mapping: (i) use of DSPs for the full mapping, (ii) 1-bit Input Mapping with LUT-based adder trees and multipliers, (iii) 1-bit Input Mapping with LUT-based adder trees and multipliers and use of DSPs in the fully connected layers (FCLs).

Table III shows that all the examined methods provide sufficient performance, with the fastest configuration being the usage of AND gates in the convolutional layers, that is enabled by the 1-bit Input Mapping, combined with the employment of DSPs in the fully connected layers.

C. Resource Utilization Evaluation

According to Table IV, the conversion of the Deep model to a Wide-and-Shallow one results to a 73.7% reduction in the on-board required memory. Subsequently, by performing the 8-bit Uint Quantization, we achieve a total memory saving of 93.4% in comparison with the original model. Compared to the naive post-training rescaling & rounding method, the proposed technique delivers a memory reduction of 27.1%.

Furthermore, as shown in Table V, the 1-bit Input Mapping reduces the LUT utilization by 10.2%, and achieves a 53.6% LUT reduction when combined with the DSP usage in the fully connected layers. Finally, we comply with the FPGA's available resources, as the exclusive employment of DSPs results in over-utilization (120 out of 80 available DSPs).

TABLE V: FPGA RESOURCE UTILIZATION

Design	LUT	FF	DSP
8-bit Uint Quantization + Exclusive Use of DSPs	11540	2530	120
8-bit Uint Quantization + 1-bit Input Mapping w/ LUTs	10360	2545	0
8-bit Uint Quantization + 1-bit Input Mapping w/ LUTs + DSPs for FCLs	5353	2536	60

TABLE VI: TF2FPGA VS CPU/GPU

Implementation	Full	Main Proc	Ex + Mem	Ex
CPU + GPU	$4.239s$	$2.851s$	$2.633s$	$1.223s$
CPU AVX2 + GPU	$3.839s$	$2.751s$	$1.933s$	$1.203s$
CPU	$3.469s$	$2.497s$	$2.211s$	$1.201s$
CPU AVX2	$2.920s$	$2.104s$	$1.942s$	$1.007s$
TF2FPGA Accelerator	—	—	—	$0.109s$

D. Speedup vs CPU/GPU Implementations

After confirming the optimization choices with the aforementioned evaluation analysis, the final FPGA accelerator incorporates a Wide-and-Shallow model with the usage of 1-bit Input Mapping, 8-bit Uint Quantization, and DSPs in the fully connected layers. Below, we perform a comparison vs a combination of CPU/GPU configurations on the same model. The CPU configurations include compilation of our Tensorflow environment with and without the AVX2 instruction set.

The model's inference engine was evaluated end-to-end as well as considering the various phases of CPU/GPU execution:

- *Full*: full Tensorflow-model execution
- *Main Proc*: omits device discovery and initialization time
- *Ex + Mem*: omits pre-processing and data serialization
- *Ex*: omits memory loading time, pure inference execution

As shown in Table VI, the proposed solution outperforms the rest of the implemented configurations. Surprisingly, CPU configurations perform better than the GPU ones, mainly due to the limited inter layer parallelism/depth, thus under-utilizing the GPU resources. However, we note that this observation is model-specific, meaning that deeper inference engines would probably be benefited more with GPU execution.

Finally we present a total overview of our contribution in Fig. 4, where we illustrate the superiority of the proposed TF2FPGA accelerator compared to the simple Wide-and-Shallow model as well as the state-of-the art Deep models.

V. CONCLUSION

In this paper, we introduced a framework for transforming CNN models into high throughput standalone CNN accelerators. Targeting increased performance and small resource utilization on the FPGA, the proposed framework enables structural transformations, i.e., Deep to Wide-and-Shallow conversion, 1-bit Input Mapping, and 8-bit Uint Quantization, as well as streaming optimizations such as the MatMul-to-FIR conversion. The experimental results show that the proposed TF2FPGA accelerator delivers negligible accuracy loss (0.89%), while offering small memory requirements and decreased LUT utilization. Finally, compared to various CPU/GPU implementations on the same model, the proposed accelerator achieves a $\times 10$ speed-up.

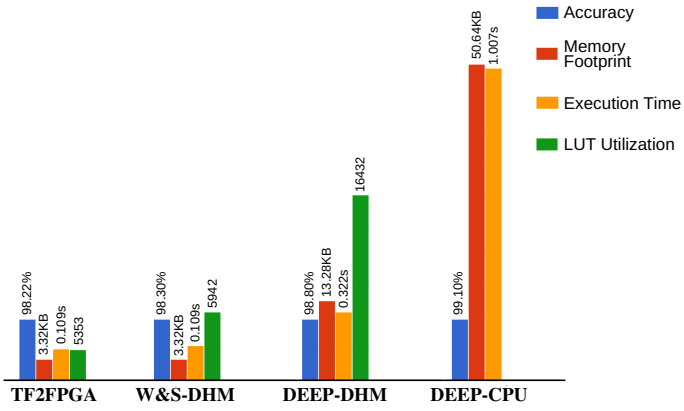

Fig. 4: Normalized evaluation results of TF2FPGA.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Conference on Neural Information Processing Systems (NIPS)*, Dec. 2012, pp. 1097–1105.

[3] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From High-level Deep Neural Models to FPGAs," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[4] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A Complete Design Flow for Mapping CNN onto Customized Hardware," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2016, pp. 24–29.

[5] S. Venieris and C. Bouganis, "fpgaConvNet: Automated Mapping of Convolutional Neural Networks on FPGAs," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Feb. 2017, pp. 291–292.

[6] M. Abadi *et al.*, "TensorFlow: A System for Large-scale Machine Learning," *CoRR*, vol. abs/1605.08695, 2016. [Online]. Available: http://arxiv.org/abs/1605.08695

[7] L. J. Ba and R. Caurana, "Do Deep Nets Really Need to be Deep?" *CoRR*, vol. abs/1312.6184, 2013. [Online]. Available: http://arxiv.org/abs/1312.6184

[8] K. Abdelouahab and M. Pelcat, "Tactics to Directly Map CNN Graphs on Embedded FPGAs," *CoRR*, vol. abs/1712.04322, 2017. [Online]. Available: http://arxiv.org/abs/1712.04322

[9] J. B. Dennis and D. P. Misunas, "A Preliminary Architecture for a Basic Data-flow Processor," in *International Symposium on Computer Architecture (ISCA)*, Jan. 1975, pp. 126–132.

[10] L. Lai, N. Suda, and V. Chandra, "Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations," *CoRR*, vol. abs/1703.03073, 2017. [Online]. Available: http://arxiv.org/abs/1703.03073

[11] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5MB model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[14] R. Chellappa, K. Fukushima, A. K. Katsaggelos, S.-Y. Kung, Y. LeCun, N. M. Nasrabadi, and T. A. Poggio, "Guest editorial applications of artificial neural networks to image processing," *IEEE Transactions on Image Processing*, vol. 7, no. 8, pp. 1093–1096, Aug. 1998.