EPJ Web of Conferences **245**, 01023 (2020) https://doi.org/10.1051/epjconf/202024501023
CHEP 2019

Highly Performant, Deep Neural Networks with sub-microsecond latency on FPGAs for Trigger Applications

Noel Nottbeck[1], *Christian* Schmitt[1,2,*], and *Volker* Büscher[1,2]

[1]Institut für Physik, Johannes Gutenberg-Universität Mainz, Mainz; Germany
[2]Cluster of Excellence PRISMA$^+$, Johannes Gutenberg-Universität Mainz, Mainz; Germany

Abstract. Artificial neural networks are becoming a standard tool for data analysis, but their potential remains yet to be widely used for hardware-level trigger applications. Nowadays, high-end FPGAs, often used in low-level hardware triggers, offer theoretically enough performance to include networks of considerable size. This makes it very promising and rewarding to optimize a neural network implementation for FPGAs in the trigger context.
Here an optimized neural network implementation framework is presented, which typically reaches 90 to 100% computational efficiency, requires few extra FPGA resources for data flow and controlling, and allows latencies in the order of 10s to few 100s of nanoseconds for entire (deep) networks.

1 Introduction

Deep neural networks are widely and very succesfully used in high energy physics for object reconstruction and identification as well as in event reconstruction and signal-background separation. However, only few examples exist yet within low-level hardware triggers, where mainly field-programmable gate arrays (FPGAs) are used. One concrete implementation is the z-vertex trigger of the Belle-II experiment [1]. In contrast to the existing 'high level synthesis for machine learning' framework [2], this work [3] uses a bottom-up approach for implementing general networks on FPGAs, which grants maximum control over each part and therefore allows for specific hardware optimizations.

The ATLAS detector [4] and its upgrades of the first level trigger system [5, 6] is hereby used as a reference, since there the availability of state of the art FPGAs as well as tight constraints on the incoming data rate of 40 MHz and on the latency (tens to few hundreds of nanoseconds for the network inference) provide an ideal test environment.

For the implementation of the neural network layers the focus is on an efficient resource usage and at the same time on an easy usability of the framework where no detailed knowledge of the underlying implementation is needed. In contrast to CPUs or GPUs, where neural networks typically rely on floating-point arithmetics, fixed point arithmetics with a per network layer configurable precision is used. The calculations are performed within the digital signal processors (DSPs) of the FPGA that allow, in the case of the Xilinx UltraScale+ XCVU9P-2 used here, one multiplication with a subsequent addition (MAC) per cycle and DSP. For the incoming data rate of 40 MHz and an FPGA processing frequency of 640 MHz this translates into a theoretical upper limit of about 100k MACs/event.

*e-mail: schmittc@uni-mainz.de

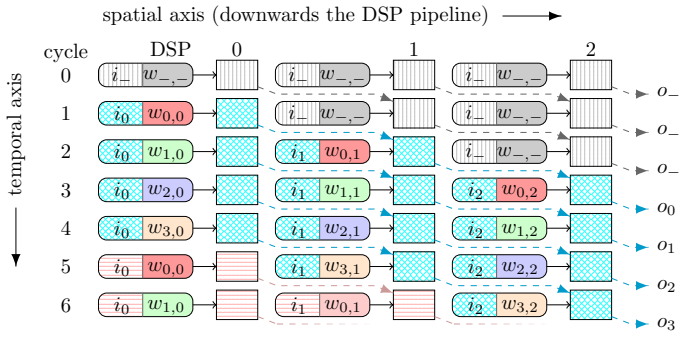

Figure 1. Schematic showing the data flow for the fully-connected layer. Gray (vertical lines) refers to no data set available/idling, while cyan (crosshatch) and pink (horizontal lines) refer to data of a first and second data set, respectively. The inputs of neighboring pipeline stages are updated in subsequent cycles and only once per data cycle, while the weight sequence for each individual DSP is repeated during each data cycle. [3]

2 Network layer implementation

As each network type has specific data flow and processing needs, each layer type is implemented individually. The final network is then constructed by pipelining several layers behind each other. One important ingredient in the implementations is the expected range of the ratio between the processing and the data frequency, $C = \frac{f_\mathrm{P}}{f_\mathrm{D}}$. Given that the maximum FPGA frequency is typically below the 1 GHz mark, the corresponding limit for the values of C in the case of e.g. ATLAS is at most ~ 20. This allows to use multiplexers for input paths, as only few different inputs will be required. Finally, to maintain synchronicity between the layers, all inputs must be read in within C cycles and all results must be available within C cycles with an arbitrary (but determinable and finite) processing delay in between.

Fully-connected layers

In fully-connected layers every neuron requires every input at some point. This fact can be exploited by implementing the neuron processing in pipelines of DSPs: the input to the DSPs stays the same and the weights are updated during each cycle as shown in figure 1. The pipeline can be divided into smaller parts to reduce the overall latency and to allow for cases where multiple new inputs are available during each cycle from the previous part of the pipeline.

2D convolution layers

Convolutional layers typically feature only few parameters but a significant amount of arithmetic operations due to the manifold application of the same kernel. The number of multiplications, N_MUL, in 2D convolution layers scales with the product of the input (V_I) and the kernel (V_K) volumes, $N_\mathrm{MUL} \approx V_\mathrm{I} \cdot V_\mathrm{K}$. This requires an efficient way to reuse both inputs and weights as much as possible in order to save resources. This is achieved by using one 'row' of data as smallest unit for the distribution to the different processing units, the so called 'row units', where a row is identified by its height and channel index and it spans the entire width of the volume as shown in figure 2. Initially, all row units begin processing data for the same output channel but of directly neighboring slices. Cycle by cycle, every row unit progresses to the next channel of the same slice. Further details can be found in [3].

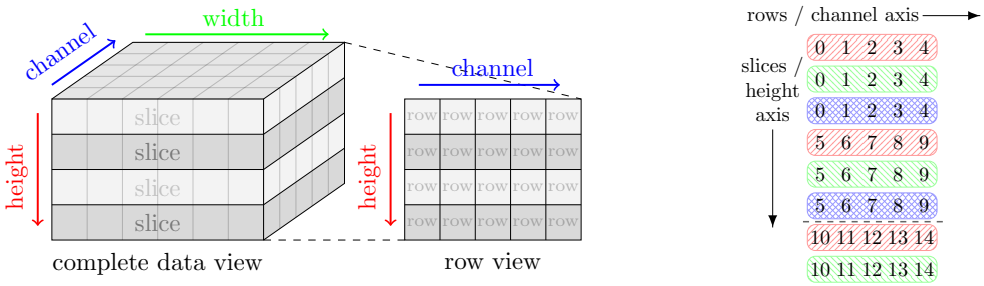

Figure 2. The Illustration on the left shows how 'slices' and 'rows' are related to the original data volume. A slice contains all elements at a given height, a row spans the entire width at a given height and channel position. The schematic on the right shows an example of the row unit allocation scheme. Positions marked with a 0 are computed first, all others require the indicated number of delay cycles. The different colors (patterns) indicate which row unit covers a given row. [3]

2D maximum pooling layers

2D maximum pooling layers are implemented using a similar strategy but since normally, i.e. in case of non-overlapping pooling, each input is uniquely assigned to an output, the row allocation can simply be done from top to bottom channel and top to bottom slice without the need for input sharing.

Network creation toolkit

A Python-based toolkit has been developed that takes a trained Keras [7] network as input together with design parameters like e.g. the bit widths to be used for the calculations in the various layers and then creates all of the needed VHDL code to implement the network on an FPGA.

3 Results

Given our target data input frequency of 40 MHz and the capabilities of the Xilinx XCVU9P-2 FPGA, possible network sizes of at most multiple 10^4 MACs have been studied. For comparison, simple deep neural networks in use by ATLAS start at a few 10^3 MACs for fully-connected networks [8].

All of the results reported in the following have been obtained using this toolkit for different networks of the MNIST digit classification task as example networks. This allowed to test all of the layer types supported by the toolkit in an easily comprehensible way without any loss of generality: the results can be directly transferred to any other task with similar layer or network complexity.

Individual layers

For fully-connected layers the main limitation is the number of available DSPs, which scales as $N_{\mathrm{DSP}} \approx \frac{N_{\mathrm{I}} \cdot N_{\mathrm{N}}}{C}$, where N_{I} is the number of inputs, N_{N} the number of neurons and C the ratio of FPGA processing frequency divided by the data input frequency. The usage of other resources of the FPGA is minimal: on average about 8 flip-flops (FF) and 2 look-up tables (LUT) are needed per DSP for the tested networks. This has to be compared to about 350 FFs and 175 LUTs per DSP that are available in the chosen FPGA.

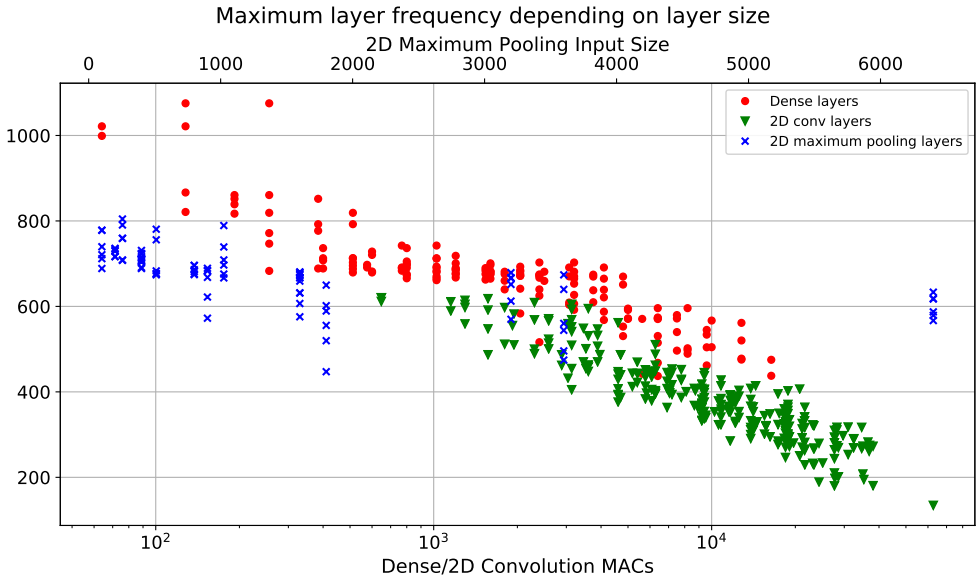

Figure 3. Plot showing the maximal layer frequency for the different layer types as a function of their complexity in terms of multiply-accumulate (MAC) operations. [3]

The increased design complexity of 2D convolution layers directly translates into an increased resource usage of about 34 FFs and 21 LUTs per DSP, which still represents only a moderate fraction of the available resources in the FPGA. The resource requirements of the 2D maxium pooling layers are in contrast negligible as no DSPs are used and per input value only 5 to 16 LUTs and between 18 to 29 FFs are needed.

The maximal achievable frequency for the different layer types as a function of the layer complexity is shown in figure 3. Fully-connected layers and 2D maximum pooling layers achieve frequencies above 400 MHz even for the most demanding layers that have been tested. The higher complexity of 2D convolution layers translates into generally lower maximal frequencies but still reach about 200 MHz even for layers with up to 40k MACs.

Entire networks

The resource requirements of entire networks are dominated by the 2D convolution layers since these are the most resource demanding. Hence also the resource ratio with respect to all available FPGA resources is close to the usage of individual 2D convolution layers. While implementing the networks the data input frequency was always kept at 40 MHz while the target processing frequency was scanned in multiples of the data input frequency. Networks with up to 15k MACs could succesfully achieve the imposed timing constraints, while larger networks did not yet allow to process data with an input rate of 40 MHz as can be seen in figure 4. With future timing improvements, especially in the 2D convolution layers, this limit of about 15k MACs is expected to be pushed to significantly higher values.

Especially for the usage within trigger applications the total network latency also plays an important role. The exact latency of a given network depends strongly on the number of layers, i.e. the depth of the network, and the overall size of the network. Networks with four layers that need a few thousand multiplications can achive latencies between 100 and

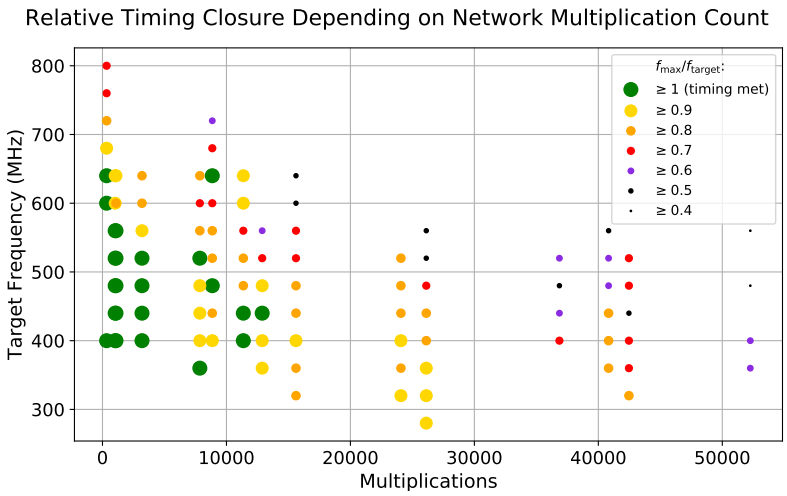

Figure 4. Plot showing the relative network timing closure depending on the network size (multiplications) and the target frequency (multiple of the 40 MHz data frequency). Every column represents one example network. [3]

150 ns while networks with six layers and several ten thousand multiplications require a few hundred nanoseconds. Figure 5 shows the obtained latencies for various networks together with the design latencies as a function of the data to processing frequency ratio C under the assumption that the timing closure is always achieved. This shows that with future frequency improvements, i.e. increased values for C, it will not only be possible to reduce the real-time latency significantly but also to save resources on the FPGA as e.g. the number of required DSPs is inversely proportional to C.

4 Summary and Outlook

A neural network implementation framework has been developed that allows to convert a trained deep neural network into a fast and efficient FPGA implementation providing a substantial inference performance at incoming data rates of e.g. 40 MHz while reaching overall latencies in the order of tens to a few hundreds of nanoseconds. This makes it possible to include such networks into high-performance detector triggers without the need of expert knowledge about FPGA implementations. The framework is Python-based and easy to use. Network size limits are only given by the available latency budget and the capabilities of the chosen FPGA.

Further improvement possibilities, especially in the 2D convolution layers, have already been identified and hence the current performance limits can be expected to be surpassed significantly in the near future.

Acknowledgements

This research is supported by the Bundesministerium für Bildung und Forschung (BMBF).

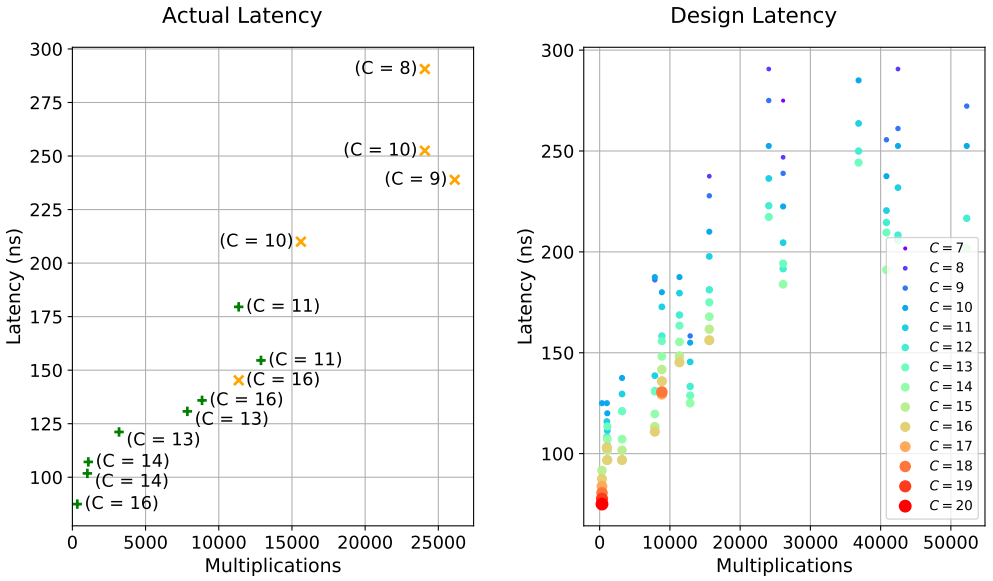

Figure 5. The achieved latencies for some of the networks are shown on the left. Networks which reached timing closure are indicated with a green cross while the ones that failed by a small fraction are shown as yellow cross. The values for the ratio between the processing and the data frequency, C, of the individual networks are indicated in brackets. Under the assumption that timing closure can be achieved, the right plot shows the network latency depending on C where each column represents one example network. [3]

References

[1] S. Neuhaus, S. Skambraks, C. Kiesling, EPJ Web Conf. **150**, 00009 (2017)

[2] J. Duarte et al., JINST **13**, P07027 (2018), 1804.06913

[3] N. Nottbeck, C. Schmitt, V. Büscher, JINST **14**, P09014 (2019), 1903.10201

[4] ATLAS Collaboration, JINST **3**, S08003 (2008)

[5] ATLAS Collaboration, *Technical Design Report for the Phase-I Upgrade of the ATLAS TDAQ System*, CERN-LHCC-2013-018, ATLAS-TDR-023 (2013)

[6] ATLAS Collaboration, *Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System*, CERN-LHCC-2017-020. ATLAS-TDR-029 (2017)

[7] *Keras: The python deep learning library*, https://keras.io/

[8] ATLAS Collaboration, *Identification of Hadronically-Decaying W Bosons and Top Quarks Using High-Level Features as Input to Boosted Decision Trees and Deep Neural Networks in ATLAS at* $\sqrt{s} = 13$ *TeV*, ATL-PHYS-PUB-2017-004 (2017)