PROCEEDINGS OF SCIENCE

Accelerating Machine Learning inference using FPGAs: the PYNQ framework tested on an AWS EC2 F1

Marco Lorusso,[a,b,*] **Daniele Bonacorsi,**[a,b] **Davide Salomoni,**[b,c] **Riccardo Travaglini,**[a,b] **Diego Michelotto,**[c] **Doina Cristina Duma**[c] **and Paolo Veronesi**[a]

[a] *INFN Bologna,*
viale Berti Pichat 6/2, Bologna, Italy
[b] *Department of Physics and Astronomy, University of Bologna,*
viale Berti Pichat 6/2, Bologna, Italy
[c] *INFN CNAF,*
viale Berti Pichat 6/2, Bologna, Italy
E-mail: marco.lorusso11@unibo.it, daniele.bonacorsi@unibo.it, d.salomoni@unibo.it, riccardo.travaglini@bo.infn.it, diego.michelotto@cnaf.infn.it, doinacristina.duma@cnaf.infn.it, paolo.veronesi@bo.infn.it

In the past few years, using Machine and Deep Learning techniques has become more and more viable, thanks to the availability of tools which allow people without specific knowledge in the realm of data science and complex networks to build AIs for a variety of research fields. This process has encouraged the adoption of such techniques, e.g. in the context of High Energy Physics. In order to facilitate the translation of Machine Learning (ML) models to fit in the usual workflow for programming FPGAs, a variety of tools have been developed. One example is the HLS4ML toolkit, which allows the translation of Neural Networks (NN) built using tools like TensorFlow to a High-Level Synthesis description (e.g. C++) in order to implement this kind of ML algorithms on FPGAs.
This paper presents the activity running at the University of Bologna and INFN-Bologna devoted to preliminary studies for the trigger systems of the Compact Muon Solenoid experiment at the CERN LHC accelerator. An open-source project from Xilinx called PYNQ is being tested combined with the HLS4ML toolkit. The PYNQ purpose is to grant designers the possibility to exploit the benefits of programmable logic and microprocessors using the Python language. The use of cloud computing in this work allows us to test the capabilities of this workflow, from the creation and training of a Neural Network and the creation of a HLS project using HLS4ML, to managing NN inference with custom Python drivers.
The main application explored in this work lives in the context of the trigger system of the CMS, where new reconstruction algorithms are being developed due to the advent of the High-Luminosity phase of the LHC.

41st International Conference on High Energy physics - ICHEP2022
6-13 July, 2022
Bologna, Italy

*Speaker

PoS(ICHEP2022)243

1. Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) [1, 2] are devices that implement circuits just like hardware, providing huge power, area and performance benefits over software, yet they can be reprogrammed cheaply and easily to implement a wide range of tasks.

Because customizing an FPGA involves storing values to the memory bits that control every routing choice, the creation of an FPGA based circuit is a process of creating a bitstream to load into the device. This is usually done starting with an application written in a hardware description language (HDL), such as VHDL or Verilog, however in this work a "higher-level" approach is followed, using tools and libraries that make it possible to finalize a FPGA design starting from a *behavioural description* written in C++ or, in the case of Neural Networks, in Python.

1.1 AWS EC2 F1 Instance

In order to test the capabilities of the implementation workflow presented in this work, cloud computing resources, more specifically Amazon Web Services' EC2 F1 instances, equipped with Xilinx FPGA acceleration cards, have been used. F1 instances are equipped with tools to develop, simulate, debug, and compile a hardware acceleration code, including an FPGA Developer Amazon Machine Image (AMI) which supports a range of development environments suited for low-level hardware developers, as well as software developers who are more comfortable with C/C++ and OpenCL environments. Once an FPGA design is complete, it can be registered as an Amazon FPGA Image (AFI), and deployed to every F1 instance needed.

2. The Implementation of a NN on FPGA

Improving the transverse momentum measurement performed by the Compact Muon Solenoid (CMS) muon Level-1 trigger, namely the momentum resolution, is very important to achieve a reduction of trigger rates. In fact, due to the rapidly decreasing shape of the inclusive muon p_T spectrum, even a relatively small reduction of the resolution can provide a significant decrease of the trigger rate at a given p_T threshold, by reducing the number of low momentum muon candidate misidentified as high momentum ones. This becomes particularly pressing in the context of future upgrades of CMS, in view of the High Luminosity LHC upgrade, to avoid using higher momentum thresholds as luminosity increases, with the consequence of losing physics acceptance.

By implementing an Artificial Neural Network (NN), a class of Machine Learning (ML) algorithms, on an FPGA, this work places itself in the search for ways to make the p_T prediction faster, other than more accurate.

2.1 The Model

The model built for this research is the next iteration of the Fully Connected Multilayer Perceptron regressor designed for my previous work in [2, 3]. Its purpose was to find an alternative algorithm to perform transverse momentum (p_T) assignment to muons in the context of the Level-1 trigger at the CMS experiment at CERN. This NN has been implemented with the following structure: the first hidden layer has 35 neurons and receives the information directly from the input layer of 27 different features with the ReLU (Rectified Linear Unit) selected as activation

function. The second layer is identical to the first one but contains 20 neurons and this is repeated for other 4 additional hidden layers with 25, 40, 20 and 15 neurons, respectively. In the end, the output layer (with only one node) closes the network. The model has been optimized for hardware implementation with *pre-training quantization* [4] and *weight pruning*.

2.2 The Implementation

The first step required for the implementation of a Neural Network on an FPGA is the conversion of the high-level code used for the creation of the model (Python + Tensorflow & QKeras) into High Level Synthesis (HLS) code. HLS describes the process of automatic generation of HDL code from *behavioural description* contained in a C/C++ script. To accomplish this task, the *hls4ml* package [5] has been used. This tool has been developed by members of the High Energy Physics (HEP) community to translate ML algorithm, built using frameworks like TensorFlow2, into HLS code.

Once the target hardware has been defined, and the trained model converted into HLS code using *hls4ml* (more details are available in [2, 3]), the project has to be imported in *Vitis*, a tool part of the Xilinx Design Suite, dedicated to developing applications for data center acceleration cards. Here the C++ code must be tweaked in order to expose the interface of the Neural Network and make it compatible with the *Application Acceleration development flow*, offered by Vitis.

Then, we can instruct Vitis to build the entire project targeting the desired hardware. This will produce a bitstream file used to flash our design onto the FPGA. Together with the firmware design, an OpenCL application can be written that can be launched on the machine that houses the FPGA to program it, start the inference and retrieve the results (as shown in the next section).

Moreover, to deploy a design on Amazon's F1 instances, the bitstream must be uploaded to an Amazon S3 Bucket and request the creation of an *Amazon FPGA Image* (AMI) using a script included in the official github repository of the AWS EC2 FPGA Hardware Development Kit [6]. This will produce a `awsxclbin` file that can be used to program Amazon's FPGAs.

2.3 The PYNQ Project

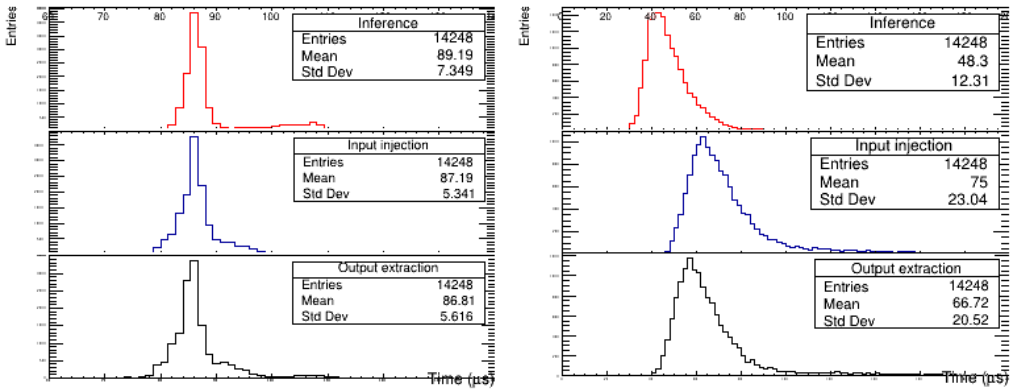
PYNQ [7] is an open-source project from Xilinx®, a prominent FPGA producer. It provides a Jupyter-based framework with Python APIs for using Xilinx platforms and AWS-F1 instances.

FPGA designs are presented as Python objects called *overlays* that can be accessed through a Python API. Creating a new overlay still requires developers with expertise in designing programmable logic circuits. Overlays, like software libraries, are designed to be configurable and re-used as often as possible in many different applications.

To date, C or C++ are the most common embedded programming languages. In contrast, Python raises the level of programming abstraction and programmer productivity. These are not mutually exclusive choices, however. PYNQ uses CPython which is written in C, and integrates thousands of C libraries and can be extended with optimized code written in C. Wherever practical, the more productive Python environment should be used, and whenever efficiency dictates, lower-level C code can be used.

PYNQ aims to work on any computing platform and operating system. This goal is achieved by adopting a web-based architecture, which is also browser agnostic. It incorporates the open-source Jupyter notebook infrastructure to run an Interactive Python (IPython) kernel and a web server directly on the ARM processor of a MPSoC or host's CPU of an acceleration card.

3. Neural Network model performance on FPGA



(a) PYNQ **(b)** OpenCL

Figure 1: Distribution of the times needed to inject data in the FPGA, perform NN inference and extract the output using the PYNQ package in Python (left) and an OpenCL application (right).

Two main aspects have been considered to study the performance of using the PYNQ package to carry out Neural Network inference on an FPGA: latency and inference accuracy.

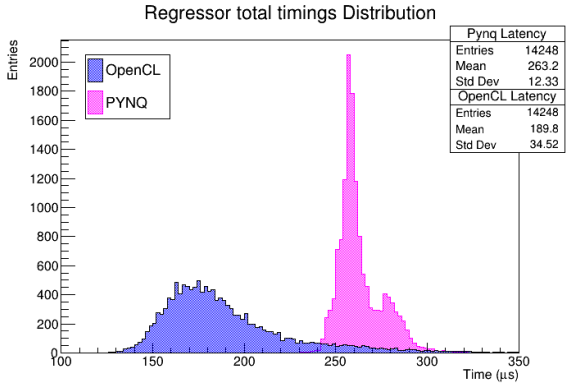


Figure 2: Total inference time distribution (input injection + inference + output extraction) using PYNQ (pink) and an OpenCL application (blue).

For the first metric, the *wall* time has been measured for the three main tasks that are executed by the host-FPGA pair for each inference that is requested. In Figure 3 the time distribution for the input injection on the FPGA card (blue), the actual inference (red) and output extraction (black) is shown for the entire validation dataset using PYNQ on the left and the OpenCL application on the right. In the PYNQ case, a degree of consistency can be seen between the different tasks. This can be explained by a common overhead caused by Python's nature as an interpreted language, which can also be considered as the main cause for the overall larger total processing time, shown in Figure 2, with respect to the application compiled in C++.

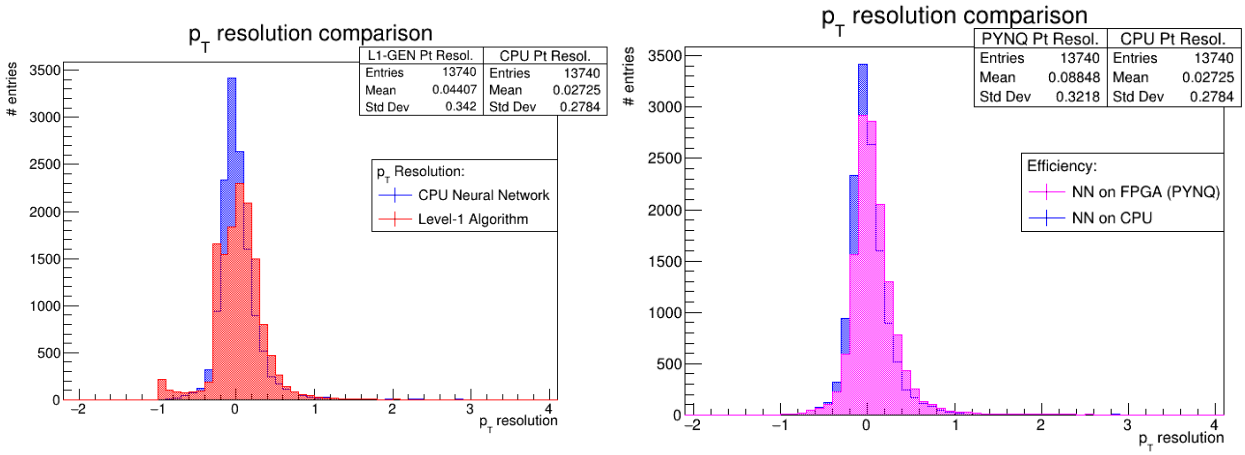
Nonetheless, the main objective of using PYNQ is offering an easier interface and less steep learning curve in dealing with accelerating algorithms using FPGAs. This means that, to achieve

the full potential of this type of hardware, the traditional approach using C/C++ application is still the way to follow.

3.1 p_T resolution histogram

To study the accuracy of the NN model implemented on the F1 instance p_T resolution histograms were used. For each entry of the dataset, the histograms were built using $\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}} - p_{T_{sim}}}{p_{T_{sim}}}$ where $p_{T_{est}}$ is the estimation of the transverse momentum, given by the model prediction or the actual algorithm used in the Level-1 trigger at CMS to perform this task, and $p_{T_{sim}}$ is the "true" transverse momentum associated to each entry of the validation set.

Firstly, the resolution of the model before the implementation on the FPGA must be checked (Figure 3a). The red histogram describes the resolution distribution of the Level-1 trigger system while the blue one shows the resolution of the predictions made by the network model running on a consumer CPU.



(a) Machine learning model (blue) v. Level-1 trigger (red) based momentum assignment.

(b) Machine learning model inference on FPGA v. a consumer CPU.

Figure 3: Transverse momentum resolution histograms.

In particular, it is possible to notice a less broad distribution for the ML resolution, resulting in an overall improvement, yet small, with respect to the Level-1 trigger system. Another noticeable detail is the small peak corresponding to the value -1: this happens when the p_T assigned by the trigger is significantly underestimated with respect to the true p_T. The Machine Learning based momentum assignment is therefore less prone to large p_T underestimation.

Having verified the accuracy of the NN model, its implementation on the FPGA available in the F1 instance can be analyzed. In Figure 3b the p_T resolution histogram obtained by performing the inference using the PYNQ environment is shown over the model resolution described before. When the assignment is performed on an FPGA, slightly worse results are produced, with a small bias towards higher values of $\Delta p_T/p_T$. This could be the effect of the loss in precision the input features have to go through due to the conversion to fixed-point representation needed to perform computations efficiently in an FPGA [2, 3]. Nevertheless, the hardware approach still appears

compatible, or in case of higher momenta, even better than the Level-1 trigger based momentum assignment.

4. Conclusions

An open-source project from Xilinx (a major FPGA producer) called PYNQ has been tested, combined with the HLS4ML toolkit, in order to program a Neural Network on an FPGA and use it to perform inference. The PYNQ purpose is to grant designers the possibility to exploit the benefits of programmable logic and microprocessors using the Python language. Cloud computing was used in this work to test the capabilities of this workflow, from the creation and training of a Neural Network and the creation of a HLS project using HLS4ML, to testing the predictions of the NN using PYNQ APIs and functions written in Python.

Hardware and software set-up, together with performance, were tested. An increase in latency of the algorithm was discovered when using PYNQ with respect to a more traditional way of interacting with an FPGA via an application written in OpenCL. This can be explained by an overhead caused by Python's nature as an interpreted language. Consistency between the predictions of a NN before and after its implementation on the FPGA was verified.

References

[1] André DeHon Scott Hauck, *Reconfigurable computing: the theory and practice of FPGA-based computation*, Systems on Silicon, Morgan Kaufmann, 2007.

[2] M. Lorusso, *FPGA implementation of Muon Momentum assignment with Machine Learning at the CMS Level-1 Trigger*, University of Bologna master thesis (unpublished).

[3] T. Diotalevi, M. Lorusso, R. Travaglini, C. Battilana and D. Bonacorsi, *Deep Learning fast inference on FPGA for CMS Muon Level-1 Trigger studies*, PoS ISGC2021, 2021, DOI: 10.22323/1.378.0005.

[4] C.N. Coelho, A. Kuusela, S. Li, et al. *Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors*, Nat Mach Intell 3, 675–686 (2021), DOI: 10.1038/s42256-021-00356-5

[5] J. Duarte et al. *Fast inference of deep neural networks in FPGAs for particle physics*, In Journal of instrumentation 13.07, July 2008.

[6] https://github.com/aws/aws-fpga

[7] http://pynq.readthedocs.io/