A FPGA-based tensor accelerator for Machine Learning

Summary

Candidate:
Francesco Angione
Supervisors:
prof. Paolo Bernardi
prof. Pedro Petersen Moura Trancoso

A.Y. 2019/2020

1

Introduction

Machine learning is one of the hot technologies today as it is being used to solve complex problems that would otherwise be very hard or costly to solve with traditional methods.
The use of commodity hardware is not the most effective and efficient way to execute Machine Learning, so the goal is to satisfy the required demands for different Machine-Learning models but at lower cost and energy consumption in order to be deployed also on mobile devices. As it is very well-known, hardware accelerators are capable, if designed correctly, of delivering a lot of improvements in terms of the latency but also in terms of energy efficiency. Thus, in order to obtain the best solution in every metric a hardware-software co-design is needed, requiring to the hardware designer a basic knowledge of machine learning algorithms.

The goal is to develop a hardware accelerator from scratch, which implements a tensor-based convolution. Exploiting a non Von Neumann architecture and reuse of weights reduces the CPU workload and boost the models' performance. The use of different arithmetic data type can drastically reduce the computations without reducing the final accuracy of the Neural Network. From a hardware perspective, the use of different arithmetic precision, such as the use of integer operations instead of floating-point operations, can lead to benefits in terms of area, energy consumption and latency.

2

System Development

The designed accelerator has a Dataflow architecture, with emphasis on weight data reuse, and it is able to execute a tensor convolution. The basic idea is a computation matrix composed in every entry of processing elements which are able to perform operation between the incoming data and the weights, which have been already loaded for exploiting a data reuse approach.
The custom hardware accelerator is not useful as it is. It has to be integrated into a ML-Framework, TensorFlow, which allows to integrate a custom hardware accelerator minimizing the efforts to change the model code and its definitions.

The workflow of the Hardware-Software development is illustrated in the Figure 2.1.

2.1 Software

The focus of the work is the inference process, pre-trained models are needed and TensorFlow Hub comes in handy for this purpose. It provides already pre-trained Machine Learning models for different domains. Moreover, TensorFlow has the feature of quantizing a post-trained model for different arithmetic precision. In the Fig. 2.1 it can be seen that the quantization process has been done offline.
TensorFlow demands as library for the accelerator a C Python-API compatible shared library. In addition, the code for using the accelerator was already written using the PYNQ environment in Python. Therefore, for allowing code reuse and decreasing the development time the Python code has been embedded in the C code (from a TensorFlow example of the delegate library), adding callbacks to Python code.

2.2 DTPU, the hardware accelerator

The hardware accelerator, named *Cogitantium*[1], *The Dumb Tensor Processing Unit*, is in charge of carrying out the tensor convolution of the neural network model, exploiting a data-flow architecture on the input data and a data reuse for the weight data.

[1]Thoughtful

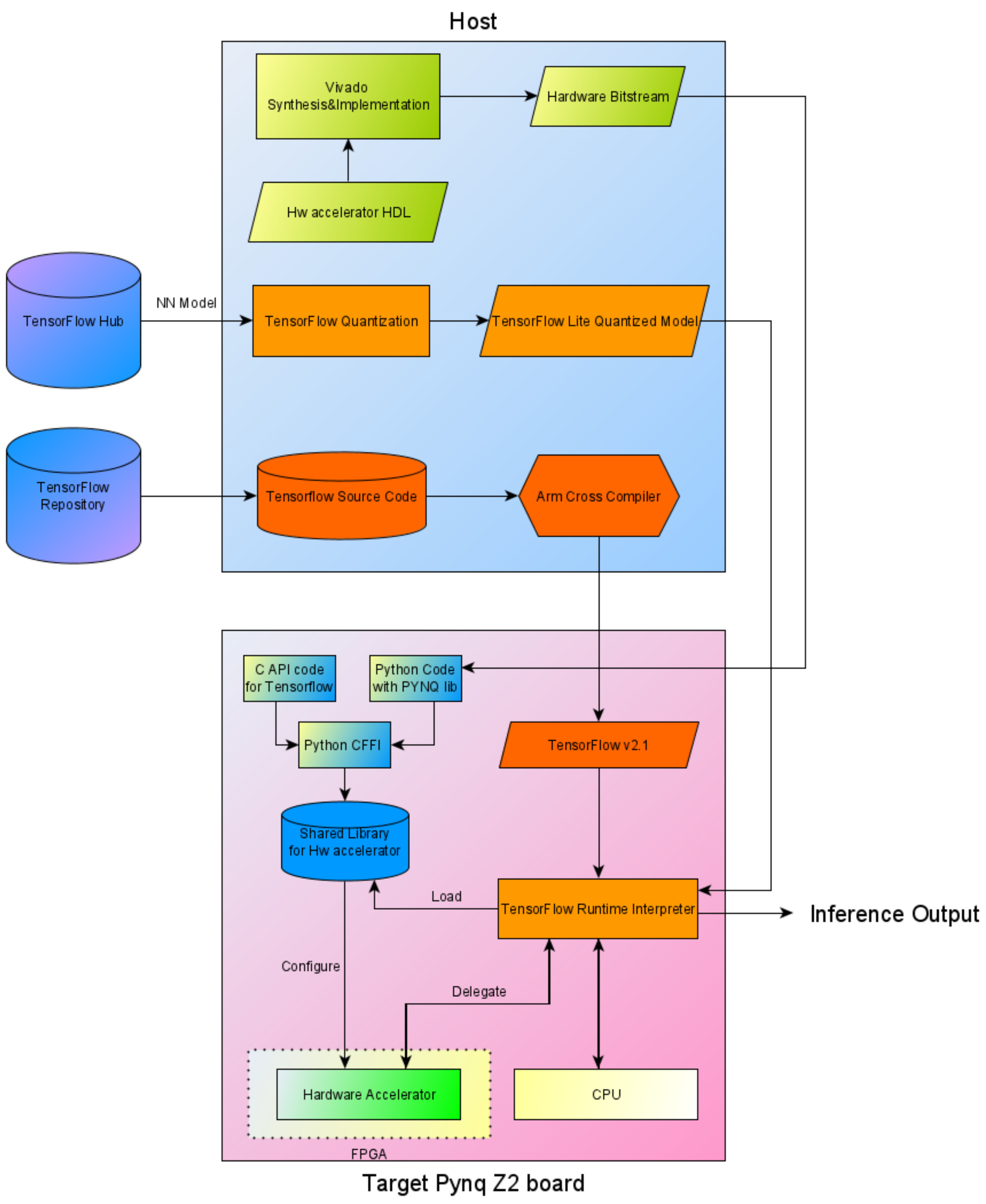

Figure 2.1: Development workflow

The work is not focused on developing embedded memories and AXI interfaces, therefore a Xilinx's IP core, which includes all those necessary subcomponents, has been used. The latter has allowed to completely focus the work on the DTPU core, which has become:

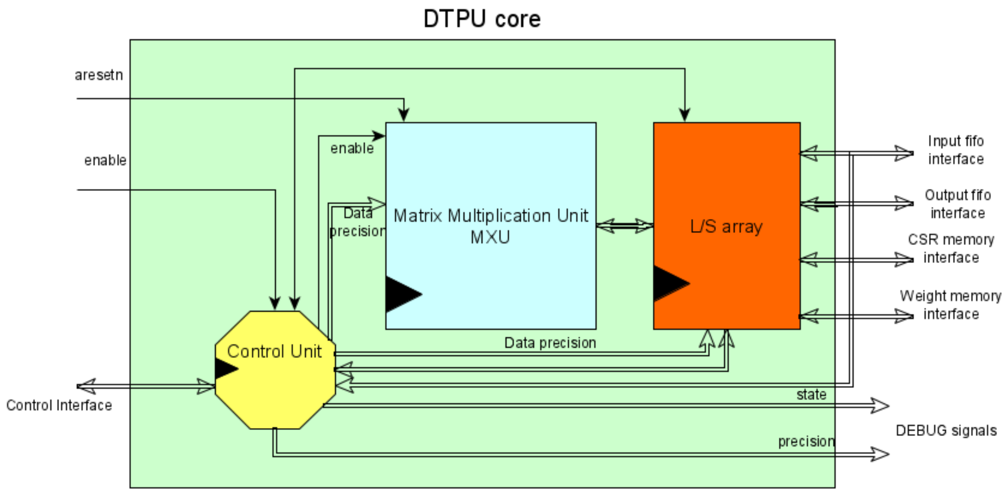

Figure 2.2: RTL view of DTPU core

Where the sub-units:

- L/S array provides the data for the Matrix Multiplication Unit, especially the weight data are reused across several executions and therefore loaded once. It also provides the correct data to the correct computation units depending on the precision.
- Control Unit is in charge of handling handshake signals for transferring the ownership of the data (data transferred by the DMA from the Main Memory), load the weights and activation in the respectively units and save the results to the output FIFO. Since it is a Data flow architecture, there is no control flow of the data in the core and this has allowed to keep the Control Unit as simple as possible.
- Matrix Multiplication Unit (Mxu) is the computation unit of the hardware accelerator, the brawn of the accelerator, a symmetric matrix of MACs with variable precision, where every MAC unit has its own weight value. It executes the tensor convolution for different arithmetic precision.

3 Results

Every domain and application could have different evaluation metrics. In the following the overall metrics are evaluated.

- Utilization factor, for integer 8 and 16 the trend is similar. It is a 1 to 1 mapping between the PEs and the DSPs. As soon as the DSP are used the utilization of LUT and FF is linear in the sizes of Matrix Multiplication unit, while the DSP utilization is quadratic. At a full utilization of DSP entities the PEs start to be implemented in logic there is a sudden rise in the LUT utilization. Increasing the clock frequency, the FPGA's utilization is reduced since with an increase of the Matrix Multiplication Unit sizes the design is not able anymore to meet the timing requirements.
- Energy and Power Consumption, the power consumption of the processing system and the static power consumption have a less impact on the total power consumption with an increase of the Matrix Multiplication Unit and the frequency. On the other hand, the dynamic power consumption, as expected, grows with a growing Matrix Multiplication Unit and the design frequency.
- Throughput, it results to be equal to the number of rows into the Matrix Multiplication Unit. Enough data needs to be provided to the accelerator in order to have all the Processing Elements working with useful data, if not, the throughput goes down.
- Latency, the execution of the models is done with different clock frequencies and data type, and as consequence a different overall latency (and power consumption). The reason about the increase in the latency, since the main goal was to reduce the latency, is that introducing the accelerator and its library comes with several overheads, mainly due to runtime transformations of matrices in a suitable format for the accelerator.
- Accuracy, it is measured for different data width and model with reference to the actual value (inference without the accelerator). In general, models have a wrong prediction. One of the reason may reside in the input data feed to the model, they are totally random. Another improvement should be to accumulate on the different bit width precision, for example compute multiplication on 8 bits integer and accumulate values on 16-bit integer. Moreover, as in every software product, bugs have not been detected but this does not mean that the written software is bug-free.

4

Conclusion

4.1 Discussion

A big portion of inference process for Neural Networks involves massive multiply and add computation, basic operation of tensor convolutions, and across several execution data, especially weight tensors, are reused.
As consequence, for speeding-up and reduce the power consumption (especially in mobile devices) of ML models a hardware accelerator has been developed. It is also designed for accommodating different data type computation request from Neural Network models, ranging from integer8/16/32/64 to floating-point 32 and brain floating-point 16. It is possible to build a custom hardware accelerator for a specific ML operation and then integrate it into a framework without changing the model nor the framework. The bottom up approach and the delegate class available in Tensorflow has allowed to fully tailor a new class of hardware accelerators which can accommodate different needs (i.e. depending on which part of the model has to be accelerated). As it has been organized, changing the core software in the Python code and the core in the hardware, it can be also used for addressing different model's operations.

4.2 Future Works

For every human artifacts, there is always work to do. In addition, for Computer Engineering artifacts there is also an important step which is the software (and in this case also of the hardware) optimization. In particular:

- Software Optimization and migration to a full C code implementation for further reducing the latency.
- A deep software/hardware testing for finding additional bugs.
- Power estimation using the simulation's switching activity in order to obtain a very precise and reliable power consumption.
- Comparison of model execution on different state-of-the-art platforms.

Following the previous recommendation, the work may arrive to a competitive level such as the one of the GPUs or other hardware platforms.