CHALMERS
UNIVERSITY OF TECHNOLOGY

POLITECNICO
DI TORINO

Hardware accelerator for Machine Learning

Erasmus Master thesis in Computer science engineering

Francesco Angione

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Hardware accelerator for Machine Learning

Francesco Angione

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Supervisor: Pedro Petersen Moura Trancoso, Department of Computer Science and Engineering, Chalmers University of Technology
Home Supervisor: Paolo Bernardi, Department of Control and Computer Engineering, Polytechnic of Turin
Examiner: Per Larsson-Edefors ,Department of Computer Science and Engineering, Chalmers University of Technology

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A robot involved into the self learning process, reading.

Abstract

Abstract text about your project in Computer Science and Engineering.

add cover image

Acknowledgements

Here, you can say thank you to your supervisor(s), company advisors and other people that supported you during your project.

Francesco Angione, Gothenburg, March 2020

Contents

List of Figures

List of Tables

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. Speech and image recognition, as well as many other complex decision-making problems such as self-driving vehicles are successfully solved with Machine Learning and Deep-Learning.
The success of Machine Learning is being driven by the current available hardware which can provide the required demands in terms of storage and compute capacity. But obviously as problems scale so do the demands and thus special hardware is being developed to address these needs. As soon as the mobile devices are well spread, the need of a custom energy efficient hardware accelerator for Machine Learning is needed [1].

The use of commodity hardware is not the most effective and efficient way to address this problem, so research is looking at solution that can satisfy the required demands but at lower cost and energy consumption in order to support Machine Learning also on mobile devices. The constant developments in the Machine Learning methods require the design of more flexible hardware accelerators [2] [1].
As it is very well-known, the hardware accelerators are capable, if designed correctly, of delivery a lot of improvements in terms of the latency but also in terms of energy efficiency[3]. Thus, in order to obtain the best solution in every metric a hardware-software co-design is needed, requiring a basic knowledge of Machine Learning algorithms.

The use different arithmetic data type or skipping multiplication with data which are almost zero can drastically reduce the computations without reducing the final accuracy of the DNN [4][5]. From a hardware perspective, especially the use of different arithmetic precision[6] can lead to benefits in terms of area, energy consumption and latency.
According to that, accuracy of operations, reliability, performance and energy efficiency are evaluated, moving through different designs with different precision and/or data gating and compared to the implementation of the same DNN inside a GPU, as well as custom hardware platform.

Machine Learning includes two processes, the training and the inference. Mainly, the work is focused on the inference process, exploring different optimizations in terms of hardware.

In the last years, the number of published papers regarding Machine Learning has growth exponentially, leading to a lot of efforts also from the companies which started to develop, deploy and sell their own hardware platforms, such as Tensor Processing Unit from Google, NVDLA from Nvidia and Gaudi and Goya, respectively for training and interference, from Habana (acquired by Intel).
One of the main ideas of those works is that during the inference process, a model does not need the high precision computations [5] [7]for achieving high accuracy into its outputs.

2
Background

In this chapter a general glance on Artificial Intelligence, and its sub-categories , is given.

> *Can a machine think?*
> — Alan Turing, Computing Machinery and Intelligence

2.1 Overview

description and reasoning of AI in general
figure of division

2.1.1 Machine Learning

description

2.1.1.1 Brain Inspired

2.1.1.2 Neural Networks

2.1.1.3 Deep Learning

math model

2.2 Applications

3
State of the Art

In this chapter various solutions available on the market are presented for running the inference process of Machine Learning models.

3.1 Overview

The role of Machine Learning has continuously growth in the past few years and a lot of efforts has been done for developing good software APIs in order to address different needs and domains.
In principle, all the Machine Learning algorithms can be run on the CPU, which already runs the OS and other Application Softwares, this leads to a lot of overheads, especially in terms memory accesses which are expensive in terms of energy and latency.

Analyzing Machine Learning algorithms it comes evident that they massively do the same operations and access to data with some kind of patterns. Thus, with the outcome of the new paradigm for the GPU, the General Purpose GPU programming it comes in handy that implementing those algorithms on a GPU, which matches the Machine Learning algorithms requirements regarding the massive operations and the reuse of data, has given a lot of advantages in terms of latency and energy efficiency. However, the capability of GPU of running Machine Learning algorithms has been pushed almost at the maximum with the increase of computation demands in modern neural networks, therefore other solutions have been explored, such as the development of specific hardware platform.

3.2 GPU

The Moore's law is reaching the end from the point of view of CPUs. However, it seems that the GPUs can still carry on the Moore's law [8].
For this reason, a lot of improvements especially from the companies have been made for developing more and more GPUs with a lower performance per watts.
As already mentioned, with the income of general purpose GPU programming paradigm, more and more Machine Learning algorithms have been designed for being run on the GPU, gathering the best fruits given by that type of architecture.

As consequences, companies such as Nvidia have started to develop GPU for boosting Machine Learning applications performance.

3.2.1 Nvidia Tesla V100

The Nvidia Tesla V100 is one of the most performant GPU at the moment on the market. The newly added Tensor Core Unit allows massive increases in throughput and efficiency.
It is able to deliver up to 125 TFPLOPS[1] for training and inference Machine Learning applications.

The core of the GPU is the Streaming Multiprocessor from the Nvidia Volta GPU.

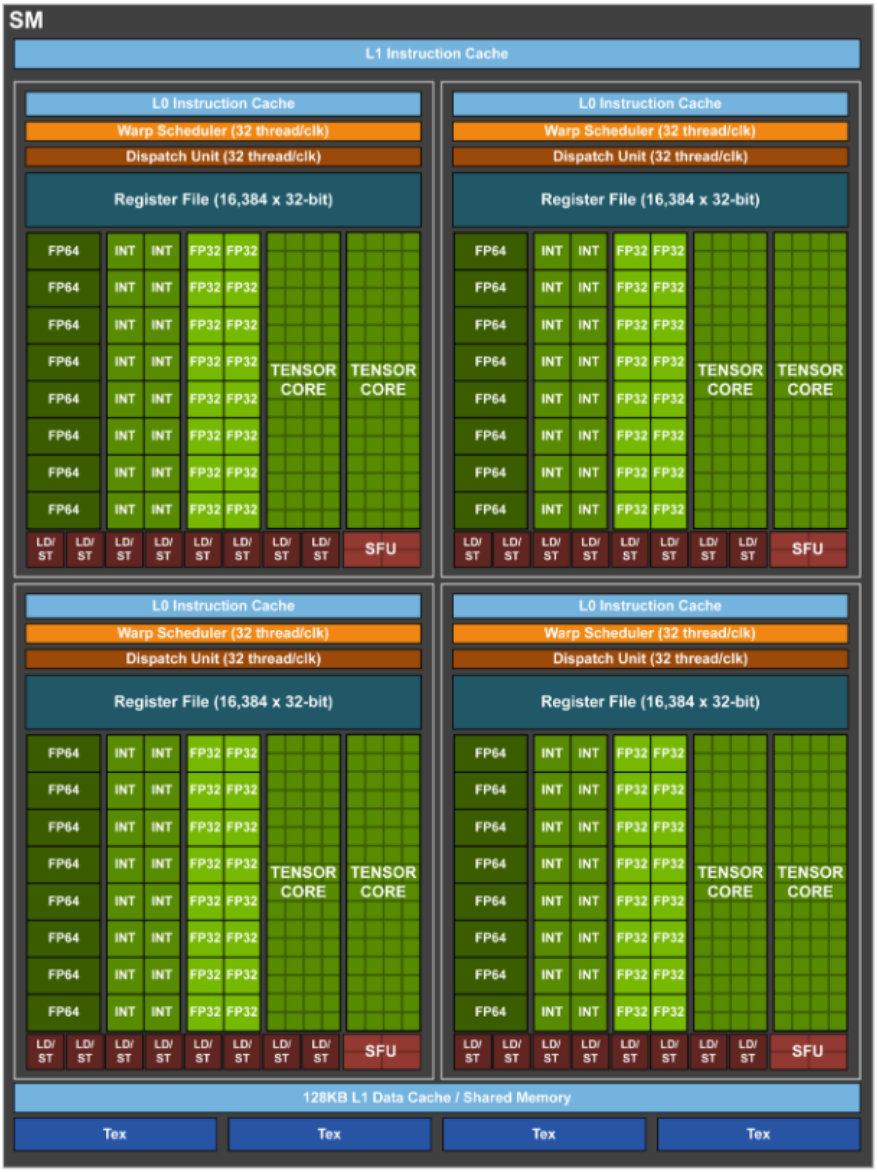

Figure 3.1: Streaming Multiprocessor Architecture

Composed of integer, FP32, FP64 units and the Tensor Core Units designed specif-

[1] floating point operations per second

ically for deep learning. With independent parallel integer and floating-point data paths, the Volta SM is also much more efficient on workloads with a mix of computation and addressing calculations.

Matrix-Matrix multiplication operations are at the core of neural network training and inference, and are used to multiply large matrices of input data and weights in the connected layers of the network. The idea is represented into the Figure 3.2.

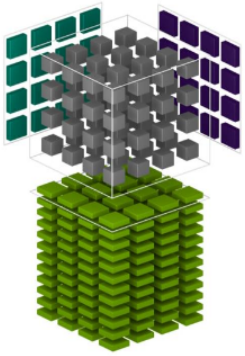

Figure 3.2: Matrix Multiplication in Tensor Core

The Tesla V100 GPU contains 640 Tensor Cores: eight per Streaming Multiprocessor. Each one of the cores is able to perform 64 floating point Fused Multiply and Accumulate operations per clock. According to Figure 3.3 the Tensor Core Units are able to compute multiplications on FP16 and accumulate on FP32, leading to a further reduction of latency and energy consumption.

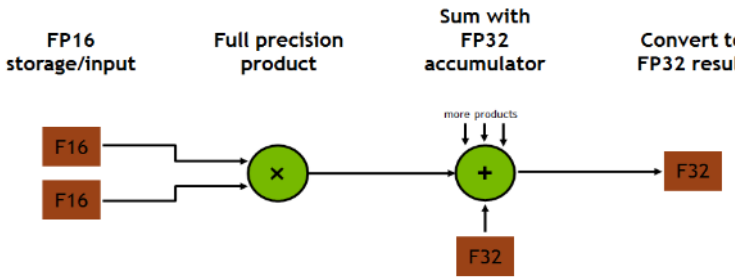

Figure 3.3: Mixed Precision Schema of a FMA unit in Tensor Core Unit

The deep learning frameworks and the CUDA Toolkit include libraries that have been custom-tuned to provide high multi-GPU performance for each one of the following deep learning frameworks in the Figure 3.4.

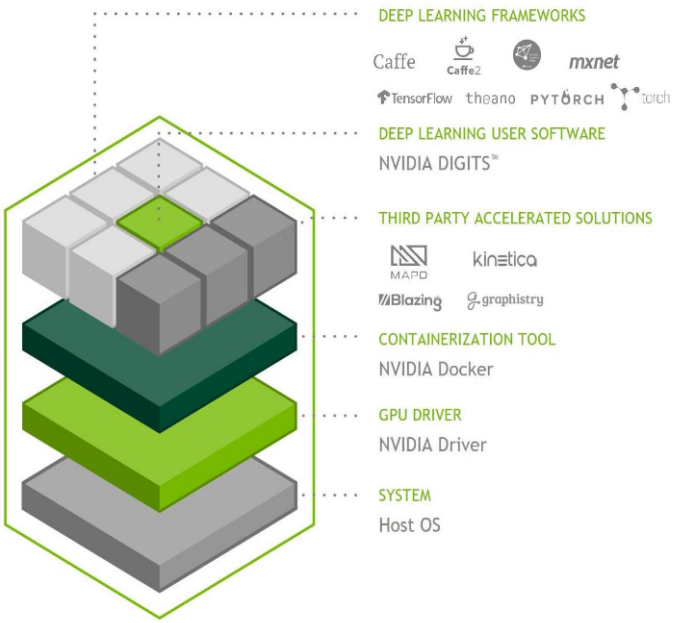

Figure 3.4: Software stack

Combining powerful hardware with software tailored to deep learning, it provides to developers and researchers solutions for high-performance GPU-accelerated deep learning application development, testing, and network training.

3.3 Hardware Platform

Instead of developing GPUs also suitable for Machine Learning applications, the companies have designed and deployed special purpose hardware accelerators.

3.3.1 NVDLA

The Nvidia Deep Learning accelerator is a free open source hardware platform from Nvidia, highly customizable and modular, which allows to design and deploy deep learning inference hardware.
The architecture comes in two configurations:

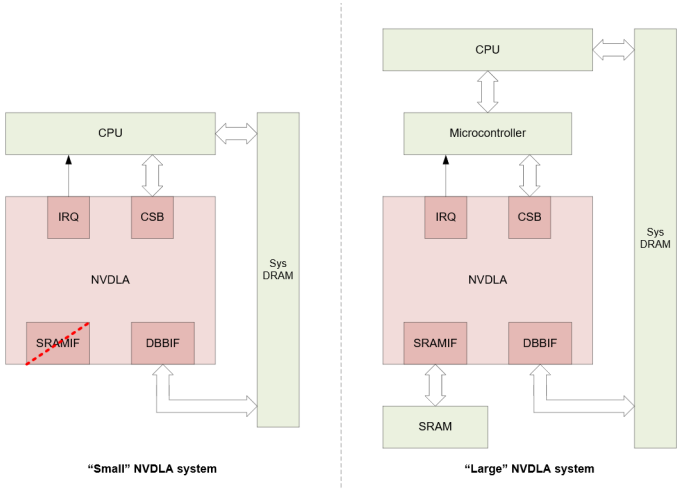

Figure 3.5: Comparsion of two possible NVDLA system

As already mentioned, the aim of the work is to develop a hardware accelerator for Machine Learning suitable for mobile devices, therefore from now on the NVDLA small system will be considered and analyzed.
The internal architecture of the NVDLA small system is:

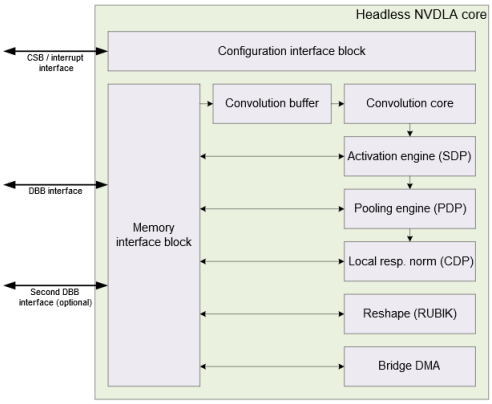

Figure 3.6: Internal architecture of NVDLA small system, Secondary DBB not considered

According to Figure 3.5, for the Small configuration of the accelerator, the processor will be in charge of programming and scheduling the operations on the NVDLA and as consequences handles the start/end of operations and possible interrupts, all of them through the CSB (Configuration Space Bus) interface which is AXI Lite compliant[9].
The data movement to/from memory are handled by the Internal memory controller though the DBB (Data BackBone) interface, which is AXI [9] compliant.

The internal architecture of NVDLA is composed by various engines, each one of them is able to perform specific Machine Learning models:

- Convolution Core: it comes in pair with the Convolution Buffer, its private memory for the data (inputs and weights). It is used to accelerate the convolution algorithms.
- Activation engine (Single Data point Operations): it performs post processing operations at the single data element level such as bias addition, Non linear function, PReLU (Parametric Rectified Linear Unit) and format conversion when the software requires different precision for different hardware layers.
- Pooling engine (Planar Data Operations): it is designed to accomplish pooling layers, i.e. it executes operation along the width and height plane.
- Local response normalization engine(Cross Channel Data operations): it is designed to address local response normalization layers.
- Reshape(Data memory and reshape operations): it transforms data mapping format without any data calculation.
- Bridge DMA: it is in charge of copying data from the Main Memory to the SRAM of the accelerator, only available into the large configuration of the system.

Another possible configuration which is worth to mention is the possibility to let the engines work separately on independent task or in a fused fashion where all of them are pipelined, working as a single entity.

According to developers the configurability of the cores ranges from arithmetic precision to the theoretical throughput that a single unit can achieve (increasing the number of internal Processing Elements). Moreover, since the engine units are independent from each other, according to the application and the model used they can be safely removed from the design.

3.3.1.1 NVDLA Software

It is also worth to mention that the accelerator comes already with a basic software stack:

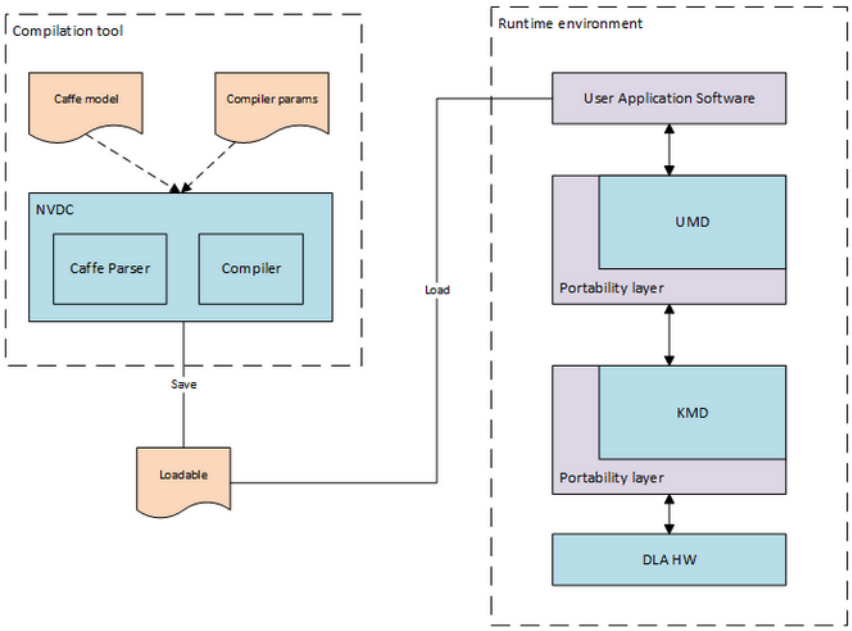

Figure 3.7: NVDLA Software stack

Where the Compilation tools are in charge of converting existing pretrained model into a set of hardware layers (for the desired precision) and programming sequences suitable for the NVDLA, the output of this process is a Nvidia Loadable file suitable for the runtime environment.
Regarding the runtime environment, it has been designed for a system in which is present an OS. It is composed in two parts: the User Mode Driver (UMD) and the Kernel Mode Driver (KMD).
The User Mode Driver loads the loadable file in memory and submits the operation to the KMD, it is also in charge of data movement from/to the accelerator.
The KMD is in charge of submit operations to the accelerator through low level functions, schedules the operations and handles the interrupts.
Both the KMD and the UMD are wrapped into portability layers which are, respectively, hardware dependent and OS dependent. In principle, for migrating the software to another OS or hw plaftorm it is enough to modify only the portability layers.

3.3.2 Google TPU

A tensor processing unit is an AI accelerator application-specific integrated circuit developed by Google specifically for neural network machine learning, particularly using Google's own TensorFlow software. It includes:

- Matrix Multiplier Unit (MXU): 65,536 8-bit multiply-and-add units for matrix operations
- Unified Buffer (UB): 24MB of SRAM that work as registers
- Activation Unit (AU): Hardwired activation functions

In Figure 3.8 a general view of TPU architecture is presented.

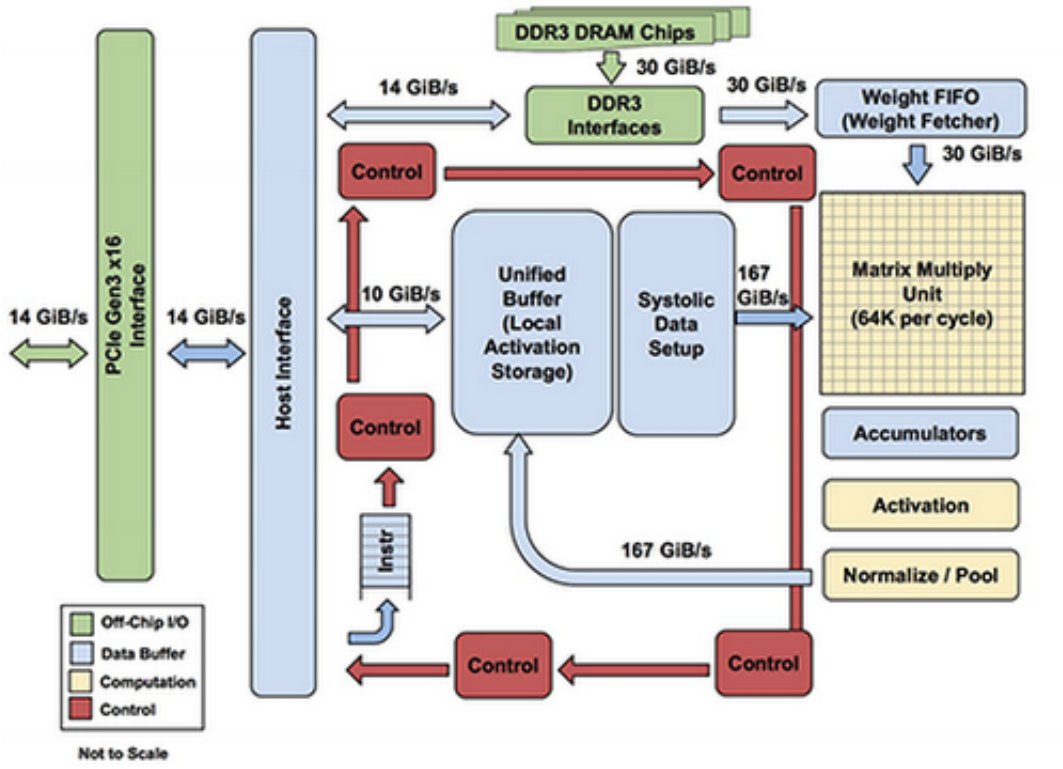

Figure 3.8: Google TPU architecture

Rather than be tightly integrated with a CPU, the TPU is designed to be a coprocessor in which the instruction are sent by the host server rather than fetched.

The matrix multiplication unit reuses both inputs many times as part of producing the output avoiding the overhead of continuously read data from memory. Only spatial adjacent ALU are connected together, which makes wires shorter and energy-efficient. The ALUs only perform computations in fixed pattern.

As far as concern the software stack, the TPU can be programmed for a wide variety of neural network models. To program it, API calls from TensorFlow graph are converted into TPU instructions.

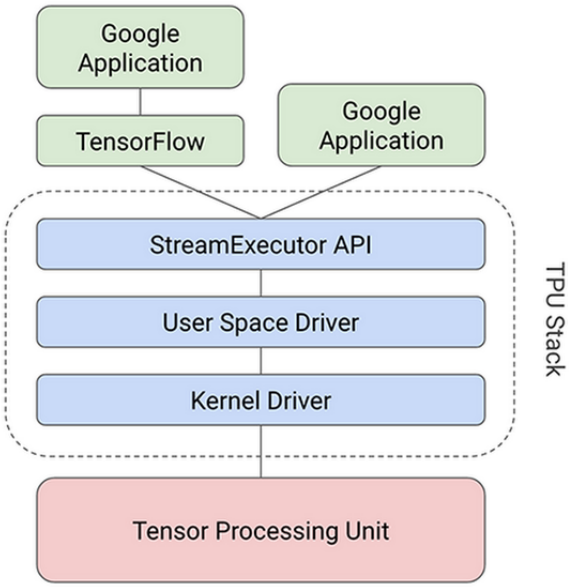

Figure 3.9: Google TPU Software Stack

3.3.3 Habana Goya HL-1000

Habana's Goya is a processor dedicated to inference workloads. It is designed to deliver superior performance, power efficiency and cost savings for data centers and other emerging applications.
It allows the adoption of different deep learning models and is not limited to specific domains. Moreover, the performance requirements and accuracy can be user-defined.

In the Figure 3.10 a high level view of the Goya architecture can be appreciated.

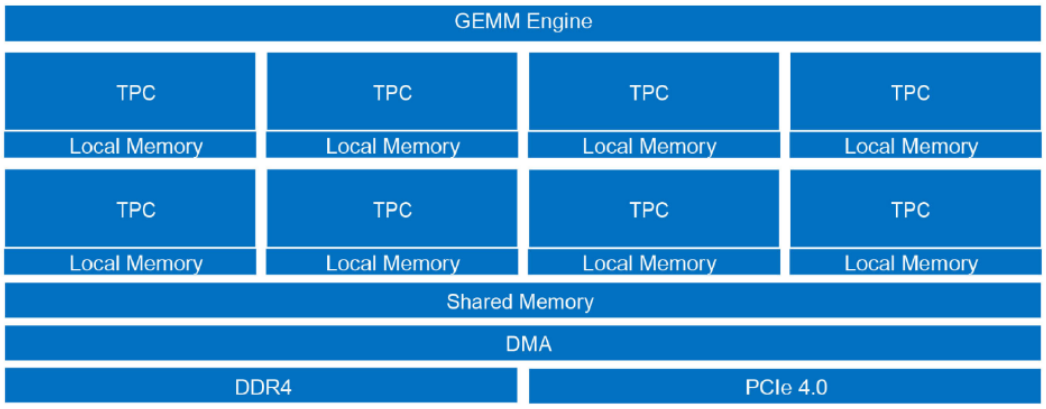

Figure 3.10: High level view of Goya architecture

It is based on scalable, fully programmable Tensor Processing Cores, specifically designed for deep learning workloads.
It also provides other flexible features such as GEMM operation acceleration, special functions dedicated hardware, tensor addressing, latency hiding capabilities and different data types support in TPC (FP32, INT32, INT16, INT8, UINT32, UINT16, UINT8).

Regarding the software stack, it can be interfaced with all deep learning frameworks. However, a model has to be first converted into an internal representation, as it can be seen in Figure 3.11.

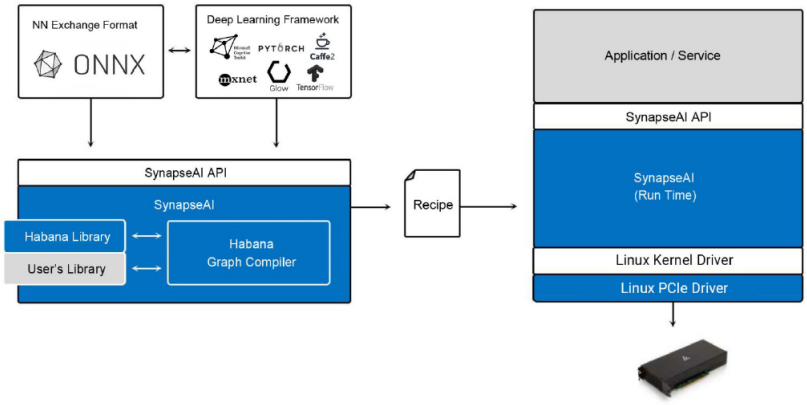

Figure 3.11: Habana Goya Software Stack

It also supports quantization of models trained in floating-point format with near-zero accuracy loss.

4

System Devolopment

In the following chapter a Top Down approach of the System development is described, with particular emphasis on the hardware development part.

4.1 Overview

The entire work is implemented on a PYNQ Z2 board from TUL, based on a Zynq-7000 SoC [10].

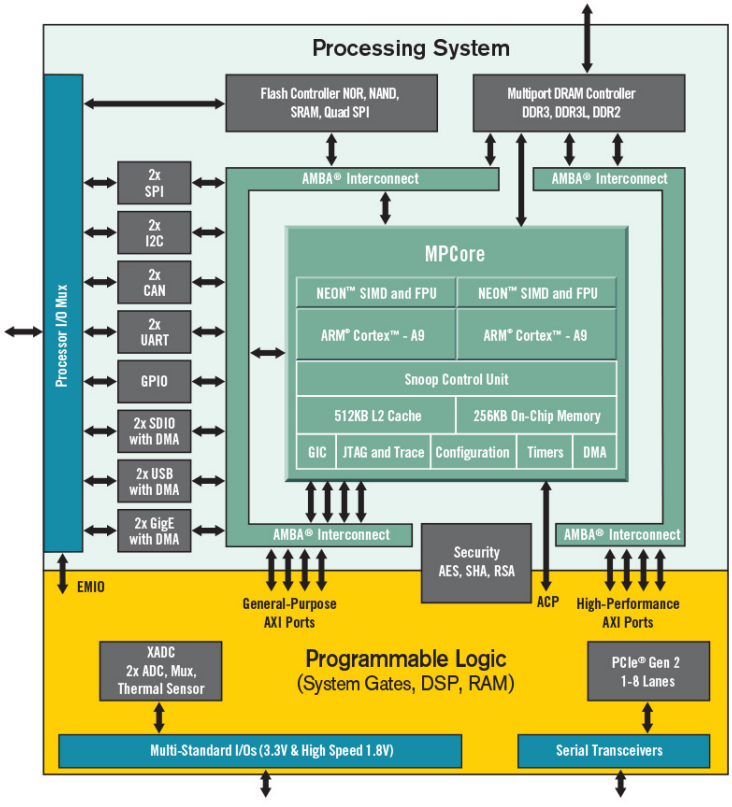

Figure 4.1: Zynq 7000 SoC

The latter is composed of a programmable logic and a processing system, the Programmable logic hosts the accelerator and the relative AXI interconnections while the processing system is in charge of running the software for programming and scheduling correctly the operation on the accelerator.

In order to speed-up the development process and use built-in library for the AXI protocol and the DMA transfers, the software is carried out through the PYNQ enviroment of the board [11] which is based on Python.
The usage of Python as basic software allows to easily integrate it with high level Machine Learning API, such as TensorFlow in this case, an end-to-end open source Machine Learning platform [12]. It has the feature of quantize a post-trained model for different arithmetic precision.
Since the aim of the work is focused on the inference process, pre-trained models are needed and TensorFlow Hub [13] comes in handy for this purpose. It provides already pre-trained Machine Learning models for different domains.

4.2 Software

todo -> driver for accelerator and model from tensorflow (also offline quantization)

4.3 System Level

As it can be seen from 4.1 , it is divided in two big block:

- Processing System: The processing system in Figure 4.2 referred as *ps7* is in charge of running the OS and the Machine Learning application, as consequence it also runs the necessary software for programming the accelerator registers and the data movement to/from main memory from/to the accelerator.
- Programmable Logic: The programmable logic (PL) hosts the entire design, from the accelerator itself to the DMAs and the AXI interconnections.
  Several single channel DMA have been used instead of using a single DMA with multiple channels, the reason is that the in the PYNQ enviroment only the drivers for the single channel DMA are provided. Furthermore, the Programmable Logic can be divided into:
  - AXI interconnections: IP cores from Xilinx[14][15] in order to connect and correctly address entities in the Programmable Logic.
  - AXI DMA: IP core from Xilinx [16] which allows data movement between main memory and accelerator memories.
  - DTPU: the actual hardware accelerator.
  - XADC: IP core from Xilinx [17] which allows to measure the temperature of the SoC, the voltages and the currents at runtime.

In the following figure, the schematic of the overall design is presented.

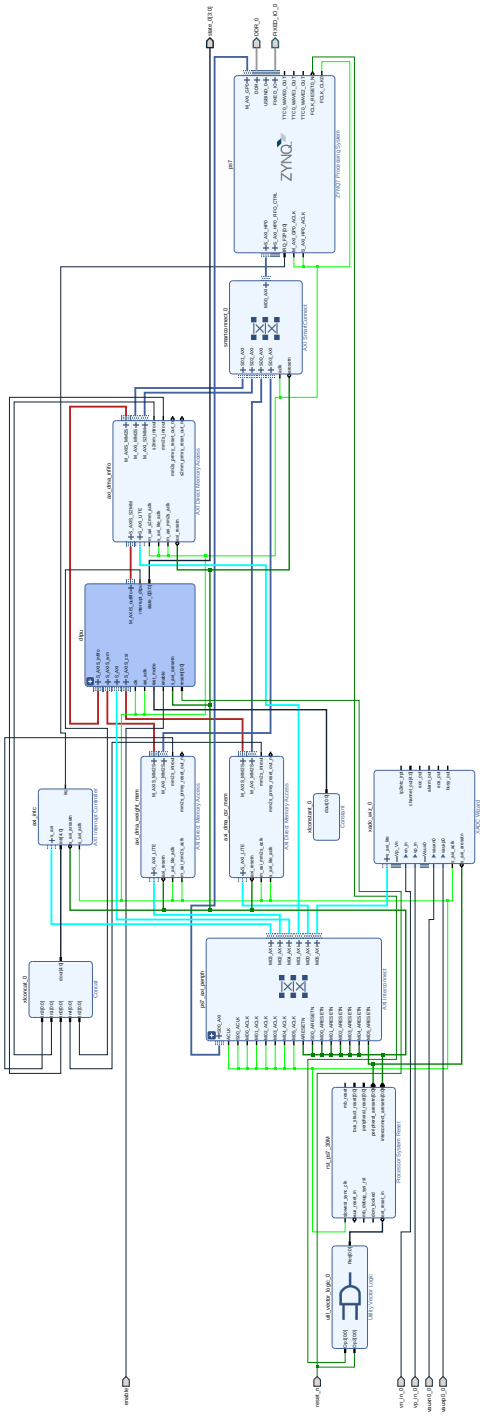

Figure 4.2: System

4.4 DTPU the hardware accelerator

4.4.1 Overview

schematic

4.4.2 Real Implementation

schematic from vivado

4.4.2.1 Axi accelerator adapter

4.4.2.2 DTPU core

5
Results

Describe you results. Use tables, diagrams etc. for illustration. results from hw up to sw

5.1 Utilization Factor

utilization factor for different size of mxu core

report power and timing

we can still compare the utilization of dtpu with the nvdla

5.2 Evaluation metrics

evaluation metrics suggested from the MIT paper

5.3 Neural Networks Models

same NN for different precision and different rows columsn?

Common NN models from MIT tutorial: LeNet (1998) ,AlexNet (2012) ,OverFeat (2013), VGGNet (2014), GoogleNet (2014), ResNet (2015)

6
Conclusion

You may consider to instead divide this chapter into discussion of the results and a summary.

6.1 Discussion

6.2 Conclusion

6.3 Future Works

Bibliography

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, "A domain-specific architecture for deep neural networks", *ACM 61 pag. 50-59* **August 2018**.

[2] V. Sze, Y. H. Chen, T. Yang, J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey", *IEEE vol. 105 no. 12 pp. 2295-2329* **Dec. 2017**.

[3] A. Rahman, S. Oh, J. Lee, K. Choi, "Design space exploration of FPGA accelerators for convolutional neural networks", **1996**.

[4] J. T. Johnston, S. R. Young, C. D. Schuman, J. Chae, D. D. March, R. M. Patton, T. E. Potok, "Fine-Grained Exploitation of Mixed Precision for Faster CNN Training", **2019**, 9–18.

[5] Y. Cai, C. Liang, Z. Tang, H. Li, S. Gong, "Deep Neural Network with Limited Numerical Precision", **2018**, (Eds.: J. Abawajy, K.-K. R. Choo, R. Islam), 42–50.

[6] H. J. L. Hao Zhang, S.-B. Ko, "Efficient Fixed/Floating-Point Merged Mixed-Precision Multiply-Accumulate Unit for Deep Learning Processors", **2018**.

[7] J. Johnson, "Rethinking floating point for deep learning", *Facebook AI Research* **2018**.

[8] Chien-Ping Lu in Proceedings of 2010 International Symposium on VLSI Design, Automation and Test, **2010**, pp. 5–5.

[9] Arm, "AMBA® AXI™ and ACE™ ProtocolSpecification", **2011**.

[10] Xilinx, "Zynq-7000 SoC, Technical Reference Manual", **2018**.

[11] Xilinx, PYNQ, `http://www.pynq.io/board`.

[12] TensorFlow, `https://www.tensorflow.org/overview`.

[13] TensorFlow Hub, `https://www.tensorflow.org/hub/overview`.

[14] Xilinx, "AXI Interconnect v2.1LogiCORE IP Product Guide", **2017**.

[15] Xilinx, "Vivado Design Suite, AXI Reference Guide", **2017**.

[16] Xilinx, "AXI DMA v7.1LogiCORE IP Product Guide", **2019**.

[17] Xilinx, "7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter , User Guide", **2018**.

recheck references

add the xilinx ip core manuals

A

Appendix 1