TÉCNICO LISBOA

Deep Neural Network on the Versat Reconfigurable Processor

João Pedro Costa Luís Cardoso

Introduction to Research in

Electrical and Computer Engineering

Supervisor(s): Prof. José João Henriques Teixeira de Sousa

January 2020

Contents

List of Tables

List of Figures

Chapter 1

Introduction

FIX: mete citaçoes nas coisas mais importantes

In this report, the problem of accelerating the execution of Deep Neural Networks (DNNs) using Coarse GRained Reconfigurable Arrays (CGRAs) is studied, with special emphasis on compiling a DNN description into code that runs on CPU/CGRA system. The Deep Versat Architecture [**?**] CGRA will be used as an implementation tool in this work.

1.1 Problem

Neural Networks have been an object of study since the 1940's, but until the beginning of this decade their applications were limited and did not play a major role in computer vision conferences. With its meteoric rise in research, several solutions to accelerate this algorithm have appeared, from FIX (FPGA) to FIX (ASIC) implementations.

Convolutional Neural Networks (CNNs) are a particular kind of DNN where the output values of the neurons in one layer are convolved with a kernel to produce the input values of the neurons of the next layer. This algorithm is compute bound, that is, its performance depends on how fast it can do certain calculations, and depend less on the memory access time. Namely the convolutional layers take approximately 90$\%$ of the computation time.

The acceleration of these workloads is a matter of importance for today's applications such as image processing for object recognition or simply to enhance certain images. Other uses like instant translation and virtual assistants are applications of neural networks and their acceleration is of vital importance to bring them into Internet of Things.

A suitable circuit to accelerate DNNs in hardware is the CGRA. A CGRA is a collection of Functional Units and memories with programmable interconnections in order to form computational datapaths. A CGRA can be implemented in both FPGAs and ASICs. CGRAs can be reconfigured much faster than FPGAs, as they have much less configuration bits. If reconfiguration is done at runtime, CGRAs add temporal scalability to the spacial scalabilty that characterize FPGAs. Moreover, partial reconfiguration is much easier to do in CGRAs compared to FPGAs which further speeds up reconfiguration time. An-

other advantage of CGRAs is the fact that they can be programmed entirely in software, contrasting with the large development time of customized Intellectual Property (IP) blocks. The Coarse Grain Reconfigurable Arrays (CGRA) is a midway acceleration solution between FPGAs, which are flexible but large, power hungry and difficult to reprogram, and ASICs, which are fast but generally not programmable.

However, mapping a specific DNN to a CGRA requires knowledge of its architecture, latencies and register configurations, which may become a lengthy process, especially if the user wants to explore the design space for several DNN configurations. An automatic compiler that can map a standard DNN description into CPU/CGRA code would dramatically decrease time to market of its users. Currently there are equivalent tools for CPUs and GPUs and even for FPGAS.

1.2 Solution

The proposed solution is a compiler that takes a configuration file from a neural network framework like Caffe or Darknet. This new tool inputs the parameters of Deep Versat, such as the number of layers and functional units, and produces the C code needed for the Versat runs. This code is run on the RISC-V picorv32 [**?**] CPU controller that has Deep Versat as a peripheral.

1.3 Report Outline

This report is composed of 4 more chapters. In the second chapter, the state-of-the-art of neural networks and the difficulties accelerating them is described. In the third chapter, the Deep Versat architecture and how to program it is explained. In the fourth chapter, CNN compiler techniques are explored. Finally, the last chapter contains the proposed solution and the plan for its execution.

Chapter 2

Deep Neural Networks

A Neural Network (NN) is an interconnected group of nodes that follow a computational model that propagates data forward while processing. The earliest Neural Networks were proposed by McCulloh and Pitts [1] , in which a Neuron has a linear part, based on aggregation of data and then a non linear part called the activation function. The issue with using only one neuron is that it isn't able to be used in non-linear separable problems. By aggregating several neurons in layers and the input of each neuron as in figure 2.1 being based on the previous layers, that problem can be eliminated.

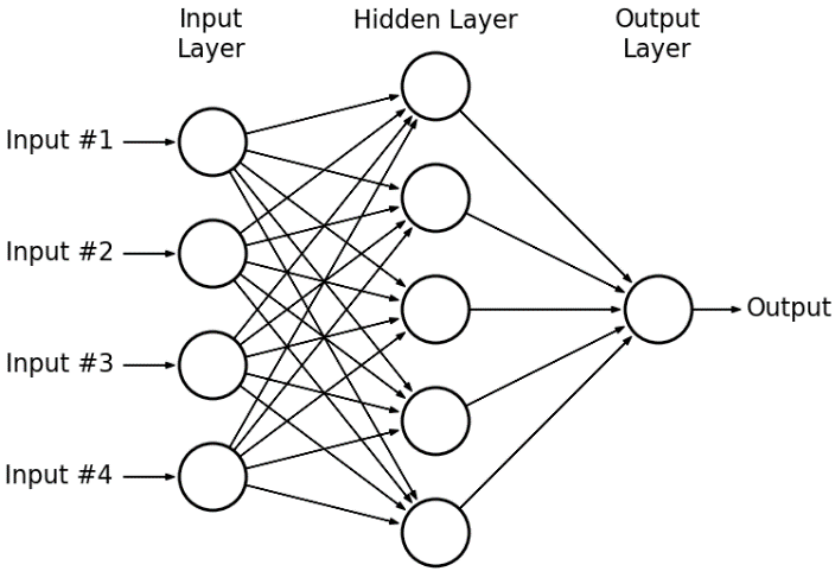

Figure 2.1: Multi Layer Perceptron- IMAGE TO BE REPLACED

Each input to a Neuron contributes differently to the output. This share is dependent on the weight value. These are obtained by training the network through various techniques, one of which is Deep Supervised Learning [2] . For a certain input, there's an expected output and the real output of the NN. Then the loss function based on these outputs is calculated and then the weight values are iteratively modified for better outputs by the Neural Network.

A Deep Neural Network (DNN) is a NN that uses this approach for learning. It has multiple hidden

layers and it can model complex non-linear relationships. If the activation function isn't a polynomial, it satisfies the Universal approximation problem [3].

One of the limitations of traditional Networks is the complexity that is given between each layer. Let's use the example of hand digit recognition problem. The MNIST data set is composed of 28x28 grayscaled images [4]. In a traditional fully connected Neural Network, a neuron from the second layer would have 28x28 weights. That's 3.136 kiloBytes per neuron of weight values while using Floating-Point 32 bit (FP32). By making the layer size constant, the computational power required grows exponentially. When building a more complex network for image recognition, the input layer grows and so by default does the number of weights.

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a class of DNNs used in Image and Video recognition due to their shift invariance characteristic. They were first proposed in the 1980's but it wasn't until 2012 with AlexNet [5] that CNNs really took off. Fundamentally, it's a regularized version of Multilayer Perceptrons (MLP).

These networks fix the issues discussed due to each neuron of the following layer being connected to a few of the previous layer.

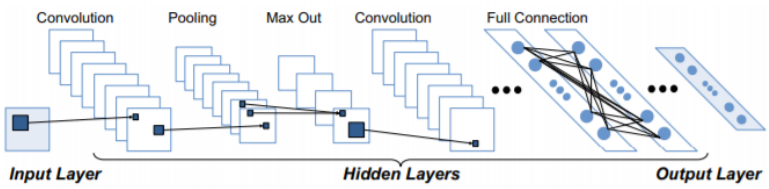

Figure 2.2: CNN- IMAGE TO BE REPLACED

2.1.1 Architecture Overview

Convolutional Layer

The most processing time consuming is the Convolutional Layer due to it's raw processing needs. It takes an input with several dimensions: image width,height and color space for the first layer, for the following convolutional layers it takes a 3D array: map width,height and number of channels. For the earlier example of the MNIST data set, it would be 28x28x1 as it's a 2D image in grayscale.

To compute a neuron in the next layer we get the convolution in equation 2.1 and image representation in figure 2.3, where x_j^{l+1} is the output, δ is the activation function, which depends on the architecture, x_i^l is the input of the convolution layer, k_{ij}^{l+1} is the kernel of said layer which is obtained by training the network and b_j^{l+1} is the bias.

$$x_j^{l+1} = \delta(\sum_{i \epsilon M_j} x_i^l * k_{ij}^{l+1} + b_j^{l+1}) \tag{2.1}$$

Thus an output neuron depends only on a small region of the input which is called the local receptive field.

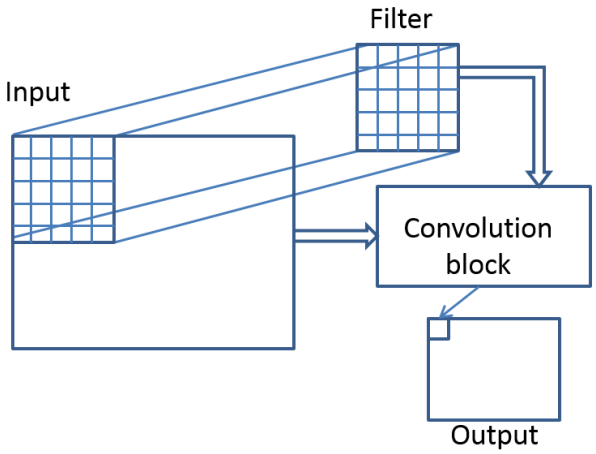

Figure 2.3: 2D CONV- TO BE REPLACED

The output's dimensions depend on several parameters of the convolution such as zero-padding and stride. The Former means to add 0's around the edges of the input matrix. The latter means the step used for the convolution, if the value is e.g 2, it will skip a pixel each iteration of the convolution. The equation in 2.2 can be used to calculate the output size. Where n is the width/height of the input of layer l, b is the width/height of the kernel, p is zero-padding while s is the stride.

$$n^{l+1} = \frac{n^l - b^l + 2 \times p}{s} \tag{2.2}$$

The number of channels of the output is equal to the number of filters in the convolutional layer.

Pooling Layer

The MaxPool or AvgPool are layers used in Convolutional Neural Networks to downsampling the feature maps to make the output maps less sensitive to the location of the features.

Maximum Pooling or MaxPool, like it's suggested in it's name groups $n*n$ points and outputs the pixel with highest value. The output will have it's size lowered by n times. The Average Pooling or AvgPool, instead takes all of the input points and calculates the average. Downsampling can also be achieved by using convolutions with stride 2 and padding equal to 1. Upsample layers can be also used that turn each pixel into n^2, where n is the amount of times the output will be bigger than the input.

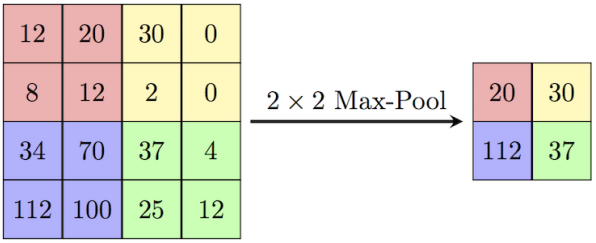

Figure 2.4: MAXPOOL- TO BE REPLACED

Fully Connected Layer

The fully Connected Layer is mostly used for classification in the final layers of the Neural Network. It associates the feature map to the respective labels. It takes the 3D vector and outputs a single vector thus it's also known as flatten. The equation in 2.3 describes the operation. Here w_{ji}^{l+1} are the weights associated with a specific input for each output pixel.

$$x_j^{l+1} = \delta(\sum_i (x_i^l \times w_{ji}^{l+1}) + b_j^{l+1}) \tag{2.3}$$

Route $\&$ Shortcut Layer

The Shortcut layer or skip connection was first introduced in Resnet [6]. It allows to connect the previous layer to another to allow the flow of information across layers. The Route layer, used in Yolov3 [7] ,concatenates 2 layers in depth (channel) or skips the layer forward. This is used after the detection layer in Yolov3 to extract other features.

Dropout Layer

This type of layer was conceived to avoid overfitting [8] by dropping the neurons with probability below the threshold. In figure 2.5, there's a graphical representation.

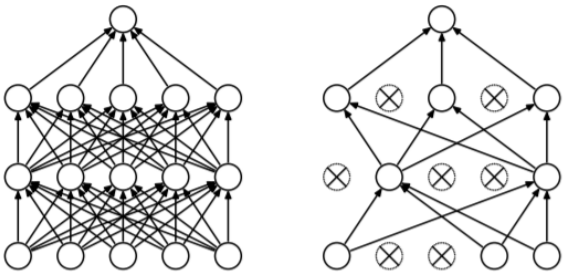

Figure 2.5: Dropout if applied to all layers, adapted from [8]

Activation Functions

Activation Functions (AF) are functions used in each layer of a Neural Network to compute the weighted sum of input and biases, which is used to give a value to a neuron.Non-linear AFs are used to transform linear inputs to non-linear outputs. While training Deep Neural Networks, vanishing and exploding gradients are common issues, in other words, after successive multiplications of the loss gradient, the values tend to tend to 0 or infinity and thus the gradient disappears. AFs help mitigate this issue by keeping the gradient in specific limits. The most popular activation functions can be found in table 2.1.

Activation Functions	Computation Equation
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Softmax	$f(x_i) = \frac{x_i}{\sum_j e^{x_j}}$
ReLU	$f(x) = \begin{matrix} x & if & x \geq 0 \\ 0 & if & x < 0 \end{matrix}$
LReLU	$f(x) = \begin{matrix} x & if & x > 0 \\ \alpha x & if & x \leq 0 \end{matrix}$
ELU	$f(x) = \begin{matrix} x & if & x > 0 \\ \alpha e^{x} - 1 & if & x \leq 0 \end{matrix}$

Table 2.1: Popular Activation functions

2.2 Frameworks for Neural Networks

To run a Neural Network model there's several popular frameworks like Tensorflow, PyTorch, Caffe and Darknet. Their propose is to offer abstraction to software developers that want to run these networks. They also offer programming for different platforms like nVidia GPU's by using the CUDA API.

2.2.1 Darknet

Darknet [9] is an open source neural network framework written in C and CUDA.It's used as the backbone for Yolov3 [7] and supports several different network configurations such as AlexNet and Resnet. It utilizes a network configuration file (.cfg) and a weights file (.weights) as input for inference.

Listing 2.1: cfg code for a Convolutional Layer used in Yolov3 [7]

```
[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky
```

In listing 2.1, there's a snippet of the file featuring a convolution layer with 32 kernels of size 3x3. It has stride of 1 and zero padding of 1, meaning the output size will be equal to the input. The input size can be calculated by analyzing the previous layers and the network parameters. The network parameters in 2.2 includes data to be used for training while only the first three parameters are needed for inference.

Listing 2.2: cfg code for the network parameters

```
[net]
```

```
width=608
height=608
channels=3

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

2.2.2 Caffe

Convolutional Architecture for Fast Feature Embedding (Caffe) [10] is also an Open source framework written in C++ with interface for Python. Caffe exports a neural network by serializing it using the Google Protocol Buffers (ProtoBuf) serialization library. Each network has 2 prototxt files:

- deploy.prototxt- File that describes the structure of the network that can be deployed for inference.
- train_val.prototxt- File that includes structure for training. it includes the extra layers used to aid the training and validation process.

The interface for python helps generate these files. For inference only the deploy file matters.

Listing 2.3: prototxt file for the input data and the first convolution layer of AlexNet [5]

```
name: "AlexNet"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
```

```
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
}
```

Chapter 3

Deep Versat

Versat is a Coarse Grained Reconfigurable Array (CGRA) Architecture. CGRAs are in-between Field Programmable Gate Arrays (FPGA) and general purpose processors (GPP). The former is fully reconfigurable and the highest performance for a workload can be achieved as the Architecture is tailored to the workload. GPPs on the other hand, aren't reconfigurable and thus slower but are more generic and can process different workloads. While FPGAs have the granularity at the gate level, CGRAs have the granularity at the functional unit level. They are configurable at run-time and the datapath can be changed in-between runs.

In this chapter, the base Versat Architecture will be explained and then the Deep Versat Architecture and it's improvements.

3.1 Versat Architecture

The Versat Architecture [11–14] is depicted in figure 3.1. It's composed by the following modules: DMA,Controller,Program Memory,Control File Registry,Data-Engine and Configuration module. The Controller accesses the modules through the control bus. The code made in assembly or C is loaded into the program Memory (RAM) where the user can write to the configuration module the versat runs. Between runs of the Data Engine, the Controller can start doing the next run configuration and calculations.

3.1.1 Data Engine

The Data Engine which is represented in figure 3.2 carries out the computation needed on the data arrays. It's a 32 bit Architecture with up to 11 Functional Units: Arithmetic and Logic Unit(ALU), stripped down ALU (ALU-Lite), Multiplier and Accumulator (MAC) and Barrel Shifter. Depending on the project and calculations, a new type of FU or the existing ones can be altered to support the algorithm. The DE has a full mesh topology, that means that each FU can be the output to another, This decreases the operating frequency.

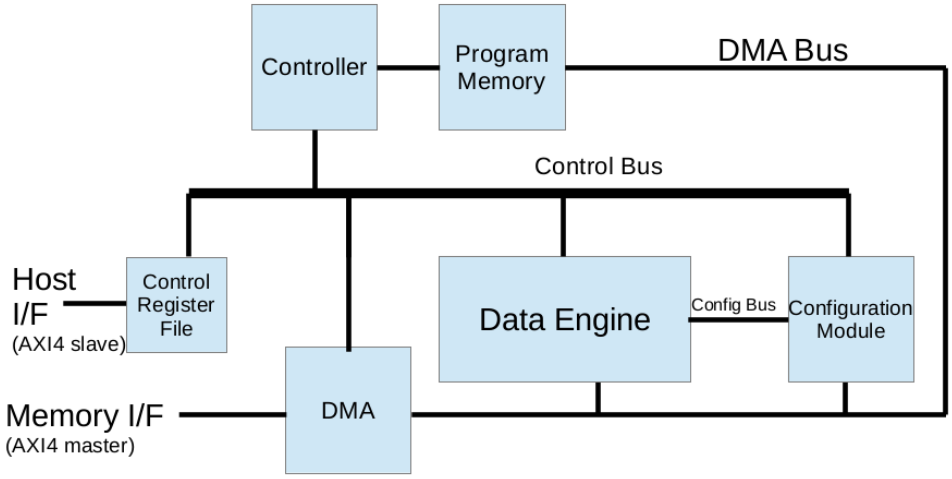

Figure 3.1: Versat Topology, taken from [12]

Each Input of a Functional Unit has a Mux with 19 entries, 8 of which are from the memories (2 from each Mem out of 4 total units) and the rest from the Functional Units (11).

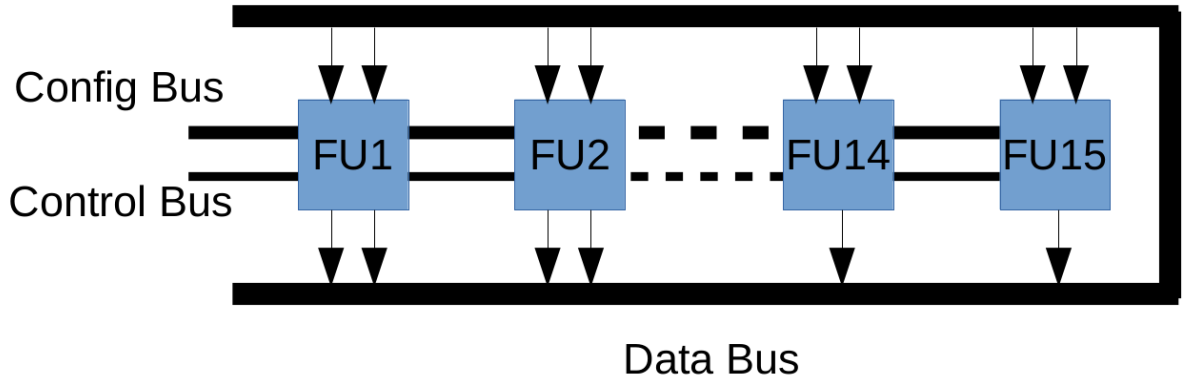

Figure 3.2: Versat Data Engine Topology, taken from [13]

The 4 Memories are dual port and for the input of both ports, there's an Address Generation Unit (AGU) that is able to reproduce two nested loops of memory indexes. The AGUs control which MEM data is the input of the FUs and where to store the results of the operation. Also, the AGUs support a delayed start to line up timings due to latencies.

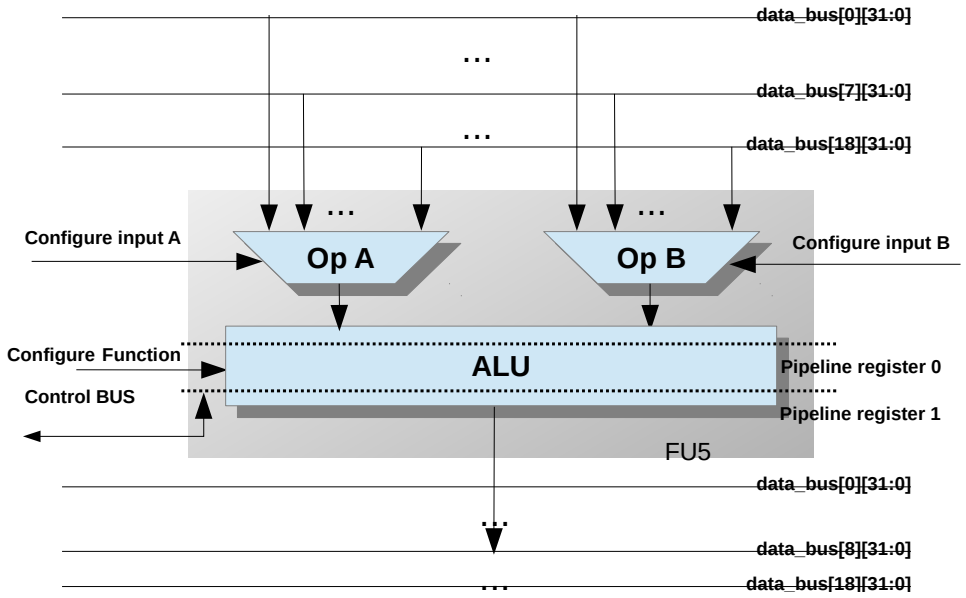

Figure 3.3: Versat Functional Unit, taken from [15]

3.1.2 Configuration Module

Versat has several configuration spaces devised for each Functional Unit, with each space having multiple fields to define the operation of the Functional unit (e.g which op for the ALU). These are accessed before the run by the controller to define the datapath.

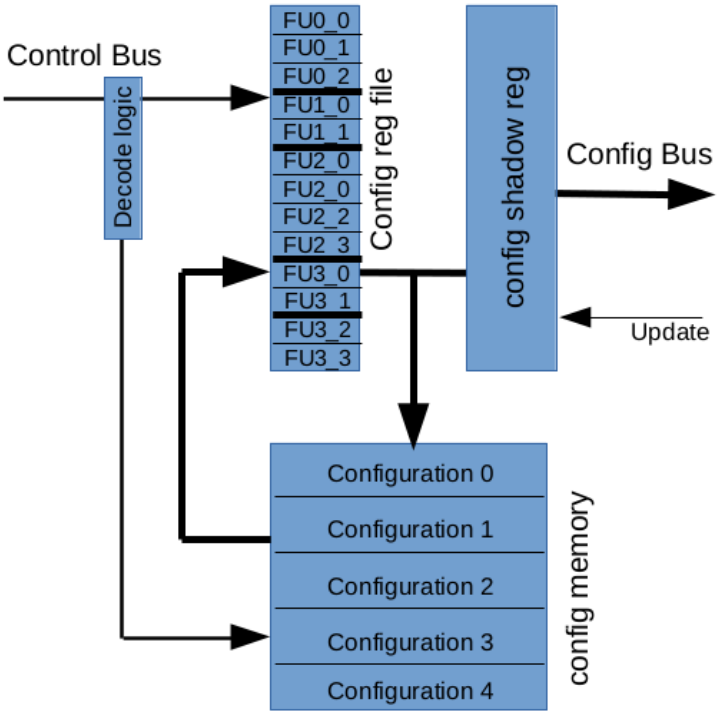

Figure 3.4: Configuration Module,taken from [12]

The Configuration Module (CM), depicted in figure 3.4, has three components:configuration memory, variable length configuration register file and configuration shadow register. The latter holds the current

configuration so the controller can change the values of the configuration file in-between runs. The decode logic finds which component to write or read, if it's the registers, it ignores read operations. Meanwhile, the configuration memory interprets both write and reads. When it receives a read, it writes into the register configuration data, when it's a write, it stores the data instead.

3.2 Deep Versat Architecture

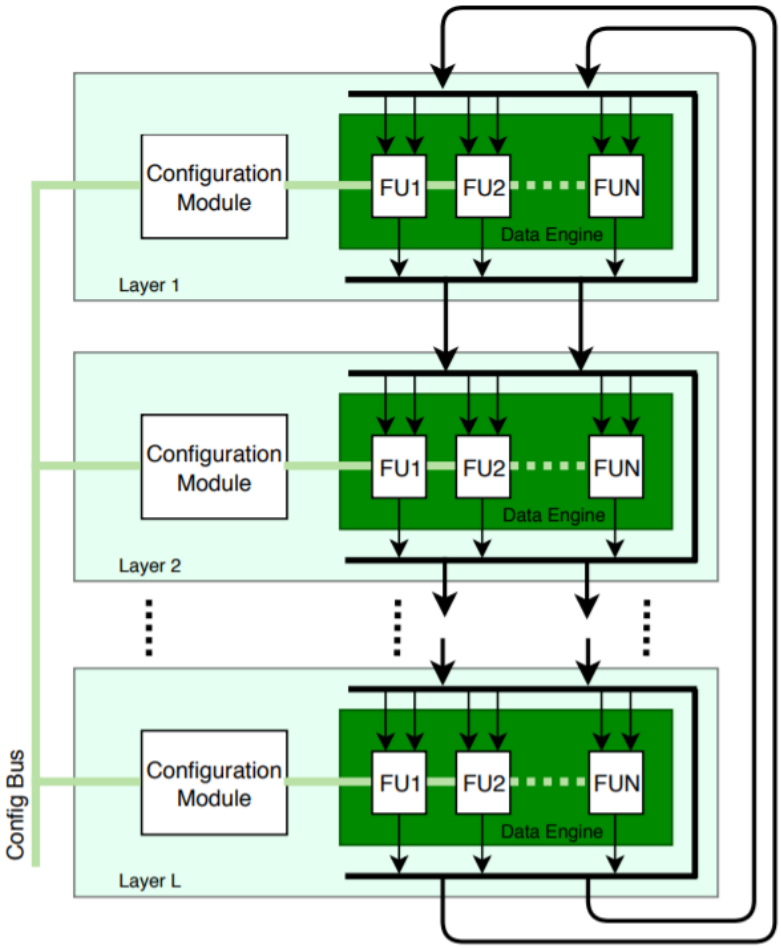

Figure 3.5: Deep Versat Architecture, taken from [16]

The Deep Versat Architecture[16] , in figure 3.5, decouples the Data Engine (DE) from all control and as such, it can be used with any CPU. It can be paired with hard cores in FPGA boards like the ZYNC board with it's A9 ARM dual core CPU's or pair it with a soft core.

It's principle is to create the concept of a Versat Core: Configuration Module (CM) and it's Functional Units (FU) connected with a control bus and a data bus. Instead of writing to a memory, there's the option to write for the next Versat Core to create more complex and more complete Datapaths, to avoid having to reconfigure a lot of times.

The number of Layers and FUs are reconfigurable pre-silicon with the only limitation that each layer is identical. To program Deep Versat, an API is generated from the Verilog .vh files.

3.2.1 Deep Versat System

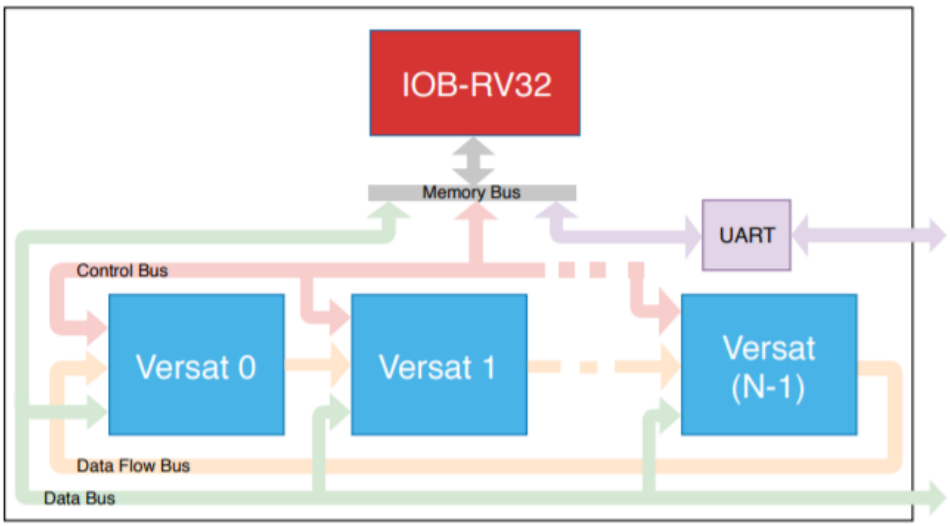

Figure 3.6: Deep Versat System, taken from [16]

To make a complete system, a new controller is needed with a more robust toolchain. In a recent dissertation [16], the IOB-RV32 processor was used which uses the RISC-V 32IM. The core is derived from the open source PicoRV32 CPU [17]. The IOB-RV32 uses it's memory bus to access peripherals in which Deep Versat and the UART module are connected as such. The control bus is used to acess the configuration modules of Deep Versat. The data bus is used to read and write large amount of data into Deep Versat. The data flow bus is reserved for inter Versat Core communication.

Peripheral	**Memory address**
UART module	12'h100xxxxx
Deep Versat control bus	8'h11xxxxxx
Deep Versat data bus	8'h12xxxxxx

Table 3.1: Deep Versat Memory Map

The memory map to address the peripherals, including deep versat, is in table 3.1. Each Versat has 15 bits of Address while the CPU addresses the peripherals with 32 bits, with 8 of those occupied to chose the peripheral in question. That leaves 9 bits to address several Versat Cores which brings the theoretical maximum versat cores to 512. The IOB-RV32 is compatible with GNU toolchain to offer better portability of code and alongside the C++ Versat API the difficulty to code for the System diminishes.

Chapter 4

CNN Compiling and Computation

In section 2.2, Neural Network Frameworks are introduced. This chapter is divided into two sections. Section 4.1 is an overview of Toolflows that map Convolutional Neural Networks using said frameworks into FPGA. Section 4.2 introduces concepts to accelerating CNNs into deployed hardware like CGRAs.

4.1 Toolflows for Mapping CNNs in FPGAs

Several software frameworks have been developed to accelerate development and high-performance execution of CNNs. Neural Networks Frameworks discussed in section 2.2 provide high level APIs together with high performance execution on Multi Core CPUs, GPUs, Digital Signal Processors (DSP) and Neural Processing Units (NPU) [18]. FPGAs provide an alternative to these architectures by being high-performance while also being low-power that can meet several requirements like throughput and latency in diverse applications. Thus, toolflows that map descriptions of CNN into hardware architecture to perform inference of the Network were created. Several toolflows and their interfaces are present in table 4.1.

Toolflow Name	**Interface**	**Year**
fpgaConvNet	Caffe & Torch	May 2016
DeepBurning	Caffe	June 2016
Angel-Eye	Caffe	July 2016
ALAMO	Caffe	August 2016
Haddoc2	Caffe	September 2016
DNNWeaver	Caffe	October 2016
Caffeine	Caffe	November 2016
AutoCodeGen	Proprietary Input Format	December 2016
Finn	Theano	February 2017
FP-DNN	Tensorflow	May 2017
Snowflake	Torch	May 2017
SysArrayAccel	C	June 2017
FFTCodeGen	Proprietary Input Format	December 2017

Table 4.1: CNN to FPGA Toolflows, adapted from [19]

4.1.1 Supported Neural Network Models

These toolflows support the most common layers in CNNs discussed in chapter 2. The acceleration target changes depending on toolflow. fpgaConvNet [20] focuses more on feature extraction while offering non accelerated support for Fully Connected layers by casting them as Convolutional Layers with 1x1 kernels.

4.1.2 Architecture & Portability

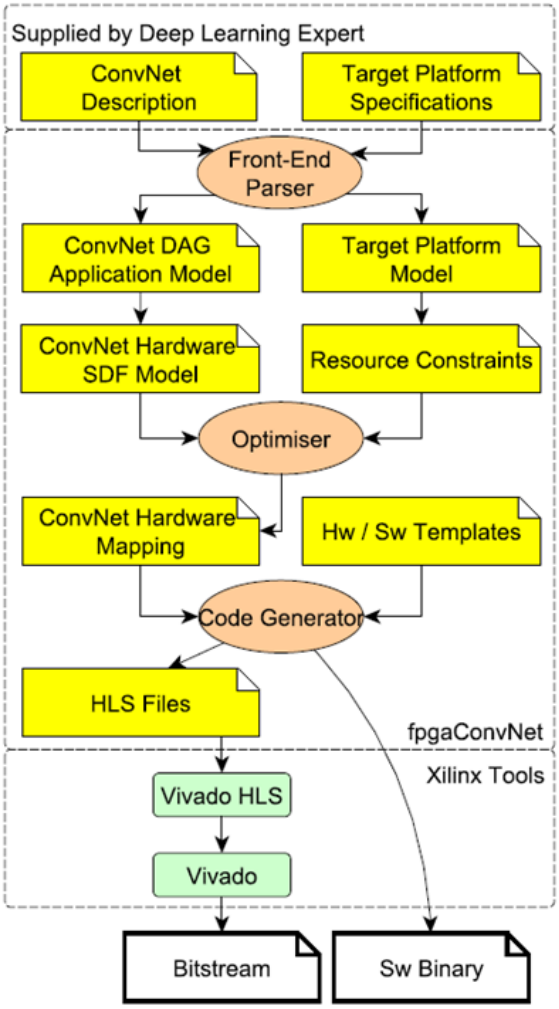

Figure 4.1: fpgaConvNet Architecture. Taken from [20]

As shown in figure 4.1, fpaconvnet architecture consits of a Front-End Parser to capture the number of layers to implement, their size and kernels. Then the architecture is optimized to the device target and Neural Network. Finally, it's mapped to hardware by using High Level Syntheses (HLS) which is a programming model for hardware based on C with a layer of abstraction.

4.2 CNN Auto Tuning Framework

Chapter 5

Proposed Work and Planning

The proposed work for the dissertation consists in the development of an Auto-Tuning Software for the Deep Versat Architecture while connected to the IOB-RV32 CPU. The compiler's propose is to be able to run any state of the art CNN on the Deep Versat system with no effort on the user side. For the proof of concept stage, Darknet and Caffe will be the frameworks chosen to be compatible with this compiler.

Deep Versat has customizable FU amounts and options, so the compiler must be able to change the datapath based on the Deep Versat Configuration.

5.1 Hardware System

For the Auto-Tuning Software to be able to model CNNs for Deep Versat, the System needs to access External Memory for the weights and inputs as for state of the art network like Yolov3 [7] has 107 total layers with weight size of 236MB. That amount can't be stored in BRAMs so the IOB-RV32 will have to load from memory (RAM) to Deep Versat and vice-versa.

INPUT IMAGE HERE

5.1.1 Acceleration on Deep Versat

Some of the activation functions discussed in chapter 2 will have to be processed in software on the core unless custom Functional Units are made for each function. Convolutional,Fully Connected,Shortcut and Route layers can be implemented on Deep Versat without any FU changes. How much it will be accelerated is based on number of cores and MACs available. Pooling needs adaptation in Hardware to run them, if not implemented, run on the RISC-V core.

5.2 Software Proposal

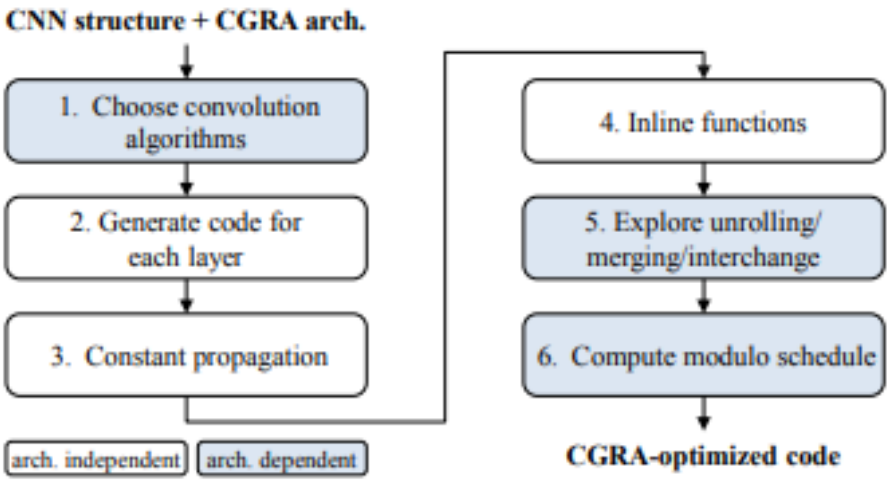

Figure 5.1: Optimizing Runs on CGRA. Taken from [21]

To build a CNN model for Versat, a parser from Configuration file to layer is needed. For darknet [9], the parser can be re-used. For Caffe, a new one would need to be built so the output of the parser is equal to other frameworks for the same CNN network. The objective to be accomplished is to support all Caffe possible layers and Darknet's layers. After parsing, the Versat dataflow and runs for each type of layer will be defined depending on current silicon set up of the CGRA. Then, the C code with the Versat runs will be written. In 5.1, a CGRA optimization flow for CNN is presented.

5.3 Planning

In fig 5.2 is presented a GANT chart with the proposed schedule of the planned work.

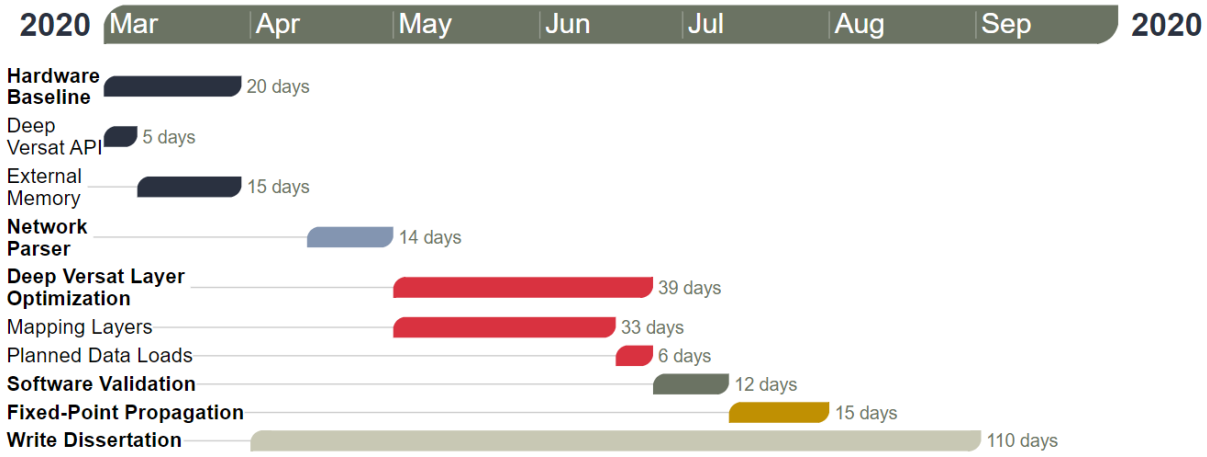

Figure 5.2: GANT chart of Proposed Work

Bibliography

[1] G. Piccinini. The first computational theory of mind and brain: A close look at mcculloch and pitts's "logical calculus of ideas immanent in nervous activity". *Synthese*, 141, 08 2004. doi: 10.1023/B:SYNT.0000043018.52445.3e.

[2] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

[3] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861 – 867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL `http://www.sciencedirect.com/science/article/pii/S0893608005801315`.

[4] mnist database of hand-written digits. URL `http://yann.lecun.com/exdb/mnist/`.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[7] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[9] J. Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. URL `http://arxiv.org/abs/1408.5093`.

[11] R. Santiago, J. D. Lopes, and J. T. de Sousa. Compiler for the versat reconfigurable architecture. REC 2017, 2017.

[12] J. D. Lopes, R. Santiago, and J. T. de Sousa. Versat, a runtime partially reconfigurable coarse-grain reconfigurable array using a programmable controller. Jornadas Sarteco, 2016.

[13] J. D. Lopes and J. T. de Sousa. Fast fourier transform on the versat cgra. Jornadas Sarteco, 09 2017.

[14] J. D. Lopes and J. T. de Sousa. Versat, a minimal coarse-grain reconfigurable array. In D. I., C. R., B. J., and M. O., editors, *High Performance Computing for Computational Science – VECPAR 2016*, pages 174–187. Springer, 2016. doi:10.1007/978-3-319-61982-8_17.

[15] J. D. Lopes. Versat, a compile-friendly reconfigurable processor – architecture. Master's thesis, Instituto Superior Técnico, November 2017.

[16] V. J. B. Mário. Deep versat: A deep coarse grain reconfigurable array. Master's thesis, Instituto Superior Técnico, November 2019.

[17] Picorv32- a size-optimized risc-v cpu. URL `https://github.com/cliffordwolf/picorv32`.

[18] A. Ignatov, R. Timofte, P. Szczepaniak, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool. Ai benchmark: Running deep neural networks on android smartphones, 10 2018.

[19] S. I. Venieris, A. Kouris, and C.-S. Bouganis. Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions, 2018.

[20] S. I. Venieris and C.-S. Bouganis. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47. Institute of Electrical and Electronics Engineers (IEEE), May 2016. doi: 10.1109/FCCM.2016.22. URL `http://dx.doi.org/10.1109/FCCM.2016.22`.

[21] I. Bae, B. Harris, H. Min, and B. Egger. Auto-tuning cnns for coarse-grained reconfigurable array-based accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37, 07 2018. doi: 10.1109/TCAD.2018.2857278.