Internship Report

NIT Andhra Pradesh Online Internship

Y V Sai Dinesh
Department of Electronics & Communications Engineering
Central University of Karnataka
sdinesh@cuk.ac.in

***Abstract*—This project proposes a framework for planning CNN accelerator on embedded FPGA for image classification. The planned framework provides a tool for FPGA resource-aware style area exploration of CNNs and mechanically generates the hardware description of the CNN to be programmed on a target FPGA. The framework consists of 3 main backends; software package, hardware generation, and simulation/precision adjustment. The software package backend is an API to the designer to style the CNN and train it in line with the hardware resources that square measure available. Not like different tools, that square measure usually unnatural by a tiny low range of inflexible templates, our flow uses Google's XLA compiler that emits LLVM code directly from a Tensorflow specification. This LLVM code will then be used with a high-level synthesis tool to mechanically generate hardware. We show that our flow permits users to get Deep Neural Networks with only a few lines of Python code.**

I. Introduction

Deep learning has emerged as a very important application space for Field-Programmable Gate Arrays (FPGAs). FPGA implementations of machine learning applications will usually run abundant quicker that computer code implementations, and may consume considerably less power than Graphic process Unit (GPU) implementations. As FPGAs seem as a part of cloud computing infrastructure, we tend to expect that these blessings can lead additional and additional designers to require advantage of the advantages of FPGAs for these applications.

Designing such applications is difficult, however. the look flow for Associate in Nursing FPGA machine learning accelerator might begin with a computer code model enforced employing a package like TensorFlow, Keras, or PyTorch. These frameworks permit the abstraction of implementation details, sanctionative non-experts to experiment with state-of-art Deep Neural Networks. At this stage, the designer will perceive the desired network size, convergence rate, etc. The user will valuate the expected success of the algorithmic rule on expected knowledge if an acceptable knowledge set is out there.

Once the topology and meta-parameters have been selected, the designer can map the circuit to a hardware implementation. This is often done manually, by writing C code with appropriate optimization directives and using a high-level synthesis tool, or writing Register-Transfer Level (RTL) code and compiling. This step is time consuming and requires hardware design expertise that limit the applicability of FPGAs in this important domain. Our flow uses Google's XLA compiler which emits LLVM code directly from a Tensorflow specification.

The framework consists of three main components that correspond to our contributions. They are as follows:

A. Software backend

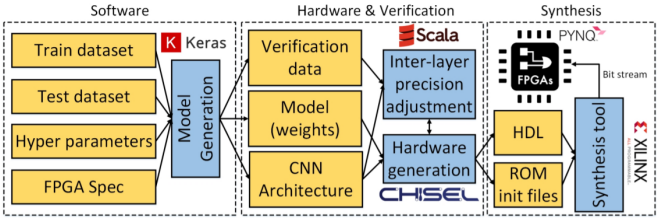
Using Python, we provide a tool for designing and training a CNN architecture. The model (weights) of the trained network and other CNN architecture parameters are used in the hardware framework to generate the hardware description language of the CNN, which is going to be implemented on the FPGA. This back-end also provides the information needed for Simulation/precision adjustment backend as well as checking the hardware resources for the designed CNN model.

B. Hardware backend

Using CHISEL , we perform automatic generation and integration of different hardware components needed for the CNN architecture designed in the software backend. The CNN model is passed to this back-end to be used in HDL generation of the CNN. The output of this backend is the HDL code that is ready to be synthesized and programmed on an FPGA.

C. Simulation/precision adjustment backend

Using Scala testing libraries and the data passed to this backend, inter-layer precision of the generated CNN hardware will be adjusted. The CNN output error introduced by varying integer and fractional part of each layer's data is minimized by this backend.

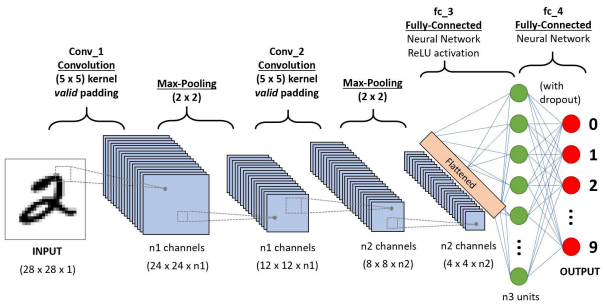


D. MLP and MNIST digit recognition

The MNIST database of handwritten digits is commonly used for evaluating a variety of image processing algorithms. It contains a training set of 60,000 examples and a test set

of 10,000 examples of 28x28 pixels. In this example, an MLP followed by a softmax is trained offline in Tensorflow using XLA and LeFlow-generated hardware is deployed in an FPGA for inference. The example including the training phase with XLA is part of the LeFlow distribution. The input layer contains 784 nodes, one for each pixel of the input image. The output layer of the MLP, which contains one output for each digit to be identified, is then fed to the softmax, resulting in 10 outputs that represent the probability distribution over the 10 different possible digits. The Tensorflow code used to generate the hardware through LeFlow. It is also important to note that it is usually not necessary to make any changes to a code that works with Tensorflow's XLA in order for it to work with LeFlow. Moreover, the values assigned to "inputs", "weights" and "bias" are not immutable in the generated circuit, since those place holders will be mapped into memories instead of being hardcoded into the design.

II. CNN



Above figure shows an typical LeNet-5 CNN for handwriting digit recognition, which is simpler than many modern CNNs, but sufficiently representative for introducing the basic CNN models. It has 2 convolutional layers, 2 max-pooling layers and 2 fully connected multilayer perceptron (MLP) layers.

Convolutional neural networks are considered as one of the most influential innovations in the field of computer vision. The success of deep learning networks grew to prominence in 2012 when utilized CNNs to win the annual olympics of computer vision, ImageNet large-scale vision recognition challenge (ILSVRC). Using AlexNet model, they achieved an astounding improvement as the image classification error dropped from 26% (in 2011) to 15%. ImageNet is a standard benchmark dataset used to evaluate the performance ImageNet Competition Results. of object detection and image classification algorithms.

However, the classic use-case of CNNs is for image and speech processing. A typical CNN is a multi-layered feed-forward ANN with a pipeline-like architecture. Specifically, each layer performs a well-known computation on the outputs of the previous layer to generate the inputs for the next layer. In general, CNNs have two types of inputs; the data to be tested or classified (also named as feature maps), and the weights.

Images, audio files, and recorded videos are examples of the input data to be classified using CNNs. On the other hand, the network weights are the data generated from training the CNN on a dataset containing similar inputs to the one being tested.

III. HARDWARE ACCELERATION OF DEEP LEARNING NETWORKS (SURVEY)

To provide more accurate results as well as real-time object recognition, for example in applications such as robots and auto-piloted cars, the size of the convolution neural network needs to be increased by adding more neural network layers. However, evolving more and new type of NN layers results in more complex CNN structures as well as high depth CNN models. Thus, billions of operations and millions of parameters, as well as substantial computing resources are required to train and evaluate the resultant largescale CNN. Such requirements represent a computational challenge for general purpose processors (GPP).

Consequently, hardware accelerators such as application specific integrated circuit (ASIC), field programmable gate array (FPGA), and graphic processing unit (GPU) have been employed to improve the throughput of the CNN. In practice, CNNs are trained off-line using the backpropagation process. Then, the off-line trained CNNs are used to perform recognition tasks using the feed-forward process. Therefore, the speed of feed-forward process is what matters. GPUs are the most widely used hardware accelerators for improving both training and classification processes in CNNs.

This is due to their high memory bandwidth and throughput as they are highly efficient in floating-point matrix-based operations. However, GPU accelerators consume a large amount of power. Therefore, their use in CNN-based applications implemented as a cloud service on large servers or in battery operated devices becomes a challenge. Furthermore, GPUs gain their performance from their ability to process a large image batch in parallel.

For some applications like a video stream, input images should be processed frame by frame as the latency of the result of each frame is critical to the application's performance. For some tracking algorithms, the result of one frame affects the process of the next frame. Recently evaluated emerging DNN algorithms on latest generations of GPUs (i.e., NVIDIA Titan X Pascal) and FPGAs (i.e., Intel Arria 10 GX 1150 and Intel Stratix 10 2800). The experimental results show that current trends in deep neural networks favor FPGA platforms as they offer higher power efficiency (a.k.a., performance per Watt). FPGA and ASIC hardware accelerators have relatively limited memory, I/O bandwidths, and computing resources compared with GPU-based accelerators.

However, they can achieve at least moderate performance with lower power consumption. The throughput of ASIC design can be improved by customizing memory hierarchy and assigning dedicated resources. However, the development cycle, cost, and flexibility are not satisfactory in ASIC-based

acceleration of deep learning networks. As an alternative, FPGA-based accelerators are currently in use to provide high throughput at a reasonable price with low power consumption and reconfigurability.

The availability of high-level synthesis (HLS) tools, using C or C++, from FPGA vendors lowers the programming hurdle and shortens the development time of FPGA-based hardware accelerators. Convolutional neural networks have a very useful property, that is, each feature map neuron shares its weights with all other neurons. It has been proved that the highest energy expense results from accessing the off-chip DRAM memory for data movement rather than computation. In other words, the energy cost of the increased memory accesses and data movement due to the large number of CNN operations often exceeds the energy cost of computation. Thus, CNN accelerators need to carefully consider this to achieve efficient architecture in terms of time and power.

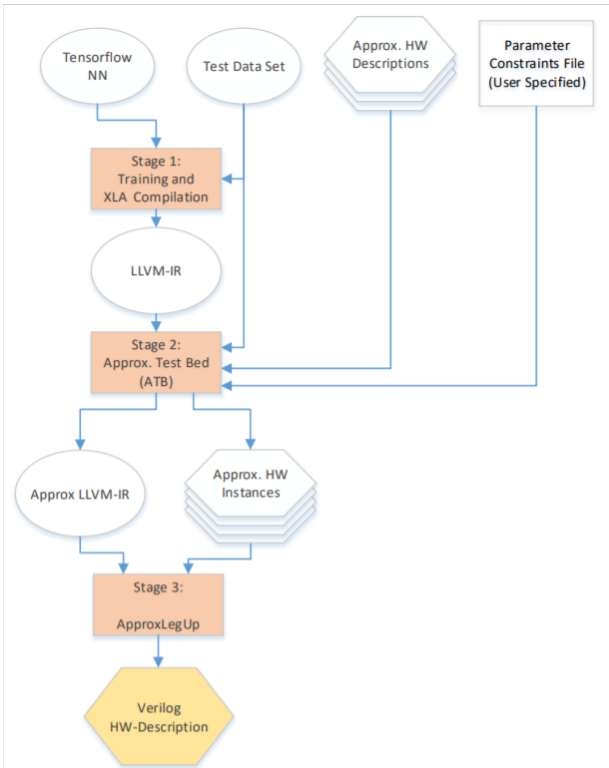
IV. Tools used for High Level Synthesis

A. LegUp

LegUp is Associate in Nursing open supply HLS tool which may be accustomed synthesize High-Level descriptions of HW systems to Verilog. It options 2 modes: pure HW and a HW/SW hybrid flow. we have a tendency to utilize the pure HW flow during this paper, however Associate in Nursing extension to a HW/SW hybrid flow is feasible. At its core LegUp options the LLVM framework and LLVM-IR2 that is employed by LeFlow to pass the outline of the NNs to LegUp. directions in LLVM-IR correspond on to HW operations. LegUp performs the classic HLS steps on the IR that area unit allocation, scheduling, binding and generation of alpha-lipoprotein. Further, LegUp permits for many optimizations like loop unrolling and dead code removal. even if LegUp is open supply, in keeping with LegUp produces HW implementations that area unit of comparable quality to industrial HLS tools.

B. LeFlow

LeFlow is a design flow, that is based on LegUp. Instead of feeding a C-Program into LegUp, LeFlow uses TF and Googles XLA compiler to compile a NN description (given in Python) to an optimized LLVM-IR. LeFlow has two stages: The first stage is implemented in Python. In this stage the description of the NN is imported and trained. After compiling it to LLVM-IR using Googles XLA compiler, the LLVM-IR is restructured such that it can be fed to LegUp. In the second stage, the restructured LLVM-IR is fed to LegUp which generates synthesizable Verilog.



V. LLVM Version Issues

An implementation-specific complexity was that our version of LegUp uses LLVM 3.5.0, while Tensorflow uses LLVM 7.0. Some differences include (1) the syntax of meta-data as well as the syntax of GEP, load, and store instructions were modified, (2) some LLVM function attributes, including speculable and nonrecurse, were added in the more recent version of LLVM. Our implementation of the LeFlow block performs transformations to address these differences.

VI. Limitations of LeFlow

- LeFlow currently uses kernels that were implemented in XLA and were originally meant to be used by CPUs. Although compiler optimizations and scheduling are able to retrieve a substantial amount of parallelism from those implementations, LeFlow would heavily benefit from an XLA back-end with kernels targeting FPGAs
- The high dimensionality of inputs/weights and the amount of parallel accesses that are typical in machine learning applications is a challenge for modern automatic memory partitioning algorithms. LeFlow would specially benefit from a machine learning specific automatic memory partitioning algorithm.
- One of the key possibilities that make deep learning networks efficient in FPGAs is the opportunity to use a customizable fixed-point bit width. Adding fixed-point support to LeFlow will be an important step in the development of this toolkit. Additionally, techniques to automatically profile the application and choose the appropriate representation could be easily explored in software with Tensorflow and deployed in hardware.

- Although it is straightforward to use Tensorflow to debug the functionality of an implementation, it is currently difficult for software developers to debug the generated hardware in terms of the original Python code. A performance debugging infrastructure suitable for software developers is another interesting venue for research.

VII. Experimental Results

MNIST Hand Written Digits & CIFAR10 dataset is used to train and test the CNN. Then converted the images to grey to have one-channel input due to our a small FPGA board. Using the software backend we can design and train a CNN. Table below shows the architecture of the CNN we targeted for classifying MNIST/CIFAR-10 dataset. I used 3×3 filters for convolutions. The reason behind choosing such a small CNN is that we are targeting a small FPGA as our hardware platform and any larger CNN would not pass the checking resource limitation phase of the Software backend. Also, I wanted to show the accuracy difference between the accelerated version on FPGA and the software version. In order to compare the software version and the version implemented on FPGA, we can classify MINIST/CIFAR-10 test dataset for both software model on Altera boards which is the embedded processor of targeted FPGA and on the CNN accelerator with two modes. HW-SM is the CNN accelerator generated in shared mode (sharing one convolution unit among all layers), and HW-EM is the CNN accelerator generated in exclusive mode (one convolution for each layer). This leverage Google's XLA compiler, which generates LLVM IR, and use this IR as input to an FPGA-oriented HLS tool. Due to mismatches between the XLA IR output and the requirements of our HLS tool, significant transformations are required in order to ensure a seamless flow.

VIII. References

- https://www.easics.com/products/deep-learning-fpga
- https://arxiv.org/pdf/1901.00121.pdf
- https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0222984
- https://www.hindawi.com/journals/ijrc/2019/7218758/
- https://www.xilinx.com/applications/megatrends/machine-learning.html
- https://www.xilinx.com/products/design-tools/ai-inference/edge-ai-platform.html3
- https://ieeexplore.ieee.org/document/8855594
- https://link.springer.com/article/10.1007/s11227-020-03325-8

Please check the below link for my codes and dependencies of this internship.

Codes: https://github.com/wikdin/Final-Project