FPGA-Accelerated Convolutional Neural Network

1st Mohammed CHELKHA
SEMi, SIGER
FMPs, USMBA
Brussels, Belgium
mohammed.chelkha@umons.student.ac.be

2nd Carlos VALDERRAMA
ELECTRONICS AND MICROELECTRONIcS UNIT
POLYTECHNIC FACULTY OF MONS
Mons, Belgium
carlos.valderrama@umons.ac.be

3rd Ali AHAITOUF
SIGER
USMBA
Fez, Morocco
ali.ahaitouf@usmba.ac.ma

***Abstract*—In recent years, FPGA has become an attractive solution to accelerate CNN classification for its flexibility, short time-to-market, and energy efficiency. The real-time evaluation of a CNN for image classification on a live video stream can require billions or trillions of operations per second. To come with a competitive re-configurable implementation satisfying both development time and flexibility, we propose using as a base a re-configurable Architecture composed by a set of image and video processing blocks. The whole architecture can be configured on-the-fly based on the image characteristics thus supporting variable image resolutions for each layer of the CNN.**

***Index Terms*—Convolutional Neural Network, FPGA, Deep learning, Coarse-Grain**

I. Introduction

Convolutional Neural Networks have been some of the most influential innovations in the field of computer vision. [1] 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (the annual Olympics of computer vision), dropping the classification error record from 26 to 15 percent, an astounding improvement at the time.

CNNs extraordinary performance comes at a high cost in terms of computing complexity. Even more effort is required for picture segmentation and scene tagging. While the latest graphics processing units (GPUs) can achieve this level of speed, there is also a need to integrate such solutions into other systems, such as vehicles, drones, or even wearable gadgets, which have stringent physical size and energy consumption constraints. As a result, future embedded CNNs will require compact, efficient, yet extremely powerful processing platforms. [2] This study aims to explore the possibilities of utilizing an existing novel flexible architecture for real time image and video processing, that takes advantage of the inherent parallelism of Filed programmable Gate Arrays (FPGAs) to achieve real-time performance. We can resume our proposal in two main contributions:

- A convolutional neural network architecture that has been designed to fit perfectly on the FPGA. The CNN is very consistent, and it achieves a satisfactory classification accuracy with low processing cost.
- A hardware/software co-design to efficiently accelerate the entire CNN on FPGAs. We propose a uniformed convolutional matrix-multiplication representation for both computation-intensive convolutional layers and pooling layers.

II. Related Work

GPUs were initially developed to accelerate graphics processing. A GPU is particularly designed for integrated transform, lighting, triangle setup/clipping, and rendering. A modern GPU is not only a powerful graphics engine but also a highly parallelized computing processor featuring high throughput and high memory bandwidth for massive parallel algorithms [3], which is dubbed as GPU computing or general-purpose computing on GPU (GPGPU). For our interest, CNNs can take advantages of the nature of algorithmic parallelism in the following aspects [4] : (i) the convolution operation of an n x n matrix using a k x k kernel can be in parallel; (ii) the sub-sampling/pooling operation can be parallelized by executing different pooling operations separately; (iii) the activation of each neuron in a fully connected layer can be parallelized by creating a binary-tree multiplier. With great parallel processing structures and strong floating-point capabilities, GPGPUs have been recognized to be a good fit to accelerate deep learning. A number of GPU-based CNN libraries have been developed to facilitate highly optimized CNN implementation on GPUs, including cuDNN [5], Cuda-convnet [6] and several other libraries built upon the popular deep learning frameworks, such as Caffe [7], Torch, Tensorflow [8], Keras, etc.

III. Background

As a typical supervised learning algorithm, there are two major phases in CNN: training phase and inference (aka feed-forward) phase. Since many industry applications would train CNN in the background and only perform inferences in a real-time scenario, we mainly focus on the inference phase in this thesis. The aim of the CNN inference phase is to get a correct inference of classification for input images. It is composed of multiple layers, where each image is fed to the first layer. Each layer receives a number of feature maps from a previous layer, and outputs a new set of feature maps after filtering by certain kernels. The convolutional layer, activation layer, and pooling layer are for feature map extraction, and the fully connected layers are for classification. When these layers are stacked, we have formed a full CNN architecture. However, knowing the overview of how CNNs operate isnt going to be sufficient to implement a CNN with real world data. Its imperative to not only understand the individual layers, but the fine points of the parameters and how they communicate with other layers too.

IV. Enhanced P2IP Architecture for Real-World Application

The P2IP is a systolic CGRA designed for real-time image and video processing targeting embedded systems. The objective of this architecture is to overcome the limitations of the existent solutions for image and video processing, combining high-performance, low-latency, and low-power consumption, with a level of flexibility. Images or frames are entered as a stream of pixels in a sequential line-scanned format progressing through the pipeline at a constant rate. The P2IP datapath works at the pixel clock frequency,delivering one processed pixel per clock cycle after a initial latency to fill the pipeline.It was projected to work between a frame source and a frame sink directly on the pixelstream. In order to allow the P2IP integration into an image processing chain, the AXI4-Stream [12] was adopted as the external interconnection protocol. The AXI4-Stream protocol is managed by the P2IP Controller which is also in charge of reading the configuration words on the configuration input port (config in) and transferring them to the processing core. The P2IP controller is the input of the P2IP configuration mechanism followed by a Configuration Decoder Tree (CDT), distributed throughout the processing core.

V. Hardware Implementation of P2IP-CNN on FPGA

A. Developping the Baseline Model

The design of the test harness is modular, and we can develop a separate function for each piece. This allows a given aspect of the test harness to be modified or inter-changed, if we desire, separately from the rest.

We can develop this test harness with three key elements. They are the preparation of the dataset, the definition of the model and the extraction of the weights and the feature maps of each layer.

B. Accelerator Design Exploration

Our CNN accelerator design on FPGA is composed of several major components, which are processing elements (PEs), on-chip buffer, external memory, and on-/off-chip interconnect. A PE is the basic computation unit for convolution and pooling. All data for processing are stored in external memory. Due to on-chip resource limitation, data are first cached in on-chip buffers before being fed to PEs. Double buffers are used to cover computation time with data transfer time. The on-chip interconnect is dedicated for data communication between PEs and on-chip buffer banks.

There are several design challenges that obstacle an efficient CNN accelerator design on an FPGA platform. First, loop tiling is mandatory to fit a small portion of data on-chip. An improper tiling may degrade the efficiency of data reuse and parallelism of data processing. Second, the organization of PEs and buffer banks and interconnects between them should be carefully considered in order to process on-chip data efficiently. Third, the data processing throughput of PEs should match the off-chip bandwidth provided by the FPGA platform. The two-level unrolled loops are implemented as concurrently executing computation engines and a tree-shaped poly structure is used. For the best cross-layer design case, the computation engine is implemented as a tree-shaped poly structure with 9 inputs from input feature maps and 9 inputs from weights and one input from bias, which is stored in the buffers of output feature maps. This architecture consists of two parts: a data access system and a PE array. The data access system includes two parts, namely, a DDR3 controller and a Cache IP. The DDR3 controller is used to exchange data between the DDR3 memory and the Cache IP. The Cache IP can provide a feature map, kernel, and weight to the function module. The PE array is the computation core of the accelerator and consists of function modules and reorganization buffers. Each PE is a pipeline architecture, the execution time between two adjacent PEs is the super-pipeline cycle. The controller is not a part of the accelerator and is used to interact with the CPU and accelerator.

C. Design of the HW-SW platform

The rapid evolution of system-on-a-chip technologies has created the need for hardware-software co-design since these two constituent elements (HW-SW) of modern embedded systems can no longer be treated separately. This forms an important gap in existing methodologies, since the designer would like to be able to evaluate a number of alternative architectures, before committing to a specific one.The design of an embedded system using a co-design approach, involves a series of actions that must be followed [13] In hardware-software co-design approaches, the whole development cycle should be based around a single model.This model evolves during the various design stages from the initial informal conceptualization of the user's requirements to the final implementation-level detailed description of the system. The next step is to refine the formal system specification so that all details - including implementation decisions - are contained in the system model. Finally, the emerged system model is translated to implementation languages like C, C++, Java etc. for software and VHDL, Verilog, Hardware C etc. for custom hardware. Our system does this by using a hybrid of layer and model parallelism together with a number of new workload/weight balancing strategies. a single on-the-fly reconfiguration is needed: each configuration computes certain layers, or a part of a single layer; each device is optimized independently with respect to its own computation.

We find this approach to be effective with performance similar to that of GPU clusters of similar size and technology,but with far better power efficiency. The limiting factor is inter-FPGA bandwidth. The framework for mapping CNN logic to distributed FPGA clusters that achieves both high efficiency and scalability; that does not suffer from issues related to mini-batch size; and that needs only a simple interconnection network as is available in any multi-FPGA system with consistent communication and reasonable bandwidth. To ensure the connection, we will use the PIO (Parallel I/O) component from Platform Designer. When adding the component, we get

to choose the direction and the width of the register, also the base address of PIO component which is very important. The ARM program will access the component according to this base address. The ARM program development will make use of this address and a given Linux shell batch file will help extract the address information to a header file hps_0.h.

Finally, we need to integrate the SoC design with our P2IP using Verilog code to instantiate the core; The Verilog code generated and modified still has to be compiled into a bit-stream for the FPGA. With the schedule and resource allocation already fixed and all timing constraints properly set, there is not much left to be configured in Quartus itself. However, the timing results reported by the Quartus can be quite different from the estimates reported by the simulation in ModelSim.

D. ARM program development

With all these steps done, the FPGA side of the CNN accelerator is complete. However, there is still a missing key component: The CPU-side software which will run the rest of the CNN layers and configure the P2IP at each step . This subsection introduces how to design an ARM C program to control the CNN FPGA-Accelerator. Altera SoC EDS is used to compile the C project. For ARM program to control the P2IP component, the registers addresses are required. The Linux built-in driver '/dev/mem' and mmap system call are used to map the physical base address of P2IP component to a virtual address which can be directly accessed by Linux application software.

VI. RESULTS

A. P2IP CNN FPGA Accelerator Performance

The P2IP CNN FPGA Accelerator is meant to be a proof-of-concept for the implementation of CNNs on the basis of a systolic CGRA for image and video processing on FPGA. The secondary goal targets a maximum throughput on the given small and low-power platform, and in consequence a good power efficiency. This section evaluates the finished design with regard to the factors resource utilization, accuracy and the throughput of the accelerator. Finally, a number of potential architectural optimizations are highlighted.

The P2IP Embedded CNN has been completely assembled and successfully taken into operation on a DE1 SoC Board. The full test system consists of :

- HPS (ARM Cortex9 with 1 GB DDR3 memory), running under Linux4.
- MNIST CNN network description and trained weights, copied to the memory
- P2IP CNN FPGA Accelerator bitstream, loaded into the FPGA fabric
- P2IP CPU-side application, feeding the input images, launching the FPGA accelerator, measuring the timing and checking the classification results.

Using the above system configuration, the P2IP Embedded CNN has been evaluated in a realistic embedded scenario.

B. Throughput and Latency

The embedded CNN's throughput is measured in terms of images per second. In a typical scenario, the CNN accelerator is configured with the network description and the trained weights beforehand, and is then utilized to classify an incoming stream of images. Therefore,the run-time per frame is measured from the moment when the FPGA accelerator is started,to the moment when the calculation of the Softmax Classification layer is finished. In the P2IP, each PE can have a different latency according to its configuration.

The NE latency (NEL) for a neighborhood window with m × n pixels in an image with dimensions M × N pixels, can be expressed as defined here :

$$NEL = N(\frac{m-1}{2}) + \frac{n+1}{2} + b$$

where N is the number of pixels in a image line, m is the number of lines in a neighborhood window, n is the number of pixels per neighborhood line, and b is the border handler latency. The Convolver Module latency for a 3x3 filter is calculated as 9 pixel clock cycles and the Reconfigurable Interconnection latency as 6 pixel clock cycles.

C. Accuracy and Error

the Keras Python CNN model scored a 94% accuracy in the test. To put the P2IP CNN model to test and confirm these numbers, we extracted the feature maps of each layer and did a similarity comparison with the Keras model. This was achived using the SSIM algorithm.

The SSIM was first introduced in the 2004 IEEE paper [14] , it was introduced as an alternative complementary framework for quality assessment based on the degradation of structural information. The Structural Similarity Index (SSIM) metric extracts 3 key features from an image: Luminance, Contrast, Structure. The comparison between the two images is performed on the basis of these 3 features. This was all implemented in Python and the scores were ranging from 0.81 to 0.68 in terms of similarity.

D. Resources Analysis

Regarding the amount of logic required by the P2IP architecture, the Table I presents the resources required by the proposed implementations. Note that since all devices from the "V" series share the same internal architecture,the resources utilization is similar for both boards.

VII. DISCUSSIONS

A. Comparison with State-of-the-Art Architectures

we confine ourselves to a summary of the most important characteristics in this section. To start with, table II repeats the comparison of the different CNN topologies, and this time includes the P2IP CNN and its key parameters. All the parameters from our CNN, were calculated based on the Pyhton-Keras model.

The comparison presented in this section is concentrated on programmable architectures that target embedded systems, performing image classification.

Components	Adaptive Logic Module	Memory (kbits)	DSP
Control and Interface			
Controller	151	0	0
Input Register	15	up to 256 MB	0
Output Register	12	up tp 256 MB	0
Total	178	-	0
PE			
PE-CD	38	0	0
Spatial processor	1221	2.3	14
Memory Controller	1838	131.1	0
Reconfigurable Interconnection	211	0	0

TABLE I
P2IP CNN RESOURCES REQUIREMENTS ON FPGA DEVICES FROM THE ALTERA CYCLONE"V" SERIES

	Conv Layers	MACCs (millions)	params (millions)	Error
P2IP CNN	Up to 20	530	1.1	19%
AlexNet	5	1140	62.4	19.7%
VGG-16	16	15470	138.3	8.1%
GoogleLeNet	22	1600	7.0	9.2%
ResNet-50	50	3870	25.6	7.0%
SqueezeNet	18	860	1.2	19.7%

TABLE II
COMPARISON OF P2IP CNN TO CNN ARCHITECTURES FROM PRIOR WORK.

the P2IP CNN has an evident advantage over the CPU implementation. The P2IP speedup factor for the CPU implementation varies from 24 to 29. Regarding the GPU implementation, the P2IP CNN presents an advantage on almost all resolutions. The GPU tends to have an increasingly better performance in function of the image resolution, while the P2IP has a constant performance.

In addition to its competitive performance, the P2IP can still offer portability and much lower power consumption when compared to GPUs and CPUs. The P2IP solution consumes 50 times less power than the CPU implementation and 90 times less power than the GPU implementation.

B. Potential Improvements

An architectural bottleneck can be seen in the prefetching of image pixels from the image cache. Although this task is executed in parallel to the actual output channel calculation to hide the prefetch latency, the current delay is relatively long. The architecture of the image cache might need to be improved to allow for more parallel read accesses, or a register-field might be used to cache the active image patch. This should be viable as the image cache occupies less than 8 % of the Block ROMs, and a total of 300 k flip-flops are still unused. An ideal image cache would have a latency of less than 5 clock cycles, which would result in a speedup factor of 1.4.

Also, 5×5 convolutions are currently not implemented efficiently: two 3×3 MACC units are used for the necessary multiplications. The potential overall speedup from utilizing all 18 multipliers in the MACC units for individual 5×5 convolutions is approximately 1.2 with the current prefetch latency of 9 clock cycles. With an ideal Image Cache, a speedup factor of nearly 1.5 could be achieved.

VIII. CONCLUSION

In this article, we demonstrated the design and implementation of a proof-of-concept FPGA-accelerated embedded Convolutional Neural Network. The P2IP CNN is designed for image classification and consists of two main components: a highly optimized and customized CNN FPGA topology, and the P2IP CNN HW-SW platform, a FPGA-based architecture for configuring and running the FPGA accelerator. The P2IP Embedded CNN has been assembled into a fully working proof-of-concept system on both the DE-1 and the SoCkit board, All Programmable platforms. This project clearly demonstrates the feasibility of FPGA-based embedded CNN implementations. The current solution already exhibits a reasonable performance, and a number of opportunities for further gains in throughput and power efficiency have been pointed out.

The tough requirements of embedded CNNs regarding the size, efficiency and computational power of the underlying computing platform are very hard to meet with the systems available today. Even though the presented P2IP CNN does not yet provide the massive amounts of computational power required for future applications of embedded image understanding, it may still serve as a steppingstone and a guide for further explorations of the FPGA as a platform for embedded CNNs. The biggest advantage of these FPGA-based systems can be seen in their scalability. Using a larger device, much higher performance can be attained at comparable efficiency figures,while most other platforms are inherently limited by the amount of computational power available on a given chip. FPGAs therefore provide a promising path towards the vision of powerful embedded CNNs and the abundance of fascinating applications which could profit from on-board image understanding

REFERENCES

[1] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115–133,, 1943.
[2] A. Karpathy. Surpassing human-level performance on imagenet classification, 2014.
[3] WikiPedia. Graphics processing unit, 2013.
[4] Magnus Halvorsen. Hardware acceleration of convolutional neural networks. MS thesis, Norwegian University of Science Technology, 2015.
[5] haran Chetlur. Cudnn: Efficient primitives for deep learning. arXiv: 1410.0759, 2014.
[6] Alex Krizhevsky. Cudaconvet2, 2013.
[7] Yangqing Jia. Caffe: convolutional architecture for fast feature embedding. International Conference on Multimedia, 2014.
[8] Tensorflow, https://www.tensorflow.org/ 2014.
[9] cs231n. Convolutional neural networks for visual recognition, 2016.
[10] D. Gschwend; C. Mayer; S. Willi. Design and implementation of a convolutional neural network accelerator asic. Semester Thesis, ETH Zürich, 2015.
[11] Keiron O'Shea; Ryan Nash. An introduction to convolutional neural networks, 2015.
[12] ARM , AXI4 , https://www.arm.com, 2010
[13] F. Vahid D. Gajski. Specification and design of em-bedded hardware-software systems. IEEE Designand Test of Computers, vol.12. pp.53-67, 1995.
[14] Zhou Wang . Alan Conrad Bovik . Hamid Rahim Sheikh. Image quality assessment: From error visibility to structural similarity. IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 13, NO. 4, 2004.