Accelerating Deep Learning Models on Field-Programmable Gate Arrays

Bilal Ayyache, Robert Mackenzie Beggs, Kyle Schnarr, and Bayley Walsh

March 2020

1 Abstract

Artificial intelligence (AI) and its derivatives are more than ever considered a viable option for solving complex problems with previously intractable quantities of parameters due to the availability of large datasets in diverse areas coupled with hardware advancement. Areas that utilize generative models, classification, natural language processing as well as many others have seen unprecedented performance benefits from the widespread development of artificial intelligence, however, these algorithms are memory intensive and require dedicated hardware to train [14, 8, 10]. Current hardware solutions to train these algorithms include individual and clustered instances of central processing units (cpus) and graphics processing units (gpus). More recently, hardware with the express purpose of employing neural networks, called tensor processing units (tpus) have been employed. In addition to the expense associated with training, additional expense is incurred when considering deployment. The large variability in price and form factor make both employing and training neural networks via field-programmable gate arrays (FPGAs) an alternative worthy of further investigation[8, 10, 11, 5, 14].

2 Introduction

Connectivity to the internet of things (IOT) is at an unprecedented high and will only continue to grow. By the end of 2020, it has been proposed that 400 Zettabytes of data will be generated by 50 billion devices [6, 1]. This quantity of data provides unparalleled opportunity for problem solving via use of AI algorithms. The impact of AI on everyday life has been comprehensive and profound. Just on phones, the impact of these algorithms is undeniable. Facial recognition to unlock a phone, autocomplete in text and email, and low light photo processing are all daily quality of life improvements birthed by AI. The fields required for these improvements, computer vision, natural language processing and generative models respectively all have benefited greatly by the aggregate of accessible datasets [8].

Furthermore, it has been proven that there is a relationship between the size of the training set and the effectiveness of the model [13, 9, 12]. The cost of training models on increasingly large datasets is a barrier to accessible use. The adjustable structure of FPGAs allow for a more efficient design for specific algorithms in which operations usually execute in fewer clock cycles, despite typically slower clock speeds as compared to fixed Von Neumann, Harvard or modified Harvard architectures found in CPU and GPU designs [5]. Utilization of this characteristic in addition to the ability of parallelization allows for the possibility of computing advantage when applying matrix operations inherent in machine learning [5].

Our proposed outline is as follows:

- Provide a background on the composition of AI algorithms.
- Present an in depth case study on acceleration.

3 Background

3.1 Convolutional Neural Networks (CNNs))

3.1.1 Convolution

To produce an output value, a kernel of weights computed during training slides over a matrix containing the original image with the goal of creating a representative matrix of smaller size [15]. Elementwise multiplication is completed between the kernel values and the original array as the filter progresses over the original array, and the values are summed. This operation starts at the upper left hand corner, progresses horizontally to the upper right hand corner, returns to the starting position, moves downwards then repeats the routine until the window rests in the lower right hand corner [10, 7, 15]. Convolution comprises of four main behaviours summarized below in terms of programming loops:

- Multiply and accumulate via kernel
- Scan feature map along a single dimension
- Perform convolution at each input feature map
- Loop across each output feature map

It is important to optimize this operation as it has been estimated that in CNNs convolution represents 90% of the operations[15].

3.1.2 Activation

Activation functions are used to determine whether a neuron should output a value. An example of an operation is the sigmoid function, $f(x) = \frac{1}{1+e^{-x}}$. Large negative values of x approach 0, while positive values approach 1. This activation makes the function ideal for many applications including classification [15, 10, 7].

3.1.3 Normalization

Normalization is used to temper the activation of neighborhoods of neurons to dampen uniform reactions and create contrast to ensure only relatively large activations are recognized [15].

3.1.4 Pooling

The size of the image is continuously reduced by pooling layers, inserted between consecutive convolution layers. This decrease corresponds to a decrease in the quantity of parameters. The amount of parameters corresponds to computation costs [11, 15].

3.1.5 Fully Connected Layers

For the purpose of classification, fully connected layers are typically the final output layers. Each neuron in a fully connected layer shares a connection with each neuron in the previous layer[11, 15, 7].

4 Design

The common design principle for the use of neural networks on FPGAs are as follows; use of a convolution unit, use of a data cache, and memory and computational optimization.

4.0.1 Convolution Unit

Within neural networks convolution operations are the most performed operations on a FPGA making up to 90% of the computations [15]. Within a hardware design the use of a convolution unit allows greater temporal locality between subregions as the data will be constantly called upon thus improving the overall computing performance of a neural network. Although due to on-chip resource limitations the computations are fed to buffers before being sent to processing elements.

4.0.2 Data Cache

The use of data cache splitting allows for an increase the throughput of the neural network's accelerator as it increases the number of input-output interfaces on the FPGA board, increasing the parallelism of the neural network implementation. Although this decreases the total memory each block of RAM has.

4.0.3 Memory and Computational Optimization

Performing memory and computational optimizations to the code of a FPGA neural network is an important step in the design of a neural network on hardware as it will allow for the increase of total throughput of the system, doing optimizations such as loop unrolling will increase the speed of the system but at the cost of more hardware usage. The optimizations to memory will increase the throughput of the system due to reducing the number of times the neural network system needs to access off-chip memory. Although most of the optimizations include using more hardware components which will cause an increase in price otherwise there is only so much optimization you can do with already build FPGA.

5 Case Studies

There have been many uses of FPGA's to accelerate the neural network process, mostly through the optimization of calculations of the convolution layer math.

5.1 Convolutional Network Processor

A group's research from 2009, [3] implemented the unique structure of FPGA's that include lots of multiply-and-accumulate (MAC) modules. These MAC modules make low-level calculations needed in CNN's fast and power efficient. Creating a unique processing module called a convolutional network processor (CNP). The CNP uses the parallelism between convolutional layers to increase the computation time of convolutional neural networks (CNN). The interface used allowed the data transmission between external memory and the FPGA 8 channels for read/write communication. The CNP allowed kernel size squared MAC calculations during each clock cycle.

5.2 MAPLE

MAPLE,[2] a FPGA board created for accelerated learning of CNN's, makes parallel streams of vector and matrix calculations. By allocating separate memory to multiple processing elements each holding a column of a matrix. Clusters of these columns are then multiplied by the rows and stored onto the a memory bank off-chip. This allows multiple streams of communication between memory and computation. MAPLE also implements in-memory processing to reduce the off-chip memory traffic. This allows the MAPLE system to process 50% faster than a NVIDIA implementation.

5.3 Neural Network Next

Neural network next (nn-X) [4] is a SoC computing system that utilizes a host processor, co-processor and external memory. The co-processor creates collections of PE's that utilizes a pooling module and a non-linear operator to accelerate computation. The PE pipelines the data using cache memory and communicates between other PE's to reduced the number of external memory traffic. This resulted in 115 times the computation speed than 2 embedded ARM processors using a face recognition model.

All of these implementations of FPGA acceleration involve the use of the parallelism of the CNN's and the unique ability of the MAC units to calculate efficiently to reduce the matrix calculation timing. They also aim to reduce the traffic between external memory and the FPGA. Only through the use of FPGA's would these methods be possible.

6 Conclusion

We aim to provide a background on the history of AI integration on FPGAs, detail common design principles of CNN acceleration algorithm implementation and present in depth case studies of specific implementations providing the details of each while considering current hardware limitations. We discover that specifically for convolution there is room for optimization. As seen, there are many ways to utilize FPGA's to take advantage of this non-optimized convolution.

References

[1] Saman Biookaghazadeh, Fengbo Ren, and Ming Zhao. "Are FPGAs Suitable for Edge Computing?" In: (Apr. 2018).

[2] Srihari Cadambi et al. *A programmable parallel accelerator for learning and classification.* 2010.

[3] Clement Farabet et al. *An FPGA-based processor for Convolutional Networks.* 2009.

[4] Vinayak Gokhale et al. *A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks.* 2014.

[5] Yufeng Hao. *A General Neural Network Hardware Architecture on FPGA.* 2017. arXiv: `1711.05860 [cs.CV]`.

[6] Mohsen Imani et al. "FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision". In: *Proceedings of the 46th International Symposium on Computer Architecture.* ISCA '19. Phoenix, Arizona: Association for Computing Machinery, 2019, pp. 802–815. ISBN: 9781450366694. DOI: `10.1145/3307650.3322237`. URL: `https://doi.org/10.1145/3307650.3322237`.

[7] Asifullah Khan et al. *A Survey of the Recent Architectures of Deep Convolutional Neural Networks.* 2019. arXiv: `1901.06032 [cs.CV]`.

[8] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. *Deep Learning on FPGAs: Past, Present, and Future.* 2016. arXiv: `1602.04283 [cs.DC]`.

[9] Trond Linjordet and Krisztian Balog. *Impact of Training Dataset Size on Neural Answer Selection Models.* Jan. 2019.

[10] Ahmad Shawahna, Sadiq M. Sait, and Aiman H. El-Maleh. "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review". In: *IEEE Access* 7 (2019), pp. 7823–7859.

[11] Naveen Suda et al. "Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale Convolutional Neural Networks". In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* FPGA '16. Monterey, California, USA: Association for Computing Machinery, 2016, pp. 16–25. ISBN: 9781450338561. DOI: `10.1145/2847263.2847276`. URL: `https://doi.org/10.1145/2847263.2847276`.

[12] Chen Sun et al. *Revisiting Unreasonable Effectiveness of Data in Deep Learning Era.* 2017. arXiv: `1707.02968 [cs.CV]`.

[13] Antonio Torralba and Alexei A. Efros. *Unbiased Look at Dataset Bias.* 2011.

[14] Sheping Zhai et al. "Design of Convolutional Neural Network Based on FPGA". In: *Journal of Physics: Conference Series* 1168 (Feb. 2019), p. 062016. DOI: `10.1088/1742-6596/1168/6/062016`.

[15] Chen Zhang et al. “Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks”. In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* FPGA ’15. Monterey, California, USA: Association for Computing Machinery, 2015, pp. 161–170. ISBN: 9781450333153. DOI: `10.1145/2684746.2689060`. URL: `https://doi.org/10.1145/2684746.2689060`.