ETH ZÜRICH

MASTER'S THESIS

A Deep Learning Library for FPGAs using OpenCL

Author:
Houssam NAOUS

Supervisor:
Dr. Torsten HOEFLER

A thesis submitted in fulfillment of the requirements for the degree of Masters of Science in Computer Science

in the

Scalable and Parallel Computing Lab
Computer Science

August 23, 2018

Declaration of Authorship

I, Houssam NAOUS, declare that this thesis titled, "A Deep Learning Library for FPGAs using OpenCL" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

ETH ZÜRICH

Abstract

Computer Science
Computer Science

Masters of Science in Computer Science

A Deep Learning Library for FPGAs using OpenCL

by Houssam NAOUS

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

Declaration of Authorship **iii**

Abstract **vii**

Acknowledgements **ix**

1 Introduction **1**
1.1 Deep Learning and Applications 1
1.1.1 History 1
1.1.2 Learning in Artificial Neural Networks 1
1.1.3 Deep Learning 2
1.2 Modern FPGAs 2
1.2.1 The Case for FPGAs 2
1.2.2 FPGAs vs GPUs 2
1.3 High-Level Synthesis and OpenCL 3
1.3.1 FPGA Workflow 3
1.3.2 High-Level Synthesis 4
1.3.3 Xilinx and Intel 4
1.3.4 OpenCL for FPGAs 4
1.4 Related Work 5
1.5 Motivation 6
1.6 Outline of Next Sections 6

2 Introduction **7**
2.1 7

A Frequently Asked Questions **9**
A.1 How do I change the colors of links? 9

Bibliography **11**

Chapter 1

Introduction

1.1 Deep Learning and Applications

1.1.1 History

The first artificial neural network traces back to 1958 where it was first conceived by psychologist Frank Rosenblatt [21]. It was called the perceptron and it was meant to model the way a human brain adapts to inputs from the external world to learn binary classification tasks. At some point someone realized that this model could be useful in pattern matching tasks. The artificial neural network is organized into layers of a single threshold logic unit that models a single neuron in the human brain. In the early days, these models were constructed physically and later on were simulated on a single computer [17]. Nowadays, learning tasks are distributed and coordinated on multiple machines to achieve a single learning task [7]. The primitive perceptron developed into a structure of layers organized and separated by non-linearities. In theory, the "Universal Approximation Theorem" states that a multi-layer perceptron with one hidden layer containing a finite number of neurons can approximate any continuous function under some assumptions on the activation function [5].

1.1.2 Learning in Artificial Neural Networks

Towards the end of 1986, Hinton's paper titled *"Learning representations by back-propagating errors"*[22] was published and it introduced the usefulness of an algorithm called back-propagation that can train an artificial neural network that is organized into layers. It proved more useful than the previously know perceptron-convergence algorithm [29] and by the end of the 1980s many scientific institutes adopted the use of neural networks and utilized them to solve many tasks [19]. Unlike standard algorithms that rely on conditional procedures and hand-crafted logic, the artificial neural network if designed properly is robust to noise and can adapt to those pattern matching tasks [28]. Artificial neural networks are exposed to thousands or millions of that are forward propagated through the weights in each of the layers. In-between each layer non-linearities are introduced and the output is compared to a specific target encoding of the output labels. From that a loss can be calculated and using a process called backpropagation [22], starting with the output layers, the network readjusts its weights to better match the target output, specifically reinforcing the connections that contribute to a correct output label.

1.1.3 Deep Learning

As computing resources were cheaper and more available and after the numerous improvements in computer hardware and architecture, scientists were able to simulate more complex networks with more neurons and deeper layers [26]. In fact, it was even proved that deeper networks with less neurons per layer proved more useful than the shallow networks [25], thus the concept of deep learning was popularized. Deep learning is only a subset of the broader concept of machine learning which consists of supervised, semi-supervised, and unsupervised learning tasks. It has proven itself useful in applications related to computer vision, speech, recognition, finance, and many others. The hype over deep learning increased even more when these networks were trainable on Graphical Processing Units (GPUs) which are capable of performing floating point operations on hundreds and thousands of cores in parallel [20]. This kind of parallelization decreased training time for these networks drastically and soon enough the suitable frameworks were developed and popularized [1, 3]. Researchers were then able to utilize those hardware for training neural networks with more data and experiment with more sophisticated network models and architectures [11, 20, 26].

1.2 Modern FPGAs

1.2.1 The Case for FPGAs

The market for Field-Programmable Gate Arrays (or FPGAs) has been increasingly growing and is expected to reach $12.98 billion by 2023 with a compound annual growth rate of 9.0% [1]. The demand for FPGAs was sparked by the need for high-throughput and low latency applications in industries such as aerospace, finance, and security. FPGAs are integrated circuits that are manufactured in a way such that they can be configured after production [4]. Using hardware descriptive languages (HDL) a hardware engineer can build a specific circuit and transfer it to the FPGA where it can reconfigure itself and rewire to implement a given circuit design. They offer a cheaper alternative to high-performing and specialized ASICs as they require less recurring engineering/manufacturing costs and less time to market which is necessary to thrive in this fast-paced economy. They offer a whole new dimension of customization in which complex instruction pipelines can be designed and implemented as opposed to the fixed instruction set architecture of a microcontroller or a generic CPU [14].

1.2.2 FPGAs vs GPUs

Application scientists have favored in the last couple of years the use of GPUs to accelerate deep learning tasks [20]. The GPUs architecture lends itself perfectly to perform parallel floating point computations needed to compute and train the networks. Moreover, what has lead to the GPUs more widespread use is a well defined programming model and tool-sets that are easily adopted by software programmers [6]. Dealing with those needs minimal experience in hardware architecture and applications have been parallelized and scaled massively. FPGAs on the other hand can offer higher power efficiency for the same computational workload as that of a GPU

[1]`https://globenewswire.com/news-release/2017/11/29/1210234/0/en/Global-Field-Programmable-Gate-Array-FPGA-Market-to-Reach-USD-12-989-1-million-by-2023.html` Last Accessed : 22/08/2018

and are intrinsically parallel devices [18]. However it's speed has not yet caught up with it's accelerator counterpart. Power efficiency is a major concern for large-scale applications operating in data centers. For that we have seen most recently both Microsoft [2] and Amazon [3] have incorporated FPGAs into their existing cloud computing infrastructure both for internal use against their data and as a service offered to clients wishing to utilize the power of reconfigurable architectures. Even though FPGAs up to date have only proven to be more power efficient than GPUs [14, 18], simulations and projections done at Intel Corporation predict that the upcoming generation of FPGAs will also compete with GPUs in terms of performance [18]. The projections show that the new Intel Stratix 10 is estimated to achieve 60% higher performance and 2.3 times less power consumption than the Titan X GPU. It is also important to note that the future of deep neural networks (DNNs) have resorted to fixed point computations and lowered precision up to the point of binary values as in Binarized Neural Networks [10]. All of these innovations have lead to irregular types of parallelism in which FPGAs excel at compared to GPUs. GPUs can only operate on a fixed set of data types and thus the trend towards lowering precision tips the scale of performance towards FPGAs [10, 23, 18]. The gap in performance between FPGAs and GPUs is getting smaller and thus it is necessary to update the toolset and design workflows in designing FPGA applications to keep up and make them more accessible for developers to be able to experiment and test.

1.3 High-Level Synthesis and OpenCL

1.3.1 FPGA Workflow

One of the main reasons that has lead to the slow adoption of FPGAs into commercial applications is the steep learning curve and technical background in hardware design required to be able to design and deploy applications. The usual workflow differs from that of a typical CPU application, however analogies can be drawn to bridge the gap between the two classes of applications. While designing a CPU application starts with a high-level programming language like C++ or Python, designing an FPGA circuit requires the use of hardware descriptive languages. The two most popular hardware descriptive languages are Verilog and VHDL [30]. Some FPGA design tools also offer the designer graphical user interfaces to draw schemas by dragging and dropping boxes. The latter, however, doesn't scale for larger projects and makes it harder to collaborate within a team. HDLs are dataflow programming languages that allow for broader descriptions of how digital circuits can communicate and execute logic in ways where other procedural languages like C fall short. An FPGA application designer also has additional stages in the design cycle where behavioural simulation as well as timing simulations and functional simulations of the design. After that there is a synthesis, placement, and fitting steps in which the designer hands his design to a piece of software that performs optimizations and tries to materialize the design in terms of a binary file which can be loaded onto the FPGA. Following that the developer runs also more tests on a development board and makes sure to fix any remaining bugs or errors. Another difficulty is that FPGA designs are not portable and thus certain parameters always need to be tuned to adapt to different boards with varying configurations. Each FPGA comes with a

[2]https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/ Last Accessed: 22/08/2018

[3]https://aws.amazon.com/ec2/instance-types/f1/ Last Accessed: 22/08/2018

different operating specification and different resource blocks [4], so in order to maximize utilization, the developer is expected to customize their design for each and every board.

1.3.2 High-Level Synthesis

High-Level Synthesis (HLS) is not a new invention at all. It has always co-existed with FPGAs. From initial research into HDLs, HLS tools have advanced into C-based dataflow programming paradigms nowadays. We see the level of abstraction rising from gate level, to register-transfer level, into algorithmic level synthesis [9, 16]. HLS is mainly motivated by the need to abstract hardware design to application programmers who should focus on designing and optimizing algorithms regardless of the underlying hardware. This isolation leaves hardware designers with the responsibility of optimizing intermediate representations of algorithms into synchronized logic blocks. Modern HLS tools begin by compiling the input specifications. Many code optimizations like code folding and dead-code elimination are carried-out to get a near-optimal input specification [9]. A control dataflow graph (CDFG) is created that parses the specification into a graph formed of nodes which are the basic blocks and connections that represent control dependencies between those blocks. The results is a register-transfer level (RTL) representation. It consists of a datapath (memory elements, interconnects, and functional units) and a control path. The controller is a finite-state machine that coordinates the flow of elements and operations on the datapath. The RTL representation is then verified to make sure it meets timing constraints and transformed into a gate-level representation that is then synthesized onto an FPGA according to a board specification file.

1.3.3 Xilinx and Intel

We have briefly motivated the main goal of HLS in abstracting the process of hardware design and mapping onto FPGAs. The two main manufacturers and technology leaders in the FPGA market are Xilinx and Intel (after the acquisition of Altera in 2015). Both companies have shifted their tool-sets to favor higher-level abstractions for synthesis but they have taken slightly different approaches. We note that Xilinx's development workflow favors more the hardware engineer by giving more control for them to view/modify designs and control most of the nuts and bolts that transform their applications into hardware [31]. Altera's tools however favor the software developer wishing to leverage hardware accelerators to achieve higher throughput for their application. The Intel FPGA developer is provided with a programming manual that suggests code improvement so that a better hardware design is generated. Both companies have adopted the use of HLS tools that can transform code in behavioural C/C++ and OpenCL descriptions into bitstreams [31, 12].

1.3.4 OpenCL for FPGAs

OpenCL^{TM}(Open Computing Language) is the open standard for cross-platform parallel programming that is a subset of the C standard [24]. OpenCL provides a programming model that fits the GPU architecture perfectly and is able to exploit parallelism through vectorized operations. GPUs are able to perform vectorized operations and have higher memory bandwidth than CPUs, enabling them to achieve

[4]`https://www.intel.com/content/www/us/en/fpga/devices.html` Last Accessed 23/08/2018

high throughput by doing parallel floating point operations. The challenge in adopting OpenCL for FPGAs is being able to not only vectorize operations but to also create efficient pipelines that fully utilize the resources available on the board. For that the Intel OpenCL SDK [12, 14] performs those optimizations and allows the user to use pre-defined pragmas in order to adapt OpenCL for FPGAs. This again beats the goal for portability across platforms when different customizations have to be introduced for FPGAs, however pipeline parallelism and complex data-flow instructions prove to be an advantage and a necessary feature that should be exploited on FPGAs [2, 14]. It is also worth noting that effort for optimizing the same OpenCL kernel differs between FPGAs and GPUs differs a lot putting FPGAs at a slight disadvantage. The reason is that with GPUs, the developer looks for the best mapping into the fixed architecture, while for FPGAs the developer guides the compiler into finding the best control and memory architecture for the given task. Moreover, compiling OpenCL kernels for FPGAs takes much longer than on GPUs as more board-specific optimizations can be done and because FPGAs allow for a broad design-exploration space.

1.4 Related Work

An implementation called "PipeCNN" [27] using OpenCL has explored the power of pipelining neural network layers to lower the overall memory bandwidth requirement in between network layers. The implementation was able to achieve a maximum throughput of 12.8GB/s on the Altera Stratix-5 board. The implementation however only covers convolution layers and fully connected layers. A single matrix-multiplication based kernels implements both of the convolution and fully connected layers and pipelines them with a pooling operation. In this implementation, full inter-layer communication is done by communicating through global memory. Local response normalization (LRN) which is done after the pooling layer is not pipelined directly and also communicated through global memory. The implementation, even though it utilizes the full memory bandwidth of the board, requires a lot of overhead for inter-layer communication and can be further reduced by pipelining more layers together.

The work of DiCecco et. al [8] utilizes the Xilinx SDAccel [5] toolset for optimizing neural network designs. They implement an FPGA backend for the popular neural network framework Caffe [13].This methodology makes use the already existing testbenches in Caffe for verifying correctness of the FPGA implementations. The authors run experiments using modern deep neural nets such as Alexnet, GoogleNet, and VGG-16 and achieve a maximum throughput of 50 GFLOPS across the 3x3 convolutions on the Xilinx Virtex 7. The authors also implement convolution using the Winograd [15] minimum filtering algorithm. This filtering scheme minimizes the number of multiplications required due to overlapping intermediate results in overlapping filter computations. The work lays the stepping stones and mentions the challenges in integrating FPGAs as accelerators for popular DNN frameworks like Caffe such as long reprogrammability times (100-400ms for FPGA vs 0,001ms-0.005ms for a GPU) and thus different types of parallelism should be exploited to fill in the gaps. FPGA implementations also require offline compilation and several

[5] /urlhttps://www.xilinx.com/products/design-tools/software-zone/sdaccel.html Last Accessed: 23/08/2018

vendor specific attributes to achieve optimal performance. The results are not impressive showing that GPU performance is still higher and the framework is still far from integrating the different layers of a DNN other than convolutions.

Zhang et. al.[32] were able to achieve 61.2 GFLOPS under 100 MHz on a VC707 FPGA. The main contribution of this work is an analysis framework which takes into consideration both the computing resources and the memory bandwidth provided by the board to guide the design space exploration phase. They balance out loop unrolling factors and loop tiling methods to balance the tradeoffs of using compute resources and memory bandwidth. The work however only focuses on the inference phase and lacks the analysis of the training phase of a CNN which could take days or weeks for a single learning task. They also focus on single layer optimizations for and not on multiple layer optimizations. This is important as compute intensive layers like convolutions can be balanced out with bandwidth hungry layers like the fully connected layers to achieve better performance.

1.5 Motivation

This work aims to leverage the benefits of OpenCL for programming FPGAs and target implementations of modern deep neural networks. The field has gained a lot of traction and so far the support for FPGA backends for accelerating neural network computations are still research based and experimental. It is also becoming increasingly important to utilize FPGA's configurable circuits for latency-sensitive and real-time DNN applications such as autonomous driving. By using FPGAs, neural networks can be accelerated and energy efficient by utilizing pipeline parallelism. Asides from that, an additional goal is not only to create networks for inference but to also accelerate training of deep neural networks on FPGAs. For that we use the Lenet network as a case study and proof of concept and implement minibatch gradient descent for training this network. We also aim to bridge the gap between research and application, so we have created a framework in Python that is able to go from open source model definitions like ONNX into material FPGA implementations. The development framework is easily extensible with modular components that also allow for individual customization and research into novel ways of accelerating the layer computations, backpropagation, and full network pipelining as a whole.

1.6 Outline of Next Sections

- Chapter 2 explains the algorithms and terminology in deep learning.
- Chapter 3 explains the toolset and hardware implementation of deep neural networks on FPGAs.
- Chapter 4 analyzes and discusses the results of the experiments of running the implementation on the FPGA.
- Chapter 5 servers as a primer and best practices in using OpenCL for FPGAs.
- Chapter 6 contains the concluding remarks and future direction of work.

Chapter 2

Introduction

2.1

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:
`\hypersetup{urlcolor=red}`, or
`\hypersetup{citecolor=green}`, or
`\hypersetup{allcolor=blue}`.
If you want to completely hide the links, you can use:
`\hypersetup{allcolors=.}`, or even better:
`\hypersetup{hidelinks}`.
If you want to have obvious links in the PDF but not the printed text, use:
`\hypersetup{colorlinks=false}`.

Bibliography

[1] Martín Abadi et al. “Tensorflow: a system for large-scale machine learning.” In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Tal Ben-Nun and Torsten Hoefler. “Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis”. In: *arXiv preprint arXiv:1802.09941* (2018).

[3] James Bergstra et al. “Theano: Deep learning on gpus with python”. In: *NIPS 2011, BigLearning Workshop, Granada, Spain*. Vol. 3. Citeseer. 2011, pp. 1–48.

[4] Katherine Compton and Scott Hauck. “Reconfigurable computing: a survey of systems and software”. In: *ACM Computing Surveys (csuR)* 34.2 (2002), pp. 171–210.

[5] Balázs Csanád Csáji. “Approximation with artificial neural networks”. In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24 (2001), p. 48.

[6] C Cuda. *Programming guide*. 2012.

[7] Jeffrey Dean et al. “Large scale distributed deep networks”. In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.

[8] Roberto DiCecco et al. “Caffeinated FPGAs: FPGA framework for convolutional neural networks”. In: *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE. 2016, pp. 265–268.

[9] Rajesh Gupta and Forrest Brewer. “High-Level Synthesis: A Retrospective”. In: *High-Level Synthesis: From Algorithm to Digital Circuit*. Ed. by Philippe Coussy and Adam Morawiec. Dordrecht: Springer Netherlands, 2008, pp. 13–28. ISBN: 978-1-4020-8588-8. DOI: `10.1007/978-1-4020-8588-8_2`. URL: `https://doi.org/10.1007/978-1-4020-8588-8_2`.

[10] Itay Hubara et al. “Binarized neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 4107–4115.

[11] Forrest Iandola et al. “Densenet: Implementing efficient convnet descriptor pyramids”. In: *arXiv preprint arXiv:1404.1869* (2014).

[12] FPGA Intel. “SDK for OpenCL”. In: *Programming Guide. UG-OCL002* 31 (2016).

[13] Yangqing Jia et al. “Caffe: Convolutional architecture for fast feature embedding”. In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[14] Griffin Lacey, Graham W Taylor, and Shawki Areibi. “Deep learning on fpgas: Past, present, and future”. In: *arXiv preprint arXiv:1602.04283* (2016).

[15] Andrew Lavin and Scott Gray. “Fast algorithms for convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4013–4021.

[16] Grant Martin and Gary Smith. “High-level synthesis: Past, present, and future”. In: *IEEE Design & Test of Computers* 26.4 (2009), pp. 18–25.

[17] John McCarthy et al. “A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955”. In: *AI magazine* 27.4 (2006), p. 12.

[18] Eriko Nurvitadhi et al. “Can fpgas beat gpus in accelerating next-generation deep neural networks?” In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2017, pp. 5–14.

[19] Yohhan Pao. “Adaptive pattern recognition and neural networks”. In: (1989).

[20] Rajat Raina, Anand Madhavan, and Andrew Y Ng. “Large-scale deep unsupervised learning using graphics processors”. In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 873–880.

[21] Frank Rosenblatt. *The Perceptron : a theory of statistical separability in cognitive systems*. eng. United States Department of Commerce, 1958.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), p. 533.

[23] Suhap Sahin, Yasar Becerikli, and Suleyman Yazici. “Neural network implementation in hardware using FPGAs”. In: *International Conference on Neural Information Processing*. Springer. 2006, pp. 1105–1112.

[24] John E Stone, David Gohara, and Guochun Shi. “OpenCL: A parallel programming standard for heterogeneous computing systems”. In: *Computing in science & engineering* 12.3 (2010), pp. 66–73.

[25] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[26] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning.” In: *AAAI*. Vol. 4. 2017, p. 12.

[27] Dong Wang, Ke Xu, and Diankun Jiang. “PipeCNN: an OpenCL-based open-source FPGA accelerator for convolution neural networks”. In: *Field Programmable Technology (ICFPT), 2017 International Conference on*. IEEE. 2017, pp. 279–282.

[28] Kaining Wang and Anthony N Michel. “Robustness and perturbation analysis of a class of artificial neural networks”. In: *Neural networks* 7.2 (1994), pp. 251–259.

[29] Bernard Widrow and Michael A Lehr. “30 years of adaptive neural networks: perceptron, madaline, and backpropagation”. In: *Proceedings of the IEEE* 78.9 (1990), pp. 1415–1442.

[30] Peter Wilson. *Design Recipes for FPGAs: Using Verilog and VHDL*. Newnes, 2015.

[31] AXI Xilinx Vivado. *Reference Guide*.

[32] 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: ACM, 2015, pp. 161–170. ISBN: 978-1-4503-3315-3. DOI: 10.1145/2684746.2689060. URL: http://doi.acm.org/10.1145/2684746.2689060.