2017 IEEE 35th International Conference on Computer Design

Convolutional Neural Networks on Dataflow Engines

Nils Voss*†, Marco Bacis†‡, Oskar Mencer†, Georgi Gaydadjiev*†, Wayne Luk*
*Imperial College London
‡Politecnico di Milano
†Maxeler Technologies Ltd.
{nvoss, mbacis, oskar, georgi}@maxeler.com w.luk@imperial.ac.uk

***Abstract*—In this paper we discuss a high performance implementation for Convolutional Neural Networks (CNNs) inference on the latest generation of Dataflow Engines (DFEs).**

We discuss the architectural choices made during the design phase taking into account the DFE chip properties. We then perform design space exploration, considering the memory bandwidth and resources utilisation constraints derived from the used DFE and the chosen architecture.

Finally, we discuss the high performance implementation and compare the obtained performance against other implementations, showing that our proposed design reaches 2,450 GOPS when running VGG16 as a test case.

I. Introduction

Machine learning in general and Neural Networks (NNs) in particular have received a lot of attention in recent years. Many different publications focus on the possibilities of high performance hardware implementations for typical machine learning and NN tasks and show that significant speed ups and power efficiency improvements are achievable [1]–[3]. The above combined with the higher flexibility make this approach a very interesting choice for machine learning even when compared to ASIC implementations [4].

Especially applications of NNs for deep learning and CNNs have particularly demanding computational and sometimes even real time requirements, e.g., a HD live feed that should be processed at frame rate. For these reasons CNNs can benefit from high performance hardware implementation, especially when huge archives of video material are considered.

One of the main challenges in the development of high performance CNN implementations is to find the right balance between on-chip and off-chip memory resources, since usually it is not possible to store all weight parameters and the inter layer computation results on-chip for networks of practical relevant sizes. For this reason CNN implementations usually rely on external system DDR memory, which has limited total bandwidth, erratic bandwidth, unpredictable access latencies and significantly contribute to the system power consumption.

In this paper we will discuss an optimised architecture for a generic high performance CNNs implementation and perform a design space exploration based on a high level model.

The main contributions of this paper are as follows:

- a high performance CNN hardware design exploiting multiple levels of parallelism in deep learning networks;
- design space exploration using the proposed architecture, driven by the characteristics of the specific device;
- a test case showing VGG16 network implementation on the latest generation of DFEs.

The paper is organised as follows. Section II gives a background on Maxeler DFEs specific features. Section III lists the state of the art work relevant to our proposal. Our architecture is explained in detail in Section IV, while the specific optimisations used and the Design Space Exploration (DSE) are described in Section V. Finally, we compare our results against related work in Section VI and draw our final considerations in Section VII.

II. MaxJ and MAX5 DFE

MaxCompiler is a streaming data-flow driven design environment. The user describes the generation of a computational data-flow graph using a Java style language, called MaxJ. The graph is then automatically mapped onto a hardware platform.

One special characteristic of MaxJ is that it provides a good trade off between designer productivity and low level hardware control [5]. For example it is possible to create custom address command generators, which allow high efficiency DDR access, even for complex, non-linear data access patterns.

Additionally MaxCompiler provides bit-accurate simulation capability, and libraries which allow easy integration into CPU code. It is build to operate with DFEs [6], which combine large chips with big and fast memory and high-speed interconnects.

The latest, fifth generation MAX5 DFE consists of a large capacity arithmetic chip, three 16GB DDR4 DIMMs (also referred as Large Memory, LMEM), 16x PCIe gen. 3 connectivity, a 40 GBit/s QSFP network interface and is fully compatible with the Amazon F1 cloud instance.

Each MAX5 DFE has 6,840 dedicated multipliers and over two million registers. Additionally it has 38 MB of on-chip SRAM fast memory which we call FMEM. This memory has an aggregated memory bandwidth of over 20 TB/s

III. Related Work

Interest into high performance implementations of CNNs increased during the last few years. The most notable work is the Tensor Processing Unit (TPU) chip, which is specifically designed for neural networks [7]. The TPU was developed by Google for their machine learning needs and offers up to 200x speedup compared to conventional CPUs. The major TPU advantages are the very short computation latency and up to 10x cost reductions compared to other solutions.

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.77

However, other companies like Microsoft follow a different path and build their own cloud using off-the-shelf chips [4]. They achieve similar performance compared to the TPU and claim lower development costs and more flexibility. This especially adds the option to follow the most recent changes in machine learning algorithms.

One notable work is *Caffeine* [1], in which the authors show the implementation of a general framework based on a matrix multiplication architecture. The proposed architecture uses a High Level Synthesis (HLS) generated systolic array which exploits data locality thanks to a *weight-major mappping* technique for both convolution and fully connected layers execution. Their implementations, integrated with the deep learning framework *Caffe*, obtained 354 GOPS throughput when configured as VGG16.

A different approach is reported in [8]. The authors perform an analysis of the CNN computation loop structure and compare different optimisations (in particular loop unrolling, tiling and interchange). Their analytical model based on the loop order and parameters is then used to perform the DSE. The best design found in the work uses loop unrolling of the external loops of the convolution layer. This allows to reuse the weights and minimise memory accesses and size requirements, and to achieve 645 GOPS performance.

In [2], the authors propose an end-to-end automation flow for systolic array design synthesis. A 2D systolic array structure improves the timing and the data reuse of the design, and is obtained from the analysis of the nested loops implementing the considered algorithm. By performing a two-phase DSE (first generic and then platform-specific), the authors are able to map an aribtrary user-defined CNN algorithm to few predesigned templates. As a result an example VGG16 design achieves 1.17 TOPS.

In [3] the authors use the 2D Winograd algorithm [9] and buffer the needed data in a line buffer. The architecture performs the 2D Winograd over a tile with a fixed stride, in order to compute multiple results at the same time. In this way, the authors have been able to implement an automatic tool flow, and reach 2.9 TOPS with their VGG16 implementation.

IV. VGG16 Architecture

A first step in developing an architecture for a high performance CNN implementation on a DFE is to determine, if off-chip memory will be required and if a fully streaming architecture is feasible.

The first layer of the VGG16 CNN has 64 output planes, where each output contains 224x224 pixels. This means that in total 3,211,264 elements should be stored for the first layer alone. If each element occupies one byte the buffering of all *layer 1* elements requires roughly 3MB.

It is necessary to compute all output planes of a given layer before the computation of the first output plane of the next layer can be finished. This means that a fully streaming architecture, where all layers are computed in parallel, requires buffers large enough to store all inter layer results preferably on-chip. This is not feasible because of the prohibitive on-chip memory requirements.

In theory it would be possible to buffer data between layers off-chip and still process each layer in parallel, but this would require an excessive off-chip memory bandwidth.

Considering the above, the proposed architecture will use Processing Elements (PEs), which all work on the same layer in parallel. The overall architecture of the convolution design can be seen in Figure 1. Each PE can process multiple pixels and multiple input feature maps in parallel, while calculating one output feature map.

A PE handles the convolution operation as well as the rectifier linear unit (ReLU) activation function [10]. Additionally it has a buffer used for the accumulation of the results in the output feature map. This structure is double buffered so that once all input feature maps are processed the output feature maps do not need to be written back to the off-chip memory from all PEs at the same time. When the output data are streamed back to the off-chip memory optional pooling can be applied during the streaming process.

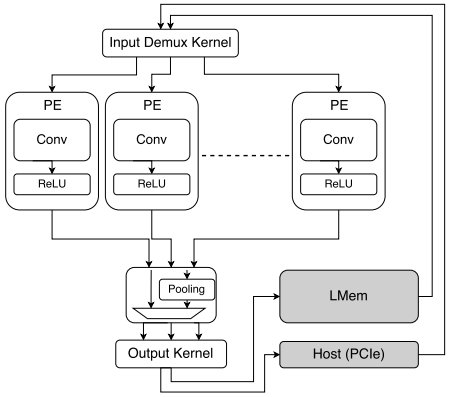

Fig. 1. Convolution design architecture, highlighting the PE connections

V. Optimisations and Design Space Exploration

In this section, we introduce a series different optimisations that are applicable to the proposed architecture, together with our DSE derived from them. The main goal of the applied optimisations is to fully utilise all available arithmetic resources in addition to an maximising usage of the on-chip memory.

A. Processing Elements count

The number of used Processing Elements (PEs) defines directly the required resource utilisation and memory bandwidth.

In particular, the more PEs are used the more on-chip memory is needed, but on the other hand the memory bandwidth requirements are reduced, as less pixels per cycle have to be processed per cycle. In addition, more PEs mean that less iterations have to be performed to generate all outputs.

To simplify control logic and to avoid stalling, the number of PEs is a divisor of the specific number of outputs.

B. Processing of Multiple Inputs in Parallel

An option to narrow the PEs memory write port width is to process multiple inputs in parallel. While this has no direct impact on the required off-chip memory bandwidth, it means that more weights have to be loaded at the same time.

Multiple parallel inputs can be used to make it easier to match the aspect ratios of the available on-chip memory or to fit the output size of each layer better to the available PEs.

C. Datatype Customisation

CNNs are ideal for custom data representations [11]–[13]. It is possible to successfully use very low precision as shown for example in [14]. However, these very low precision types usually also lead to changes to the network architecture.

Because of the specific CNNs characteristics we consider only fixed point datatypes in this work. Fixed point operations require significantly less area and power compared to their floating point counterparts. One specific reason for the above is that area is expended on normalisation and denormalisation around every operation.

The precise fixed point type used is determined by simulating the system (with a device simulator or using software libraries for fixed point computation) and checking the classification results on a test set against a reference floating point version. A different option to optimise the trade off between the used area and the achieved performance is to use asymmetric arithmetic. For example the weights can be represented with less bits than the actual network data. This also helps when the port widths of the available hardware multipliers are asymmetric.

D. Design Space Exploration

Our DSE is performed taking into account all previously described optimisations, such as a fixed point datatype, a variable PE number and number of inputs per cycle. In order to improve the timing characteristics we divide the chip into three super regions, which are programmed separately. This removes the need for communication between parts of the algorithm, which are placed on opposite ends of the chip. As a result we only perform the design space exploration for one super region (at a third of the external IO bandwidth) and then replicate our design three times in space.

Figure 2 shows the design space points by applying the optimisations described in this Section. The implementation choice starts from a range of number of PEs and pixels per cycle (*pipes*). We can then prune the design space by removing all the non-feasible points. These are points in which the architectural and application-specific limits are not respected.

The same exploration can be performed by considering the Roofline Model [15], which is a visual model initially used to analyse the attainable performance in multicore systems. The model is based on two terms: the Computation to Communication (CTC) ratio, which represents the number of operations per memory byte access, and the computational performance, represented in GOPS. Ceilings are then added to the graph: the *peak bandwidth* ceiling (left to middle) and

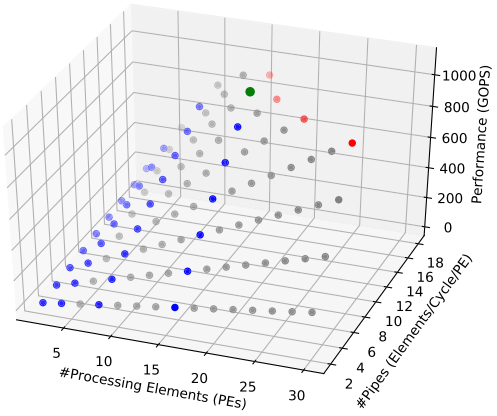

Fig. 2. Design Space of the proposed architecture for the VGG16 network. Blue dots are considered in the DSE (chosen design point in green). Red and grey dots where not considered because of resources and complexity constraints.

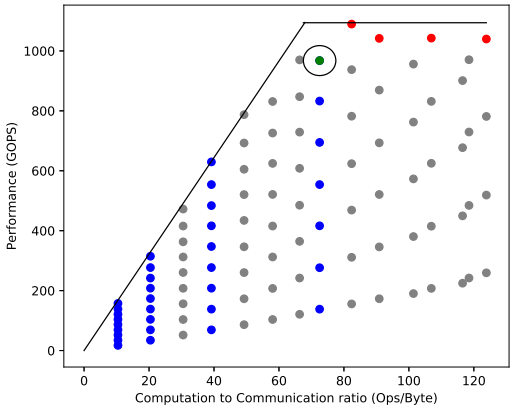

Fig. 3. Roofline Model representation of our Design Space for VGG16, showing the peak bandwith and performance ceilings. Blue dots are considered in the DSE (chosen design point in green). Red and grey dots were not considered because of resources and complexity constraints.

the *peak performance* ceiling (right). These bounds depend on the chosen architecture and compute device, and are useful to decide which parts of the design to optimise and which not.

The graph in Figure 3 represents the design space points based on their expected CTC ratio and performance at 240MHz, which is our target frequency. As it can be seen from the picture, our design parameters choice represents the best choice in terms of performance, without becoming memory bound and having a certain margin of the resources utilisation (useful for timing constraints).

VI. Experimental Results

In this section, we present our implementation results, in terms of performance and consumption of power/resources. We also perform a comparison with previous work to validate our model and implementation.

TABLE I
PERFORMANCE COMPARISON

	[1]	[8]	[2]	[3]	*Our Work*
Precision	16 bits fixed	16 bits fixed	16 bits fixed	16 bits fixed	18-27 bits fixed
Freq (MHz)	150	150	231.85	200	240
Logic cell (K)	300	161	313	600	788
SRAM (Kb)	$1,248 \times 18$	$1,900 \times 20$	$1,668 \times 20$	$1,824 \times 18$	$3,128 \times 18$
Multipliers	2,833	1,518	1,500	2,520	6,057
TeraOp/s	0.354	0.645	1.17	2.9	2.45

A. Experimental Setup

We implement and evaluate our design on a Maxeler MAX5 DFE (described in Section II). The DFE contains over 1 million logic elements, 6,840 multipliers and 38 MB of on-chip memory. In addition, 48GB of LMEM, organised as three independent 16GB DDR4 channels and a 16x PCIe gen. 3 link to the host processor are available.

The host application runs on a Dual-Socket server with Intel®Xeon®E5-2643V4 (6 cores @3.40GHz) CPUs.

B. Results and Comparison

In order to test our methodology, we implement the convolutional layers of the VGG16 network. The implemented design consists of 16 PEs for each of the three super regions. Each PEs receives at each clock cycle 14 values (7 values $\times$ 2 feature maps).

The total consumption of resources consumption and the performance of our design can be seen in Table I, along with related work. Our design uses 67% of the available logic, 89% of the multipliers, and 53% of the memory resources. To measure the actual performance of our implementation, we show a test application which performs a forward pass over a batch of images. The test results show an average throughput of 84.5 images per second at a 240MHz frequency achieving 2,450 GOPS. It can be assumed that due to the low resource utilisation higher frequencies can be achieved with more aggressive timing optimisations.

It has to be noted that most of other work at high performance uses 16 bits precision data types and 8-16 bits precision for the weights. This is due to the used multiplier architecture, which allows tiling of the hardware multipliers into two 16×16 bits multipliers. Our work uses instead a higher precision data type (27 bits data, 18 bits weights), considering the specific hardware multiplier implementation using 27 by 18 multipliers, which cannot (easily) be tiled with lower size operators. Considering these limitations, our work positions between [2] and [3], and is one of the fastest implementations compared to state of the art, while providing an especially good accuracy. In order to significantly improve the performance further we will employ the Winograd transformation in future research.

VII. CONCLUSIONS

In this work, we presented our design space exploration and the implementation of a widely used CNN on the Maxeler MAX5 DFE. The design space exploration was tailored to our base design, composed by processing elements, each computing part of the network in an efficient way, by merging the convolution and pooling execution. We then estimated the expected performance and resources usage, along with the required external memory bandwidth, which is seen as the limiting factor in most designs. We used both, standard design space exploration exploration and the Roofline model. Finally, we implemented a VGG16 test case showing high throughput.

Future steps to improve our architecture include the implementation of a Winograd-based design, in order to further reduce the computational complexity of each processing element. Work in progress considers the implementation of the Fully-Connected and other layers with a similar model and analysis and also the automation of the proposed approach.

REFERENCES

[1] C. Zhang *et al.*, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *ICCAD*, 2016.

[2] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 29.

[3] L. Lu *et al.*, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *FCCM*, 2017.

[4] K. Freund. (2017, August) Microsoft: FPGA Wins Versus Google TPUs For AI. [Online]. Available: https://www.forbes.com/sites/moorinsights/2017/08/28/microsoft-fpga-wins-versus-google-tpus-for-ai/amp/

[5] N. Voss *et al.*, *Rapid Development of Gzip with MaxJ*. Cham: Springer International Publishing, 2017, pp. 60–71.

[6] O. Pell and V. Averbukh, "Maximum performance computing with dataflow engines," *Computing in Science Engineering*, vol. 14, no. 4, pp. 98–103, July 2012.

[7] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," ser. ISCA, 2017.

[8] Y. Ma *et al.*, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *FPGA*, 2017.

[9] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," *CoRR*, 2015.

[10] M. D. Zeiler *et al.*, "On rectified linear units for speech processing," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[11] V. Gokhale *et al.*, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[12] A. Tisan and J. Chin, "An end-user platform for FPGA-based design and rapid prototyping of feedforward artificial neural networks with on-chip backpropagation learning," *IEEE Transactions on Industrial Informatics*, June 2016.

[13] S. I. Venieris and C. S. Bouganis, "fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs," in *FCCM*, 2016.

[14] Y. Umuroglu *et al.*, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *The 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[15] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, 2009.