Design Space Exploration of FPGA-Based Deep Convolutional Neural Networks

Mohammad Motamedi, Philipp Gysel, Venkatesh Akella and Soheil Ghiasi
Electrical and Computer Engineering Department, University of California, Davis
{mmotamedi, pmgysel, akella, ghiasi}@ucdavis.edu

Abstract— Deep Convolutional Neural Networks (DCNN) have proven to be very effective in many pattern recognition applications, such as image classification and speech recognition. Due to their computational complexity, DCNNs demand implementations that utilize custom hardware accelerators to meet performance and energy-efficiency constraints. In this paper we propose an FPGA-based accelerator architecture which leverages all sources of parallelism in DCNNs. We develop analytical feasibility and performance estimation models that take into account various design and platform parameters. We also present a design space exploration algorithm for obtaining the implementation with the highest performance on a given platform. Simulation results with a real-life DCNN demonstrate that our accelerator outperforms other competing approaches, which disregard some sources of parallelism in the application. Most notably, our accelerator runs $1.9\times$ faster than the state-of-the-art DCNN accelerator on the same FPGA device.

I. Introduction

Deep Convolutional Neural Networks (DCNN) have recently led to impressive progress in many challenging machine learning problems, such as machine vision, natural language processing and speech recognition.

The complexity of DCNNs presents their real-time performance as a major challenge that hinders their widespread deployment, particularly in resource-constrained embedded systems. In this paper, we address this topic via systematic architecture exploration. Further, we develop a custom accelerator for efficient implementation of DCNNs' test phase. We restrict our discussion to the test phase, and thus, assume that a trained model with known weights is already available.

Prior work on acceleration of DCNN implementations includes several platform technologies. One appproach involves utilization of commodity Graphics Processing Units (GPU) [7] whose software programmability renders them particulary well suited for research on DCNN models and acceleration of the training phase. The significant energy dissipation of commodity GPUs, however, prohibits their integration in energy constrained and mobile embedded systems. Another approach includes development of ASIC chips [3], which offer the well-known advantage of performance and energy efficiency at the disadvantage of significant fabrication cost and limited flexibility. The tradeoff appears to be unattractive at the moment, given the market size and the fluid and rapidly-evolving state of research in DCNN models. Another group of researchers focus on FPGA based accelerator design [2, 10]. Reasonable price, low power consumption and reconfigurability of FPGAs are characteristics that make them attractive for DCNN implementation.

Several groups have attempted to build DCNN accelerators by focusing on custom computation engines. This approach relies on the implicit assumption that DCNNs are computationally bounded [1, 2, 9]. At the other extreme, some prior work have viewed the issue as a memory bandwidth problem. As an example, Peeman *et al.* focus on maximization of the reuse of on-chip data [8]. Interestingly, DCNNs require both high memory bandwidth as well as high computation resources. Thus, an optimized accelerator needs to judiciously strike a balance between the two interdependent criteria [10]. Focus on one aspect while ignoring the other, is bound to result in a sub-optimal architecture.

In this paper, we present a systematic approach for the design and development of an FPGA-based DCNN accelerator. Specifically, we design an architecture template that is capable of exploiting all sources of parallelism in a DCNN. We develop a model that considers performance and data transfer for the proposed architecture. This allows us to estimate the architecture's feasibility and performance with respect to a specific FPGA. The model enables us to prudently quantify the tradeoff of exploiting one source of parallelism vs. another. Subsequently, we develop a design space exploration algorithm, which yields the most efficient architecture that would be feasible on the target platform. Our main contribution includes advancing the state of the art in DCNN acceleration [10] via exploiting more sources of parallelism. Experimental results demonstrate that we substantially outperform prior work on DCNN accelerators.

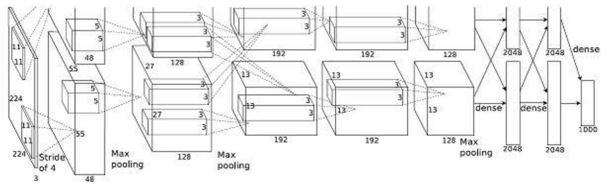

Fig. 1. A sample real-life DCNN, proposed by Krizhevsky *et al.*, which won the ImageNet contest in 2012 (AlexNet) [7]. In this DCNN $\langle N_1, M_1, K_1 \rangle = \langle 3, 48, 11 \rangle$.

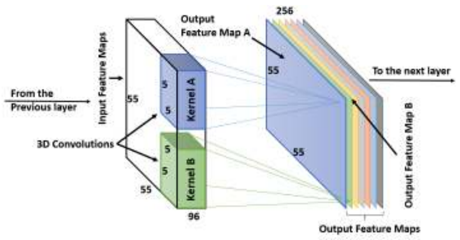

Fig. 2. Layer 2 from the DCNN which is shown in Fig. 1.

II. Convolutional Neural Network

A. Background and Overview

DCNNs form a subset of artificial neural networks in which, the transformation from the input to output feature maps is determined by a set of convolution kernels. Querying a DCNNs in the test phase, which is the focus of this paper, requires using the forward path of the trained network.

Fig. 1 illustrates an example DCNN in which, the first five layers from the left are convolutional layers and the last three layers are fully connected layers. More than 90% of the execution time in DCNNs is spent in the convolutional layers [4]; therefore, in this study we only concentrate on accelerating the convolutional layers. In a DCNN there are L convolutional layers $l_1, ..., l_L$ each of which, consists of a set of convolution kernels along with a pooling and sub sampling functions (in Fig. 1, $L = 5$). By sliding one stack of convolution kernels over an input layer and continuously computing convolutions, a single output feature map can be generated. This sliding is performed by different strides (S) in different layers. Each layer is associated with a stack of convolution kernels of size $K_l \times K_l$. The size of this stack is N_l, and in each layer, there are M_l different stacks, each of them is used to build one output feature map. Variables N_l and M_l are the number of input and output feature maps in layer l respectively. Equation (1) defines a 3D convolution which is used in DCNNs. Variables R_l and C_l are the number of rows and columns in the input feature maps of layer l respectively. IFM, OFM and W stand for the Input Feature Maps, Output Feature Maps and weights of the convolution kernels.

As it is illustrated in Fig. 2, every output feature map is the result of the 3D convolution of a kernel with all of the input features maps (i.e., the number of output feature maps are equal to the number of kernels).

B. Parallelism in DCNNs

In each DCNN, there are several sources of parallelism (Fig. 3). To achieve the best possible speedup, all of these sources should be recognized and exploited properly.

- **Inter Layer Parallelism**
  As $\forall\ l \in \{1, 2, .., L\} : IFM_{(l+1)} = OFM_l$, there is data dependency between layers; hence, different layers cannot be executed in parallel. On the other hand, since real life DCNNs are very large, it is infeasible to implement all layers in a pipelined fashion. Even for small CNNs, pipeline based implementations do not always provide the best performance [5].
- **Inter Output Parallelism**
  Different output feature maps are totally independent of each other, and theoretically all of them can be computed in parallel. To do so, Equation (1) should be calculated in parallel for different values of m.
- **Inter Kernel Parallelism**
  Each pixel in each output feature map is the result of a set of convolutions. However, as these convolutions are independent for different output pixels, it is possible to compute all of them concurrently. This is another source of data level parallelism that can be exploited. Therefore in order to exploit this source of parallelism, Equation (1) can be calculated for different values of r and c concurrently.
- **Intra Kernel Parallelism**
  Finally, there is a considerable amount of parallelism in each convolution. A convolution is essentially a set of multiplications and additions. As each multiplication between a weight in a kernel and a pixel in an input feature map is independent from another multiplication, all of them can be performed in parallel.

If there were unlimited area, bandwidth and on chip memory, all of the aforementioned sources of parallelism could be exploited to expedite the neural network as much as possible. However, in practice, this is infeasible. Therefore, the challenge is to find the optimal combination of different parallelism sources that both minimizes the execution time and satisfies the constraints of the target chip. In this paper we find the optimal solution by introducing an architecture which can utilize all of the parallelism sources and optimizing design parameters for it.

III. Proposed Architecture Template

The proposed architecture is shown in Fig. 4. The first layer which is named A consists of blocks of T_k multipliers. The multipliers can be used concurrently to compute a portion of the required multiplications of a convolution. The results of these multiplications are accumulated using

$$\begin{aligned}
&\forall\ l \in \{1,2,3,...,L\}; && l : \text{layers}\\
&\forall\ r \in \{1,2,3,...,R_l\}, && r : \text{row in feature maps}\\
&\forall\ c \in \{1,2,3,...,C_l\}, && c : \text{column in feature maps}\\
&\forall\ m \in \{1,2,3,...,M_l\}: && m : \text{output feature maps in layer } l
\end{aligned}$$

$$OFM[l][m][r][c] = \sum_{n=1}^{N_l} \sum_{i=-\lfloor \frac{K_l}{2} \rfloor}^{\lfloor \frac{K_l}{2} \rfloor} \sum_{j=-\lfloor \frac{K_l}{2} \rfloor}^{\lfloor \frac{K_l}{2} \rfloor} IFM[l][n][r+i][c+j] \times W[l][n][m][i+\lfloor K_l/2 \rfloor][j+\lfloor K_l/2 \rfloor] \quad (1)$$

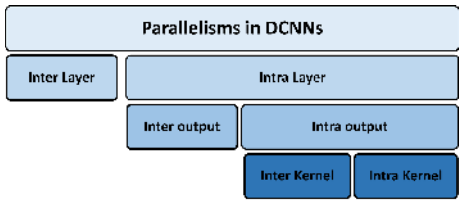

Fig. 3. Available parallelism sources in DCNNs

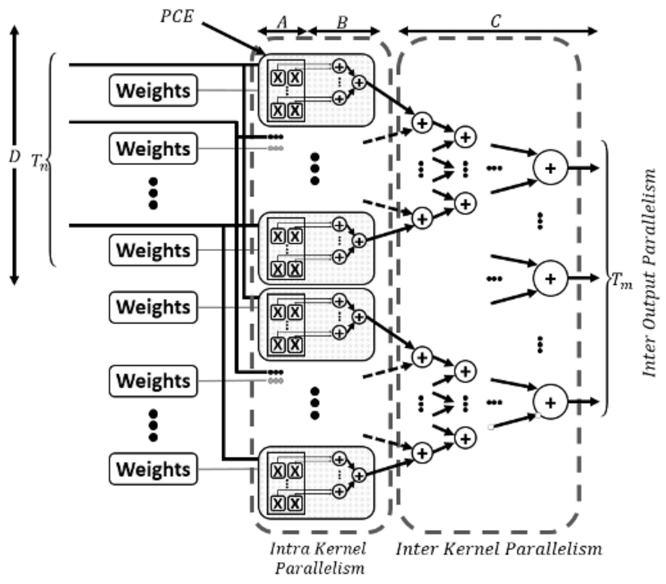

Fig. 4. Proposed Architecture. Multipliers in layer A and corresponding adders in B form Parallel Convolution Engines (PCE) that can exploit intra kernel parallelism. Combination of PCEs along with the corresponding adders in part C are designed for exploiting inter kernel parallelism . T_m replication of unit D provide the ability to exploit inter output parallelism.

corresponding adder trees (B). Combination of one multiplier block from layer A and corresponding adders from layer B are called **Parallel Convolution Engine** (PCE) in this paper. PCEs provides the ability of exploiting intra kernel parallelism. Each convolution kernel has two inputs: input feature maps and corresponding weights. In the proposed architecture, it is possible to feed in T_n different kernels of the same kernel stack to the convolution engines along with T_n different input feature maps. The results should be added together using the adder stacks, which are labeled with C. The combination of convolution engines along with the corresponding adders, which are labeled as D, are designed for exploiting inter kernel parallelism.

In order to provide the ability of using the inter output parallelism in this architecture, unit D is replicated T_m times. Hence, each replica can be used to compute an output feature map in parallel with other replications. The architecture which is shown in Fig. 4, has the ability to utilize all of the different parallelism sources which exist in a DCNN. Yet, in order to achieve the optimal solution, it is important to determine the appropriate values for T_K, T_m and T_n. We will offer a technique to find those values in the following sections.

The limited amount of on-chip memory mandates a tiled data transfer. We are using tiling in the kernel level as well as feature map level. For DCNNs that have large kernel sizes, tiling in the kernel level improves the performance drastically. This tiling which is extended to the kernel level provides us with the opportunity to search for the optimized architecture among a larger number of candidates (i.e., the design space is a superset of the one in [10]). In tiling, a tile with the dimension of $T_r \times T_c$ is fetched for each input feature map in each iteration. Likewise, for each pixel of this tile, a tile of weights with the dimension of $T_i \times T_j$ is fetched. For an optimal architecture, it is required to determine the suitable values for T_r, T_c, T_i and T_j.

IV. Analytical Model

In this section, we develop an analytical model that shows the relation between different design parameters and attainable performance. This model is also used to indicate the required memory bandwidth for the design. Given the goal of minimal execution time, this model can be used to calculate how many times each module should be replicated.

TABLE I
Optimal Architecture Parameters for Different Degrees of Flexibility (FPGA: Xilinx Virtex7 485t, DCNN: AlexNet [7]). L stands for Layer.

L	Layer Specific Solution					Layer Specific Solution (Fixed T_k)					Static Solution				
	T_m	T_n	T_k	Cycles	GFLOPS	T_m	T_n	T_k	Cycles	GFLOPS	T_m	T_n	T_k	Cycles	GFLOPS
1	16	3	10	117975	89	48	3	3	124025	85	16	3	9	127050	83
2	4	24	5	233280	96	10	16	3	255879	87	16	3	9	279936	80
3	15	32	1	79092	95	16	10	3	79092	95	16	3	9	87204	86
4	15	32	1	118638	95	32	5	3	118638	95	16	3	9	129792	86
5	10	48	1	79092	95	10	16	3	79092	95	16	3	9	86528	86
S				628077					656726					755642	

A. Computation Model

The number of execution cycles is equal to the number of MAC (Multiplication and Accumulation) operations. This number can be computed using Equation (2) where M, N, R, C, K and P are number of output feature maps, number of input feature maps, number of rows, number of columns, size of the convolution kernels and the pipeline overhead respectively.
As each convolution includes one multiplication and one addition, the total number of required operations can be shown by Equation (3). Hence, the computation roof can be defined and calculated as shown in Equation (4).

$$\text{Number of Execution Cycles} = \lceil\frac{M}{T_m}\rceil \times \lceil\frac{N}{T_n}\rceil \times \frac{RC}{T_r T_C} \times \lceil\frac{K}{T_i}\rceil \times \lceil\frac{K}{T_j}\rceil \times (T_r T_c \times \lceil\frac{T_i T_j}{T_k}\rceil + P) \quad (2)$$

$$\text{Num of Ops} = 2 \times R \times C \times M \times N \times K \times K \quad (3)$$

$$\text{Computation Roof} = \frac{\text{Number of Operations}}{\text{Number of Execution Cycles}} = \frac{2 \times M \times N \times K^2}{\lceil\frac{M}{T_m}\rceil \times \lceil\frac{N}{T_n}\rceil \times \lceil\frac{K}{T_i}\rceil \times \lceil\frac{K}{T_j}\rceil \times \lceil\frac{T_i \times T_j}{T_k}\rceil} \quad (4)$$

The optimal architecture minimizes the number of execution cycles, i.e., maximizes the computation roof. Notice that the nominator in Equation (4) only depends on the DCNN. Hence, for a particular neural network, the total number of operations are constant. A well designed accelerator is able to achieve a computation roof that fully utilizes all of the resources of a particular FPGA.

B. On-Chip Buffers

The estimated computation roof can be achieved if with the selected parameters, the required data transmission is less than the maximum available bandwidth. Computations are performed on the input feature maps and weights and the results are stored as output feature maps. Hence, three different buffers are required to hold the necessary data. The required sizes for input feature maps, weights and output feature maps are shown in Equations (5), (6) and (7) respectively.

$$\beta_{in} = T_n(ST_r + T_i - S)(ST_c + T_j - S) \times 4 \text{ Bytes} \quad (5)$$

$$\beta_{wght} = T_m \times T_n \times T_i \times T_j \times 4 \ Bytes \quad (6)$$

$$\beta_{out} = T_m \times T_r \times T_c \times 4 \ Bytes \quad (7)$$

It is possible to prove that for the most efficient implementation of Equation (1), the number of loads and stores can be calculated using Equations (8), (9) and (10).

$$\alpha_{in} = \frac{M}{T_m} \times \frac{N}{T_n} \times \frac{R}{T_r} \times \frac{C}{T_c} \times \frac{K}{T_i} \times \frac{K}{T_j} \quad (8)$$

$$\alpha_{wght} = \frac{M}{T_m} \times \frac{N}{T_n} \times \frac{R}{T_r} \times \frac{C}{T_c} \times \frac{K}{T_i} \times \frac{K}{T_j} \quad (9)$$

$$\alpha_{out} = 2 \times \frac{M}{T_m} \times \frac{R}{T_r} \times \frac{C}{T_c} \quad (10)$$

Using these values and buffer sizes, the required Computation To Communication ratio (CTC) can be computed as shown in Equation (11). The CTC is a measure for reuse of data which is fetched to the on-chip memory.

$$CTC = \frac{\text{Total required computation}}{\text{Total Required communication}} = \frac{2 \times M \times N \times R \times C \times K^2}{\alpha_{in} \times \beta_{in} + \alpha_{wght} \times \beta_{wght} + \alpha_{out} \times \beta_{out}} \quad (11)$$

V. Experimental Results

To find the set of parameters that maximize the performance, we explore the design space by enumerating over all possible configurations and computing the *Computation Roof*. This enumeration must be performed under four constraints:

1. The sum of all required buffers in a design must be less than or equal to the available on chip memory.
2. The required bandwidth should be less than or equal to the available bandwidth on a particular platform.
3. Only a certain number of Computational Engines (CE) can be implemented on any chip. We adopt the number of CEs from [10] to enable a fair comparison. However, as we use Parallel Convolution Engine, the number of

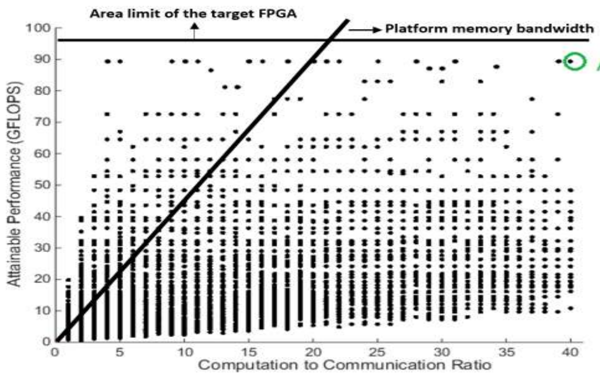

Fig. 5. Design space exploration for Layer 1 of the DCNN of Fig. 1

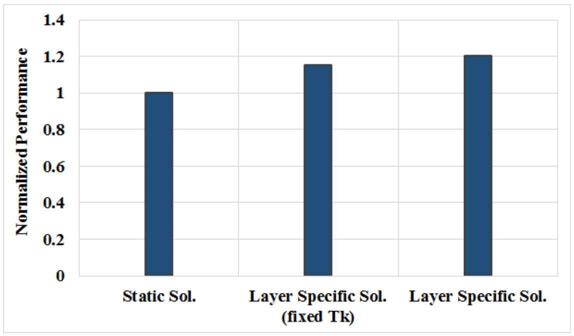

Fig. 6. Normalized performance for dynamic and static configuration

CEs should be decreased proportional to T_k as shown in Equation (12).

$$T_m \times T_n \leq \frac{\# \, CEs}{T_k} \tag{12}$$

We find the optimal design parameters for AlexNet [7] under the constraints of Xilinx Virtex7 485t FPGA. The results are shown in Table I.

A. Impact of Reconfigurability

We considered three different scenarios with different degrees of reconfigurability. Initially, the accelerator supports flexible T_m, T_n and T_k. Subsequently, T_k is fixed for every layer. Finally T_m, T_n and T_k are fixed for all layers. The normalized performance of these three experiments is shown in Fig. 6. Performance is defined as $1/(execution\ cycles)$.

Layer Specific Optimal Solution: In this case T_m, T_n and T_k are fully flexible and the optimal solution is shown in Table I. This configuration delivers the best performance but demands a very complex interconnection network.

The design space for layer 1 of the DCNN is shown in Fig. 5. For any point in the design space, the slope of the line from the origin to that point gives the required bandwidth. The target of this exploration is to find the design with the highest GFLOPS. Among points with highest GFLOPS, point A is the best candidate because it has the best CTC.

Layer Specific Optimal Solution (Fixed T_k): In the subsequent experiment, (T_k) is fixed across different layers. As it is shown in Table I, due to the reduced flexibility, the performance drops by 4.5%. However, the architecture no longer needs to support dynamic sizes for PCE across different layers.

Static Optimal Solution: In this experiment T_m, T_n and T_k are fixed and the optimal solution is shown in Table I. For a completely static accelerator the number of execution cycles increases by 13% compared to the dynamically reconfigurable accelerator. However, for the comparison between dynamic and static reconfigurability in [2, 10] and this paper, the required area for the interconnection network is not taken into account. Hence, the speedup of 13% is only a very loose upper bound. This indicates that the achievable speedup of dynamic reconfigurability can never be better than 13%.

B. Performance Comparison

Performance Comparison Versus CPU: Based on the execution time which is reported in [10] the proposed accelerator in this paper has a speedup of 23.24X and 6.4X compared to the single and 16 threaded CPU based implementation.

Performance Comparison Versus Other Accelerators: The performance of different accelerators are reported for different DCNNs on different FPGAs. Hence, before comparing the performances, it is required to normalize the data as described in [10]. The result are shown in Table II. Compared to the previous approaches, the proposed accelerator has the highest performance density. Our approach is 37% faster compared to the state of the art solution (ISFPGA2015 [10]).

In order to determine which accelerator performs better when more resources are available, the design space is explored for an FPGA with 2 times more area and bandwidth than Virtex7 485t and the results are shown in Table III. For this FPGA, our accelerator can achieve a speedup of 1.9X compared to the offered approach in [10]. This shows that the proposed solution can utilize the resources better for more sophisticated chips which will be offered in the future. The normalized performance (divided by [10]) for the proposed solution and [10] are compared in Fig. 7.

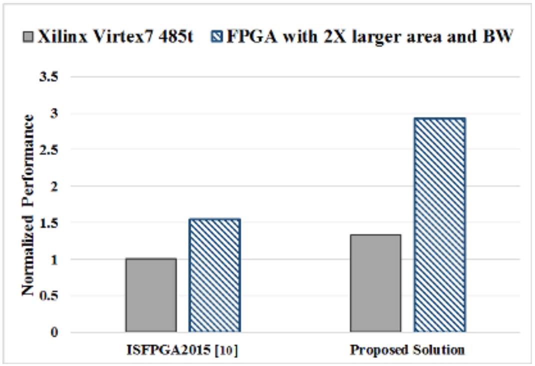

Fig. 7. Normalized performance for proposed solution and ISFPGA2015 [10] on Xilinx Virtex7 485t and an FPGA with 2X larger area and bandwidth

TABLE II
Performance Comparison

	ICCD2013[8]	FPL2009[6]	PACT2010[1]	ISCA2010[2]	ISFPGA2015[10]	Proposed Sol.
Precision	fixed point	48bits fixed	fixed point	48bits fixed	32bits float	32bits float
Frequency	150 MHz	125 MHz	125 MHz	200 MHz	100 MHz	100 MHz
FPGA Chip	VLX240T	SX35	SX240T	SX240T	VX485T	VX485T
CNN Size	2.74 GMAC	0.26 GMAC	0.53 GMAC	0.26 GMAC	1.33 GFLOP	1.33 GFLOP
Performance	17 GOPs	5.25 GOPs	7.0 GOPs	16 GOPs	61.62 GFLOPs	84.2 GFLOPs
GOPs/Slice	4.5E-04	3.42E-04	1.9E-04	4.3E-04	8.12E-04	11.09E-04

TABLE III
Static Optimal Solution on an FPGA with 2X larger area and bandwidth than the baseline FPGA (Virtex7 485t). DCNN: AlexNet [7].

	ISFPGA2015 [10]				Proposed Accelerator					Speedup
	T_m	T_n	Cycles	GFLOPS	T_m	T_n	T_k	Cycles	GFLOPS	
Layer 1	48	3	366025	29	48	3	5	75625	139	4.84
Layer 2	64	12	145800	154	64	3	5	116640	192	1.25
Layer 3	64	15	41067	182	64	3	5	43602	171	0.94
Layer 4	64	15	59319	189	64	3	5	64896	172	0.91
Layer 5	64	15	39546	189	64	3	5	43264	172	0.91
Total			651757					344027		1.89

VI. Conclusion

In this paper, a new accelerator for DCNNs is proposed. This accelerator can effectively leverage all of the available sources of parallelism to minimize the execution time. Moreover, we proposed an improved tilling technique that increases the performance by utilizing the tiling in the convolution kernel level. We also developed an analytical model for the proposed architecture and used that model to determine optimized design parameters for a real-life neural network and a particular FPGA. Experimental results show that the proposed solution outperforms all previous work. Specifically, our accelerator has a speedup of 1.9X compared to the state-of-the-art DCNN accelerator.

References

[1] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf. A programmable parallel accelerator for learning and classification. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 273–284. ACM, 2010.

[2] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 247–257, New York, NY, USA, 2010. ACM.

[3] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al. Dadiannao: A machine-learning supercomputer. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pages 609–622. IEEE, 2014.

[4] J. Cong and B. Xiao. Minimizing computation in convolutional neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2014*, pages 281–290. Springer, 2014.

[5] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109–116. IEEE, 2011.

[6] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. Cnp: An fpga-based processor for convolutional networks. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 32–37. IEEE, 2009.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal. Memory-centric accelerator design for convolutional neural networks. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pages 13–19. IEEE, 2013.

[9] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf. A massively parallel coprocessor for convolutional neural networks. In *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, pages 53–60. IEEE, 2009.

[10] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.