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. Leverages all sources of parallelism in DCNNs, in this paper we propose an FPGA-based accelerator architecture. We develop analytical feasibility and performance estimation models that take into account various design and platform parameters. Consequently, we develop a design space exploration algorithm using which, we obtain the implementation with the highest performance for a given target FPGA 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× 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 and development of a custom accelerator for efficient implementation of DCNN feed-forward 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 approach involves utilization of commodity Graphics Processing Units (GPU) [7] whose software programmability renders them particularly 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 programmability 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 to 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 [10]. We develop performance and memory data transfer models for the template architecture based on which, we estimate both feasibility, with respect to a specific FPGA device with known resources, and performance of a particular instantiation of the architecture. The models enable 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 contributions include advancing the state of the art in DCNN acceleration [10] via consideration of all sources of parallelism, development of the associated performance and memory transfer estimation models, and design space exploration apparatus. Experimental results demonstrate that we substantially outperform prior work on DCNN accelerators.
As it is shown 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 $\forall \ l \in \{1, 2, 3, ..., L\}$ : $IFM_{(l+1)} = OFM_l$, there is data dependency between layers; hence, they cannot be executed in parallel. On the other hand, since real life DCNNs are very large, they cannot be implemented in a pipeline fashion. Even for small CNNs, pipeline based implementation does 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 pixels, it is possible to compute all of them concurrently. This is another source of data level parallelism that can be exploited. Therefore in Equation (1), in order to exploit this source of parallelism, calculation should be done for different values of $c$ and different values of $r$ 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, it is required to determine for a particular neural network and a specific chip, what parallelism sources should be used to what extent to minimize the execution time. 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 that can be used concurrently to compute a portion of the required multiplications of a convolution. The results of these multiplications are accumulated using corresponding adder trees (B). Combination of one multiplier
∀ l ∈ {1, 2, ..., L};  \ l \ : \ \text{layers}  \\
∀ r ∈ {1, 2, ..., R_l};  \ r \ : \ \text{row in feature maps}  \\
∀ c ∈ {1, 2, ..., C_l};  \ c \ : \ \text{column in feature maps}  \\
∀ m ∈ {1, 2, ..., M_l};  \ m \ : \ \text{output feature maps in layer} \ l  \\

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

\text{Fig. 3. Available Parallelism Sources in DCNNs}

block from layer A and corresponding adders form layer B are called \text{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 different kernels of the same kernel stack to the convolution engines along with different input feature maps. The results should be added together using the adder stacks, which are labeled with C. The combination of convolutional 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 (unlike [7]), 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. Yet, for an optimal architecture, it is required to determine the suitable values for T_r, T_c, T_i and T_j.

\text{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 \ \text{replication of unit D provide the ability to exploit inter output parallelism.}

\text{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. This model can be used to determine implementing how many replica of each module on a FPGA yields the minimal execution time.

\text{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)
in which \( P \) is the pipeline overhead.

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).

Number of Execution Cycles =
\[
\left[ \frac{M}{T_m} \right] \times \left[ \frac{N}{T_n} \right] \times \frac{RC}{T_j} \times \left[ \frac{K}{T_i} \right] \times \left[ \frac{K}{T_j} \right] \times (T_r \times C) + P
\]
\[ (2) \]

Num of Ops = \[ 2 \times R \times C \times M \times N \times K \times K \] \[ (3) \]

Computation Roof = \[
\frac{\text{Number of Operations}}{\text{Number of Execution Cycles}} = \frac{2 \times M \times N \times K^2}{\left[ \frac{M}{T_m} \right] \times \left[ \frac{N}{T_n} \right] \times \frac{RC}{T_j} \times \left[ \frac{K}{T_i} \right] \times \left[ \frac{K}{T_j} \right] \times (T_r \times C) + P}
\]
\[ (4) \]

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 DCNNs. 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.

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

\[
\beta_{wght} = T_m \times T_n \times T_i \times T_j \times 4 \text{ Bytes} \]
\[ (6) \]

\[
\beta_{out} = T_m \times T_n \times T_i \times T_j \times 4 \text{ Bytes} \]
\[ (7) \]

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).

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

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

\[
\alpha_{out} = 2 \times \frac{M}{T_m} \times \frac{R}{T_r} \times \frac{C}{T_i} \]
\[ (10) \]

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.

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

**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 CEs should be decreased proportional to \( T_k \). As it is shown in Equation (12).

\[
T_m \times T_n \leq \frac{\# \text{CEs}}{T_k} \]  
\[ (12) \]

We find optimal design parameters for the DCNN which is shown in Fig. 1 Under the bandwidth and area constrains of Xilinx Virtex7 485t FPGA.

**A. Impact 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
TABLE I
Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>fixed point</td>
<td>fixed point</td>
<td>fixed point</td>
<td>48bits fixed</td>
<td>32bits float</td>
<td>32bits float</td>
</tr>
<tr>
<td>Frequency</td>
<td>150 MHz</td>
<td>125 MHz</td>
<td>125 MHz</td>
<td>200 MHz</td>
<td>100 MHz</td>
<td>100 MHz</td>
</tr>
<tr>
<td>FPGA Chip</td>
<td>VLX240T</td>
<td>SX35</td>
<td>SX240T</td>
<td>SX240T</td>
<td>VX485T</td>
<td>VX485T</td>
</tr>
<tr>
<td>CNN Size</td>
<td>2.74 GMAC</td>
<td>0.26 GMAC</td>
<td>0.53 GMAC</td>
<td>0.26 GMAC</td>
<td>1.33 GFLOP</td>
<td>1.33 GFLOP</td>
</tr>
<tr>
<td>Performance</td>
<td>17 GOPs</td>
<td>5.25 GOPs</td>
<td>7.0 GOPs</td>
<td>16 GOPs</td>
<td>61.62 GFLOPs</td>
<td>84.2 GFLOPs</td>
</tr>
<tr>
<td>GOPs/Slice</td>
<td>4.5E-04</td>
<td>3.42E-04</td>
<td>1.9E-04</td>
<td>4.3E-04</td>
<td>8.12E-04</td>
<td>11.09E-04</td>
</tr>
</tbody>
</table>

TABLE II
Layer Specific Optimal Solution, Layer Specific Optimal Solution with fixed \( T_k \) and Static Optimal Solution for the proposed accelerator (FPGA: Xilinx Virtex7 485t, DCNN: AlexNet [7]). \( L \) stands for Layer.

<table>
<thead>
<tr>
<th>( L )</th>
<th>( T_m )</th>
<th>( T_n )</th>
<th>( T_k )</th>
<th>Cycles</th>
<th>GFLOPS</th>
<th>( T_m )</th>
<th>( T_n )</th>
<th>( T_k )</th>
<th>Cycles</th>
<th>GFLOPS</th>
<th>( T_m )</th>
<th>( T_n )</th>
<th>( T_k )</th>
<th>Cycles</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>3</td>
<td>10</td>
<td>117975</td>
<td>89</td>
<td>48</td>
<td>3</td>
<td>3</td>
<td>124025</td>
<td>85</td>
<td>16</td>
<td>3</td>
<td>9</td>
<td>127050</td>
<td>83</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>24</td>
<td>5</td>
<td>233280</td>
<td>96</td>
<td>10</td>
<td>16</td>
<td>3</td>
<td>255879</td>
<td>87</td>
<td>16</td>
<td>3</td>
<td>9</td>
<td>279936</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>32</td>
<td>1</td>
<td>79092</td>
<td>95</td>
<td>16</td>
<td>10</td>
<td>3</td>
<td>79092</td>
<td>95</td>
<td>16</td>
<td>3</td>
<td>9</td>
<td>87204</td>
<td>86</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>32</td>
<td>1</td>
<td>118638</td>
<td>95</td>
<td>32</td>
<td>5</td>
<td>3</td>
<td>118638</td>
<td>95</td>
<td>16</td>
<td>3</td>
<td>9</td>
<td>129792</td>
<td>86</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>48</td>
<td>1</td>
<td>79092</td>
<td>95</td>
<td>10</td>
<td>16</td>
<td>3</td>
<td>79092</td>
<td>95</td>
<td>16</td>
<td>3</td>
<td>9</td>
<td>86528</td>
<td>86</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td>628077</td>
<td>656726</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>755642</td>
<td></td>
</tr>
</tbody>
</table>

three experiments is shown in Fig. 6. Performance is defined as \( 1/\text{(execution cycles)} \).

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 II. 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. In this Figure, the red line is the available bandwidth and any black point indicates a potential design with a certain CTC and GFLOPS. 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 II, due to the reduced flexibility the performance is dropped 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 II.

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 I. Compared to the previous approaches, the proposed accelerator has the highest performance density. The speedup of our approach is 37\% compared to the second best 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 for the proposed solution and the solution which is offered in [10] are compared in Fig. 7.

VI. Conclusion

In this paper, a new accelerator for DCNNs is proposed. This accelerator can effectively leverage all of the available parallelism sources 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**


