Optimizing Frequency Domain Implementation of CNNs on FPGAs

Hanqing Zeng, Ren Chen, Viktor K. Prasanna

Computer Engineering Technical Report Number CENG-2017-03

Ming Hsieh Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, California 90089-2562

July 2017

Optimizing Frequency Domain Implementation of CNNs on FPGAs

Hanqing Zeng, Ren Chen, Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, California 90089
{zengh,renchen,prasanna}@usc.edu

ABSTRACT

Convolutional neural networks (CNNs) have become popular for many machine learning applications. Recently, accelerators either in ASIC or using FPGA have been proposed to achieve low latency for classification as well as reduce the energy consumption. However, with increasing variety of deep learning models and proliferation of FPGA devices and families, realizing a high-performance design becomes challenging. In this paper, we propose an algorithm-architecture co-design methodology based on the computational characteristics of CNN models and the features of underlying hardware to realize high performance designs. To speed up various CNN models, our methodology consists of two levels of optimization for convolution in the frequency domain: (1) At the algorithm level, we reduce the total number of operations. We propose a new technique called *Concatenate and Pad (CaP)*. By applying it with its dual operation *Overlap and Add (OaA)*, we can match various image sizes to any given FFT size. Our hybrid algorithm based on *native* FFT, *OaA*-FFT and *CaP-OaA*-FFT achieves significant computation reduction for a wide range of CNNs. (2) At the architecture level, our goal is to maximize the utilization of limited on-chip resources. We develop highly efficient hardware building blocks to support our algorithmic optimizations. We develop a simplified performance model which captures the constraints of a given hardware device such as external bandwidth, logic resources and on-chip memory. The performance model leads to a reduced design space which is explored using bounded parameter scan to realize a high performance FPGA design for a CNN. We illustrate our methodology by proposing two FPGA designs for AlexNet using throughput and latency as performance metrics. Experimental results show that the two designs achieve throughput of 163.4 *GOPS* and latency of 12.4 *ms* respectively, on the Intel HARP heterogeneous platform. Our designs significantly improve the state-of-the-art implementations of AlexNet on FPGAs.

KEYWORDS

Convolutional Neural Networks; Fast Fourier Transform; FPGA; Overlap and Add; Optimization

1 INTRODUCTION

Convolutional Neural Networks (CNNs) are gaining increasing popularity in a variety of applications including computer vision, signal processing and natural language processing [14, 24, 26]. To achieve better accuracy, deeper and larger CNN models are being explored. This leads to high throughput requirement [19]. To embed CNNs into real time applications, single image classifcation time is critical. This results in low latency requirement [4]. Furthermore, with the progress in deep learning, there is increasing diversity in terms of CNN models and accelerators. Therefore, a flexible accelerator design for large scale CNNs to realize high throughput and low latency becomes a crucial problem to address.

In recent years, many CNN accelerators have been implemented on GPU [3, 13, 22], ASIC [10] and FPGA [20, 25, 27]. Energy efficiency, reconfigurability and unprecedented logic density make FPGAs attractive for high performance CNN processing. Yet difficulties exist to fully make use of such platforms in deep learning applications. The first challenge is to deal with the large problem size. Motivated by the huge amount of operations required by spatial convolution (the convolution that performs sliding window operation of the kernel over the image), alternatives such as Winograd transform [15] and frequency domain convolution [18] have been proposed. Recent implementations of these algorithms have shown promising results [5, 28]. The other challenge lies in large variance of the parameters among different convolution layers, such as the image size, kernel size and number of feature maps. For resource efficiency, the same hardware module is reused for different layers. Yet some hardware modules don't have the flexibilty to perform well under the computational requirements of different CNN layers. A design containing such modules is very inefficient, because large amounts of runtime reconfiguration will be required by the many layers in a deep CNN. In addition, the design space of feasible hardware designs over all the layers can be huge. Limited design space exploration in [16] takes as long as 5 hours on two desktops.

In this work, we improve the performance of CNNs running on FPGAs by identifying designs that perform well on all convolution layers. We propose an optimization flow with algorithm-architecture co-design, based on convolution in frequency domain. To deal with the variance of image sizes and kernel sizes, we propose a dual operation of *Overlap and Add (OaA)* [12], called *Concatenate and Pad (CaP)*. We prove that frequency domain convolution using *CaP-OaA* performs equally well as the native frequency domain convolution in terms of number of operations. Yet the *CaP-OaA* approach needs *no* hardware reconfiguration for different layers, while the native frequency domain convolution incurs significant reconfiguration overhead for every layer. We further propose a hybrid algorithm which combines *OaA* and *CaP-OaA* with the native frequency domain convolution. The hybrid algorithm is able to specifically optimize either throughput or latency for a given CNN. To address the variance in the number of feature maps, we go into the architecture level by proposing an efficient design space

exploration algorithm. We first develop high performance hardware modules to support our hybrid convolution algorithm, and derive an analytical performance model for our architecture. Our design space exploration algorithm is based on the observation that optimal configuration of a single layer is a good indication for the optimal configuration of the entire network. Thus the design space for a complete CNN can be bounded by the local optimum of each individual layer.

The main contributions of this work are:

- We analyze the complexity of frequency domain convolution using *Overlap and Add (OaA)* specifically in the CNN context. We propose a dual operation of *OaA* called *Concatenate and Add (CaP)*, and design a hybrid algorithm that combines *OaA*, *CaP* and the native frequency domain convolution.
- We show that the hybrid algorithm significantly reduces the computation complexity for a wide range of CNN models. Compared with spatial convolution, it results in 57.5% reduction in the number of operations for AlexNet, without needing for runtime reconfiguration.
- We derive a comprehensive performance model which takes into account details of the CNN and hardware constraints such as peak bandwidth to external memory, and available on-chip memory, logic and DSP resources. We propose a methodology to significantly reduce the design space of this performance model.
- We develop a highly-optimized hardware design, with each individual modules in the form of parameterized soft IP cores. The building blocks such as run-time configurable 2D FFT module can be easily configured for different CNNs and FPGAs.
- We implement our design on the Intel HARP heterogeneous platform, where the FPGA is responsible for our hybrid algorithm, and the CPU performs other light-weight operations such as overlapping and padding.
- We demonstrate our design methodology by developing two designs for AlexNet, one optimized for high throughput and the other for low latency respectively. The designs show throughput of 163.4 *GOPS* and latency of 12.4 *ms* on the Intel HARP platform [1].

2 BACKGROUND AND RELATED WORK

State-of-the-art convolutional neural networks for image classification can achieve very high accuracy over a thousand input categories [14, 24]. A CNN is formed by stacking various layers together. A typical design includes four types of layers: convolution layers, rectified linear unit (ReLU) layers, pooling layers and fully connected layers. Convolution layers extract features out of the input 2D images. Pooling layers enable the network to gradually shift its focus from local features to global features. ReLU layers add non-linearity into the network. Finally, the partially processed features are fed into fully connected layers to complete the classification.

2.1 Convolution in Frequency Domain

Let the input matrix to a convolution layer be I, whose dimension is $f_{in} \times l_{img} \times l_{img}$; the kernel of the layer be K, whose dimension is $f_{out} \times f_{in} \times l_{kern} \times l_{kern}$. The convolution layer does the convolution operation over the last two dimensions of I and K, and accumulates over f_{in} dimension. As a result, the output to that layer is of dimension $f_{out} \times l'_{img} \times l'_{img}$.

Spatial convolution basically performs sliding window operation. Alternatively, a more efficient algorithm is by computing in the frequency domain [18]. Convolution in the spatial domain is equivalent to Hadamard product ($\circ$) in the frequency domain. The algorithm is summarized by *Equation 1*, where $\boldsymbol{\mathcal{F}}$ is Fourier transform, and $\boldsymbol{\mathcal{F}}^{-1}$ denotes inverse Fourier transform.

$$I * K = \boldsymbol{\mathcal{F}}^{-1}\Big(\boldsymbol{\mathcal{F}}(I) \circ \boldsymbol{\mathcal{F}}(K)\Big) \tag{1}$$

There are two noteworthy points. First of all, typical convolution is companied by stride S. Striding is applied directly to the sliding window in spatial convolution, and can be achieved in the frequency domain by subsampling the output matrix after the inverse Fourier transform. Secondly, the I and K matrix need to be padded to the same size before the Fourier transform. This ensures that the Hadamard product is valid. Since the input images to the first few convolution layers are quite large, computing FFT on the full I matrix is not always preferable. We analyze the frequency domain convolution with various image sizes in *Section 3*.

2.2 Overlap and Add (*OaA*)

To avoid the operation on a large matrix, the idea is to divide the original image matrix into smaller tiles. After performing frequency domain operation on each tile with the kernel, the resulting tiles are combined to form the final output [2].

The following describes the basic procedures of computing FFT using Overlap and Add (*OaA*) technique. Given an $l_{img} \times l_{img}$ input image I and an $l_{kern} \times l_{kern}$ kernel K, we convolve them using N point 2D FFT units (subject to $N > l_{kern}$). First, we divide I into tiles $T^{in}_{i,j}$ of size $l_{tile} \times l_{tile}$ (where $l_{tile} + l_{kern} - 1 = N$). Then after zero padding to size $N \times N$, we can compute the intermediate output tiles using *Equation 2*. To get the final matrix R, we move the output tiles $T^{out}_{i,j}$ so that pixel $(0,0)$ of each tile is overlapped with pixel $(i \cdot l_{tile}, j \cdot l_{tile})$ of R. Each pixel in R is the sum of corresponding pixels in $R_{i,j}$. The operation is summarized by *Equation 3*.

$$T^{out}_{i,j} = \boldsymbol{\mathcal{F}}^{-1}\Big(\boldsymbol{\mathcal{F}}(T^{in}_{i,j}) \circ \boldsymbol{\mathcal{F}}(K)\Big) \tag{2}$$

$$R[p][q] = \sum_{i,j} \Big(T^{out}_{i,j}[p - i \cdot l_{tile}][q - j \cdot l_{tile}]\Big) \quad \text{where} \quad \begin{cases} 0 \leqslant p - i \cdot l_{tile} < l_{tile} \\ 0 \leqslant q - j \cdot l_{tile} < l_{tile} \end{cases} \tag{3}$$

In the above equations, the square brackets $[*][*]$ indicates the pixel index within the 2D matrix and all indices i, j, p, q starts from 0.

The Overlap and Add technique together with convolution in frequency domain reduces the complexity of spatial convolution from $\mathcal{O}(l^2_{img} \cdot l^2_{kern})$ to $\mathcal{O}(l^2_{img} \cdot \log l_{kern})$ [12] for a single frame. Further analysis of such complexity in the CNN context will be conducted in *Section 3*.

Table 1: AlexNet Specification

Layer	Image Size	Kernel Size	Input Features
CONV1	224×224	11×11	3 (RGB)
CONV2	55×55	5×5	96
CONV3	27×27	3×3	256
CONV4	13×13	3×3	384
CONV5	13×13	3×3	384

2.3 CNN Models

ImageNet competition in recent years has produced many well-designed CNNs for image classification [14, 24]. Popular designs use multiple convolution layers which gradually extract features from local to global scale when proceeding to deeper layers. The computation is conducted on data of high dimensions (images: $Batch \times f_{in} \times l_{img} \times l_{img}$; kernels: $f_{out} \times f_{in} \times l_{kern} \times l_{kern}$), where $Batch$ specifies the size for batch processing. From the specifications shown in *Table 1* , we observe how the network extracts out more and more features when images are transformed to lower and lower resolution.

2.4 CNN Accelerators

To speedup the inferencing of large scale CNNs, much effort has been spent with the help of dedicated hardware accelerators. Projects in [16, 23, 25, 27] target at spatial convolution. They optimize the hardware by applying techiniques such as loop tiling/unrolling and data mapping. Little optimization is done from the computation complexity point of view, and the design space of their designs can be huge. On the other hand, works in [5, 9, 28] attempt to apply Winograd transform or frequency domain convolution to reduce the number of operations. The promising results shown by them inspire further study into these algorithms. Our work proceeds from [28] to further optimize frequency domain convolution on FPGAs with a novel algorithm-architecture co-design.

3 ALGORITHM LEVEL OPTIMIZATION

Performing convolution in frequency domain will reduce the computation complexity from $\mathcal{O}\left(l_{img}^2 \cdot l_{kern}^2 \cdot f_{in} \cdot f_{out}\right)$ to $\mathcal{O}\left(l_{img}^2 \cdot f_{in} \cdot f_{out} + l_{img}^2 \cdot \log l_{img} \cdot (f_{in} + f_{out})\right)$ for inference [18]. This analysis is based on the assumption that the FFT size is equal to the image size, and FFT is conducted on the entire input image. We refer to the convolution under such assumption as the *native* frequency domain convolution. We should note that the above complexity equation sets the theoretical lower bound, under the assumption hard to be satisfied by realistic hardware. Generally, variance in the image sizes is very large for different layers, and the target hardware does not have such high flexibility for FFT of various sizes. Efficient FFT hardware designs such as the radix-x based architecture [11] can only support limited number of distinct FFT sizes (e.g., sizes of x^i) under reasonable amount of runtime reconfiguration. In addition, even if the hardware is able to support all the FFT sizes required by different layers, using such hardware results in low resource efficiency, since the gap between the maximum and minimum image sizes is very large. The problem of hardware runtime inflexibility is addressed by the *OaA* technique, as tiling mitigates image scaling across layers. Although recent FPGA implementation of CNNs using *OaA* has shown good performance [28], a detailed analysis of this technique specifically in the CNN context is needed to optimize the implementation.

This section proposes two general algorithmic optimizations which reduce the computation complexity of frequency domain convolution at low hardware cost. Starting from the *OaA* complexity analysis when applied to CNN, we first propose a hybrid algorithm combining frequency domain convolution with and without *OaA*. We show that this approach performs well for both large and small images, and requires only small FFT sizes. Then we further improve the algorithm's flexibility and performance by proposing a dual operation of *OaA*, which we call *Concatenate and Pad (CaP)*. Due to the limited number of FFT sizes supported in hardware, we propose a second hybrid algorithm combining *OaA* and *OaA-CaP* with the native frequency domain convolution. It achieves further computation complexity reduction and hardware simplification without requiring any FFT hardware reconfiguration. The proposed algorithm is a key component in our algorithm-architecture co-design.

3.1 Optimization for Images of Different Scales

Image scaling is common in CNNs. *Table 1* shows that from the first convolution layer to the last, images shrink more than 17 times in AlexNet. The kernel sizes are reduced accordingly. Previous study [28] dealt with images of different scales by partitioning the large images using *OaA*. We further analyze the computation complexity of the *OaA* approach when applied to CNNs, and show how FFT of small sizes can reduce the number of operations significantly for all convolution layers.

By using *OaA*, we perform FFT, Hadamard product and IFFT on each input tile. Following the notation in *Section 2.2*, the total number of operations needed by convolving a full image can be calculated as:

$$O_{total} = \left(O_{tile,FFT} + O_{tile,Hadamard} + O_{tile,IFFT} + O_{tile,OaA}\right) \cdot \left\lceil \frac{l_{img} + l_{kern} - 1}{N - l_{kern} + 1} \right\rceil^2 \tag{4}$$

where:

$$\begin{aligned} O_{tile,FFT} &= C_1 \cdot N^2 \cdot \log N \cdot f_{in} \\ O_{tile,IFFT} &= C_2 \cdot N^2 \cdot \log N \cdot f_{out} \\ O_{tile,Hadamard} &= C_2 \cdot N^2 \cdot f_{in} \cdot f_{out} \\ O_{tile,OaA} &= C_3 \cdot N \cdot (l_{kern} - 1) f_{out} \end{aligned}$$

And C_* are constants for the cost of addition or multiplication.

Previous work [12, 28] note that this equation is asymptotically bounded by $\mathcal{O}(l_{img}^2 \cdot \log N)$, and considers it to be superior to the native FFT whose complexity is $\mathcal{O}(l_{img}^2 \cdot \log l_{img})$. We argue that this conclusion is valid only for single frame convolution and does not apply to CNNs when f_{in}, f_{out} come into picture. Most of the time except for the first layer, f_{in}, f_{out} are at least as large as N or l_{img}, so for the domain of the CNN problem, $O_{tile,FFT}$ and

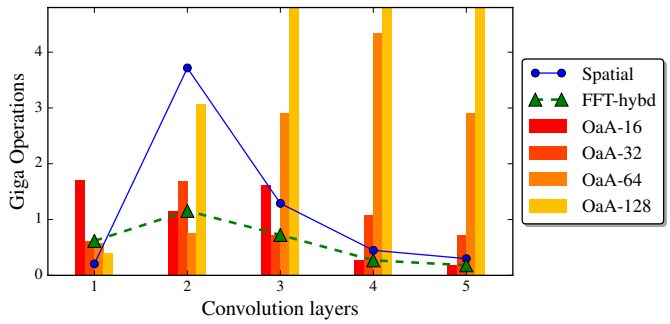

Figure 1: AlexNet: Effect of FFT size on total number of operations. "FFT-hybd" is produced by the hybrid algorithm using native FFT convolution and FFT convolution with *OaA*.

$O_{tile,IFFT}$ are much less than $O_{tile,Hadamard}$ ($\log N \ll f_{in}, f_{out}$). Thus, the complexity of *OaA* when applied to CNN should instead be $\mathcal{O}(l_{img}^2 \cdot f_{in} \cdot f_{out})$. *Equation 4* can be approximated as:

$$O_{total}^{approx} \approx C_2 \cdot f_{in} \cdot f_{out} \cdot \left(\frac{l_{img} + l_{kern} - 1}{1 - (l_{kern} - 1)/N} \right)^2 \tag{5}$$

The domain of N is: $l_{kern} - 1 < N \leqslant l_{img} + l_{kern} - 1$.

Note that $\frac{\partial O}{\partial N} = -C_4 \cdot \frac{N}{(N - l_{kern}+1)^3}$; Thus we observe that: (1) number of operations decreases as N increases; (2) the benefit of increasing N diminishes when N becomes sufficiently larger than $(l_{kern} - 1)$. By *Observation (1)*, N should not fall in the left end region of its domain. By *Observation (2)*, N should not be in the right end region. In this case, there is little benefit to increase N to match the image size when l_{img} is very large. Thus, it can be concluded that the middle ground between $(l_{kern} - 1)$ and $(l_{img} + l_{kern} - 1)$ is the most suitable region for N.

The motivation for *OaA* is to avoid using large-size FFT when dealing with large images. Thus, for small images, we can simply apply native frequency domain convolution. Then the number of operations equals to $C_2 \cdot f_{in} \cdot f_{out} \cdot (N')^2$, where N' is the smallest FFT size such that $N' \geqslant l_{img} + l_{kern} - 1$. We can use the hybrid algorithm that uses the frequency domain convolution with *OaA*, as well as the native frequency domain convolution.

Figure 1 illustrates our analysis for AlexNet. By using the *OaA* technique, FFT with small variance in sizes can process quite well images of different scales. The green line shows the performance of the hybrid frequency domain algorithm. The first two layers use *OaA* based convolution for 32, 16 points, and the last three layers use native frequency convolution for 32, 16, 16 points respectively. Compared with spatial convolution, our approach reduces the number of operations by 50.6%, using only two small FFT sizes of 16 and 32.

This section proposes an optimization by applying *OaA* in the CNN context. The optimization greatly relieves the reconfiguration requirement for FFT hardware, while preserving the native frequency domain convolution's advantage in terms of computation complexity. We will show in the next section our second optimization, which further reduces the number of operations and completely eliminates the need for hardware reconfiguration.

3.2 Optimization for Exploiting Limited Number of FFT Sizes

The analysis of *Section 3.1* on both *OaA* based and the native FFT neglects the wasted computation incurred by image padding to match FFT sizes. However, this may result in sub-optimality implementations due to the limitations of realistic FFT hardware. Without reasonable amount of reconfiguration, high performance FFT hardware is only able to support limited number of different FFT sizes. As an example, consider the case of radix-4 FFT design. The FFT sizes of $4, 16, 64, 256$ need to handle images of any sizes ranging from ten to several hundred.

The key problem is mismatch between image sizes and FFT sizes. Since we have little control over the FFT sizes under the hardware constraints, we address this problem by changing image sizes through input data reshaping.

Algorithm 1 Operations by a convolution layer

```
1:  procedure ConvLayer
2:      ▷ I is the input image array
3:      ▷ K is the kernel array
4:      for i_b = 1 to Batch do
5:          for i_out = 1 to f_out do
6:              for i_in = 1 to f_in do
7:                  T^in[i_b][i_out][i_in] ← I[i_b][i_in] * K[i_out][i_in]
8:              end for
9:              T^out[i_b][i_out] ← Σ_{j=1}^{f_in} T^in[i_b][i_out][j]
10:         end for
11:     end for
12: end procedure
```

The operation of the CNN can be summarized by *Algorithm 1*. The operation under concern is the convolution in *line 7*. For the nested loop, we would like to manipulate one of the *Batch*, f_{out}, f_{in} dimensions so that l_{img} of image I can be resized. Index i_b takes effect on I and is independent of K, so *Batch* is the appropriate dimension. We refer to the manipulation of *Batch* and l_{img} dimensions as the *reshaping* of input data.

We propose a dual operation of *OaA*, *Concatenate and Pad (CaP)*, and apply it together with *OaA* to have the flexibility on the l_{img} dimension. We need to ensure that reshaping does not affect the output result of convolution, and introduces negligible overhead.

We define the *CaP* operation as follow. Given a set of d^2 images I of equal size $l_{img} \times l_{img}$, we arrange the images in a $d \times d$ mesh, with x pixels of zero value between the vertically or horizontally adjacent images. *CaP* outputs a large image I^{CaP} by concatenating I and the zero pixels in between. x is defined as the size of zero padding. *Figure 2* illustrates such process. The following theorem shows that by choosing appropriate x, we can produce the correct output of convolution:

Theorem 3.1. *For a given kernel K of size $l_{kern} \times l_{kern}$, the convolution using K on a set of images I is identical to performing convolution using K on I^{CaP} iff the padding x is at least $l_{kern} - 1$.*

Proof. The proof of $x_{min} = l_{kern} - 1$ can be done from the perspective of spatial or frequency domain convolution. We prove

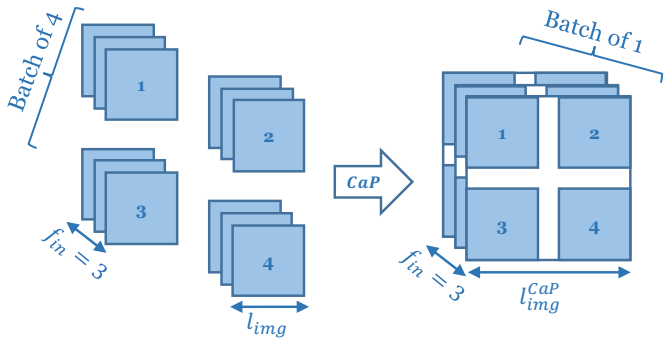

Figure 2: Illustration of the *CaP* algorithm.

the theorem by *OaA* in frequency domain to illustrate the dual operation.

Define $\ominus$ as the partition operation by OaA. $\ominus(I, l_{tile})$ outputs a set of tiles after partitioning I into $l_{tile} \times l_{tile}$ tiles T. Define $\oplus^k$ as the concatenation operation. $\oplus^k(I)$ outputs a large image by concatenating the 2D mesh of $\{I\}$ with k pixels of *margin* between adjacent images; $\oplus^{-k}(I)$ outputs an image by concatenating the 2D mesh of I with k pixels of *overlap* between adjacent images. Define T^p as the matrix by zero-padding T with p pixels after the last pixels along the two dimensions.

Performing convolution after *CaP* can be expressed by the following set of equations:

$$
\begin{aligned}
I^{CaP} &= \oplus^{x}(I) \\
\{I^x\} &= \ominus^{l_{img}+x}(I^{CaP}) \\
T &= \boldsymbol{\mathcal{F}}^{-1}\left(\boldsymbol{\mathcal{F}}((I^x)^{l_{kern}-1}) \circ \boldsymbol{\mathcal{F}}(K^{x+l_{img}-1})\right) \\
&= I^x * K = (I * K)^x = T_*^x \\
I^{CaP} * K &= \oplus^{-(l_{kern}-1)}(T) = \oplus^{-(l_{kern}-1)}(T_*^x) \\
&= \oplus^{x-(l_{kern}-1)}(T_*) \\
&= \oplus^{x-(l_{kern}-1)}(I * K)
\end{aligned}
$$

From the $\oplus^{x-(l_{kern}-1)}$ operator in the last step, it can be concluded that convolution on I^{CaP} is identical to convolution on the set of I, *iff* $x - (l_{kern} - 1) \geqslant 0$. Thus, the minimum value of x to avoid aliasing between adjacent images is $(l_{kern} - 1)$. ■

Based on *Theorem 3.1*, we can *CaP* the images from a batch so that input of $Batch \times f_{in} \times f_{out} \times l_{img}^2$ is reshaped to $\frac{Batch}{d^2} \times f_{in} \times f_{out} \times (l_{img}^{CaP})^2$, where $l_{img}^{CaP} = d \cdot l_{img} + (d-1) \cdot l_{kern}$, and d is referred to as the *Batch* folding factor. We can then apply the *OaA* technique on I_{img}^{CaP} to complete the convolution. Abbreviate FFT based convolution with *OaA* as *OaA*, and FFT based convolution with *CaP* and *OaA* as *CaP-OaA*. We analyze in a convolution layer, how much improvement can be brought by *CaP-OaA* compared with *OaA* and the native approach. We assume that the hardware supports the following FFT sizes: $\mathbb{N} = [N_1, N_2, ..., N_m]$, where $N_1 < N_2 < ... < N_m$. *Theorem 3.2* evaluates the optimal complexity for *CaP-OaA*. Let $\mathbf{K}_i = \frac{l_{img}+l_{kern}-1}{N_i-(l_{kern}-1)}$.

Theorem 3.2. *The minimum computation complexity of CaP applied to a CNN is* $C_2 \cdot f_{in} \cdot f_{out} \cdot \mathbf{K}_m^2 \cdot N_m^2$, *where* C_2 *is some constant. To achieve such complexity, the Batch folding factor should be a multiple of* $\frac{N_m-(l_{kern}-1)}{\gcd(l_{img}+l_{kern}-1, N_m-(l_{kern}-1))}$.

Proof. Based on *Equation 4*, the complexity of *CaP-OaA* approach averaged for a single image is:

$$
\begin{aligned}
O_{CaP-OaA} &= C_2 \cdot f_{in} \cdot f_{out} \cdot \left\lceil \frac{l_{img}^{CaP} + l_{kern} - 1}{N_i - (l_{kern} - 1)} \right\rceil^2 \cdot \frac{N_i^2}{d^2} \\
&= C_2 \cdot f_{in} \cdot f_{out} \cdot \left\lceil d \cdot \mathbf{K}_i \right\rceil^2 \cdot \frac{N_i^2}{d^2}
\end{aligned}
\tag{6}
$$

The minimum computation complexity can thus be obtained by the following inequality:

$$
O_{CaP-OaA} \geqslant C_2 \cdot f_{in} \cdot f_{out} \cdot \mathbf{K}_i^2 \cdot N_i^2 \geqslant C_2 \cdot f_{in} \cdot f_{out} \cdot \mathbf{K}_m^2 \cdot N_m^2
$$

We can set d as a multiple of $\frac{N_m-(l_{kern}-1)}{\gcd(l_{img}+l_{kern}-1, N_m-(l_{kern}-1))}$ to eliminate the ceiling function, and choose $N_i = N_m$. Then $\min\{O_{CaP-OaA}\} = C_2 \cdot f_{in} \cdot f_{out} \cdot \mathbf{K}_m^2 \cdot N_m^2$.

Thus, $O_{CaP-OaA}^{min} = C_2 \cdot f_{in} \cdot f_{out} \cdot \mathbf{K}_m^2 \cdot N_m^2$. ■

Corollary 3.3. *The computation complexity of CaP-OaA is always no higher than OaA. The minimum ratio of CaP-OaA's number of operations over OaA's number of operations is* $\max_i \left(\frac{\mathbf{K}_m}{\lceil \mathbf{K}_i \rceil} \cdot \frac{N_m}{N_i}\right)^2$.

Corollary 3.4. *The computation complexity of CaP-OaA is lower than native FFT if* $\frac{N_l}{N_m} > \mathbf{K}_l$ *(where* N_l *denotes the minimum FFT size such that* $N_l > l_{img} + l_{kern} - 1$*). The minimum ratio of CaP-OaA's number of operations over native frequency domain convolution's number of operations is* $\left(\frac{\mathbf{K}_m \cdot N_m}{N_l}\right)^2$.

The corollaries can be easily proven by directly comparing the optimal *CaP-OaA* complexity with the optimal *OaA* complexity and the optimal native approach complexity. For *OaA*, since we have no further information to evaluate the effect of the ceiling function, we need to calculate $\lceil \mathbf{K}_i \rceil \cdot N_i$ for all $N_i \leqslant l_{img} + l_{kern} - 1$.

Figure 3 shows the effect of including *CaP* in the frequency domain convolution. We compare the algorithm of *CaP-OaA*, with the hybrid algorithm using native approach and *OaA*. We plot under two different FFT constraints. The blue points are under the assumption of $\mathbb{N} = \{4, 16, 64, 256\}$. The red dots show the performance in the case of more hardware support, $\mathbb{N} = \{4, 8, 16, 32, 64, 128, 256\}$. The kernel size is assumed to be 3×3. There are very few points where *CaP-OaA* is slightly worse than the native approach. Most of the time, for both FFT sequences, the extra flexibility brought by *CaP* benefits the computation complexity significantly.

One additional benefit of applying *CaP-OaA* is that it makes one-size FFT sufficient for all layers. Images in deeper layers can be *CaP*-transformed to eliminate the domain constraint of $N \leqslant l_{img} + l_{kern} - 1$ discussed in *Section 3.1*. Evaluation on AlexNet shows that the *CaP-OaA* technique reduces 57.5% of computation compared with spatial convolution, using only one-size FFT of size 32.

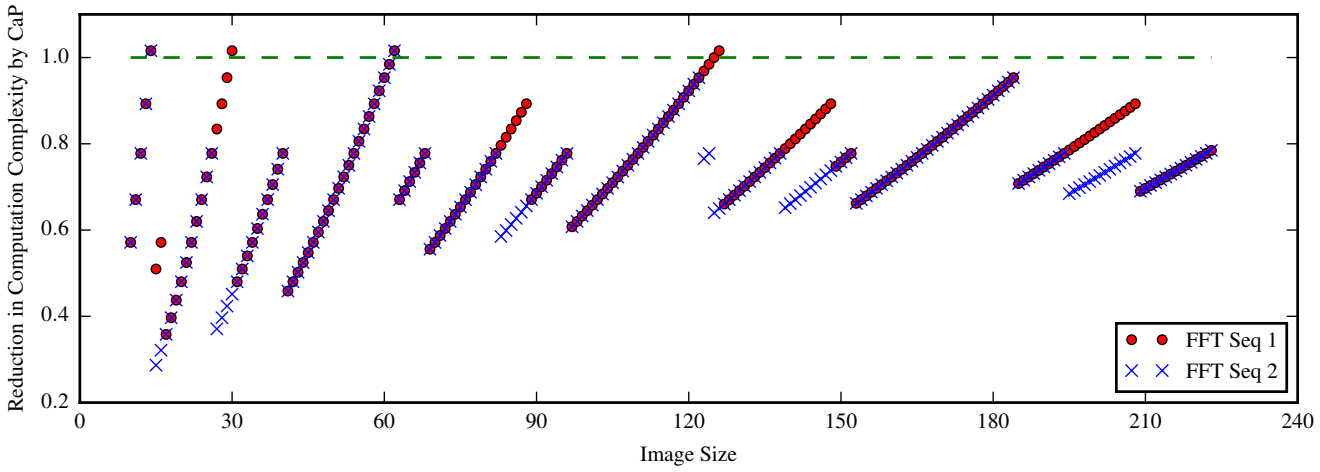

Figure 3: Effect of *CaP* optimization. y axis is calculated by dividing the number of operations of *CaP-OaA* by the number of operations of the hybrid algorithm discussed in *Section 3.1.*

3.3 Discussion on the Various Optimization Techniques

The various optimization techniques discussed so far can be viewed in a unified way. By setting *Batch* folding factor to 1, *Cap-OaA* reduces to *OaA*. By setting $N = l_{img} + l_{kern} - 1$, *OaA* reduces to the native frequency domain convolution. Note that, there is no pre-processing overhead incurred by *OaA* or *CaP*. *CaP* completely eliminates the need to reconfigure hardware when processing different layers.

We can perform optimization by an hybrid algorithm that combines all the three techniques: native approach, *OaA* and *CaP-OaA*. The three algorithms can be applied separately on different layers. Significant reduction in computation complexity can thus be achieved for arbitrary CNNs with very little hardware.

We should also notice that *CaP* is based on batch processing. In this case, although throughput is further optimized, latency may be compromised. Thus, using hybrid algorithm that uses *OaA* and native algorithm can result in a low latency design, while further inclusion of *CaP-OaA* can result in a high throughput design.

4 SYSTEM ARCHITECTURE

This section gives an overview of the architecture design on FPGA to perform the hybrid algorithm for frequency domain convolution. The architecture consists of three main modules to perform FFT, Hadamard product and IFFT operations on the image and kernel data.

4.1 Streaming 2D FFT Module

The 2D FFT on an $N \times N$ matrix involves two phases of computation. In the first phase, we perform N-point 1D FFT on the N rows of the input matrix. In the second phase, we perform N-point 1D FFT on the N columns of phase 1's output matrix. Matrix transpose operation is involved between the two phases. We design resource efficient hardware for both the 1D FFT and the matrix transpose operation.

The radix-x N-point streaming 1D FFT architecture consists of $\log_x N$ computation stages. To meet the bandwidth requirement of input data, we can fold each stage f_{FFT} times so that input parallelism of stage 0 is decreased from N words to N/f_{FFT}. Streaming permutation of size $m_i = \frac{N}{x^{i-1}}$ is required between stages i and $(i+1)$ to support folding [6]. Thus, the total memory consumption by the permutation network for the $\log_x N$ stages is: $2N + \sum m_i \approx 3N$, where the $2N$ term is due to the bit-reversal in the last stage [7].

To perform the matrix transpose of size $N \times N$ between the two phases of 2D FFT, we utilize the Streaming Permutation Network (SPN) proposed in [8]. The SPN can perform arbitrary stride permutation by the in-place permutation in time algorithm. The SPN only requires size $N \times N$ single-port memory.

Our proposed streaming 2D FFT module consists of P_{FFT} units of $N \times N$ 2D FFT. Each 2D FFT unit can be folded q_{2DFFT} times so that it contains N/q_{2DFFT} pipelines of N-point 1D FFT. Each 1D FFT pipeline can also be folded q_{1DFFT} times so that the data parallelism of each pipeline is N/q_{1D}. The total data parallelism to the 2D FFT module is $\frac{P_{FFT} \cdot N^2}{q_{1DFFT} \cdot q_{2DFFT}}$.

4.2 Multiply-and-Accumulate Module

The Hadamard product is performed by the Multiply-and-Accumulate (MAC) module. Inputs to the MAC module are the kernel and image tiles after the Fourier transform. Outputs of the MAC are the resultant tiles after the Hadamard product computation. The P_{img} image tiles and P_{kern} kernel tiles fed into the MAC module in one cycle should have the same index in the f_{in} dimension. Then after f_{in} cycles, the accumulator can complete the reduction on dimension f_{in}. Data parallelism of $P_{img} \cdot P_{kern} \cdot N^2$ is required to support the full permutation of the set of P_{img} image tiles and the set of P_{kern} kernel tiles. If input is available every cycle, the output rate of the MAC module is $P_{img} \cdot P_{kern}$ tiles per f_{in} cycles. The required input rate for image tiles is P_{img} tiles per f_{out}/P_{kern} cycles. Thus, the ratio of the input and output rate is f_{in}/f_{out}. We can easily verify that the MAC module design is work optimal.

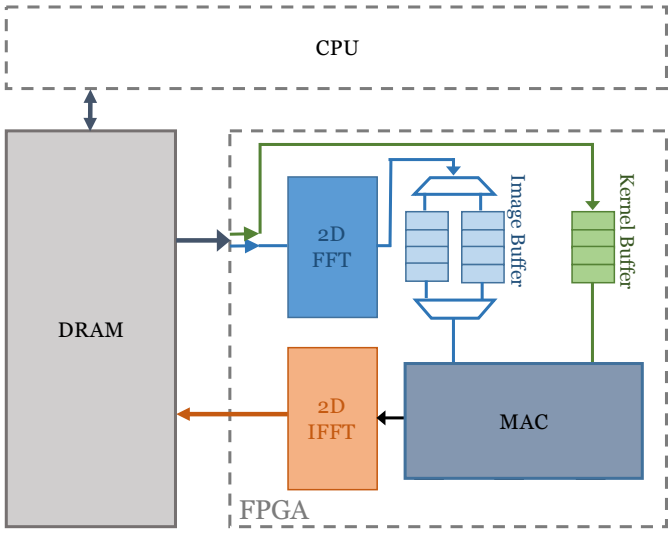

Figure 4: Overall System Design

Table 2: Summary of Resource Consumption and Throughput

	2D FFT	MAC
Logic	$2 \cdot \log_x N \cdot (\frac{N}{q_{1DFFT}} - 1) \cdot \frac{N}{q_{2DFFT}} \cdot P_{FFT} \cdot \mathbf{L}$	$P_{img} \cdot P_{kern} \cdot N^2 \cdot \mathbf{L}$
Memory	$(2 \cdot \frac{3N}{f_{2DFFT}} + N^2) \cdot P_{FFT} \cdot \mathbf{D}$	——
Throughput	$\frac{P_{FFT} \cdot N^2}{q_{1DFFT} \cdot q_{2DFFT}}$	$P_{img} \cdot P_{kern} \cdot N^2 / f_{in}$

We can also fold the MAC module by q_{MAC} so that the granularity is not an entire N^2 matrix. The data parallelism after folding is $P_{img} \cdot P_{kern} \cdot N^2 / q_{MAC}$.

4.3 Overall System Design

The full system design is shown in *Figure 4*. The FFT modules in dashed lines before the kernel buffers will be discussed in *Section 5.1*. The images tiles are processed by the 2D FFT module before they are stored in the image buffer. The output tiles are transformed back to spatial domain by the 2D IFFT modules before they are written to external memory. We use double buffering technique to hide the memory latency of image and kernel tiles.

The resource consumption and throughput for the FFT and MAC modules are summarized in *Table 2*. Constant **L** refers to the logic consumption for one complex number adder and multiplier. **D** refers to the memory consumption for one complex number.

5 ARCHITECTURE LEVEL OPTIMIZATION

In this section, we proceed to the optimization at the architecture level so that we can perform the most efficient mapping for our optimized algorithm, given any CNN and target FPGA. The challenges are two-folded. On the one hand, the various kinds of hardware modules as well as the ample resources on chip create a huge design space to explore. On the other hand, since the same hardware is deployed for accelerating all the convolution layers of a CNN, it is non-trivial to identify a global optimal configuration for the entire CNN.

As a simple example, we consider the input/output rate requirement for an arbitrary architecture designed for the convolution layer. If the input data is of shape $f_{in} \times l_{img} \times l_{img}$ and the output is of shape $f_{out} \times l'_{img} \times l'_{img}$, then regardless of the underlying hardware, the required ratio of input to output data rate should be approximately f_{in}/f_{out}. However, such ratio differs greatly across layers. For AlexNet, there is a 30x variation between the first and the last layer.

Given the complexity of the problem, we derive an analytic performance model for the architecture described in *Section 4*. We formulate our problem as a non-linear optimization problem and propose a technique to greatly reduce the design space to be explored.

5.1 Data Flow

There have been much discussion on how to design the data flow so that memory reuse or locality can be exploited to a maximum extent. Unlike the case of spatial convolution where memory access pattern in the deep nested loop is complicated, our approach using frequency domain convolution simplifies the memory design by the feature of the algorithm. *CaP* and *OaA* are analogous to the loop tiling and unrolling [16] in spatial convolution. Using these techniques, we can scale up or down the tile and image sizes based on the available hardware resources. The key difference from spatial convolution is the Hadamard product. It conducts element-wise operation on the matrices, meaning that all the loop carried dependencies are automatically eliminated. This property ensures that massive parallelism can be exploited on the FPGA device.

The priority for designing our data flow is to exploit as much parallelism as possible for the MAC module. During inference, the kernel data is unchanged, one option is to directly store the kernels in frequency domain and transfer them to FPGA. This saves FFT computation on the kernel data. Thus, we have more hardware resource budget for the MAC module to improve the overall system throughput. However, the above approach has negative impact on the system bandwidth. Suppose the convolution in frequency domain uses N-point FFT. Then if we transfer to hardware the kernels in frequency domain, instead of the original kernels in spatial domain, the total data communication on kernels will increase from $\mathcal{O}(l_{kern}^2)$ to $\mathcal{O}(N^2)$. Depending on the value of N, l_{kern} and the hardware bandwidth, logic resource, either kind of kernel data flow can be more suitable. Thus, there is an optional FFT module before the kernel buffer in *Figure 4*. In terms of data reuse, we use the image oriented design so that image reuse is maximized. To consume one image buffer, we reload the kernel buffer several times to make a full pass of the kernel data.

5.2 Analysis of IO and Computation Cost

The analysis in this section is based on the assumption that kernels in frequency domain are loaded on FPGA. Then we don't have the kernel FFT modules. The analysis in this section can be easily modified for the other kernel data flow.

We consider the time period in which one image buffer is consumed completely. We call this one *round* of computation. The peak throughput of the FFT, MAC and IFFT modules can be found in *Table 2*. We denote the peak throughput as T_{FFT}, T_{MAC} and T_{IFFT} (the equations for them are summarized in *Table 2*).

The full external bandwidth (B_{sys}) is shared by FFT module (B_{FFT}), kernel buffer (B_{kern}) and IFFT module (B_{IFFT}). The amount of data transferred to FFT is $D_{FFT} = M_{img}$, and the amount of data to kernel buffer is $D_{kern} = f_{in} \cdot f_{out} \cdot N^2$, based on our data reuse scheme. The IFFT module produces $D_{IFFT} = \frac{f_{out}}{f_{in}} \cdot D_{img}$ amount of output.

The IO time t^{IO}_{round} is determined by the largest time needed by D_{FFT}, D_{kern} and D_{IFFT}. Since data is streamed from MAC, IFFT to external memory, we deal with D_{IFFT} later in the computation cost calculation. The bandwidth B_{FFT} and B_{kern} should be assigned based on the workload D_{FFT}, D_{kern} as well as the peak throughput of the subsequent FFT module. This is shown in *Equation 7*.

$$\begin{aligned} B_{FFT} &= \min\left\{\frac{D_{FFT}}{D_{FFT}+D_{kern}}(B_{sys}-B_{IFFT}), T_{FFT}\right\} \\ B_{kern} &= (B_{sys}-B_{IFFT}) - B_{FFT} \end{aligned} \tag{7}$$

The IO time is calculated as: $t^{IO}_{round} = \frac{D_{FFT}}{B_{FFT}} \geqslant \frac{D_{kern}}{B_{kern}}$.

The computation time t^{comp}_{round} is defined as the time period for consuming one image buffer by the MAC and IFFT module. It is bounded by the peak throughput of the two modules, as well as the time for loading the image buffer. The throughput or output bandwidth of the MAC and IFFT module is shown by *Equation 8*.

$$B_{IFFT} = \min\left\{\frac{D_{IFFT}}{t^{IO}_{round}}, T_{MAC}, T_{IFFT}\right\} \tag{8}$$

By using *Equation 7* as well as the relation among t^{IO}_{round}, D_{FFT} and D_{IFFT}, we express *Equation 8* in the following way:

$$B_{IFFT} = \min\left\{T_{MAC}, T_{IFFT}, \frac{f_{out}}{f_{in}}T_{FFT}, \frac{Q}{1+Q}B_{sys}\right\} \tag{9}$$

where $Q = \frac{f_{out}}{f_{in}} \cdot \frac{M_{img}}{M_{img}+f_{in}\cdot f_{out}\cdot N^2}$.

Equation 9 shows how the system throughput would be determined by the various hardware resources and the CNN specification. T_{MAC}, T_{FFT}, T_{IFFT}, B_{sys} and M_{img} captures the hardware constraints of logic resources, external bandwidth and on-chip memory. f_{in}, f_{out} and N reveal the impact of CNN specification on the hardware performance. The problem is that for different convolution layers, f_{in}, f_{out}, N differs. We can use the *CaP-OaA* technique discussed in *Section 3* to reduce or even remove the variance of N. We will deal with the variance of f_{in} and f_{out} in *Section 5.3*.

The computation time for one *round* is given by: $t^{comp}_{round} = \frac{D_{IFFT}}{B_{IFFT}}$.

In the above design, the image buffer consuming rate never exceeds the producing rate. Thus, the total time of a *round* is $t_{round} = t^{comp}_{round}$. For the double-buffering to completely hide the memory latency, $\min\left\{T_{MAC}, T_{IFFT}\right\} \leqslant \min\left\{\frac{f_{out}}{f_{in}}T_{FFT}, \frac{Q}{1+Q}B_{sys}\right\}$ has to be satisfied.

For the other kind of design containing the kernel FFT module, corresponding changes need to be made for the above equations.For example, D_{kern} is changed to $f_{in} \cdot f_{out} \cdot l^2_{img}$. We can derive its performance model using the similar idea as the above discussion.

5.3 Design Space Exploration

The processing of a convolution layer involves several *round*s. Total time for processing a convolution layer is given by:

$$t_{layer} = f_{in} \cdot \frac{(l^*_{img})^2}{M_{img}} \cdot t_{round} = \frac{f_{out} \cdot (l^*_{img})^2}{B_{IFFT}} \tag{10}$$

where l^*_{img} is image size after the padding or *Batch* folding operation by the hybrid algorithm discussed in *Section 3*.

The goal of our architecture optimization is to identify the configuration of different modules so that throughput or latency regarding the entire CNN is optimized. We define t_{CNN} as the time to process a single image by all convolution layers. For batch processing in the case of *CaP-OaA*, t_{CNN} is averaged over the d^2 images in I^{CaP}. Thus, regardless of the metric being throughput or latency, we minimize t_{CNN}. We formulate an optimization problem as follow:

$$\begin{aligned} \underset{\mathbb{H}}{\text{minimize}} \quad & \sum_{\text{layer } i} t^i_{layer} \\ \text{subject to} \quad & \mathbf{LOGIC}_{FFT,MAC,IFFT} \leqslant \mathbf{LOGIC}_{sys} \\ & \mathbf{MEMORY}_{FFT,MAC,IFFT} \leqslant \mathbf{MEMORY}_{sys} \end{aligned}$$

where the set of architectural parameters $\mathbb{H}$ includes N, q_{1DFFT}, $q_{1DIFFT}, q_{2DFFT}, q_{2DIFFT}, P_{FFT}, P_{IFFT}, M_{img}, M_{kern}, P_{img}, P_{kern}$, q_{MAC}. The parameters are defined in *Section 4* and *5*.

$\mathbf{LOGIC}_{FFT,MAC,IFFT}$ and $\mathbf{MEMORY}_{FFT,MAC,IFFT}$ refer to the logic or on-chip memory consumption by the FFT, MAC, IFFT modules. The terms $\mathbf{LOGIC}_{sys}$ and $\mathbf{MEMORY}_{sys}$ refer to the available logic and on-chip memory on the FPGA board.

The design space is of very high dimension. We can remove two variables by the following observation: (1) N can be chosen by our algorithm level optimization; (2) M_{kern} does not affect the performance because of the image oriented data reuse; (3) P_{img}, P_{kern} always appear in the form of $P_{img} \cdot P_{kern}$, The design space now consists of 9 variables.

The main difficulty in further simplification of this optimization problem is the min operation in the B_{IFFT} term. This min function captures what kind of hardware resource is the bottleneck of our design. The reason why we can not evaluate the min function is that, for different convolution layers, the bottleneck can be different. In other words, if our target is to optimize a single layer only, the design space will be much smaller. For a single layer optimization problem, we can prove by contradiction that there exists an optimal configuration where all the hardware resources are fully utilized, such that:

$$B_{IFFT} = T_{MAC} = T_{IFFT} = \frac{f_{out}}{f_{in}}T_{FFT} = \frac{Q}{1+Q}B_{sys} \tag{11}$$

Thus, the original optimization problem now contains three more equality constraints. The number of free variables in the design space is reduced from 9 to 6. The new design space is defined by $\mathbb{H}' = \{M_{FFT}, P_{img}, q_{1DFFT}, q_{1DIFFT}, q_{2DFFT}, q_{2DIFFT}\}$.

For the optimization problem on a complete CNN, we can use the optimum of a single convolution layer as an indication. Based on *Equation 9* and *Table 2*, we observe how different hardware modules or resource become the performance bottleneck when f_{in} and f_{out} vary with the convolution layers. For the first few layers when f_{in}, f_{out} are both small and the ratio f_{out}/f_{in} is large, T_{MAC}, $\frac{f_{out}}{f_{in}}T_{FFT}$ and $\frac{Q}{1+Q}B_{sys}$ are of higher value. In this case, T_{IFFT} will be the bottleneck. Similarly, for deeper layers, the bottleneck may change to other hardware modules.

We propose an efficient design space exploration algorithm for a complete CNN, by bounding the design space defined by $\mathbb{H}$ with the optimum for each single layer. The single layer optimum is obtained by exploring the design space defined by $\mathbb{H}'$. The procedure is shown by *Algorithm 2*.

Algorithm 2 Design Space Exploration with Indication

procedure DesignSpaceExploration
  ▷ Conduct design space exploration on a complete CNN.
  ▷ OPT stores the optimum for single layer.
  ▷ $OPT[i] \in \mathbb{H}$
  $OPT[] \leftarrow NULL$
  for $i = 0$ to Layer l **do**
    Perform design space exploration over $\mathbb{H}'$
    $OPT[i] \leftarrow$ optimum for layer i
  end for
  ▷ $Range[d]$ stores the range of value for dimension d
  $Range[] \leftarrow NULL$
  for $d = 0$ to Dimension D **do**
    $Range[d] \leftarrow [\min_i\{OPT[i][d]\}, \max_i\{OPT[i][d]\}]$
  end for
  Design space exploration of the entire CNN with the design space bounded by $Range[]$.
end procedure

The key to our algorithm is that it transforms a large design space exploration problem into limited number of sub-problems of much smaller sizes. The design space exploration algorithm implemented in C++ executes in several minutes on a desktop platform.

6 EXPERIMENTS

6.1 Experimental Setup

We choose the Intel Heterogeneous Research Platform (HARP) [1] as our primary experimental platform. The target platform with integrated CPU and FPGA is shown in Figure 5. Intel HARP is a pre-production of Intel QuickAssist QPI FPGA Platform where Intel Xeon E5-2600 v2 processor and Altera Stratix V FPGA are integrated. The data communication between CPU and FPGA is through a coherent shared memory workspace. The DRAM access granularity for FPGA is one cache line width, which is 64 bytes. The CPU and FPGA communicate through Intel QuickPath Interconnection (QPI) [29] with 6.4 GT/s theoretical bandwidth[1]. The measured peak bandwidth from the shared memory to FPGA is $5GB/s$. The

[1]Giga Transactions per second

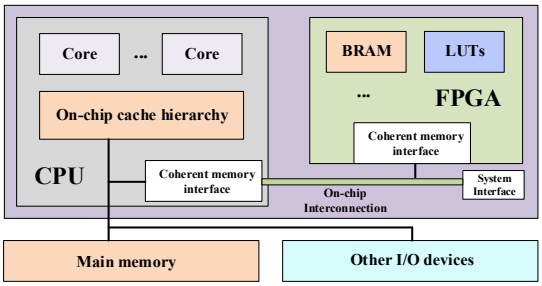

Figure 5: Target architecture integrating general purpose processors and FPGA

Altera Stratix V FPGA in HARP has 234720 ALMs and 256 DSP blocks. The FPGA on-chip memory is of size $6.25MB$.

In our experiment, we distribute the workload between CPU and FPGA as follows. The FPGA performs the FFT, Hadamard product and IFFT operations for the input image tiles and the kernel tiles. The CPU is responsible for other light-weight operations including partitioning the images into tiles and combining the tiles into the final output, according to the the *OaA* and *CaP* algorithm discussed in *Section 3*.

6.2 Low Latency or High Throughput Design

Based on our algorithm-architecture co-design, we propose two implementations of AlexNet on the HARP platform. The low-latency design chooses 64-point FFT for the first two layers, and 16-point FFT for the last three layers. It uses the hybrid algorithm described in *Section 3.1*. The high-throughput design chooses 64-point FFT for all the five layers. It uses the hybrid algorithm described in *Section 3.2*. The data precision is 16-bit fixed point. To save external memory bandwidth, we use the design containing kernel 2D FFT modules. Two sets of hardware configuration $\mathbb{H}$ are generated by our architectural design space exploration algorithm, as shown in *Table 4*. The data parallelism of the kernel 2D FFT module is 4. For the two configurations, the high throughout design requires *configuration 1*, and the low latency design requires both *configuration 1* and *2*. The low latency design switches between the two different configurations by runtime reconfiguration. The radix-4 FFT design explained in *Section 4.1* can change between 16-point and 64-point without too much overhead in time or hardware resource. In our design, it is realized by bypassing the last stage of the 64-point 1D FFT pipeline. For the MAC module, note that in *Table 4*, the data parallelism of MAC for the two configurations are the same. Thus, the only thing to do for MAC module reconfiguration is to change the memory read locations into the image and kernel buffers. Note that no runtime reconfiguration is involved in the high throughput design. This is one benefit of using the *CaP* algorithm.

6.3 Performance Evaluation

We compare our work with the state-of-the-art AlexNet implementations on FPGAs, as shown in *Table 3*. We report the throughput and latency for all the five convolution layers of AlexNet. The host

Table 3: Comparison with Other AlexNet Implementations on FPGAs

	DATE'16 [21]	FPGA'16 [25]	FPL'16 [17]	FPGA'17 [28]	Our Design	
					Low Latency	High Throughput
FPGA	Virtex-7 VC707	Stratix-V GSD8	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7
Frequency (*MHz*)	160	–	100	200	200	200
Precision	Fixed 32-bit	Fixed 8-16 bits	Fixed 8-16 bits	Floating 32-bit	Fixed 16-bit	Fixed 16-bit
DSP Utilization	2688 (96%)	234 (91.4%)	256 (100%)	224(87.5%)	232(90.6%)	256 (100%)
Logic Utilization	45K (9%)	152K (65%)	121K (52%)	201K (86%)	200K (85%)	199K (85%)
On-chip RAM	543 (53%)	1744 (71%)	1552 (61%)	2000 (80%)	2143 (85%)	2130 (85%)
Latency (*ms*)	–	20.1	12.75	40.81	**12.4**	51.2
Throughput (*GOPS*)	147.82	72.4	114.5	83.0	117.7	**163.4**

Table 4: Hardware Configuration of the Two Designs

Conf.	M_{img} (MB)	$\frac{P_{MAC} \cdot N^2}{q_{MAC}}$	$\frac{P_{FFT} \cdot N^2}{q_{1DFFT} \cdot q_{2DFFT}}$	$\frac{P_{IFFT} \cdot N^2}{q_{1DIFFT} \cdot q_{2DIFFT}}$
1	2.4	$\frac{1 \cdot 64^2}{4}$	$\frac{1 \cdot 64^2}{32 \cdot 16}$	$\frac{1 \cdot 64^2}{64 \cdot 16}$
2	2.4	$\frac{4 \cdot 16^2}{1}$	$\frac{1 \cdot 16^2}{8 \cdot 4}$	$\frac{1 \cdot 16^2}{16 \cdot 4}$

CPU computes concurrently with FPGA for other operations involved in the CNN. The CPU processing time completely overlaps with the processing time of the FPGA.

The performance of our two designs is as expected. The low latency design achieves latency of 12.4 *ms*, which is lower than all the other implementations listed in *Table 3*. The high throughput design achieves throughput of 163.4 *GOPS*, which is the best among all the listed implementations. The performance improvement comes from the two-level optimization discussed in the previous sections. The algorithmic optimization reduces the required number of operations, and the architectural optimization identifies the most suitable configuration of the hardware modules. The architectures of the high throughput and the low latency designs are very similar. The same hardware can be reused by the two designs with optional runtime reconfiguration of the 2D FFT modules.

To get deeper understanding of our implementation, we analyze the experiment data based on *Equation 9*. *Table 5* shows the FPGA throughput for each convolution layer in the case of the low latency design. The middle columns show the peak throughput achievable given the hardware configuration of FFT, MAC, IFFT, and the peak bandwidth to external memory.

Table 5: Analysis on the Bottleneck of Our System (Low Latency Design). Unit of the values: $ComplexWord/Sec$

	T_{MAC}	T_{IFFT}	$\frac{f_{out}}{f_{in}} T_{FFT}$	$\frac{Q}{1+Q} B_{sys}$	Bottleneck
CONV1	341.3	8.0	128.0	**6.1**	B_{sys}
CONV2	10.7	8.0	10.7	**4.2**	B_{sys}
CONV3	4.0	8.0	6.0	**3.2**	B_{sys}
CONV4	2.7	8.0	4.0	**1.9**	B_{sys}
CONV5	2.7	8.0	2.7	**1.8**	B_{sys}

From *Table 5*, we conclude that the external bandwidth is the bottleneck of the system. This observation is verified by the measured latency. The single image classification latency should be at least the data transfer time of image data, kernel data and output data. The time t_{mem} can be easily calculated by dividing the total amount of data with the peak memory bandwidth. The measured latency of 12.4*ms* is very close to t_{mem}, meaning that our design saturates the memory bandwidth. Given a system with higher external memory bandwidth, the performance of our design can be further improved.

7 CONCLUSION

In this paper, we presented a novel algorithm-architecture co-design methodology to improve the latency or throughput of large scale CNNs on FPGAs. Our design was based on two observations on state-of-the-art CNNs – large variance in image sizes l_{img}, and large variance in the number of feature maps f_{in} and f_{out}. Our algorithm optimization was based on the *OaA* technique applied to frequency domain convolution. To address the variance of l_{img}, we proposed a dual operation of *OaA* called *Concatenate and Pad (CaP)*. This technique reduces the computation complexity of convolution, and eliminate the hardware overhead for runtime reconfiguration. To address the variance of f_{in} and f_{out}, we proposed a system design to support our algorithmic optimization. We further formulated the analytical performance model for our system. We derived an efficient design space exploration algorithm by reducing the number of variables using the optimum related with a single convolution layer.

We demonstrated our co-design methodology by implementing two designs of AlexNet on the Intel HARP platform. Our low-latency implementation achieved latency of 12.4 *ms*, and our high-throughput implementation achieved throughput of 163 *GOPS*. Our implementations outperform state-of-the-art implementations of AlexNet on FPGAs significantly.

In the future, we will further generalize the co-design methodology to support wider range of CNNs and FPGAs. We will implement techniques such as layer partitioning to handle the cases where on-chip memory is not large enough for a large f_{in} value. We will also compare our work with GPU and multi-core implementations.

REFERENCES

[1] 2015. Intel Inc. Xeon+FPGA Platform for the Data Center. (2015). https://www.ece.cmu.edu/calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf

[2] 2017. Overlap-add method. (2017). "https://en.wikipedia.org/wiki/Overlap-add_method"

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. CoRR abs/1603.04467 (2016). http://arxiv.org/abs/1603.04467

[4] B. Ahn. 2015. Real-time video object recognition using convolutional neural network. In 2015 International Joint Conference on Neural Networks (IJCNN). 1–7. DOI: https://doi.org/10.1109/IJCNN.2015.7280718

[5] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. 2017. An OpenCL™Deep Learning Accelerator on Arria 10. (2017), 55–64. DOI: https://doi.org/10.1145/3020078.3021738

[6] R. Chen, H. Le, and V. K. Prasanna. 2013. Energy efficient parameterized FFT architecture. In 2013 23rd International Conference on Field programmable Logic and Applications. 1–7. DOI: https://doi.org/10.1109/FPL.2013.6645545

[7] R. Chen, N. Park, and V. K. Prasanna. 2013. High throughput energy efficient parallel FFT architecture on FPGAs. In 2013 IEEE High Performance Extreme Computing Conference (HPEC). 1–6. DOI: https://doi.org/10.1109/HPEC.2013.6670343

[8] Ren Chen, Sruja Siriyal, and Viktor Prasanna. 2015. Energy and Memory Efficient Mapping of Bitonic Sorting on FPGA. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15). ACM, New York, NY, USA, 240–249. DOI: https://doi.org/10.1145/2684746.2689068

[9] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi. 2016. Caffeinated FPGAs: FPGA Framework For Convolutional Neural Networks. ArXiv e-prints (Sept. 2016). arXiv:cs.CV/1609.09671

[10] Clément Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello. 2010. Hardware accelerated convolutional neural networks for synthetic vision systems. In Proceedings of 2010 IEEE International Symposium on Circuits and Systems. IEEE, 257–260.

[11] Mario Garrido, J. Grajal, M. A. Sánchez, and Oscar Gustafsson. 2013. Pipelined Radix-2K Feedforward FFT Architectures. IEEE Trans. Very Large Scale Integr. Syst. 21, 1 (Jan. 2013), 23–32. DOI: https://doi.org/10.1109/TVLSI.2011.2178275

[12] Tyler Highlander and Andres Rodriguez. 2016. Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add. CoRR abs/1601.06815 (2016). http://arxiv.org/abs/1601.06815

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14). ACM, New York, NY, USA, 675–678. DOI: https://doi.org/10.1145/2647868.2654889

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems. 1097–1105.

[15] Andrew Lavin. 2015. Fast Algorithms for Convolutional Neural Networks. CoRR abs/1509.09308 (2015). http://arxiv.org/abs/1509.09308

[16] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17). ACM, New York, NY, USA, 45–54. DOI: https://doi.org/10.1145/3020078.3021736

[17] Yufei Ma, N. Suda, Yu Cao, J. s. Seo, and S. Vrudhula. 2016. Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA. In 2016 26th International Conference on Field Programmable Logic and Applications (FPL). 1–8. DOI: https://doi.org/10.1109/FPL.2016.7577356

[18] Michaël Mathieu, Mikael Henaff, and Yann LeCun. 2013. Fast Training of Convolutional Networks through FFTs. CoRR abs/1312.5851 (2013). http://arxiv.org/abs/1312.5851

[19] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric Chung. 2015. Accelerating Deep Convolutional Neural Networks Using Specialized Hardware. (February 2015). https://www.microsoft.com/en-us/research/publication/accelerating-deep-convolutional-neural-networks-using-specialized-hardware/

[20] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16). ACM, New York, NY, USA, 26–35. DOI: https://doi.org/10.1145/2847263.2847265

[21] A. Rahman, J. Lee, and K. Choi. 2016. Efficient FPGA acceleration of Convolutional Neural Networks using logical-3D compute array. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE). 1393–1398.

[22] Dominik Scherer, Hannes Schulz, and Sven Behnke. 2010. Accelerating Large-scale Convolutional Neural Networks with Parallel Graphics Multiprocessors. In Proceedings of the 20th International Conference on Artificial Neural Networks: Part III (ICANN'10). Springer-Verlag, Berlin, Heidelberg, 82–91. http://dl.acm.org/citation.cfm?id=1886436.1886446

[23] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 1–12. DOI: https://doi.org/10.1109/MICRO.2016.7783720

[24] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR abs/1409.1556 (2014). http://arxiv.org/abs/1409.1556

[25] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16). ACM, New York, NY, USA, 16–25. DOI: https://doi.org/10.1145/2847263.2847276

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. CoRR abs/1409.4842 (2014). http://arxiv.org/abs/1409.4842

[27] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. 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). ACM, New York, NY, USA, 161–170. DOI: https://doi.org/10.1145/2684746.2689060

[28] Chi Zhang and Viktor Prasanna. 2017. Frequency Domain Acceleration of Convolutional Neural Networks on CPU-FPGA Shared Memory System. (2017), 35–44. DOI: https://doi.org/10.1145/3020078.3021727

[29] Dimitrios Ziakas, Allen Baum, Robert A Maddox, and Robert J Safranek. 2010. Intel® quickpath interconnect architectural features supporting scalable system architectures. In High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on. IEEE, 1–6.