Improving CNN Performance on FPGA Clusters through Topology Exploration

Ruihao Li
Shandong University
Qingdao, Shandong, China

Ke Liu
Shandong Normal University
Jinan, Shandong, China

Xiaojun Cai*
Shandong University
Qingdao, Shandong, China

Mengying Zhao
Shandong University
Qingdao, Shandong, China

Lizy K. John
The University of Texas at Austin
Austin, TX, USA

Zhiping Jia
Shandong University
Qingdao, Shandong, China

ABSTRACT

Field Programmable Gate Array (FPGA) platform has been a popular choice for deploying Convolution Neural Networks (CNNs) as a result of its high parallelism and low energy consumption. Due to the limited on-chip computation and storage resources, FPGA clusters are becoming promising candidates to improve CNN throughputs. In this paper, we first put forward strategies to optimize the inter-board resource allocation in FPGA clusters. Then we model the multi-board cluster problem based on dynamic programming to get the optimal topology of the FPGA clusters. Experimental results show that typical well-known CNNs with our proposed FPGA cluster topology obtains an average throughput 4.33× than single-board solutions and 1.87× than other state-of-the-art multi-board solutions.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**;

KEYWORDS

FPGA Clusters, Convolution Neural Networks, Machine Learning Acceleration

ACM Reference Format:
Ruihao Li, Ke Liu, Xiaojun Cai, Mengying Zhao, Lizy K. John, and Zhiping Jia. 2021. Improving CNN Performance on FPGA Clusters through Topology Exploration. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3412841.3441893

1 INTRODUCTION

Convolution Neural networks (CNNs) have demonstrated dramatic performances in various artificial intelligence applications. Multi-core CPUs, GPUs and ASICs have been used to implement the CNN inference tasks [2, 4, 6, 37]. However, the limited degree of parallelism of CPUs and the high energy consumption of GPUs make these two platforms not the optimal candidates for timing-constrained applications [13, 32]. ASICs can benefit from the high degree of parallelism and low energy consumption, but it is not a minor work to update ASIC architectures corresponding to the changes of CNN structures [9, 15, 16, 25, 26]. In this context, field programmable logic arrays (FPGAs), which are widely used in cyber-physical systems (CPS), are becoming attractive candidates for accelerating CNNs [1, 14, 24, 35]. The high-performance computing units on FPGAs guarantee a high throughput when executing programs in parallel, and the reconfiguration characteristic enables FPGAs support different kinds of neural networks. A large number of mature frameworks have been proposed to accelerate CNNs using FPGAs, which are divided into two categorizes, the Streaming Architecture and the Single Computation Engine [30].

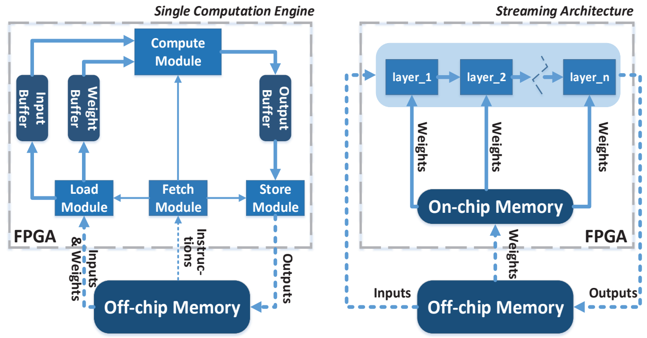


Figure 1: Dataflow of single-board solutions.

The Streaming Architecture uses multiple hardware modules to deploy each CNN layer, and in FPGA systems the hardware modules are intellectual property (IP) cores. Each IP core is optimized separately to improve the parallelism of the selected CNN layer. As shown in Figure 1, all hardware modules ($layer\_1$, $layer\_2$, $\cdots$, $layer\_n$) are linked as a pipeline, and intermediate results between IP cores are transmitted in the form of data streams. Therefore, when CNN streams transfer through the whole system, these hardware modules will work in parallel to process the data, which represents different stages in a pipeline system. The Single Computation Engine is a more general solution for different kinds of CNNs. CNN operations are abstracted as Single Instruction Multiple Data (SIMD) instructions, and only one IP core is used for computation. Control units and on-chip buffers are designed for instruction decoding,

*corresponding author: xj_cai@sdu.edu.cn

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441893

data fetching and CNN dataflow scheduling. The Single Computation Engine is applied in the latest Xilinx machine learning acceleration card Alveo [33], and prior works have proposed the Instruction Set Architectures (ISAs) to implement CNNs efficiently on FPGAs [3, 18].

Nonetheless, FPGAs are still facing great challenges to implement large-scale CNNs. The structures of state-of-the-art CNNs are becoming more and more complicated due to the growing demands for accuracy, leading to an increment of computation and memory requirements [24]. The capacity of FPGA on-chip memory is relatively limited (kilobytes level) [34] comparing with the size of current CNN weight parameters (megabytes level) [20]. This memory capacity gap introduces huge amount of data movement between off-chip and on-chip memory, which is one of the major factors affecting the CNN throughput.

A solution to the limited on-chip memory capacity is to deploy CNNs on FPGA clusters instead of one single board, on the basis of performance-oriented resource allocation strategies. Prior works have proposed to connect FPGA boards as a linear cluster based on the Streaming Architecture framework, in order to avoid the reconfiguration due to the resource limitation of one single board [7, 8, 11, 12, 23, 36]. However, different layers of CNNs often employ different shapes of tensors, leading to diverse computation and communication behaviors [31]. Based on the roofline model [18], some of the layers are memory bounded layers, while other layers are computing bounded layers. Such imbalance among different layers make FPGA cluster solutions difficult to make full use of both computation and storage resources of each FPGA when simply using the linear topology. In addition, large-scale CNNs require a huge number of FPGA boards to deploy them, but such design may not be an efficient solution when small-scale CNNs are deployed.

In this paper, we observe that the topology of FPGA clusters has significant effects on the performance of CNNs. We design a robust quantitative model which establishes a relationship between the topology of FPGA clusters and CNN throughput. Specifically, here are the technical contributions of this paper.

- We analyzed current FPGA solutions and discussed the necessity of investigating the topology of FPGA clusters.
- We proposed strategies to get the optimal FPGA cluster topology to maximize the global throughput based on dynamic programming algorithms. A combination of data-parallelism and model-parallelism is applied.
- We discussed solutions for splitting particular layers and deploying fully connected layers, which are special cases in our model.
- We deployed different kinds of CNNs on Xilinx PYNQ and Xilinx UltraScale+ ZCU102 platforms to evaluate the performance of our model. Our proposed solution achieved an average throughput 4.33× than single-board solutions and 1.87× than other state-of-the-art multi-board solutions.

2 BACKGROUND AND MOTIVATION

In this section, we will first briefly introduce the FPGA on-chip resources. Then we will illustrate the inefficiencies of the two state-of-the-art single FPGA board frameworks and why FPGA clusters can address these issues. Furthermore, the possible throughput

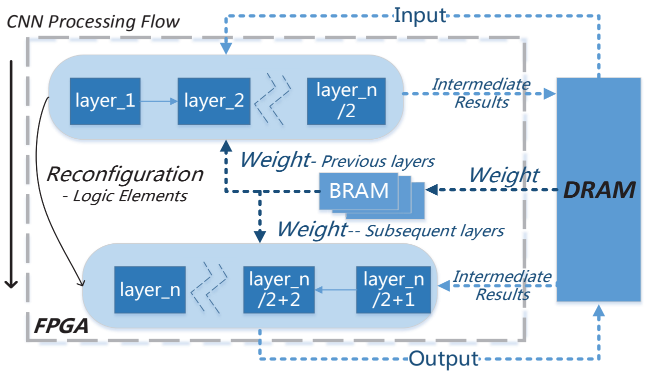


Figure 2: Reconfiguration in streaming architecture.

Table 1: VGG16 configurations.

Layer Name	Input Size	Output Size	Weight Size	Operations	Weight Reuse Time
conv3-64	224 * 224 * 3	224 * 224 * 64	3 * 3 * 3 * 64	173,408,256	224 * 224
conv3-128	112 * 112 * 64	112 * 112 * 128	3 * 3 * 64 * 128	1,849,688,064	112 * 112
conv3-256	56 * 56 * 128	56 * 56 * 256	3 * 3 * 128 * 256	1,849,688,064	56 * 56
conv3-512	28 * 28 * 256	28 * 28 * 512	3 * 3 * 256 * 512	1,849,688,064	28 * 28
conv3-512	14 * 14 * 512	14 * 14 * 512	3 * 3 * 512 * 512	924,844,032	14 * 14
FC-4096	7 * 7 * 512	4096	7 * 7 * 512 * 4096	205,520,896	1
FC-4096	4096	4096	4096 * 4096	33,554,432	1
FC-1000	4096	1000	4096 * 1000	8,192,000	1

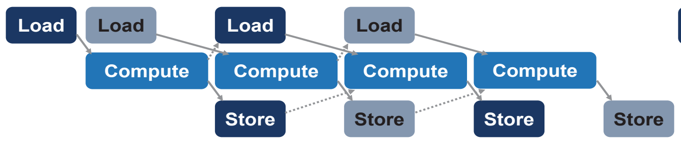
improvement when FPGA cluster topology is introduced will also be discussed.

2.1 FPGA Resources

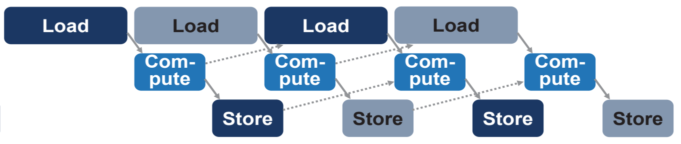
State-of-the-art FPGAs consist of Look Up Table (LUT), Flip Flop (FF), Digital Signal Process (DSP), Blocked RAM (BRAM) and other logic resources [34]. Based on the mature frameworks implementing CNNs on FPGAs, the high-throughput BRAM resources are primarily used to store the weight parameters which are reused frequently, and the distributed RAM is another candidate for storing small-scale ephemeral data during the computation. LUT resources are used for comparison operations or other logic operations in convolution and pooling layers, and FFs can act as buffers to store intermediate results. DSP units are usually used for complex computation tasks (multiplication and addition) in convolution and fully connected layers. To achieve a better performance and energy efficiency, making full use of these FPGA on-chip resources is necessary.

2.2 Inefficiency of Single-board Solutions

2.2.1 ***Streaming Architecture.*** In the Streaming Architecture, the potential issue of deploying the entire network in a pipeline is that on-chip resources of one single FPGA board are not sufficient to deploy all CNN layers. The limited on-chip memory capacity is not enough for storing all weight parameters on-chip, which means data movement between off-chip and on-chip memory or IP core reconfiguration is required. Figure 2 shows a typical workflow of the Streaming Architecture. Assuming the on-chip memory can only store half of the weight parameters, so only the previous half layers can be deployed on the FPGA board at the beginning. When the first element of the input data stream passes through all of the deployed layers (first half layers), it cannot be loaded back to on-chip acting as the input for those subsequent half layers directly.



(a) Expected pipeline.



(b) Real pipeline.

Figure 3: Pipeline of single computation engine.

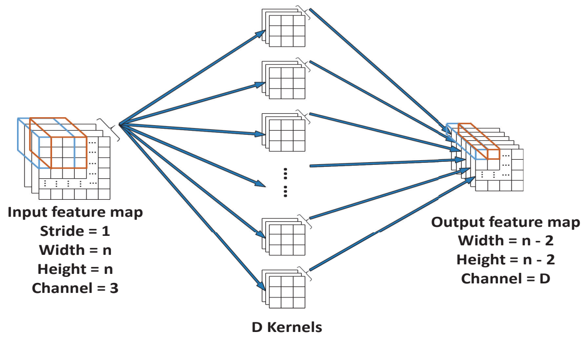


Figure 4: Convolution in single computation engine.

The reason is that these computation IP cores are still processing the left parts of the input data stream, making neither the BRAM able to be refreshed nor the whole on-chip design can be reconfigured. This data dependency is regarded as pipeline hazards, which affects the overall throughput. The reconfiguration time can be omitted if the batch size of the input data stream is large enough [29], but it will be an issue in real time systems, like self-driving cars or medical systems.

2.2.2 ***Single Computation Engine.*** In the Single Computation Engine, a full utilization of the compute module is the promise of an expected throughput. In other words, the idle cycles of DSP units should be as short as possible. To improve the utilization of the compute module, the double-buffer technique can be applied to amortise the overhead of instruction decoding and data fetching, which is shown in Figure 3a. The prerequisite of the double-buffer technique is that the time of loading data is shorter than the time of computation. If the loading time is longer than computing time, the pipeline will be described as which shown in Figure 3b. Therefore, the on-chip computation resources (DSP units) are underutilized, and the deployed CNN cannot achieve its maximal throughput, which is the real case.

In real cases, the fetched data are usually stored in discontinuous memory addresses in DRAM, which has a long random access latency. We will use the example of the first layer in VGG16 [25] (configurations shown in Table 1), which is one of the most typical CNN, to illustrate this situation. In this layer, the size of one input feature map is $224 * 224 * 3$, while the size of each weight kernel is $3 * 3 * 3$ with a dimension of 64. In each instruction, $(3 * 3 * 3)$ elements of the input feature maps will do multiplication and addition operations with the $(3 * 3 * 3) * 64$ weight kernel elements, producing $1 * 1 * 64$ outputs, which is shown in Figure 4. We notice that the $3 * 3 * 3$ elements of the input are stored in three different rows in DRAM, so loading these elements to on-chip memory requires three DRAM read requests. We measured the DRAM read latency in Xilinx PYNQ and UltraScale+ ZCU102, which is $35 - 40$ clock cycles, so the total loading time is more than 100 clock cycles. The loading time is longer than the computation time, since typical FPGA platforms contain hundreds of DSP units, which supports the $3 * 3 * 3$ multiplications being processed in parallel. As a result, for such kind of layer, the Single Computation Engine framework cannot obtain a satisfied performance due to the large number of DSP idle cycles.

2.3 Importance of Cluster Topology

The lesson we learn from the above two examples is that CNNs cannot achieve their maximal performance with single-FPGA solutions. There are prior works trying to implement CNNs on FPGA clusters. Jiang et al. [12] focused on heterogenous FPGA cluster solutions, which is difficult to be implemented due to the lack of unified inter-board data transmission protocols. There are other prior works [8, 11, 17, 23, 36] focusing on homogeneous FPGA cluster solutions. However, with an increasing number of FPGA boards, the topology of FPGA clusters becomes more complex and could affect the overall performance, which is not fully explored in their designs.

In distributed CNN systems, there are two kinds of parallelism, data-parallelism and model-parallelism [19]. In the data-parallelism, single board CNN implementations are naively duplicated, and input batches are partitioned and assigned to each board. In the model-parallelism, the CNN model is divided and deployed on different boards, and the inputs will pass through each board as data streams. In real-time image/video processing systems, when the input batch size is small, the overhead introduced by inter-board data transmission will contribute a large proportion to the overall inference time, making the data-parallelism a better solution. In datacenter systems, when the batch size is large, this inter-board data transmission overhead can be amortized, making the performance of model-parallelism better than data-parallelism. And in some other systems, a combination of these two kinds of parallelism will make the system achieve a better throughput. With different parallelism modes, different cluster topology will be applied. In this work, we will discuss how to balance these two kinds of parallelism by exploring the FPGA cluster topology.

3 METHODOLOGY

In this section, we will discuss the techniques for modeling the FPGA cluster solutions. Our design is built atop of the Streaming Architecture framework, and CNN layers are considered as pipeline stages. At first, we develop strategies to balance the processing time of each stage to improve the overall throughput. We make a combination of data-parallelism and model-parallelism. As shown in Figure 5, each linear sub-cluster in our system represents one entire pipeline which implements one entire CNN (model-parallelism

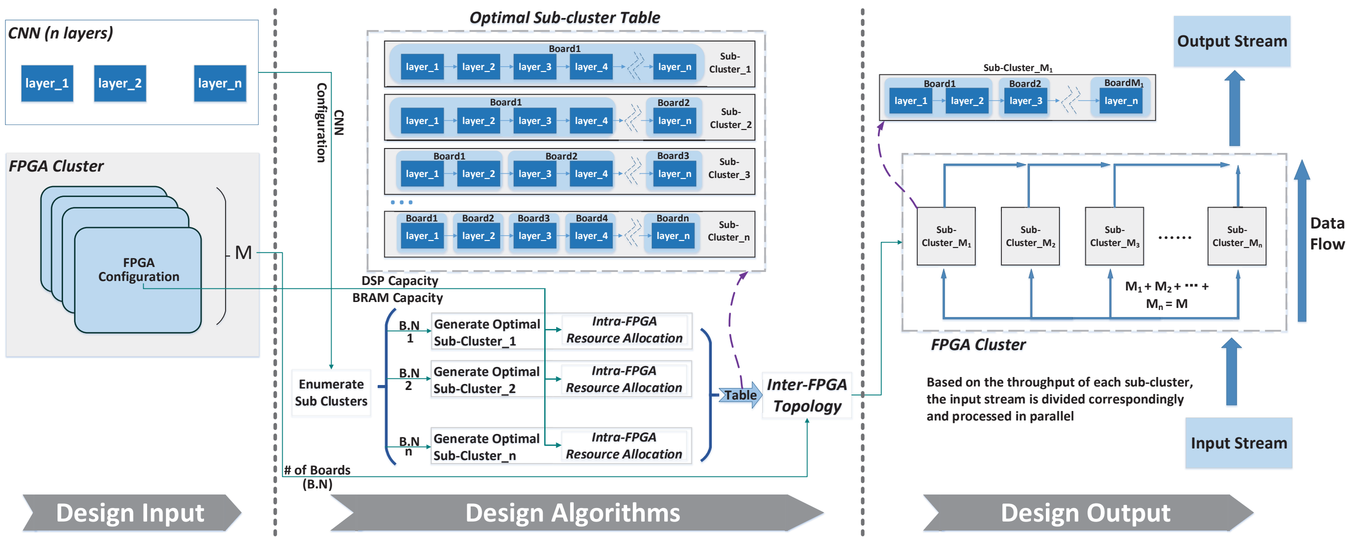


Figure 5: Illustration of proposed design flow for FPGA clusters.

intra sub-cluster), and each sub-cluster works independently processing inputs of different batches in parallel (data-parallelism inter sub-cluster). In addition, special cases for large-scale convolution layers and fully connected layers are also discussed in the last part of this section.

3.1 Overall Design Flow

In this part, we will give an overview of the design flow of our system. As shown in Fig. 5, the inputs of our system are the FPGA configuration (the number of LUTs, BRAMs, DSPs, FFs, etc.), total number of boards (M), and the CNN configuration (information of the n layers, see Table 1 as an example). The outputs of our system are the topology of the FPGA clusters, the partition of the input batches, and the resource allocation inside each sub-cluster.

Here are the steps to generate the desired output. At first, we figure out the optimal sub-cluster table of all possible board numbers ($1 \sim n$). Since there are n CNN layers, we set the maximum number of boards in one sub-cluster as n. We use intra-FPGA resource allocation strategy based on the Streaming Architecture framework in [30] to get the optimal resource allocation inside each sub-cluster. We then analyze the inter-board data transmission protocols to characterize the overhead. Furthermore, we design algorithms based on the optimal sub-cluster table to get the optimal topology of the FPGA clusters by dynamic programming.

3.2 CNN Dataflow

For the overall CNN dataflow, we use a combination of state-of-the-art techniques to achieve a better performance in a more efficient way. Prior efforts have proposed solutions for CNN acceleration, and work Eyeriss [4, 5] presented a taxonomy of these existing CNN dataflows. Scale-sim [21] even modeled the off-chip on-chip data transmission bandwidth requirements for the Weight Stationary (WS), Output Stationary (OS), and Input Stationary (IS) Dataflow. The WS dataflow, which refers to pre-store all weight parameters in on-chip memory before inference starts, will be applied to convolution layers in our model, since different pixel elements in one input feature map share the same weight parameters, as shown in Table 1. For fully connected layers, considering the data reuse is less frequent than the convolution layers, IS dataflow will be applied, which means input feature maps instead of weight parameters will be pre-stored in on-chip memory.

3.3 Cluster Topology

In this part, we will elaborate the methodology to get the optimal FPGA cluster topology, and we divide our design into three parts. Firstly, we discuss the idealistic situation. When the number of input feature maps is approaching infinite like datacenter systems, which means the ratio of the inter-board data transmission time to the total inference time is approaching zero. In this case, the inter-board data transmission overhead can be omitted. However, in most situations, we must take the inter-board data transmission latency and bandwidth into consideration. Therefore, we give a quantitative analysis of the inter-board data transmission protocols, and then build models to figure out the optimal FPGA cluster topology under such circumstance.

3.3.1 ***Case Ignoring Inter-board Transmission Latency.*** When the number of input feature maps is nearly infinite, or in other words, the input data stream is a non-stop dataflow, the inter-board data transmission latency will have little effect on the global throughput, and we propose the model based on complete knapsack problem to solve it.

Firstly, we use the optimal intra-board resource allocation strategy from work [29] to get the Optimal Sub-cluster Table in Figure 5. If CNN with n convolution layers is deployed on a k−board sub-cluster, there will be $C(n-1, k-1)$ different combinations to divide the n layers into k parts. The time complexity of enumerating all possible combinations is non-polynomial (NP), so we add several constrains to speedup the procedure.

The CNN configuration is considered as one prerequisite, and the total size of weight parameters of all convolution layers is part of this information. Based on the FPGA configuration, the minimal number of FPGA boards that can provide enough on-chip memory for storing these weights can be calculated, and we define this number as k_{min}. To reduce the computation for searching the optimal sub-cluster, the throughput of each sub-cluster containing boards less than k_{min} is denoted as 0. We iteratively increase the number of boards from k_{min} to k_{max} ($k_{max} <= n$), and calculate the throughout of each k. To shrink the computation, we define k_{max} as the point where $Throughput\ of\ k_{max} > Throughput\ of\ k_{max+1}$, which means the iteration will be stopped if the throughout does not increase as the number of boards increases.

In CNNs, convolution layers contribute more than 90% of the arithmetic operations [30], so we focus on the resource allocation for convolution layers in this part. As the number of boards increases, the inter-board data transmission bandwidth requirement will also increase, which will be discussed in Section 3.3.2. Additionally, there are also situations that k_{min} exceeds the number of layers of the selected CNN, and this situation will be discussed in Section 3.4.1. Strategies for deploying fully connected layers and other kinds of layers will be discussed in Section 3.4.2 to make our system more robust.

With the Optimal Sub-cluster Table, our model can be abstracted as a complete knapsack problem. We define $v[i]$ as the throughput of an FPGA cluster containing i boards. The i-board cluster can be consisted of $1 - i$ sub-clusters, and the the global throughput is a sum of the throughput of each sub-cluster. We use $dp[i][j]$ to represent the maximum throughput of a j-board cluster consisting of only the first i kinds of sub-clusters. Since there are j boards, there will be at most j kinds of sub-clusters, and we define the sub-cluster containing i boards as the ith kind of sub-cluster. Then the recurrence relation of our model can be written as:

$$\begin{aligned} dp[i][j] = \max\{&dp[i-1][j],\ dp[i-1][j-k*i] \\ &+ k*v[i] \mid 0 \le k*i \le j\} \end{aligned} \tag{1}$$

Then this problem can be solved using the complete knapsack model with a time complexity of $O(M|M|)$, where M represents the total number of boards. Prior work [8] showed that the linearity is up to 60 FPGAs per cluster with 150 Gb/s as the inter-board communication constraint, so the time complexity is not an issue for state-of-the-art CPUs.

3.3.2 *Inter-board Data Transmission*.

In FPGA cluster solutions, the inter-board data transmission bandwidth and latency are needed to be considered. If the inter-board data transmission bandwidth is lower than the throughput of any board in the sub-cluster, the global throughput will be constrained by the bandwidth, which will become the bottleneck of our model.

As the network goes deeper, the channel of the input feature map of each layer will increase. In this situation, one element of the output in the last several CNN layers is generated by more multiplication and addition operations than the first several CNN layers. For instance, in VGG16 the size of input feature map of the first layer is $224 * 224 * 3$, the weight size is $3 * 3 * 3 * 64$, and the output size is $224 * 224 * 64$, so each element of output is generated by $3 * 3 * 3$ multiplication and addition operations. In a similar manner, we can calculate this value in the last convolution layer, which is $3 * 3 * 512$ multiplication and addition operations. Therefore, the inter-board transmission bandwidth requirement for the last several layers is less than these first several layers. If the inter-board data transmission bandwidth makes the need of the first convolution layer, it will satisfy other convolution layers in the network.

According to the Xilinx Aurora 64B/66B [28] protocol, the inter-board data transmission bandwidth is $400Gbps$ with 16-lane GTY transceivers. The Xilinx XC7VX690T platform contains $3,600$ DSP units, and we assume that the data precision is 16 fixed-point and the FPGA platform runs at a frequency of 300MHz, then the bandwidth requirement of VGG16 is $\frac{300*10^6*16bps}{3*3*3/3600} = 640Gbps$ assuming a fully pipelined multiplication and addition in DSP units. Considering the data dependency that addition has to be done after multiplication (the throughput is $320Gbps$ if reduced by half), the $400Gbps$ bandwidth meets the requirement of our system. Though some platforms like Xilinx ZCU102 only supports up to 12-lane GTY transceivers, Xilinx PYNQ only supports the low-bandwidth protocol LVDS [22], these platforms contain fewer DSP units, which means that the bandwidth requirement is less than that of Xilinx XC7VX690T.

As for the inter-board data transmission latency, the Xilinx Aurora 64B/66B white book [28] claims that the maximal transmission latency is 54 or 55 clock cycles. In order to achieve the maximal bandwidth, more data will be packed in one data frame, resulting in a longer latency. We use different frames sizes in Xilinx Vivado to measure different data transmission latency. The results shows that the data transmission latency fluctuates from 40 to 55 clock cycles with different frame size. When inter-board transmission bandwidth satisfies the throughput requirement, we can reduce the frame size manually to achieve a lower inter-board data transmission latency.

3.3.3 *General Model*.

As aforementioned, we omitted the inter-board data transmission latency in datacenter systems. However, in real-time systems, input images have to be processed immediately as soon as they arrive. In this situation, the **batch size** has to be considered since it cannot be infinity as which in the datacenter model. The real-time model is abstracted as a normal integer programming problem, and we propose Algorithm 1 to reduce the time complexity of it.

We define $f[k][x]$ as the maximum throughput of a k-board sub-cluster processing x inputs (inputs per second). x is limited by the input image size and network frame size. For example, assuming the network frame size is $147KB$, the x for VGG16 is $\frac{147KB}{224*224*3\ Bytes} = 1$, and the x for LeNet-5 is $\frac{147KB}{28*28*1\ Bytes} = 192$. For each k, we use the same constrains as Section 3.3.1 to get the Optimal Sub-cluster Table for generating the matrix $f[k][x]$. We define $g[i][j]$ as the maximum throughput of a i-board cluster processing j inputs. The i-board cluster can be consisted of $1 - i$ sub-clusters. Then the system throughput can be written in the following recursive manner:

$$g[i][j] = \max\{g[i-k][j-x] + f[k][x]\} \tag{2}$$

The time complexity for this dynamic programming problem is $O(M^2N^2)$, where M, N represents the number of boards and input images respectively. We design Algorithm 1 based on ternary search to reduce time complexity to $O(MNlogMlogN)$. There are two dimensions in the objective function, we need to search each element in both dimensions k, x to get the global optimal solution.

Algorithm 1 Solution to inter-FPGA Optimization Problem.

```
Input: The number of FPGA boards $M$; The number of input feature maps
    $N$; The inference time of one cluster matrix $f[M][N]$
Output: The global optimal inference time matrix $g[M][N]$
 1: ternary_search$(g, f)$;
 2:
 3: function TERNARY_SEARCH$(g, f)$:
 4:     for each $i \in [1, M]$ do
 5:         for each $j \in [1, N]$ do
 6:             initialize the lower bound $left = 1$;
 7:             initialize the upper bound $right = i$;
 8:             while $right - left \geq 1$ do
 9:                 $mid \Leftarrow (right + left)/2$;
10:                 $midr \Leftarrow (right + mid)/2$;
11:                 $res\_mid \Leftarrow$ search_x $(g, f, mid, i, j)$;
12:                 $res\_midr \Leftarrow$ search_x $(g, f, midr, i, j)$;
13:                 if $res\_mid < res\_midr$ then
14:                     $right \Leftarrow midr$;
15:                 else
16:                     $left \Leftarrow mid$;
17:                 end if
18:             end while
19:         end for
20:     end for
21: end function
22:
23: function SEARCH_X$(g, f, k, i, j)$:
24:     initialize the lower bound $left = 1$;
25:     initialize the upper bound $right = j$;
26:     while $right - left \geq 1$ do
27:         $mid \Leftarrow (right + left)/2$;
28:         $midr \Leftarrow (right + mid)/2$;
29:         $res\_mid \Leftarrow g[i-k][j-mid] + f[k][mid]$;
30:         $res\_midr \Leftarrow g[i-k][j-midr] + f[k][midr]$;
31:         if $res\_mid < res\_midr$ then
32:             $right \Leftarrow midr$;
33:         else
34:             $left \Leftarrow mid$;
35:         end if
36:     end while
37:     return $res\_mid$;
38: end function
```

The function *search_x* is for getting the optimal element in dimension x with an arbitrary value in dimension k, which is achieved by comparing the midpoint value with right quartering point and updating it iteratively until finding the extreme value point (Line 26-36). In a similar manner, the optimal element in dimension k is obtained in function *ternary_search* (Line 6-18) by calling *search_x* iteratively.

3.4 System Fine Tune

As aforementioned, the case of large-scale convolution layers, the resource requirement of which exceeds the on-chip resources of one entire FPGA board, will be discussed in this section. In addition, we will discuss methodologies to implement fully connected and other kinds of layers, since the solutions to convolution layers have been addressed in Section 3.3.

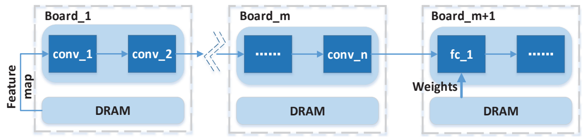


Figure 6: Solutions to fully connected layers.

3.4.1 ***Splitting up Particular Layers.*** As the network goes deeper, the size of input channels as well as weights will increase. As a result, the last several convolution layers usually consume more on-chip memory than these first several layers. In some cases, one FPGA board is not sufficient for implementing one of such kind of layer. To make our system support this special case, we design Algorithm 2 to deploy such kind of layer.

We divide the D dimensions of weight parameters into β parts and deploy them on β different boards correspondingly. For instance, the ith board is used to generate the $[(i-1) * \frac{D}{\beta} : \beta]$ channels (Line 1-6) of the outputs. Different from deploying one entire layer within one board, the ith board deploying the partial layer needs to not only transfer the outputs of its previous $i-1$ boards (Line 18-20) but also the initial inputs to the next board (Line 12-17).

To make full use of hardware resources on each board, the β parts do not need to be divided equally, which is the flexibility of our design. For example, we can deploy $1/3$ of the selected layer and the entire its previous layer on the first board, and deploy the left $2/3$ of that layer on the second board. If the β parts are equally partitioned, the hardware resources might be insufficient for deploying $1/2$ of the selected layer and the entire its previous layer. In contrast, there could be resource under utilization on the second board. With the splitting technique, the situation of deploying large-scale CNNs on FPGA clusters with limited resources can be well addressed.

3.4.2 ***Fully Connected Layers & Other Layers.*** The computation in fully connected layers can be considered as a special case of convolution layer, but there also exists differences. We choose WS dataflow for convolution layers since most convolution layers are computing intensive and weight parameters are reused more frequently than fully connected layers, as shown in Table 1. As a result, we choose to buffer input feature maps instead of weights when implementing the fully connected layers, which is known as IS dataflow. We use a combination of two types of CNN dataflow, shown in Figure 6, to use FPGA resources more efficiently.

For other kinds of layers, like pooling layers which do not require weight parameters, we fine tune the resource allocation for these kinds layers after assigning the BRAM and DSP resources for convolution and fully connected layers, since LUT resources can perform the functionality of both computation and storage. Such heterogeneous dataflow system can be implemented is the result of the flexibility of FPGA platforms, and the IP cores on each board can be easily reconfigured for various types of CNNs.

4 EVALUATION

4.1 Experiment Setup

In Section 4.2.1, we implement LeNet-5 [16] on Xilinx PYNQ clusters to evaluate the necessity of considering the topology of FPGA clusters. Since this kind of small-scale network cannot meet the

Algorithm 2 Splitting up Particular Layers.

```
Input: The batch size of input feature maps $N$; The number of parts this
    layer should be divided into $\beta$; The initial weight matrix $W[D][k*k*C]$; The input stream $in$;
Output: The output stream $out$;
for each $i$ in $[1, \beta - 1]$ do
    initialize intermediate data stream $connect_i^{i+1}$ to transmit data from
    the $i$th board to the $i+1$th board.
end for
divided_layer($in, connect_1^2, W[0 : \frac{D}{\beta} - 1][k*k*C], 1$);
· · · · · ·
divided_layer($connect_{\beta-1}^{\beta}, out, W[(\beta-1)*\frac{D}{\beta} - 1 : \beta - 1][k*k*C], \beta$);

function DIVIDED_LAYER($in, out, W, id$):
    initialize $A[k*k*C]$ to store one column of input;
    for each element in $[0, N-1]$ do
        for each column in input matrix do
            for each i in $[0, k*k*C-1]$ do
                $A[i] \Leftarrow in.read()$;
                if $id < \beta$ then
                    $out.write(A[i])$;
                end if
            end for
            for each i in $[0, id*\frac{D}{\beta} - 1]$ do
                $out.write(in.read())$;
            end for
            for each i in $[0, \frac{D}{\beta} - 1]$ do
                calculate out one output pixel $p$;
                $out.write(p)$;
            end for
        end for
    end for
end function
```

accuracy requirement of most applications, we deploy AlexNet [15], VGG16 [25], and ResNet18 [9] on Xilinx PYNQ, UltraScale+ ZCU102 in Section 4.2.2 and Section 4.2.3 for a more comprehensive evaluation. We use Xilinx Vivado HLS to develop the IP cores and use RTL language to glue the computing, communication and control modules by Xilinx Vivado.

We propose strategies to make a combination of measurement and simulation results to make our evaluation methodology not constrained by the number of FPGA boards. In our experiment, we use one single board to implement the partitioned parts of the CNN one by one, and measure the execution time of them separately. Then we use the transmission latency measured in Section 3.3.2 to calculate the overall performance. The intermediate results generated by each board are sent back to off-chip memory instead of transferring to the next board, and we use the busy signal of the DMA module to detect when the intermediate result transmission from on-chip to off-chip ends. This time has to be measured on real FPGA boards, since the simulation results cannot provide accurate DRAM memory behavior of Xilinx SOC systems.

4.2 Experimental Results

4.2.1 ***FPGA Cluster Topology Evaluation.*** Firstly, we deploy LeNet-5 with 16-bit fixed-point weight parameters on Xilinx PYNQ clusters to evaluate our dynamic programming model in Section 3.3.3.

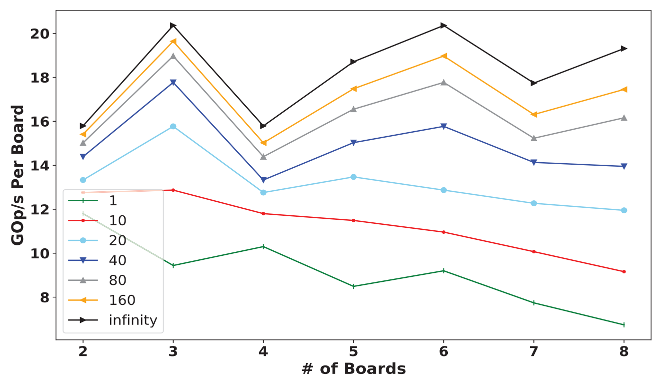
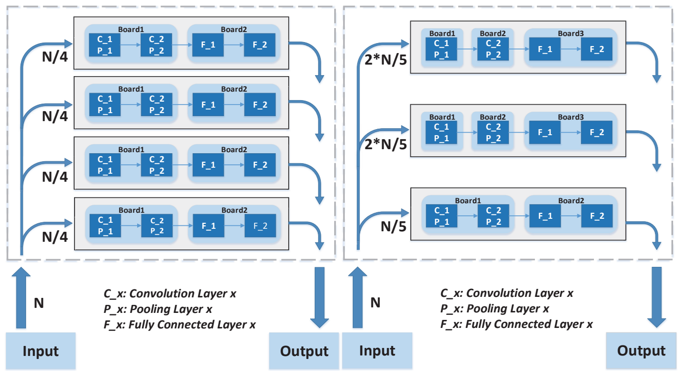


Figure 7: Performance of LeNet-5 with different batch size.



(a) Small input batch size. **(b) Large input batch size.**

Figure 8: Optimal topology of deploying LeNet-5.

We use the unit *GOp/s/board* (giga operations per second per board) to evaluate the performance. In Figure 7, the x-axis represents the number of boards containing in the cluster, and the y-axis represents the performance of LeNet-5. As the result suggests, an increment of board numbers will give a higher global throughput, but it cannot guarantee a higher average performance of each board. The reason is that LeNet-5 is implemented more efficiently on a 3-board sub-cluster than a 2-board sub-cluster if batch size is infinity.

In addition, we conclude that the optimal topological structure as well as the throughput of FPGA clusters varies as a consequence of the modification of input batch size. Our proposed technique generates different optimal topology for different input batch sizes. For example, topology in Figure 8a is adopted when input batch size is small while that in Figure 8b is derived when the input batch size is large. When the input batch size is relatively small, the transmission time occupies larger proportion of the total inference time than the situation of large input batch size, which causes the optimal topology difference.

4.2.2 ***FPGA Cluster Performance Evaluation.*** In this part, we deploy AlexNet, VGG16 and ResNet18 on Xilinx PYNQ clusters and make a comparison with state-of-the-art single-board and multi-board solutions, the results of which are shown in Table 2. Due to the limited resources of PYNQ, the last several convolution layers of VGG16 and ResNet18 cannot be deployed on one single board, so we use the strategy in Section 3.4.1 to deploy those layers. Due to

Table 2: Performance comparison with other single-board and multi-board frameworks.

	fpgaConvNet [29]	VTA [18]	FINN [27]	Work [36]	FPDeep [8]	Our Design	Our Design	Our Design
FPGA Platform	Zynq XC7Z020	Zynq XC7Z020	Zynq XC7Z045	XC7VX690T	XC7VX690T	Zynq XC7Z020	Zynq XC7Z020	Zynq XC7Z020
DSPs	220	220	900	3600	3600	220	220	220
BRAMs	0.6MB	0.6MB	2.4MB	6.5MB	6.5MB	0.6MB	0.6MB	0.6MB
Precision	fix-16	fix-8	1-bit	fix-16	fix-16	fix-8	fix-8	fix-8
BenchMark	VGG16	ResNet18	CNV-max	AlexNet	AlexNet	AlexNet	VGG16	ResNet18
Performance(GOp/s)	48.53 (CONV) **14.63***(Overall)	48.67 (CONV) **7.31***(Overall)	2465.5 (Overall) **75.33***(Overall)	207†(Overall) **25.3***†(Overall)	1022†(Overall) **124.91***†(Overall)	**147.21**†(Overall)	**142.55**†(Overall)	**131.52**†(Overall)

* We use the ratio of convolution layer running time to total inference time in VTA, which is 15%, to deduce the overall throughput of these frameworks. Besides, the throughput of each work is formalized to the throughout with 8-bit fixed-point data precision and 220 DSP units.

† For FPGA cluster solutions, we use the throughput per board (total throughput / number of boards) instead of global throughput to make a fair comparison.

the linear sub-cluster board number limitation, which is 60 [8], we choose 8-bit in stead of 16-bit fixed point [10] as the data precision to save on-chip resources.

We choose several single-board solutions, including the Streaming Architecture framework fpgaConvNet [29] which achieves the best throughput per DSP unit in [30], the Single Computation Engine framework VTA [18] which is one of latest open-source frameworks, and FINN [27] which performs better because of the precision compression. Besides, we also choose other two FPGA cluster solutions, Work [36] and FPDeep [8], to make comparisons with our design. The performance of FPDeep is evaluated by cycle-accurate simulator, and the performances of other four works are evaluated on Xilinx FPGA platforms.

The fpgaConvNet achieves higher throughput than VTA because the Streaming Architecture preforms better than Single Computation Engine framework when the batch size is large. The FINN achieves a relative satisfied performance because it does not require reconfiguration or BRAM refreshment, but the data compression (1-bit precision) will lead to a low inference accuracy. FPGA cluster solutions achieve higher throughput than fpgaConvNet and VTA by reducing data movement between on-chip and off-chip memory and avoiding FPGA reconfiguration. Our design achieves the performance 10.06× as fpgaConvNet, 20.14× as VTA, and 1.95× as FINN. FPDeep achieves better performance than Work [36] by exploring the deeply-pipelined manner of CNNs. Due to a more balanced resource allocation strategy when FPGA cluster topology is considered, our solution achieves a better performance than other FPGA cluster designs.

4.2.3 ***FPGA Platform Selection.*** In this part, we will make a discussion of how to choose the appropriate FPGA platform for deploying various types of CNNs. Our design achieves a satisfied performance when deploying CNNs on FPGA clusters, but when the CNN resource requirement overwhelms FPGA on-chip resources, like the size of weights is tens or hundreds of times as the FPGA BRAM capacity, the overall throughput will be extremely sensitive to the change of board numbers. Since WS dataflow is applied, all weight parameters have to be pre-stored on-chip. The memory bounded layers will consume more BRAM resources than computing bounded layers, making the BRAM bandwidth of these computing bounded layers not sufficient to achieve the theoretical parallelism.

For instance, as Table 3 suggests, the minimal number of PYNQ boards for deploying VGG16 is 19. Under such circumstance, the first 5 convolution layers are deployed on the first board, since the

Table 3: Deploying large-scale CNNs.

Zynq XC7Z020	# of Boards	19	20	21	22	23
	GOps / board	34.51	40.98	56.21	77.86	142.55
UltraScale+ ZCU102	# of boards	4	5	6	7	8
	GOps / board	685.65	712.23	867.71	1030.97	1714.27

weight size of the first several layers is small. Under this circumstance, there is an unbalanced workload allocation inside the first board, making it the bottleneck of the system. To obtain a more balanced pipeline, we need to assign more boards for the first 5 layers, and the total throughput increases nearly 2× with a only 7% board number change.

Instead of weight compression [27], we can choose to deploy such large-scale networks on FPGA platforms with more on-chip resources like Xilinx ZCU102. The results in Table 3 show that 2× of global throughput increment requires at least 40% board number change. In this way, the throughput of the ZCU102 cluster will not be as sensitive as the PYNQ solution. In conclusion, even FPGA cluster demonstrates satisfied performance, we should also guarantee the CNN weights scale the same order of magnitude as the BRAM capacity of our selected FPGA platform.

5 CONCLUSION

In this paper, we analyze the CNN dataflow and figure out the inefficiencies of current single-board FPGA frameworks. The huge amount of data movement between off-chip and on-chip memory is the real bottleneck. We employ FPGA clusters and put forward proper design schemes for optimizing inter-FPGA implementations to maximize the throughput of CNNs. We develop a dynamic programming model to partition CNN layers into several homogeneous FPGA boards and find the potential optimal topology of the cluster. Experimental results show that our proposed schemes for FPGA clusters achieve a higher throughput per board than single-board based solutions. Since the FPGA cluster topology is considered, our design achieves a higher throughput than other FPGA cluster solutions.

6 ACKNOWLEDGEMENT

We thank all the anonymous reviewers for the detailed comments on the paper. This work was supported by NSFC-Shandong Joint Fund (U1806203), Major scientific and technological innovation project in Shandong Province (2019JZZY010449). Any opinions, findings, conclusions, or recommendations are those of the authors and do not necessarily reflect the views of these funding agencies.

REFERENCES

[1] Aman Arora, Zhigang Wei, and Lizy K John. Hamamu: Specializing fpgas for ml applications by adding hard matrix multiplier blocks. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 53–60. IEEE, 2020.

[2] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *arXiv preprint arXiv:1903.03129*, 2019.

[3] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594, 2018.

[4] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 44(3):367–379, 2016.

[5] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE, 2014.

[7] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.

[8] Tong Geng, Tianqi Wang, Ahmed Sanaullah, Chen Yang, Rui Xu, Rushi Patel, and Martin C Herbordt. Fpdeep: Acceleration and load balancing of cnn training on fpga clusters. In *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, pages 81–84. IEEE, 2018.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. IEEE, 2016.

[10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[11] Weiwen Jiang, Edwin H-M Sha, Xinyi Zhang, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Achieving super-linear speedup across multi-fpga for real-time dnn inference. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–23, 2019.

[12] Weiwen Jiang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Lei Yang, Xianzhang Chen, and Jingtong Hu. Heterogeneous fpga-based cost-optimal design for timing-constrained cnns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2542–2554, 2018.

[13] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4154–4165, 2020.

[14] Weiwen Jiang, Xinyi Zhang, Edwin H-M Sha, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. MIT Press, 2012.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Ruihao Li, Ke Liu, Mengying Zhao, Zhaoyan Shen, Xiaojun Cai, and Zhiping Jia. Maximizing cnn throughput on fpga clusters. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, page 319. ACM, 2020.

[18] Thierry Moreau, Tianqi Chen, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Vta: An open hardware-software stack for deep learning. *arXiv preprint arXiv:1807.04188*, 2018.

[19] Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, et al. Deep learning training in facebook data centers: Design of scale-up and scale-out systems. *arXiv preprint arXiv:2003.09518*, 2020.

[20] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[21] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[22] Xilinx Inc.:"LVDS Source Synchronous 7:1 Serialization and Deserialization Using Clock Multiplication". Xapp585.

[23] Junnan Shan, Mihai T Lazarescu, Jordi Cortadella, Luciano Lavagno, and Mario R Casu. Cnn-on-aws: Efficient allocation of multi-kernel applications on multi-fpga platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[24] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 535–547. IEEE, 2017.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, pages 4278–4284. AAAI, 2017.

[27] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[28] Xilinx Inc.:"Aurora 64B/66B v11.1 Product Guide.". pg074.

[29] Stylianos I Venieris and Christos-Savvas Bouganis. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47. IEEE, 2016.

[30] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions. *ACM Computing Surveys (CSUR)*, 51(3):56, 2018.

[31] Xuechao Wei, Yun Liang, and Jason Cong. Overcoming data transfer bottlenecks in fpga-based dnn accelerators via layer conscious memory management. In *DAC*, pages 125–1, 2019.

[32] Zhigang Wei, Aman Arora, Pragenesh Patel, and Lizy John. Design space exploration for softmax implementations. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 45–52. IEEE, 2020.

[33] Xilinx. Accelerating dnns with xilinx alveo accelerator cards.

[34] Xilinx. Xilinx products overview. https://www.xilinx.com/products/silicon-devices/fpga.html, 2020.

[35] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason 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.

[36] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. Energy-efficient cnn implementation on a deeply pipelined fpga cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 326–331. ACM, 2016.

[37] Xingyao Zhang, Shuaiwen Leon Song, Chenhao Xie, Jing Wang, Weigong Zhang, and Xin Fu. Enabling highly efficient capsule networks processing through a pim-based architecture design. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 542–555. IEEE, 2020.