A Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Work and Weight Load Balancing

Tong Geng* Tianqi Wang† Ahmed Sanaullah* Chen Yang* Rushi Patel* Martin Herbordt*
*Department of Electrical and Computer Engineering; Boston University, Boston, MA, USA
†Department of Physics; University of Science and Technology of China, Hefei, China

Abstract—To improve flexibility and energy efficiency of Convolutional Neural Networks, a number of cloud computing service providers—including Microsoft, Amazon, and Alibaba—are using FPGA-based CNN accelerators. However, the growing size and complexity of neural networks, coupled with communication and off-chip memory bottlenecks, make it increasingly difficult for multi-FPGA designs to achieve high resource utilization and performance, especially when training. In this work, we present new results for a scalable framework, FPDeep, which helps users efficiently map CNN training logic to multiple FPGAs and automatically generates the resulting RTL implementation. FPDeep is equipped with two mechanisms to facilitate high-performance and energy-efficient training. First, FPDeep improves DSP slice utilization across FPGAs by balancing workload using dedicated partition and mapping strategies. Second, only on-chip memory is used in the CONV layers: a) FPDeep balances CNN weight allocation among FPGAs to improve BRAM utilization; b) training of CNNs is executed in a fine-grained pipelined manner, minimizing the time features need to be cached while waiting for back-propagation leading to a reduced storage demand. We evaluate our framework by training AlexNet, VGG-16, and VGG-19. Experimental results show FPDeep has good scalability to a large number of FPGAs, with the limiting factor being the inter-FPGA bandwidth. With 6 transceivers per FPGA, FPDeep shows linearity up to 83 FPGAs. FPDeep provides, on average, 6.36x higher energy efficiency than GPU servers.

Keywords-component — CNN Training; FPGA Cluster; High Performance Computing

I. INTRODUCTION

Convolutional Neural Networks (CNNs) are the most commonly deployed solution in applications such as image classification and object detection. CNN inference is a computation-intensive application and many efforts have been made to accelerate it. Here we focus on training, which is substantially more challenging.

![Image](https://example.com/image.png)

Fig. 1. Illustration of computations involved in DNN training.

There are two key steps (Fig. 1): Forward (FP) and Back Propagation (BP). The red datapath refers to FP. It calculates the errors of output features in the final layer. Starting with an input image (Cat), neurons in each layer are evaluated with weights $w_i$. Errors are calculated by comparing inference results to the label “Cat” in the training dataset. BP has two sub-steps: Error Back-propagation (EB-green) and Weight Gradient calculation (WG-orange). In EB, errors are back-propagated through the network. In WG, using the errors of each layer, gradients of the weights are calculated ($\frac{\partial \text{error}}{\partial w_i}$).

In our previous work [1], we proposed an FPGA-cluster-based training framework for DNN, FPDeep. The main contributions of that framework are as follows:

1. FPDeep uses model parallelism to map CNN training logic to multiple FPGAs, e.g., tightly-coupled FPGA clusters.
2. FPDeep provides fine-grained inter/intra-layer partitioning and mapping which improve workload balancing among FPGAs by increasing the flexibility of workload allocation thus leading to improved utilization. Multiple FPGAs can cooperatively compute the same layer, while multiple layers can also be mapped to the same device.
3. FPDeep uses fine-grained pipelining to minimize the time that activations need to remain available while waiting for back-propagation and the time intermediate features need to be cached while waiting for the complete layer, thus greatly reducing storage demand for activations.

Several challenges from the original FPDeep design are addressed here. First, for layers with a small number of input channels, such as the $1^{st}$ layers of the most of CNNs, the proposed intra-layer partitioning methodology, IFP (Input Feature Partition), is not useful. Thus, the $1^{st}$ layer needs to be preprocessed by the host, which limits performance. Second, the utilization of BRAMs among devices are not balanced. Finally, off-chip memory still needed to be used in certain cases, which again limits performance. In this paper, the above challenges are addressed. The main contributions are as follows:

1. An extended partitioning and mapping methodology, OFP (Output Feature Partition), is proposed to support layers with a limited number of input channels.
2. Weight load balancing is proposed to balance the weight allocation among FPGAs, reducing the number of devices needed so that only on-chip memory is required.
3. Additional and more detailed case studies are provided. Training of AlexNet, VGGNet-16 and VGGNet-19 are implemented using FPDeep. The scalability, performance and energy efficiency are evaluated and demonstrated.

II. RELATED WORK

There are two methods to map CNN applications to a cloud/cluster: Data Parallelism and Model Parallelism.

Data Parallelism (DP) is the popular approach in CPU, GPU [2], [3] and FPGA clouds, such as Catapult and CDSC
A. CNN Mapping Strategy

Assume there are $T_{FP}$ FPGAs in a target FPGA cluster. The given CNN is partitioned into $T_{FP}$ segments and each segment is mapped to one FPGA with criterion of maximum throughput under given hardware constraints. The inter- and intra-layer mapping mechanisms guarantee that the output mapping scheme provides balanced workloads among FPGAs.

1) Inter-Layer Mapping: The computation resources of $T_{FP}$ FPGAs are allocated to $L$ layers in proportion to their computational requirements, i.e., numbers of operations for FP, EB, and WG. For example, let a 2-layer CNN be mapped onto 7 FPGAs. Each FPGA has 2700 DSPs. The numbers of total operations of layer 1 and layer 2 are 182M and 83M respectively. The computational intensity of layer 1 is 2.2 times that of layer 2. To obtain balanced workload among $T_{FP}$ FPGAs, $T_{FP} \times DSP_{P, FP}$ DSP resources are allocated to each layer proportional to their arithmetic operations. In this example, 12954 and 5946 DSPs are allocated to layer 1 and layer 2, respectively. Four out of 7 FPGAs are fully allocated to the 1$^{st}$ layer, while 2 FPGAs are fully allocated to the 2$^{nd}$ layer. The leftover FPGA works for both layers, with 80% of DSP resources working for layer 1 and 20% of DSP resources working for layer 2.

2) Intra-layer Partitioning and Mapping: The next step is dividing the workload of each layer into $M$ segments and mapping them onto $M$ FPGAs. As just mentioned, 4.8 FPGAs are allocated to the 1st layer. Hence, layer 1 need to be partitioned into 4.8 segments. There are two methods for intra-layer partitioning: Output Feature Partition (OFP) and Input Feature Partition (IFP).

OFP: OFP is illustrated in Fig.2 (A). 256 output features are partitioned into 4.8 segments, including 4 big segments containing 53 output features and 1 smaller segment containing 44 features (0.8 segment). Each FPGA calculates a certain segment of output features. As calculations of each output feature require all input features, the 192 input features are broadcast to the 5 FPGAs. In training, features need to remain available while waiting for back-propagation. Therefore, the 192 features are cached in every FPGA, which can result in a large storage overhead.

IFP: Fig.2 (B) illustrates how the target CONV layer is partitioned and mapped onto 4.8 FPGAs using IFP. 192 input features and corresponding weights are partitioned into 4.8 segments including four big segments each containing 40 features, and one smaller segment containing 32 features (0.8 segment). Each FPGA receives one of the five segments and performs partial evaluations. Each output feature is calculated by summing up the related partial results from the 4.8 FPGAs. Using IFP, each input feature is only stored in 1 FPGA, which substantially reduces memory overhead compared with OFP.

Selection of IFP and OFP: Since OFP needs more storage, IFP is almost always used to partition a layer, unless the...
number of input features ($F_i$) is too small to be partitioned into $N$ (the number of FPGAs allocated to this layer) segments. For most of existing CNNs, such as AlexNet and VGGNet, OFP is only used in the 1st layer where $F_1$ is 3.

B. Weight Load Balancing Strategy

Fig. 3 (A-B) shows the weight and activation amounts of VGGNets. From the first layer to the last layer, numbers of activations are decreasing while the number of weights is increasing. The decrease of activations is because the dimensions of feature maps are reduced by pooling layers. The increase of weights is because the input and output channels of the latter layers are more than the former ones.

Using clusters with a small number of FPGAs to accelerate these two networks, the memory demands of weights for the FPGAs allocated to the latter layers increase to the point where the on-chip memories in each FPGA are not big enough to cache the allocated weights. To make it possible to map big networks to a small number of FPGAs, weight balancing must be used. Fig. 3 (C-E) shows the weight balance methodology. According to the average size of the CNN weights, the weights of the latter layers are stored in a distributed way. For example, the weights of layer 8 are stored in FPGAs 3 and 1. During computation, the weights stored in FPGA 1 are transferred to the destination FPGA 3 through the 1D network together with activations. The transport of weights does not tighten the constraint of inter-FPGA communication, because the activations in the latter layers are diminishing, resulting in a much lower inter-board communication requirement of activations than for the former layers. Our experiments demonstrate the benefits of this approach: only on-chip memory is used for CONV layers.

C. 1-D Topology

Our framework operates in a deeply-pipelined manner using a simple 1-D topology. The overall architecture is shown in Fig. 4 (B). For an $l$-layer CNN, FPGAs are divided into $l$ sets. Set $i$, which works for layer $i$, consists of $N_i$ FPGAs connected in a 1-D topology. Adjacent sets are also connected in a 1D topology. Any type of physical link can be used. This means that FPDeep can be mapped directly onto many existing platforms such as Catapult, Catapult2, and almost any tightly-coupled FPGA cluster. In Fig. 4, $F[l]$ stands for output features of layer $l$. $E[l]$ stands for errors backward propagated from layer $l$. There are 6 key data-paths. Steps 1, 2 and 3 are for FP, while 4, 5 and 6 are for BP.

1. Output features from layer $(l-1)$ are allocated to FPGAs of layer $l$ according to IFP results. Each FPGA caches $SS[l][n]$ features allocated to it and propagates the rest to the next node.

2. Using the $SS[l][n]$ features cached from Datapath 1, each FPGA calculates $F_o$ partial results of output features at layer $l$. The partial features produced from node $j$ are propagated to node $j+1$ through Datapath 2. After adding up partial features produced by nodes $j$ and $j+1$, the updated partial features are propagated continuously to the next node.

3. Weights are transferred from the node where they are cached for weight load balancing to the node where they are used to compute the output features.

4. In each cycle, errors from layer $(l+1)$ are back-propagated to FPGAs of layer $l$ through Datapath 3. Each FPGA caches all errors and propagates them backward to its preceding node.

5. Using errors from Datapath 3, each FPGA calculates $SS[l][n]$ out of $F_i$ errors and propagates them to the preceding node. Node $j$ propagates the errors calculated by itself first and then the errors transferred from node $j+1$.

6. The gradients of weights are transferred from the node where they are produced to the node where they are cached for weight load balancing.

IV. IMPLEMENTATION

As shown in Fig. 4, each FPGA instance includes FP, WG, and EB modules, as well as a memory subsystem to cache weights, gradients and activations. Each accelerator has 6 interconnection modules to communicate with its neighbors. An FPGA can be allocated to multiple layers. The architectures implemented on FPGAs working for single layer and multiple layers are illustrated in Fig. 4 (A) and (C).

1) Memory Subsystem: The memory subsystem includes BRAM-based modules storing feature, weights and gradients. 1. Feature RAM (FRAM) caches input features mapped to the target FPGA until they are consumed in back-propagation and provides input features as operators to FP and WG modules.

2. Local Gradient Buffer (LGB) caches the gradients of the weights stored in LWRAM.

3. Balanced Gradient Buffer (BGB) caches the gradients of the weights stored BWRAM. These gradients are generated by and transferred from the node where the corresponding weights are consumed.

4. Local Weight RAM (LWRAM) caches weights used as operators to produce the output features at the local FPGA.

5. Balanced Weight RAM (BWRAM) BWRAM caches the weights mapped to the local device for weight load balancing. These weights are used as operators in other nodes where on-chip memory is not enough to cache all the required weights. Both LWRAM and BWRAM are updated by WG.

2) FP, EB, WG: These three modules are used to calculate activations, errors, and gradients of the model respectively; see [1] for details. In other work [7], we proposed an accelerator for the inference of MLP, which is the foundation of the architecture of the FC layer in this work.
V. EXPERIMENTAL RESULTS

A. Single FPGA Implementation and Experiments

Results in this subsection are collected using a single FPGA board. We map three of the most widely used CNNs—AlexNet, VGGNet-16 and VGGNet-19—onto a 15-FPGA cluster. For each network, the RTL generator creates 15 bitfiles, one for each FPGA. We evaluate the design of each FPGA separately with a Xilinx VC709 board (Virtex7 XC7VX690T) and gather resource utilization, throughput and bandwidth requirements. Stochastic rounding is used during low-precision fixed-point training. In [12], it is proven that CNNs can be trained using only 16-bit fixed-point numbers when using stochastic rounding and incur no degradation in classification accuracy.

1) Workload Balance: Fig. 5 (A-F) shows resource utilization of each FPGA and resource allocations among AlexNet, VGG-16, and VGG-19 layers. As shown in Fig.5 (B, D, F)’s DSP utilization report, the mapping is well-balanced. The usage of DSP slices is roughly 80% and the throughput of each FPGA matches, around 1 TOPS. Only on-chip BRAM is used in the FPGAs that work solely on the CONV layers (FPGAs 1-14) and utilization of BRAMs is under 80%. The highest bandwidth requirement among 15 FPGAs for the 3 networks is 18.6 Gb/s.

2) Weight Load Balancing: As Section IV shows, in FPDeep, each FPGA needs on-chip memory subsystem to store weights(WRAM) and feature map (FRAM). The demand for on-chip BRAM varies greatly among FPGAs. Fig. 5 (A, C, E) shows the effect of weight balance technology. AlexNet is mapped to an FPGA cluster without weight balancing, while VGG-16/19 is implemented with weight balancing. We observe that VGG-16/19’s BRAM utilization is much more balanced than AlexNet’s. Weight load balancing makes it possible to map large networks onto clusters with a small number of devices.

In our experiments, we use a cluster with 15 FPGAs. For AlexNet, which has only 7 layers, the on-chip memory of each FPGA is sufficient to store the required weights without weight balancing. For VGGNet-16/19, there are many more layers containing many more weights than AlexNet. As mentioned in Section III.B, the memory demands of weights of the latter layers increase greatly to the point that the required size of LWRAM, in addition to FRAM and GB, is larger than the capacity of on-chip memory. Thus, to avoid using off-chip memory, weight balancing is used in our implementation of VGGNet-16/19.

3) Performance and Power Efficiency: Table I shows a comparison of performance and power efficiency among Titan X GPU [11], Tesla K80 GPU [13], a previous FPGA [11] implementation, and our work. FPDeep provides performance that is 5× higher than previous FPGA work and comparable to the Titan X GPU. We evaluate energy efficiency with respect to GOPs/J. FPDeep provides 8.8× and 5.6× better energy efficiency than the Titan X and the previous FPGA work, respectively. Compared with the K80, FPDeep provides 5.7× better energy efficiency.

B. Cluster-Level Performance Evaluation

We use a cycle-accurate simulator to evaluate cluster-level performance. Alexnet, VGG-16, and VGG-19 are mapped onto clusters of sizes 5 to 85. To demonstrate that the workload among FPGAs remains balanced in variously sized clusters, we present the proportions of idle stages. Fig. 6 (B)(D)(F) shows that the proportion of idle stages is always under 5%. When the number of FPGAs is more than 30, the number of idle stages is stable with a fluctuation of only 0.5% to 1%. Generally, as the number of FPGAs increases, the proportion of idle stages decreases. The reason is that during IFP and OFP, the number of DSPs allocated to each layer is rounded to a multiple of $K \times K$. With more FPGAs and more DSP resources, the effects of rounding error are reduced.

Computation and communication are the two most critical
TABLE I

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN Model</td>
<td>AMD A10</td>
<td>Titan X</td>
<td>Tesla K80</td>
<td>AlexNet-VGG-16</td>
<td>AlexNet-VGG-16</td>
</tr>
<tr>
<td>Configuration</td>
<td>1 GPU</td>
<td>1 GPU</td>
<td>1 GPU</td>
<td>1 GPU</td>
<td>1 GPU</td>
</tr>
<tr>
<td>Performance (GOPS)</td>
<td>34.23</td>
<td>1385</td>
<td>2330</td>
<td>2018</td>
<td>6.55</td>
</tr>
<tr>
<td>Power efficiency (GOPS/J)</td>
<td>0.39</td>
<td>4.22</td>
<td>7.87</td>
<td>6.86</td>
<td>8.28</td>
</tr>
</tbody>
</table>

Fig. 6. Roofline Models and Idle stages of AlexNet, VGGNet-16/19. In (A)(C)(E), BWC refers to bandwidth constraint.

constraints in system throughput optimization. The roofline plots of AlexNet, VGG-16, and VGG-19 are shown in Fig.6 (A)(C)(E). We observe that whole system throughput shows excellent linear scalability, up to a constraint imposed by the inter-FPGA communication bottleneck. For example, with 150 Gbps as the inter-board communication constraint, FPDeep shows linearity up to 83, 56, and 70 FPGAs for Alexnet, VGG-16, and VGG-19, respectively. As each transceiver (of that generation) can reach a maximum rate of 28 Gb/s, using 6 transceivers per FPGA achieves this number.

Since high-end FPGAs frequently have more than 50 transceivers, scaling to much larger clusters is possible. The reason that the bandwidth required by VGG-16 is larger than that for VGG-19 is straightforward. VGG-19 has more layers and thus more workload. During partitioning, with the same overall hardware resources, each layer of VGG-19 is allocated with fewer resources.

VI. CONCLUSION

In this paper, we propose a framework, FPDeep, which maps training logic of CNNs to a multi-FPGA cluster efficiently with workload and weight balancing, and also automatically generates RTL implementations for the target networks. With FPDeep, clusters of FPGAs work in a deeply-pipelined manner using 1-D topology; this enables the accelerators to map directly onto existing platforms, including Catapult, Catapult2, and almost any tightly-coupled FPGA cluster [15], [16]. FPDeep uses two mechanisms to facilitate high-performance and energy-efficiency: 1) various fine-grained partition and mapping strategies to balance workload among FPGAs, and 2) balancing weights allocation among FPGAs, which improve the on-chip memory utilization, leading to reduced storage demand to the point where only on-chip memory is required for CONV layers. Experiments show that FPDeep has the good scalability to a large number of FPGAs. The bottleneck is the inter-FPGA communication bandwidth. Using Alexnet, VG-GNets as benchmarks, with 6 transceivers per FPGA, FPDeep shows linearity up to 83 FPGAs. We evaluate energy efficiency with respect to GOPs/J and find that FPDeep provides 5.7x to 8.8x higher energy efficiency than GPU servers.

REFERENCES