FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters

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

***Abstract*—FPGA-based CNN accelerators have advantages in flexibility and power efficiency and so are being deployed by a number of cloud computing service providers, including Microsoft, Amazon, Tencent, and Alibaba. Given the increasing complexity of neural networks, however, it is becoming challenging to efficiently map CNNs to multi-FPGA platforms. In this work, we present a scalable framework, FPDeep, which helps engineers map a specific CNN's training logic to a multi-FPGA cluster or cloud and to build RTL implementations for the target network. With FPDeep, multi-FPGA accelerators work in a deeply-pipelined manner using a simple 1-D topology; this enables the accelerators to map directly onto many existing platforms, including Catapult, Catapult2, and almost any tightly-coupled FPGA cluster. FPDeep uses two mechanisms to facilitate high-performance and energy-efficiency. First, FPDeep provides a strategy to balance workload among FPGAs, leading to improved utilization. Second, training of CNNs is executed in a fine-grained inter- and intra-layer pipelined manner, minimizing the time that features need to remain available while waiting for back-propagation. This reduces the storage demand to where only on-chip memory is required for convolution layers. Experiments show that FPDeep has good scalability to a large number of FPGAs, with the limiting factor being the FPGA-to-FPGA bandwidth. Using six transceivers per FPGA, FPDeep shows linearity up to 60 FPGAs. We evaluate energy efficiency in GOPs/J and find that FPDeep provides up to 3.4 times higher energy efficiency than the Tesla K80 GPU.**

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

I. Introduction

Convolutional neural networks (CNNs) have become the dominant approach in applications such as image classification and object detection. Many FPGA cluster-based accelerators have been proposed to improve CNN efficiency. These FPGA clusters have generally been used in what could be called batch-data-parallel [1]–[5]. Each FPGA executes all layers of CNN, and each layer starts only after the previous layer has completed. However, since configurations applied in different CNN layers vary, FPGAs need to be reconfigured between layers to achieve an optimal design for each layer; this can lead to substantial overhead. Also, as the large number of weights and intermediate features often do not all fit in on-chip memory, off-chip memory must be used, which further reduces efficiency.

Another method, which we call *Model Parallelism*, is to use multiple FPGAs in a deep pipeline [6], [7]. Each FPGA accelerates only certain layers. It can therefore deploy optimal hardware for those layers without reconfiguration. Two problems still remain. First, the pipeline is not seamless; a certain layer might not be able to start until the previous layer is finished. All features must therefore be cached until the last feature of a layer is obtained. This creates a large storage requirement that requires off-chip memory. Second, the computational intensity varies greatly and this unbalanced workload diminishes the overall performance. These two problems exist in both inference and training, but they are magnified in training. First, in training, all features of the hidden layers must be cached until their corresponding errors arrive through Back Propagation (BP); training thus requires much larger memory than inference. Second, the number of operations per layer triples (due to BP).

We propose a novel FPGA-cluster-based training framework for CNNs, FPDeep, that addresses both problems: efficient pipelining and load balancing. Our overall method is to extend Model Parallelism. The main contributions are as follows.

1. Balanced Workload: FPDeep provides intra- and inter-layer partitioning and mapping.

2. Only on-chip memory is needed for the convolution (CONV) layer. FPDeep uses a fine-grained inter- and intra-layer pipeline. This minimizes the time that features need to remain available while waiting for back-propagation.

3. Simple topology. FPDeep uses a 1-D topology. This enables simple mapping onto any platform with that or a more general interconnect, including Catapult, Catapult2, and almost all tightly-coupled FPGA clusters [8].

4. Good Scalability. The limiting factor is FPGA-to-FPGA bandwidth. With N transceivers per FPGA, FPDeep shows linear scaling up to $10 * N$ FPGAs.

The rest of the sections of this paper cover, respectively, CNN training, FPDeep architecture, FPDeep design details, experimental results, and conclusion.

II. Preliminaries

In this section, we review the key steps of CNN training: Forward (FP) and Back Propagation (BP).

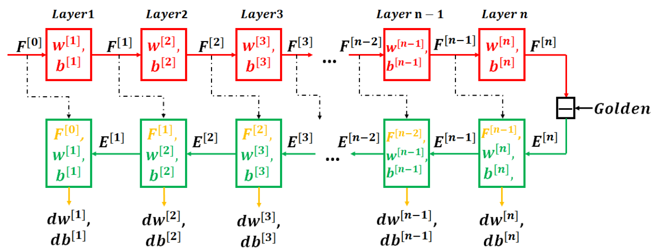

Fig. 1. Illustration of computations involved in DNN training.

FP is shown in red in Figure 1. FP calculates the errors $E[n]$ of output features at the final layer $F[n]$. Starting with an input $F[0]$, neurons in each layer are evaluated with weights w and bias b. When neurons in the last layer n are evaluated, we get the final output features $F[n]$ of the network. $E[n]$ is calculated by comparing $F[n]$ with the golden result (Golden), as labeled in the training dataset.

BP includes two sub-steps: Error Back-propagation (EB) and Weight Gradient calculation (WG). EB is shown in green in Figure 1. As $E[n]$ is back-propagated through the network, the errors of the output feature maps at each layer are calculated. As shown in Figure 1, to compute the errors of the output features of layer l, the errors, weights, and bias of layer $l+1$ are needed. WG is shown in orange. Using the error of a certain layer, gradients of the weights dw and bias db (used in this layer) are calculated. To calculate gradients of the weights and bias of layer l, feature maps of layer $l-1$ are used. Thus, feature maps need to remain available while waiting for BP.

III. Framework Design

A. Overview of the FPDeep Framework

FPDeep is a framework that helps engineers map training of a CNN onto FPGA clusters and build RTL implementations according to a parameterized mapping scheme. As shown in Figure 2, there are two key components in FPDeep: Mapping and Implementation.

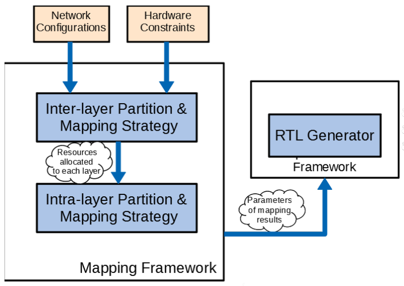
Network Configurations
Hardware Constraints
Inter-layer Partition & Mapping Strategy
Resources allocated to each layer
Intra-layer Partition & Mapping Strategy
Mapping Framework
Parameters of mapping results
RTL Generator
Framework

Fig. 2. Overview of FPDeep framework

The Mapping Framework partitions a CNN into a number of segments and maps each segment onto an FPGA. There are two sets of input parameters in the Mapping Framework: network configurations and hardware constraints. Network configurations include numbers (Fi,Fo) and sizes ($Ci{*}Ri$, $Co{*}Ro$) of input and output feature maps, filter size K, stride size S, padding size P_S, activation function of each layer and also the layer number, layer type, pooling function, and pooling size P. Hardware constraints include number of available FPGAs in the target cluster T_FP, on-chip memory resources per FPGA chip B_p_FP, DSP resources per FPGA chip DSP_p_FP, and available transceiver channels per FPGA board $Trans\_p\_FP$. These parameters are fed into the mapping framework. Through inter- and intra-layer partitioning and mapping, a parameterized mapping scheme is created.

In the Implementation Framework, the RTL generator generates RTL implementations for each FPGA based on the parameterized mapping result.

B. Partitioning and Mapping Strategy

1) Inter-layer mapping: The computation resources of T_FP FPGAs are allocated to different layers in proportional to their computational requirements, i.e., numbers of operations for FP and BP. In Figure 3, two CONV layers are given as an example to show the inter-layer mapping strategy.

The computational requirement of layer 1 is 2.2 times that of layer 2. Thus, $68\% * T_FP * DSP\_p\_FP$ DSPs are allocated to layer 1. Assuming there are 7 FPGAs in the target cluster (Figure 3), 4 out of 7 FPGAs are allocated fully to the 1^{st} layer, while the balance are allocated to the 2^{nd}. The leftover FPGA works for both layers with 80% of DSP resources working for

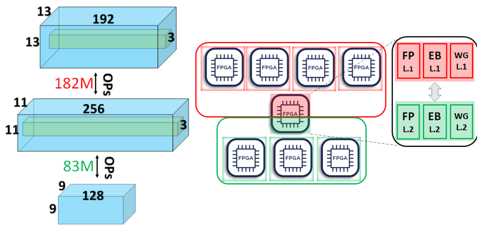

Fig. 3. A 2-layer CNN illustrating iner-layer mapping. The 4th FPGA works for both layers.

layer 1 and 20% DSP resources working for layer 2. In this method, this FPGA has replaced 2 separate FPGAs. The number of FPGAs working for layer l is defined as $N[l]$.

As mentioned in Section II, FP and BP at the same layer share weights and input features. To reduce inter-board data exchange, FP and BP operations using the same data are mapped to the same FPGA. To balance the workload of FP, EB, and WG, computation resources in each FPGA are reallocated proportionally to their operation count. For the 1^{st} layer in Figure 3, in each FPGA, 29%, 42% and 29% DSP resources are allocated to FP, EB and WG, respectively.

2) Intra-layer Partitioning and Mapping: This step partitions the workload of each layer into N segments and maps them onto N FPGAs. For illustration, FP of the 1^{st} CONV layer in Figure 3 is used as an example to show the intra-layer partitioning and mapping strategy. As there are 4.8 FPGAs allocated to FP, the FP workload in this layer is partitioned into 4.8 segments with Input Feature Partitioning (IFP).

In Figure 4, 192 input features and corresponding weights are partitioned into 5 segments including four big segments, each containing 41 features (SS), and one smaller segment containing 28 features. 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 5 FPGAs. To calculate each partial output feature on a single FPGA, a total of $SS \times K \times K$ (369) DSPs are needed. While each FPGA provides 795 DSPs for FP, we have multiple output features (B) being evaluated in parallel.

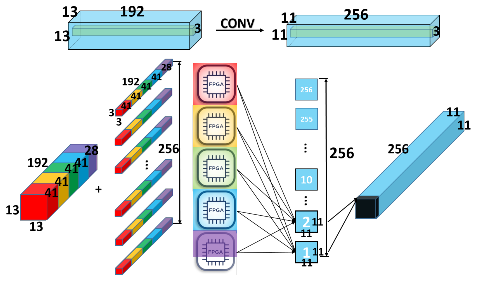

Fig. 4. Illustration of IFP

C. Fine-grained pipeline

Fine-grained inter- and intra-layer pipelining is used, minimizing the time that features need to remain available.

1) Intra-layer pipeline: The pseudo code of the CONV layer is shown in Figure 5. Each layer is partitioned and mapped to N FPGAs with the IFP strategy. Loop 5 is unrolled. Each FPGA computes B output features in parallel. Hence, Loop 3 is tiled with a tile size of B. Loops 4 through 8 are executed in parallel.

2) Inter-layer pipeline: Figure 6 compares FPDeep with the traditional inter-layer pipeline where each layer starts only after

```
for Ro in Image.height do
 for Co in Image.width do
  for fo in out.fm.num step B do
// operations below are executed in parallel
   for b in B do
    for fi in SS do
     for k in Kernel.height do
      for l in Kernel.width do
        out.fm[fo+b][Ro][Co] +=                      FPGA-1
            weights[fo+b][fi+0*SS][k][l]*
        in.fm[fi+0*SS][S*Ro+k][S*Co+l];
              ......
        out.fm[fo+b][Ro][Co] +=                      FPGA-n
            weights[fo+b][fi+N*SS][k][l]*
        in.fm[fi+(N-1)*SS][S*Ro+k][S*Co+l];
```

Fig. 5. Pseudo code of a CONV layer in FPDeep

the previous layer has finished. After B features are computed, they are propagated to the next layer and evaluated immediately. Since the workload is balanced, the production rate of output features of each layer is similar to the consumption rate of input features of the next layer. Thus, in the steady state, there are always enough input features ready to be consumed by every layer. As a result, the only on-chip memory used is for training the CONV layers.

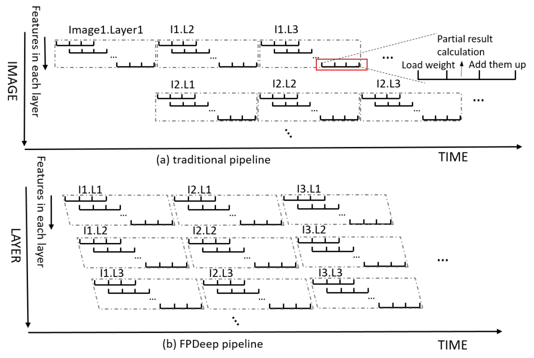

Fig. 6. Pipeline of traditional accelerator and FPDeep

D. 1-D Topology

The overall architecture of the FPDeep accelerator is shown in Figure 7. For an n-layer CNN, FPGAs are divided into n 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. There are 4 key datapaths.

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

2. Using the SS features cached from Datapath 1, each FPGA calculates Fo partial results from 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 continuously propagated to the next node.

3. 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 backwards to its preceding node.

4. Using errors from step 3, each FPGA calculates SS out of Fi errors and propagates them to the preceding node. Node j propagates the errors calculated by itself first and then the errors transferred from node $j+1$.

IV. Implementation

As shown in Figure 7, each FPGA instance includes FP, WG, and EB modules and a memory subsystem to cache weights and features. Each accelerator has two interconnection modules to communicate with its neighbors.

1) Memory Subsystem: The memory subsystem contains RAM for feature maps (FRAM) weights (WRAM). FRAM caches input features which are mapped to the target FPGA until they are consumed in back-propagation. WRAM caches weights mapped to the target FPGA and provides weights to FP and EB as operators for convolutions.

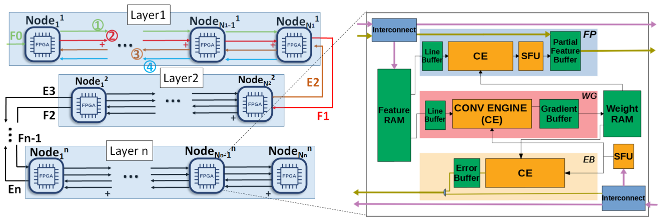

Fig. 7. Overall architecture of FPDeep

2) FP: The Line Buffer (LB) reads input features from the FRAM and feeds them to the Convolution Engines (CE) which are implemented as 2D systolic arrays. The CEs perform convolutions with weights from the WRAM and input features from LB. In the SFU, the output features are activated, normalized, and sampled.

3) EB: The EB module consumes errors of the next layer propagated by succeeding nodes and produces errors of the target layer. To calculate an error of a certain input feature map, errors of all output feature maps need to be convolved with respect to all weight filters. Errors of these input features are cached in the LB and propagated to the preceding node.

4) WG: WG consumes errors of the next layer propagated from the succeeding neighbor and calculates gradients of weights and biases. To obtain gradients of weights, errors of output feature maps are used as a filter and convolved with input feature maps cached in the FRAM. Gradients are cached in the Gradient Buffer and used to update weights in WRAM.

V. Results

A. Single FPGA Implementation and Experiments

Results in this initial report are generated with a single FPGA board. We use FPDeep to map Alexnet onto a 10-FPGA cluster; the RTL generator creates 10 bitfiles, each for one FPGA. The experimental setup is shown in Figure 8(A). We evaluate the design of each FPGA separately with a Xilinx VC709 board (Virtex7 XC7VX690T) and gather resource utilizations and throughput and bandwidth requirements. In FPDeep, stochastic rounding is used during low-precision fixed-point training. In [9], 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.

Figure 8(B-E) shows resource utilizations of each FPGA and resource allocations among AlexNet layers. As Figure 8(B) shows, 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 only on the CONV layers (FPGAs 1-9) and utilizations of BRAMs are under 80%. The highest bandwidth requirement among these 10 FPGAs is 24.9 Gb/s.

Table I shows the performance and power efficiency comparison among Titan X GPU [6], Tesla K80 GPU [10], a previous FPGA [6] implementation, and our work. FPDeep provides performance which is 5× higher than previous FPGA work and comparable to Titan X GPU. We evaluate energy efficiency with respect to GOPs/J. FPDeep provides up to 7.6× better energy efficiency than Titan X and 4× better than the previous FPGA work. Compared with the K80, FPDeep provides 3.4× better energy efficiency.

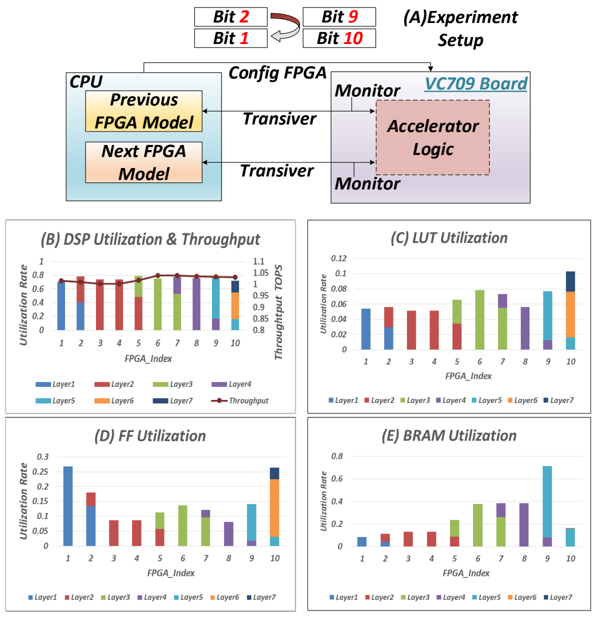

Fig. 8. Experiment results and utilization report

TABLE I
ALEXNET CLUSTER-LEVEL EXPERIMENTAL RESULT

	CPU [6]	GPU [6]	GPU [10]	FPGA [6]	Our Work
Device	AMD A10	Titan X	Tesla K80	XC7VX690T	
Config	1 CPU	1 GPU	1 GPU	4 FPGA	10 FPGA
Precision	float	float	float	fix16	fix16
Performance (GOPS)	34.23	1385	2822	207 (Per FPGA)	1022 (Per FPGA)
Power efficiency (GOPS/J)	0.39	4.22	9.41	6.55	31.97

B. Cluster-Level Evaluation

We use a cycle-accurate simulator to evaluate cluster-level performance. Alexnet is mapped onto clusters of sizes 5 to 60. To demonstrate that the workload among FPGAs remains balanced in different sized clusters, we present the proportion of idle stages. Figure 9(A) shows the proportion of idle stages is stable with fluctuation of only 5%. This leads to good scalability. A roofline model is shown in Figure 9(B). The interconnection bandwidth is the bottleneck of the overall system. That is, the system throughput presents excellent linear scalability, until reaching that constraint. For example, with 150 Gb/s as the inter-board communication constraint, FPDeep shows linearity up to 60 FPGAs. As each transceiver can reach maximum rate up to 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.

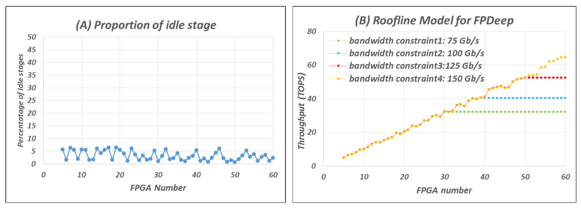

Fig. 9. Scalability of FPDeep.(A) shows the proportion of idle stages in pipeline when cluster scaling up. (B) shows the roofline model of FPDeep.

VI. Conclusion

We propose a scalable framework that helps engineers map training logic of CNNs to a multi-FPGA cluster or cloud and automatically build RTL implementations for the target network. Multi-FPGA accelerators work in a deeply-pipelined manner using a simple 1-D topology, which enables the accelerators to map directly onto most existing parallel platforms. FPDeep provides a strategy to balance workload among FPGAs, leading to high utilization and performance. Training of CNNs is executed in a fine-grained pipelined manner, which reduces the time that features need to be cached when waiting for back-propagation, thereby reducing the storage demand to the point where only on-chip memory is required for the convolution layers. Experiments show that FPDeep has good scalability to a large number FPGAs. The bottleneck is the inter-FPGA communication bandwidth. But using only 6 transceivers per FPGA, FPDeep still shows linearity up to 60 FPGAs. We evaluate energy efficiency with respect to GOPs/J and find that FPDeep provides up to 3.4 times higher energy efficiency than Tesla K80 GPU.

At this time, our FPGA cluster-level experimental results are gathered based on single FPGA board and a cycle-accurate simulator. In our current work, we are implementing the whole system on AWS and an FPGA cluster with direct interconnects.

References

[1] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates," in *FCCM*, 2017, pp. 152–159.

[2] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *IEEE Design & Test of Computers*, vol. 28, no. 2, pp. 6–15, 2011.

[3] G. Hegde, N. Kapre *et al.*, "CaffePresso: Accelerating Convolutional Networks on Embedded SoCs," *TECS*, vol. 17, no. 1, p. 15, 2018.

[4] D. J. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "High performance binary neural networks on the Xeon+FPGA platform," in *FPL*, 2017, pp. 1–4.

[5] R. L. Lian, "A framework for fpga-based acceleration of neural network inference with limited numerical precision via high-level synthesis with streaming functionality," Ph.D. dissertation, University of Toronto, 2016.

[6] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster," in *Proc Int. Symposium on Low Power Electronics and Design*, 2016, pp. 326–331.

[7] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-CNN: An FPGA-based framework for training Convolutional Neural Networks," in *ASAP, 2016*. IEEE, 2016, pp. 107–114.

[8] A. George, M. Herbordt, H. Lam, A. Lawande, J. Sheng, and C. Yang, "Novo-G#: A Community Resource for Exploring Large-Scale Reconfigurable Computing Through Direct and Programmable Interconnects," in *IEEE High Perf. Extreme Computing Conf.*, 2016.

[9] A. A. K. G. Gupta, Suyog and P. Narayanan, "Deep learning with limited numerical precision," in *ICML, 2015*, 2016, pp. 1737–1746.

[10] "Tensorflow Alexnet benchmark," https://www.leadergpu.com/articles/428-tensorflow-alexnet-benchmark.