Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster

Chen Zhang[1], Di Wu[2], Jiayu Sun[1], Guangyu Sun[1,3], Guojie Luo[1,3], and Jason Cong[1,2,3]

[1]Center for Energy-Efficient Computing and Applications, Peking University, Beijing, China

[2]Computer Science Department, University of California, Los Angeles, USA

[3]PKU/UCLA Joint Research Institute in Science and Engineering

{*chen.ceca, jiayusun, gsun, gluo*}*@pku.edu.cn,* {*allwu, cong*}*@cs.ucla.edu*

ABSTRACT

Recently, FPGA-based CNN accelerators have demonstrated superior energy efficiency compared to high-performance devices like GPGPUs. However, due to the constrained on-chip resource and many other factors, single-board FPGA designs may have difficulties in achieving optimal energy efficiency. In this paper we present a deeply pipelined multi-FPGA architecture that expands the design space for optimal performance and energy efficiency. A dynamic programming algorithm is proposed to map the CNN computing layers efficiently to different FPGA boards. To demonstrate the potential of the architecture, we built a prototype system with seven FPGA boards connected with high-speed serial links. The experimental results on AlexNet and VGG-16 show that the prototype can achieve up to 21× and 2× energy efficiency compared to optimized multi-core CPU and GPU implementations, respectively.

1. INTRODUCTION

Convolutional neural networks (CNN) have demonstrated significant success in many domains, including computer vision and audio recognition. They achieved initial success for handwritten character recognition in the 90s and since then became the state-of-the-art in large-scale visual recognition and classification. Countless studies in improving the accuracy of CNN models have been made in both academia and industry [1, 2, 3].

Along with the evolution of CNN models, a general trend is to scale up both the network size and computation complexity. To satisfy the growing demand on computation capability, researchers have used or created various high-performance hardware platforms, including GPGPU or customized accelerators such as FPGA and ASIC to improve performance and efficiency [4, 5, 6, 7, 8].

FPGA-based accelerators have been gaining popularity in accelerating large-scale CNN models because they can achieve lower latency and consume much less power compared to GPGPUs; they are also more flexible compared to ASICs. They are especially favored in datacenter-scale deployments [9]. Previous work mostly focused on single-board implementation [8, 10, 11, 12, 13]. However, the single-board design has several inefficiencies. First, FPGAs typically have fewer on-chip floating-point units and memory buffers, which limits their overall computing power. Second, the compute kernels in a CNN model can have different requirements for compute resource, memory bandwidth and on-chip buffers. As a result, it is extremely difficult to balance the resource allocation on a single FPGA for different layers in the CNN.

Table 1: Computation and memory complexity of different CNN layers in AlexNet [14]

	CONV	LRN	POOL	NL	FC
Total OPs (10^6)	591	5.5	0.5	0.0	37.7
Percentage	**93.2%**	0.8%	0.1%	0.0%	6%
Weight Size (MB)	10	0	0	0	224
Percentage	4.31%	0.0%	0.0%	0.0%	**95.7%**
DSPs per OP	1-5	**11**	3	0	1-5

The second issue can be better explained in Table 1, which is a detailed analysis of the feed-forward stage of layers in AlexNet [14], one of the most studied CNN models that won the 2012 ImageNet contest. From Table 1, the following conclusions can be drawn:

Convolution (CONV) layers are compute-intensive; they consume 4.31% of the total weights but occupy more than 93.2% of the total arithmetic operations. *Fully-connected (FC)* layers are memory bandwidth intensive; they contain only 6% of all the arithmetic operations but require 95.7% of the total weights. *Local response normalization (LRN)* layers are resource-costly; although they contain a small amount of arithmetic operations, it requires a lot of FPGA's DSP resources due to its power operations. *Pooling (POOL)* and *Non-linear (NL)* layers do not consume much computing power or memory bandwidth, and do not require a large amount of FPGA resources.

Focusing on this issue, we present a deeply pipelined multi-FPGA architecture to accelerate the feed-forward stage of large-scale CNNs. We also build a prototype of seven FPGA boards connected with high-speed serial links in a ring network to demonstrate the performance and energy efficiency of the proposed architecture. To partition an existing CNN model on the system efficiency, we propose a dynamic programming algorithm to explore the optimal mapping of each CNN layer on different FPGAs. In summary, this paper has the following contributions:

- We propose a quantitative model for mapping CNNs to the FPGA cluster. To our best knowledge, we are the first to optimally explore the design space of mapping CNNs on multiple FPGAs.
- We develop a prototype system to demonstrate that a deeply pipelined FPGA cluster can achieve comparable

ISLPED '16, August 08-10, 2016, San Francisco Airport, CA, USA

DOI: http://dx.doi.org/10.1145/2934583.2934644

performance to implementations on a high-end GPU while consuming much less energy.

2. BACKGROUND

In this section a review of the concepts used in this paper and a description of computation kernels in CNN are presented.

Feed-forward: The feed-forward step of CNN is classifying an input image. It is composed of multiple feature extraction layers, followed by classification layers. Each layer receives several feature maps from the previous layer and outputs a new set of feature maps. Figure 1 illustrates a general model of the feed-forward computation. The convolution layer, activation layer, and pooling layer are the main components in extraction layers; the fully connected layers are used in classification layers.

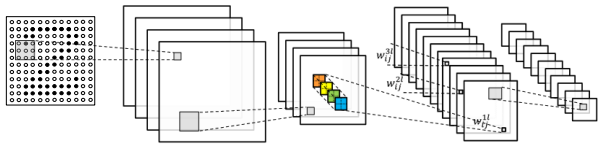


Figure 1: Feed-forward of a CNN model

CONV: CONV layers are the main components of a CNN model. CONV computation is to extract feature information by adopting a filter on feature maps from previous layers. It receives N feature maps as input and outputs M feature maps. During the computation of N kernels, each sized in $K \times K$, the kernals slide across corresponding input feature maps with element-wise multiplication-accumulation to produce one output feature map. Assuming $b^m_{r,c}$ represents a pixel of the m^{th} output feature map; $a^n_{x,y}$ represents a pixel of the n^{th} input feature map; $\omega^{m,n}_{i,j}$ represents the weight in the convolution kernel between output m and input n, and S denotes the kernels' sliding stride—then the computation of the CONV layer can be expressed as:

$$b^m_{r,c} = \sum_{n=0}^{N-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \omega^{m,n}_{i,j} \cdot a^n_{S \cdot r+i, S \cdot c+j} \quad (1)$$

LRN: LRN layers are sometimes applied after CONV layers. The LRN layer normalizes each pixel with pixels from n neighboring feature maps at the same position. Assuming $b^m_{r,c}$ represents a pixel in the m^{th} output feature map, and $a^m_{r,c}$ is a pixel in the m^{th} input feature map, then the computation of the LRN layer can be written as:

$$b^m_{r,c} = a^m_{r,c} / (k + \alpha \cdot \sum_{j=max(0,m-\frac{n}{2})}^{min(N-1,m+\frac{n}{2})} (a^j_{r,c})^2)^\beta \quad (2)$$

where k, α, β are parameters calculated based on the training and validation dataset.

POOL: POOL layers are used to achieve spatial invariance by sub-sampling neighboring pixels, normally finding the max value in a neighborhood in each input feature map.

NL: NL layers are used to adopt a non-linear function on each pixel of feature maps from previous layers to mimic the biological neuron's activation.

FC: FC layers are used to make final predictions. The FC layer takes "features" in a form of vectors, from feature extraction layers and outputs a new feature vector. Its computation pattern is exactly a dense matrix-vector multiplication or inner-product. A few cascaded inner-products finally output the classification result of CNN. Assuming a^n and b^m represent elements in the n^{th} input feature and m^{th} output feature respectively and $\omega^{m,n}$ represents the weights, then the computation of the FC layer can be written as:

$$b^m = \sum_{n=0}^{N-1} \omega^{m,n} \cdot a^n \quad (3)$$

3. DEEPLY PIPELINED FPGA CLUSTER AND PROTOTYPE SYSTEM DESIGN

The computation of the feed-forward phase of a CNN model is a streaming flow from the first layer to the last based on the description in Section 2. As a result, in the proposed architecture of the deeply pipelined FPGA cluster, multiple FPGAs are connected in a ring network, which is illustrated in Figure 2(a). On each of the FPGAs, a computation engine is customized for a specific one or more CNN layers according to optimization methods in Section 4. High-speed serial links are used as an interconnection between two FPGA boards, and a FIFO-based protocol is implemented on the inter-FPGA links.

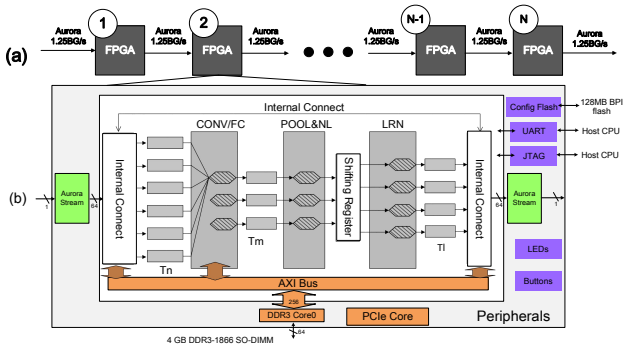


Figure 2: Logical organization of FPGA cluster

3.1 Interconnection

Low latency and high bandwidth are the requirements for the inter-FPGA connection due to the large volume of input and output data for each CNN layer, especially when the cluster is processing images in batches. A bidirectional board-to-board connection using the FPGA high-speed serial transceiver and receiver is implemented using the Xilinx Aurora protocol [15]. Each link has a bandwidth of 750MB/s, and the latency is typically around 100 FPGA cycles.

We also implemented a low-overhead flow control protocol to avoid overflow of the input buffer of each FPGA. In our protocol, the buffer state of FPGA II is back-propagated to FPGA I using the same serial link to transfer data.

3.1.1 Computation Kernels

Figure 2(b) illustrates the mapping of CNN computation engines on each FPGA. We build three computation engine models for the five kernels. Based on the computation patterns of CNN kernels, we use multiple channels to provide concurrent data streams to an SIMD computation engine for parallel processing. The number of channels are parameterized and constrained by FPGA on-chip BRAM and DSP number. The overall design space exploration methodology is described in Section 4.

CONV and *FC* layers can both be accelerated on a generalized CONV engine. For CONV layers, we make a par-

allelism model of T_n and T_m channels for input and output feature maps, respectively. FC layers are data-intensive and are usually bounded by bandwidth for FPGA platforms. We solve this problem by batching the input feature maps for FC, which enables the sharing of the weight matrix.

POOL and *NL* layers can usually be merged with the computations of CONV layers to mitigate the latency and resource cost. Therefore, the POOL and NL engines have T_m parallel channels, which is the same as the output of CONV engines.

LRN layers are DSP resource-costly. An improper resource allocation for LRN layers can make it the bottleneck of the system and degrade overall performance significantly. This can be illustrated by the results in Section 5. In the LRN engine, We model a partitioning of T_l parallel PEs for LRN layers.

3.2 Prototype System Design

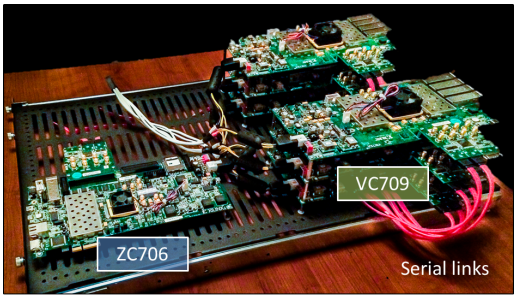


Figure 3: Physical implementation of FPGA cluster

To demonstrate the performance and energy-efficiency of the proposed architecture, we build a hardware prototype consisting of six Xilinx VC709 boards. A snapshot of the system is shown in Figure 3. We use SATA cables for the inter-FPGA communication, which are plugged to an Xilinx XM104 extension board. Each XM104 board provides two SATA ports, which are used to connect the two neighboring FPGAs in the ring network.

Since all the FPGA boards are connected as a ring, it is very easy to select the number of FPGA boards used in an experiment by reconnecting the SATA cables. In our experiments, we can evaluate different configurations ranging from one FPGA to six FPGAs.

Besides the six VC709 FPGA boards used for CNN computation, we also used another Xilinx ZC706 board, which serves as a controller. On this board, there is a Zynq FPGA that has dual-core ARM processors and FPGA programmable fabrics on the same chip. A Linux OS is running on the Zynq to control the operation of the whole system. It feeds images to the downstream FPGA and collects the results from the FPGA at the end of the pipeline. It can also use the Ethernet to communicate with a host CPU machine. Using this design, we can eliminate a CPU host machine in the system to improve its energy efficiency. We also connect all the FPGA boards to a high-efficiency ATX power supply from a PC to reduce the energy cost even further.

It is also possible to connect one or more FPGA boards to the PCIE slot of a host CPU machine, so that the system can be integrated into datacenters. This configuration is more similar to the design of Microsoft's Catapult system [16] and can be more flexible in terms of deployment. We leave this opportunity for future exploration.

4. MULTI-FPGA DESIGN SPACE EXPLORATION

In this section we discuss the design space exploration of different mappings of the feed-forward stage of the CNN layer to an FPGA cluster.

4.1 Problem Formulation

In the deeply pipelined multi-FPGA architecture introduced in the previous section, CNN layers are implemented as compute engines on different FPGA boards. The ultimate goal is to find the best linear mapping of each CNN layer i to j to k^{th} FPGA to form a pipeline of K FPGAs, such that the overall latency is minimized or the overall throughput is maximized. We assume that K, which is the total number of FPGAs in the pipeline cluster, is smaller than N CNN layers.

The complexity of an exhaustive enumeration is exponential to the number of layers N and FPGAs K, since the design space of the mappings are as many as $\binom{N-1}{K-1}$.

In the following section we present a polynomial-time design space exploration solution using **dynamic programming**. The algorithm effectively reduces the problem of multi-FPGA design space exploration to a single-FPGA, which is solvable by extending existing solutions such as [8].

4.2 Optimized Multi-FPGA Mapping Solutions

The following equations present the solutions of throughput maximization over the whole design space, where their correctness can be proven by induction.

Inter-Board Data Transfer Model.

As discussed in Section 2, once a neural network is trained, its weights become constants. We assume that all weights are pre-stored in DRAM on corresponding FPGA boards during configuration time. So the data transferred between two FPGAs are the output feature maps of layer i, which is the last layer of the first FPGA. Then the data-transfer latency is:

$$T_{ext}(i) = D_i / BW_{ext} \tag{4}$$

where D_i is the size of the output feature maps of layer L_i and BW_{ext} is the bandwidth of the board-to-board transmission.

Latency minimization solution.

The latency is measured as the time to process one image. We define $L(i, j, k)$ as the minimal latency of mapping layer i to j on k FPGA. Therefore, the final solution of the *latency-minimization* mapping of a CNN model on up to K FPGAs is:

$$\min_{k=1\ldots K} L(1, N, k) \tag{5}$$

We can recursively compute $L(i, j, k)$ as the following:

$$L(i,j,k) = \begin{cases} L(i,j,1), & k = 1 \\ \min_{r=i}^{j-1} \left(\begin{array}{l} L(i,r,k-1)+ \\ L(r+1,j,1)+ \\ T_{ext}(r) \end{array} \right), & k \geq 2 \end{cases} \tag{6}$$

In the equation, $L(i, j, 1)$ is the minimum latency of implementing layer i to layer j on one FPGA, which can be obtained using existing work. In a CNN model with N total layers, there are $N*(N+1)/2$ different $L(i, j, 1)$. After these single-board solutions are computed, we need $O(N^2 * K)$ computations to obtain the final result.

Throughput maximization solution.

The throughput can be measured as both the overall *giga-operations per second (GOPS/s)* or *images per second*. We define $T(i, j, k)$ as the maximal throughput of mapping layer i to j on k FPGA. Therefore, the final solution of the *throughput maximization* mapping of a CNN model on up to K FPGAs is:

$$\max_{k=1...K} T(1, N, k) \tag{7}$$

Since we also care about the overall energy efficiency, we define a metric of "throughput over power," which will be *GOPS/J* or *images/J*. Assuming each FPGA board consumes P Watt, the *energy efficiency maximization* solution is:

$$\max_{k=1...K} \frac{T(1, N, k)}{P \cdot k} \tag{8}$$

Both of these solutions can be obtained by recursively computing $T(i, j, k)$ as follows:

$$T(i,j,k) = \begin{cases} T(i,j,1), & k = 1 \\ \max_{r=i}^{j-1} \min \begin{pmatrix} T(i,r,k-1), \\ T(r+1,j,1), \\ \frac{1}{T_{ext}(r)} \end{pmatrix}, & k \geq 2 \end{cases} \tag{9}$$

In the equation, $T(i, j, 1)$ is the maximum throughput achieved by implementing layer i to layer j on one FPGA, which can be obtained using existing work. In a CNN model with N total layers, there are $N*(N+1)/2$ different $T(i, j, 1)$s. After these single-board solutions are computed, we need $O(N^2 * K)$ computations to obtain the result.

4.3 Single-FPGA Solution

In the previous sections, we presented a dynamic programming algorithm that reduces the problem of multi-FPGA design space exploration to the sub-problems of single-FPGA design space exploration. For the latency-minimization problem, the single-board optimization can be formulated as:

$$L(i, j, 1) = \min \sum_{r=i}^{j} L(r), \ s.\ t.\ R \leq R_{FPGA} \tag{10}$$

where $L(r)$ is the compute time for layer r, R is the total resource consumed by layer i to j, and R_{FPGA} is the available resource on FPGA. For the throughput-maximization problem, the single-board optimization can be formulated similarly.

The execution cycle estimation for **CONV**, **POOL** and **NL** layers is formulated as an equation that relates the total number of arithmetic operations and loop unrolling factors in [8, 12]. Zhang et al. [8] proved that the uniformed configuration, in which two or more layers share the same hardware, only increases the layer-specific configuration within 5% in terms of execution cycles. Considering the frequency benefit from the reduced circuit complexity, the best uniformed configuration solution that is found by searching the whole design space already has the highest GOP/s performance. In this work we use a similar formulation.

FC layers can be viewed as a convolution [17] with 1×1 kernels on 1×1 feature maps. Since the amount of weights is usually much larger than input feature maps, FC layers are memory-bound according to [12]. In a max-throughput design, we choose to batch the input feature maps. Assuming a batch size of K, then the FC computation becomes a convolution of 1×1 kernel on K feature maps. Note that batching input feature maps does not improve the latency of each feature map.

LRN layers normalize input feature maps by each pixel with pixels from neighboring feature maps at the same position, whose computation pattern is described in Section 2. It outputs the same number of feature maps as that of the input. By deciding the unrolling factor for parallelizing multiple output feature maps, we are able to explore the trade-off between the execution cycles and resource cost. The total number of execution cycles are defined by

$$\begin{aligned} execution\ cycles &= \frac{total\ arithmetic\ operations}{unroll\ factor} \\ &= \frac{M \cdot R \cdot C \cdot (N+4)}{U_{lrn}} \end{aligned} \tag{11}$$

where the notations follow equation 2. To compute the pixels from neighboring output feature maps from CONV, a stack of registers is used to buffer pixels from each of the CONV engine's output BRAM buffers, which does shifting operations to feed LRN PEs with nearby pixels.

In summary, the unrolling factors of CONV and FCN layers are $\langle T_m, T_n, T_k \rangle$, the POOL and NL layers unrolling factor is T_m, and the LRN layer unrolling factor is T_l, which defines each single-FPGA implementation. The total execution cycles of multiple layers on one single FPGA is the sum of the execution cycles of all layers.

5. EXPERIMENTAL RESULT

5.1 Experiment Setup

The baseline system used in our evaluation is a workstation of an eight-core AMD A10-5800K CPU working at 3.8GHz, and a NVidia Titan X GPU [18] plugged into the PCI-E slot. OpenBLAS and cuDNN library are used for software implementations.

For our prototype FPGA cluster (described in Section 3), we use Xilinx Vivado tools to synthesize FPGA bitstreams. The power consumption of both the baseline system and our prototype system is measured by a power meter.

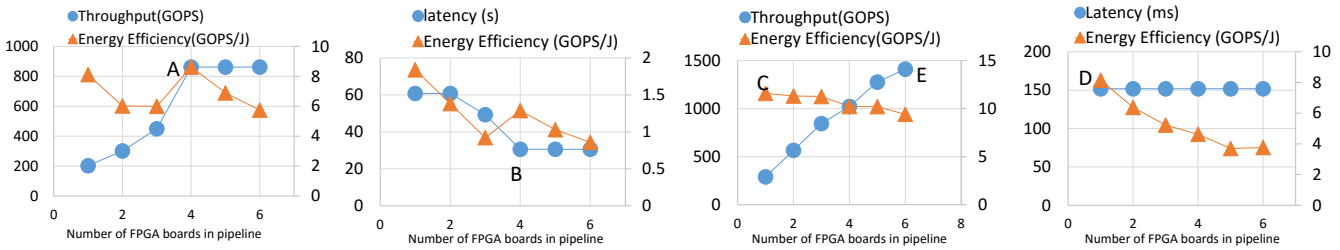
5.2 Evaluation of CNN Mappings

We first evaluate the mapping results under different objectives on the prototype system using two CNN models: AlexNet[14] and VGG-16 [19].

Throughput and energy-efficiency maximization: The throughput evaluations for AlexNet, based on different FPGA cluster sizes, are illustrated in Figure 4a and Figure 4c. For AlexNet, the best throughput is achieved at design A, which uses four FPGAs. Because the throughout is flattened after four FPGAs, the best energy efficiency is also achieved in design A. For VGG-16, the overall throughput increases with more FPGA boards but the energy efficiency does not. Therefore, the most energy-efficient design for VGG is design C in Figure 4c using one FPGA, and design E achieves the highest throughput using six FPGAs.

Figure 5 illustrates the detailed execution time breakdowns for design A and C. Design A is shown in Figure 5b, where all the layers are pipelined on four FPGAs, and the total execution time for each FPGA is balanced. Design C is shown in Figure 5a, where every layer in VGG executes sequentially.

Latency-minimization results: Figure 4b and Figure 4d illustrate the trade-offs between the FPGA cluster size and



(a) Throughput-maximization solutions for AlexNet (b) Latency-minimization solutions for AlexNet (c) Throughput-maximization solutions for VGG-16 (d) Latency-minimization solutions for VGG-16

Figure 4: Design space exploration of throughput (GOPS/s) and energy efficiency (GOPS/J) and latency (ms) maximization for AlexNet and VGG-16

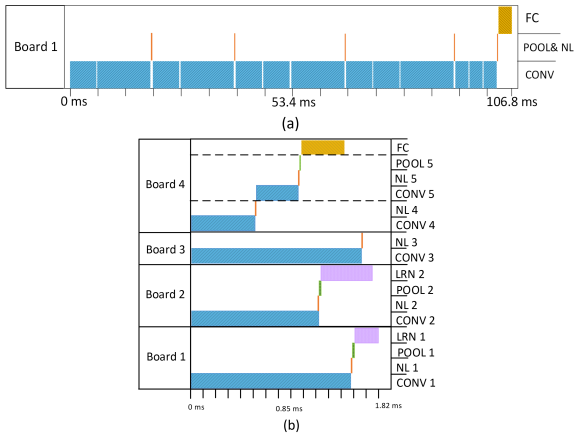


Figure 5: Execution time per image and FPGA board assignment of each layer in (a) design C and (b) design A

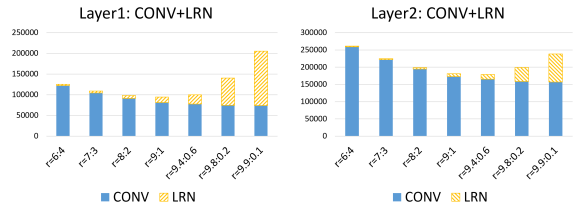


Figure 6: Performance comparison between CONV and LRN with different resource budgets. The horizontal axis 'r' denotes the ratio of DSP resource for CONV and LRN; higher 'r' means more DSPs are used for convolution and less for LRN. The vertical axis denotes execution cycles.

the best latency achieved for AlexNet and VGG-16 respectively. For AlexNet, the best latency is also achieved with four FPGAs. For VGG-16, the latency varies little due to its regular computation pattern.

Designs B and D achieve the best latency for VGG-16 and AlexNet respectively. Since in the latency-minimization design the FC layers are not batched, the total time is dominated by weight transfer. As a result, designs B and D achieve less energy efficiency compared to designs A and C, which are obtained from throughput-maximization solutions.

Observations: Several observations can be draw from Figure 4. First, the computation pattern of AlexNet has more variation than VGG, especially since it contains a LRN layer while VGG does not. As shown in Table 1, LRN is very resource-costly, so it can become a system bottleneck when insufficient resource is allocated. This problem is illustrated in Figure 6. Second, the layer configuration of AlexNet has larger variations than VGG-16. Two out of five layers in AlexNet are followed by LRN layers, while VGG-16 has repeated CONV, NL and POOL layers. As a result, more FPGA resources in the pipeline help to get better acceleration for different layers. In addition, the convolution kernel sizes vary from 11×11 to 3×3 in AlexNet, which introduces more imbalance between the workload of different layers. Having more FPGA boards can improve workload balancing by allowing different configurations for each layer. In VGG-16, all layers use 3×3 kernels, so the imbalance issue is minimal.

5.3 Overall System Performance and Energy Efficiency

Table 2 presents the comparisons between the designs generated on our FPGA cluster under different objectives and multiple baselines, including previous work on single-FPGA implementations.

For the designs generated from either throughput or energy optimization, the overall throughput is the highest among the previous single-FPGA accelerator designs for AlexNet and VGG. The energy efficiency is less than one previous work on Zynq [12] because the Zynq board is for embedded systems, while our FPGA cluster incurs inevitable overheads like board-board communications and an additional FPGA as the master node (ZC706 in our case). However, we are still higher than previous throughput-oriented FPGA accelerator designs [13]. The best throughput achieved by our prototype system is less than the GPU results, but it achieves 1.6× to 2× higher energy efficiency.

In all of our implementations, the top-1 and top-5 accuracy of the FPGA implementation of AlexNet and VGG are less than $< 2\%$ compared to the software implementation.

6. CONCLUSION

In this paper we propose a deeply pipelined multi-FPGA architecture and apply various strategies to optimize CNN mapping to multi-FPGA platforms. We also propose a dynamic programming method to efficiently explore the design space for both efficiency and latency. We build a prototype of up to six FPGAs to verify our idea. With two famous CNN models as case studies, we demonstrate that our prototype can achieve up to 21× and 2× energy efficiency compared to optimized multi-core CPU and GPU implementations respectively. The overall throughput and latency results of the prototype system are also better than existing

Table 2: Comparison of CPU, GPU, FPGA implementations

	CPU	GPU+CPU	Work [12]	Work [13]	Design A	Design B	Design C	Design D	Design E
Device	AMD A10	NVIDIA Titan X	Zynq XC7Z045	Stratix-V GSD8	Virtex-7 VX690t	Virtex-7 VX690t	Virtex-7 VX690t	Virtex-7 VX690t	Virtex-7 VX690t
Technology	32nm	28nm	28nm	28nm	28nm	28nm	28nm	28nm	28nm
CNN Model	AlexNet	AlexNet	VGG & AlexNet	VGG& AlexNet	**Alex**	**Alex**	**VGG**	**VGG**	**VGG**
Precision	float	float	fixed(16b)	fixed(8-16b)	fixed(16b)	fixed(16b)	fixed(16b)	fixed(16b)	fixed(16b)
Accuracy Top-1	-	54.3%	68.02%	66.58%	52.4%	52.4%	66.51%	66.52%	66.51%
Accuracy Top-5	-	78.7%	87.94%	87.48%	77.83%	77.83%	86.89%	86.92%	86.88%
Frequency (MHz)	3,800	$\sim$ 1,000	150	120	150	150	150	150	150
Power (Watt)	87.3	328.3	9	19.1	126	126	35	35	160
Objective	-	-	-	Throughput	**Energy***	**Latency***	**Energy***	**Latency***	**Through.***
Batch Size	16	256	1	1	16	1	2	1	2
# of FPGAs	-	-	1	1	**1+4**‡	**1+4**‡	**1+1**‡	**1+1**‡	**1+6**‡
Through. (GOPS)	**34.23**	**1385.5**	**137.0**	**117.8**	**825.6**	**128.8**	**290**	**203.9**	**1280.3**
Latency (ms)	**83.6**	**89.7**	**224.6**	**262.9**	**104**	**30.6**	**213.6**	**151.8**	**200.9**
E.-E.(GOPS/J)	**0.39**	**4.22**	**15.2**	**6.17**	**6.55**†	**1.02**†	**8.28**†	**5.83**†	**8.00**†

* Optimization objective for 'Energy' uses the metric of 'GOPS/J' and objective for 'Latency' uses 'second/image.'
‡ 1+N represents 1 Zynq board plus N VC709 boards.
† The power consumption also include the ZC706 board; therefore the overall energy-efficiency is less than reported in Figure 4.

single-FPGA implementations. The proposed architecture can also be easily integrated with the latest single-board FPGA design to achieve better energy efficiency.

7. ACKNOWLEDGMENT

This work is partially supported by the financial contributions from Fujitsu, Google, MSRA, Mentor Graphics, and an equipment donation from Xilinx. We thank the UCLA/PKU Joint Research Institute and Chinese Scholarship Council for their support of our research.

8. REFERENCES

[1] R. Girshick *et al.*, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 580–587.

[2] A. Coates *et al.*, "Deep learning with cots hpc systems," in *Proceedings of the 30th international conference on machine learning*, 2013, pp. 1337–1345.

[3] O. Yadan *et al.*, "Multi-gpu training of convnets," *arXiv preprint arXiv:1312.5853*, p. 17, 2013.

[4] Y. Q. C. Jia, "An Open Source Convolutional Architecture for Fast Feature Embedding," http://caffe.berkeleyvision.org, 2013.

[5] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM SIGPLAN Notices*, vol. 49, no. 4. ACM, 2014, pp. 269–284.

[6] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*. IEEE, 2014, pp. 609–622.

[7] M. Sankaradas *et al.*, "A massively parallel coprocessor for convolutional neural networks," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*. IEEE, 2009, pp. 53–60.

[8] C. Zhang *et al.*, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 161–170.

[9] J. Ouyang *et al.*, "Sda: software-defined accelerator for large-scale dnn systems," in *Hot Chips: A Symposium on High Performance Chips*, 2014.

[10] C. Farabet *et al.*, "Cnp: An fpga-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 32–37.

[11] S. Chakradhar *et al.*, "A dynamically configurable coprocessor for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 247–257.

[12] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[13] N. Suda *et al.*, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[14] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira *et al.*, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[15] "Aurora 64B/66B," http://www.xilinx.com/products/intellectual-property/aurora64b66b.html.

[16] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *41st Annual International Symposium on Computer Architecture (ISCA)*, June 2014.

[17] J. Long *et al.*, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[18] NVIDIA, "Gpu-based deep learning inference: A performance and power analysis," *NVIDIA Whitepaper*, vol. 2, 2015.

[19] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.