An Efficient Mapping Approach to Large-Scale DNNs on Multi-FPGA Architectures

Wentai Zhang[1,*], Jiaxi Zhang[1], Minghua Shen[2,†], Guojie Luo[1,3,‡], and Nong Xiao[2]
[1]Center for Energy-Efficient Computing and Applications, Peking University
[2]School of Data and Computer Science, Sun Yat-Sen University
[3]Peng Cheng Laboratory
Email: *rchardx@gmail.com, †shenmh6@mail.sysu.edu.cn, ‡gluo@pku.edu.cn

***Abstract*—FPGAs are very attractive to accelerate the deep neural networks (DNNs). While single FPGA can provide good performance for small-scale DNNs, support for large-scale DNNs is limited due to higher resource demand. In this paper, we propose an efficient mapping approach for accelerating large-scale DNNs on asymmetric multi-FPGA architectures. In this approach, the neural network mapping can be formulated as a resource allocation problem. We design a dynamic programming-based partitioning to solve this problem optimally. Experimental results using the large-scale ResNet-152 demonstrate that our approach deploys sixteen FPGAs to provide an advantage of 16.4x GOPS over the state-of-the-art work.**

I. Introduction

Deep neural networks (DNNs) are powering many modern artificial intelligence (AI) applications, from autonomous driving, to detecting cancer or playing complex games. FPGAs have been very attractive to accelerate DNNs with high performance and energy efficiency. Typically, FPGA-based DNN accelerators need more resources to address the increasingly complicated problems. Moreover, in single-FPGA, resource reuse is a promising approach to accelerate the small-scale DNNs such as AlexNet and VGG-Net. However, support for large-scale DNNs to obtain ideal performance is very difficult. There are few works about partitioning and mapping the DNNs on multi-workloads [1]–[3]. [1] partitions DNNs pipelined architecture on multi-FPGAs forming in a ring and they use SATA cables with XM104 extension board for the communications between FPGAs. [2], [3] also provide partitioning strategies to map DNNs on mobile devices. Their target is to minimize the latency or data transmission time, while our target is to optimize the throughput.

Advances in interconnection technologies like 10/40GbE and Infiniband, have enabled multi-FPGA prototype that provides a combination of gate capacity and communication bandwidth to strive for higher performance. The asymmetric multi-FPGA architecture, as shown in Figure 1, is widely adopted to accelerate the compute-intensive applications. In this architecture, FPGAs are attached to servers with PCIe, and servers are connected through Ethernet switches. However, it is non-trivial to efficiently map large-scale DNNs on an asymmetric multi-FPGA architecture due to the following challenges:

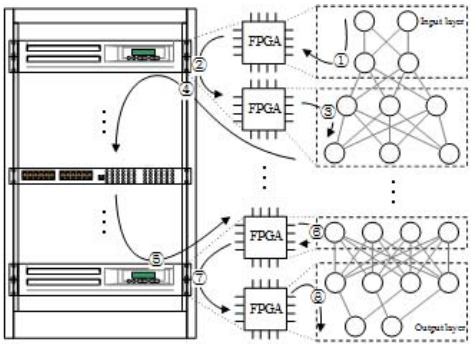

Fig. 1. A typical asymmetric multi-FPGA architecture: the solid line denotes the data flow of neural networks.

1) Topology among FPGAs: It has diversity in practical deployment that directly affects the solution space exploration.
2) Bandwidth in the system: It is non-uniform in multiple FPGA environment that directly affects the system design.
3) Resource allocation: In DNNs, each layer has different resource requirements. It is very crucial to allocate the resources, which aims at achieving the maximal throughput performance.

To address these challenges, we propose a resource- and bandwidth-aware mapping approach that partitions the large-scale DNNs into the asymmetric multi-FPGA architecture. This partitioning algorithm is based on dynamic programming, and topology and bandwidth are always used to guide the partitioning to obtain the maximal performance while maintaining the resource-performance trade-off in a single-FPGA. The major technical contributions of our work are threefold:

- We formulate a mapping problem to enable large-scale DNNs on an asymmetric multi-FPGA architecture.
- We present an optimal partitioning algorithm based on dynamic programming to provide the maximal performance in throughput.
- We prove that multi-FPGA architecture is scalable in implementing large-scale DNNs with sufficient bandwidth, and existing approaches on a single FPGA can be upgraded to the multi-FPGA version using our approach.

978-3-9819263-2-3/DATE19/©2019 EDAA

II. Problem Formulation

FPGA topology is a pre-determined graph $G(V, E)$ and $v_i \in V$ is an identity of FPGA i. The board specification is defined by functions, and they are $\overline{BRAM}(v_i)$, $\overline{DSP}(v_i)$, $\overline{FF}(v_i)$, and $\overline{LUT}(v_i)$. Moreover, each FPGA has its own DRAM, and we have FPGA-DRAM bandwidth $D(v_i)$. $e_i = (v_p, v_q)$ is a connection between two FPGAs v_p, v_q, and $B(e_i)$ or $B(v_p, v_q)$ is the data transfer bandwidth. Neural network is a linear structure and contains several connected layers, from input layer to output layer. We use A to denote a neural network, and A consists of N layers $L_1, L_2, L_3, \cdots, L_N$.

In order to avoid ambiguity, we give some assumptions below. First, all the layers of the design are connected through FIFOs interface. This simplifies the estimation of throughput between FPGAs, without loss of generality. Second, there do not exist loop backs or branches in deep neural networks. For those with branches such as LeNet or ResNet, we eliminate the branches by grouping layers in a single branch into a block. This makes sure that most of the neural networks are in linear forms. Third, partitioning is monotonous, which means if layer i and j are mapped onto FPGA v_x, any layer k is on FPGA v_x as well, where $i \leq k \leq j$.

With the above definitions and assumptions, we formally formulate the problem as follows: Given the topology, board, and neural network specifications, our objective is to find a mapping such that the layers L_i are allocated onto the v_x to gain optimal throughput, and that there is no violation of resource limitation.

III. Mapping Approach

Our approach is based on dynamic programming, and it is capable of dealing with host-to-host and FPGA-to-FPGA connection relationships. First, we perform neural network pre-analysis to figure out the resource usage and throughput for each layer or several layers detailed in Section III-A. Second, a dynamic programming algorithm is implemented to find the optimal partitioning based on input network and throughput pre-analysis. Section III-B shows our dynamic programming partitioning and mapping algorithm. Third, after partitioning, we re-organize the C/C++ source code to implement connection infrastructure, such as FIFO interface. Fourth, we use Xilinx Vivado tools to synthesize, place and route the design for each FPGA, and generate the programming files. Finally, we download all the programming file to the corresponding FPGAs to complete the whole design flow.

A. Neural Network Pre-Analysis

In this section, we describe the pre-analysis of the layers for a single FPGA. E is an estimation function of throughput for a single FPGA, and $E(i, j, x)$ indicates the estimation throughput of mapping layer i to j onto FPGA v_x. This is a familiar problem: using a single FPGA to accelerate the neural network. According to the related works, there have already been many approaches to solve this problem. Actually, the solver of E does not affect our partitioning approach, and any existing method can be used.

In the beginning, we need to solve $E(i, i, x)$ because it is the subproblem of $E(i, j, x)$. For single layer, $E(i, i, x)$ can be solved by adapting a relatively naive method [4]. We use this naive method in our implementation and experiments as an example, since we focus on resource allocation and communication optimization. In addition, the code optimization techniques used here will not affect the partitioning algorithm, though it would generate less optimal final throughput.

```
for(r=0;r<R;r+=Tr){ //feature row
 for(c=0;c<C;c+=Tc){ //feature column
  for(m=0;m<M;m+=Tm){ //output feature
  for(n=0;n<N;n+=Tn){ //input feature
    //load weights
    //load input/output maps
    for(tr=r;tr<min(r+Tr,R);++tr){
     for(tc=c;tc<min(c+Tc,C);++tc){
      for(tm=m;tm<min(m+Tm,M);++tm){
      for(tn=n;tn<min(n+Tn,N);++tn){
        for(i=0;i<K;i++){ //kernel width
         for(j=0;j<K;j++){ //kernel height
          //on-chip data computation
          out[tm][tr][tc]+=w[tm][tn][i][j]
                  *in[tn][S*tr+i][S*tc+j]
}}}}}}}}}}
```

Listing 1. Convolutional layer

After dealing with $E(i, i, x)$, we are able to calculate $E(i, j, x)$. Thus, $E(i, j, x)$ has the following equation:

$$E(i, j, x) = \min\{E(k, k, x) | i \leq k \leq j\} \quad (1)$$

When each layer is allocated with a unrolling factor, it will consume certain amount of hardware resources. The problem of calculating $E(i, j, x)$ actually becomes how to allocate proper unrolling factors for layer i to j, and the resources consumed are not exceeded in FPGA v_x ($\overline{BRAM}(v_i)$, $\overline{DSP}(v_i)$, $\overline{FF}(v_i)$, and $\overline{LUT}(v_i)$).

We use binary search to solve this problem. For a determined throughput, each layer k has a minimum unrolling factor which satisfies this throughput requirement, because the resource requirement is a monotonous function of unrolling factor. In the beginning, we set lower- and upper-bound LB, UB for the binary search. Each time we check $(LB+UB)/2$, if it can be satisfied the resource limitation, we set $LB = (LB+UB)/2$, or $UB = (LB+UB)/2$ otherwise. When LB and UB are very close, the search is finished, and $(LB + UB)/2$ is the answer. The time complexity of this method is a logarithmic function of the search range $UB-LB$.

In summary, the estimation function $E(i, j, x)$ provides a foundation of the following dynamic programming algorithm. Pre-analysis can reduce the repeated computation. As subproblems, the solver of $E(i, j, x)$ can be varied. Any existing method for a single FPGA board can adopted to solve $E(i, j, x)$.

B. Dynamic Programming Partitioning

Our goal is to get maximum throughput, and we want to partition the neural network based on these parameters such as FPGA board specification and connection bandwidth. We deploy dynamic programming to solve this optimization problem. A dynamic programming algorithm will divide the original problem into subproblems, and examine the previously solved subproblems to integrate their solutions to give

Algorithm 1 Solution for Equation (2)

```
1: function DPSOLVER(l, s, x)
2:     if m(l, s, x) = 1 then
3:         return T(l, s, x)
4:     end if
5:     if |s| = 1 then
6:         T(l, s, x) ← E(1, l, x)
7:     else
8:         T(l, s, x) ← 0
9:         for k from 1 to l − 1 do
10:            for all (2^y & (s − 2^x)) > 0 do
11:                tmp ← min{B(v_x, v_y), E(k + 1, l, x)}
12:                tmp ← min{DPSOLVER(k, s − x, y), tmp}
13:                T(l, s, x) = max{T(l, s, x), tmp}
14:            end for
15:        end for
16:    end if
17:    m(l, s, x) = 1
18:    return T(l, s, x)
19: end function
                                        ▷ solution starts from here
20: m(l, s, x) ← 0
21: Result = 0
22: for l from 1 to N do
23:    for s from 1 to 2^|V| − 1 do
24:        for all (2^x & s) > 0 do
25:            Result = max{Result,DPSOLVER(l, s, x)}
26:        end for
27:    end for
28: end for
29: return Result
```

the best solution for the given problem. Deep neural networks have many layers and are in linear forms, which is probable to be divided.

We define that $T(l, s, x)$ represents the maximum throughput (MB/s or GB/s) result while the current status is (l, s, x) in dynamic programming. In the status, l indicates that layer 1 to l have already been mapped. s is the status of current FPGA(s) we used. For example, if we have five FPGAs v_1, v_2, v_3, v_4, v_5 and v_2, v_3, v_5 are used, it is $\{2, 3, 5\}$ and can be encoded as $\underline{01101}_2$. x is an indicator of the last FPGA board v_x used, because the monotonous property ensures the last FPGA is needed. Our target is $\arg\max_{s,x} T(N, s, x)$, which implies any used status of FPGAs and last used FPGA board could lead to the optimal result.

In summary, our state transfer equation is formulated as

$$T(l,s,x) = \begin{cases} \max\limits_{0<k<l, y\in s-\{x\}} \{\min\{T(k, s-\{x\}, y), \\ \qquad\qquad B(v_x, v_y), \\ \qquad\qquad E(k+1, l, x)\}\} & \text{if } |s| > 1 \\ E(1, l, x) & \text{if } |s| = 1 \end{cases} \tag{2}$$

where E is detailed in Section III-A, and $E(k + 1, l, x)$ indicates the performance of mapping layer $k+1$ to l onto FPGA v_x. Obviously, we need to pre-analyze E to avoid repeated calculation. In other words, for a consecutive segment of layers, we use a single FPGA to estimate maximum throughput result, and we use that result to further perform system-level dynamic programming. After pre-analysis, we could get the value of any E in $O(1)$ time.

In Algorithm 1, we will present the solution for Equation (2). We use memorize search to realize the solution, because Equation (2) is not linear. In this algorithm, our solution starts from line 18. At the beginning, we initialize the mask variable m. m has the same status with T. When $m(l, s, x) = 1$, it means that $T(l, s, x)$ has already been calculated. The space complexity of this algorithm is $O(N \cdot 2^{|V|} \cdot |V|)$, and the transfer cost for every state is $O(l \cdot |s|)$. The time complexity of the algorithm is $O(N \cdot 2^{|V|} \cdot |V|)$. Moreover, the number of FPGAs used to accelerate a neural network cannot be too large, in case that the solution space is too enormous.

In summary, our dynamic programming algorithm is based on $E(i, j, x)$ and the status s of FPGAs used. The better optimization is used in $E(i, j, x)$, the better overall throughput $T(l, s, x)$ can achieve. The time complexity implies that the amount of FPGAs cannot be too large in order to ensure the running time of our algorithm is reasonable.

IV. Evaluation

a) Experimental Setup: The accelerator design is implemented with Xilinx SDx 2017.1. Pre-synthesis resource reports and performance analysis are used for our neural network pre-analysis and dynamic programming partition. The implementation is built on the Alpha Data ADM-PCIE-8V3. It is connected to our server with 16 lane PCIe 3.0, which provides a maximum bandwidth of 15.75 GB/s. Our servers run on Intel Xeon CPU E5-2650v3 (@ 2.30GHz) with 25MB cache. It communicates with FPGAs through PCIe, and uses sockets through Ethernet to reach other servers. We set up two kinds of Ethernet connections: GbE and 10GbE. GbE environment is based on traditional switches and cable links. 10GbE relies on specialized adapter card, such as Synology's and Mellanox's, and we use ConnectX-3 Pro EN adapters and SX1016 switches. In our experiments, we will focus on implementing ResNet-152 to verify our algorithm's correctness and performance.

b) Experimental Results: There are many factors that affect the final results. First, FPGAs topology and specifications are the most significant ones. We have presented them in the environment setup. In our experiments, we will change the topology to examine the results. Second, bandwidth limitation in the system is important as well. The slowest communication channel could be the host-to-host connection. It is usually implemented by Ethernet, and is much slower than FPGA-to-host connection.

We will test our algorithm on 1-4 servers, and name these topologies Topo A-D. Each server has four FPGAs, and therefore we have 4-16 FPGAs in our environment. For host-to-host connection, we will examine the difference between GbE, and 10GbE. In the beginning, the design flow will use partitioning algorithm to obtain the partitioning scheme, and the resource utilization. After that, we have the chance to review the results to make sure the following steps such as synthesis, placement and routing is able to run without mistakes.

Final resource utilization of our design is completed after SDx finishes the placement and routing. In Table II, we show

TABLE I
PERFORMANCE COMPARISON TO OTHER RESNET-152 IMPLEMENTATIONS.

	CPU (16threads)	GPU	[5]	FPGA (Baseline)	Ours (Topo D)	Ours ([5]+Topo D) (estimated)
Precision	float32	float32	fixed16	fixed16	fixed16	fixed16
Device	Xeon E5-2650v2	TITAN X	Stratix-V	Virtex Ultrascale	Virtex Ultrascale	Virtex Ultrascale
Frequency	2.6GHz	-	150 MHz	150MHz	150MHz	150MHz
Performance (GOPS)	119	1661	226.47	15.6	256.6	3714.1
Normalized Performance	7.6x	106.5x	14.5x	1x	16.4x	239.1x

TABLE II
RESOURCE UTILIZATION IN TOPO A.

Modules	BRAM_18K	DSP48E	FF	LUT
Available	1728	768	1075200	537600
FPGA 1	1406 (81.4%)	146 (19.0%)	278234 (25.9%)	416348 (77.4%)
FPGA 2	1404 (81.3%)	144 (18.8%)	274346 (25.5%)	400468 (74.5%)
FPGA 3	1418 (82.1%)	148 (19.3%)	286552 (26.7%)	448224 (83.4%)
FPGA 4	1486 (86.0%)	156 (20.3%)	309124 (28.8%)	476574 (88.6%)

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT TOPOLOGIES.

	Topo A	Topo B	Topo C	Topo D
Frequency	150MHz	150MHz	150MHz	150MHz
Throughput (MB/s)	143	334	450.45	583.4
Performance (GOPS)	62.9	146.8	198.135	256.6
Normalized Performance	1x	2.3x	3.15x	4.1x

TABLE IV
PERFORMANCE (GOPS) COMPARISON OF HOST-TO-HOST BANDWIDTH.

	GbE	10GbE
Bandwidth (MB/s)	117.875	1162.24
Topo A	62.9	62.9
Topo B	51.8	146.8
Topo C	51.8	198.135
Topo D	51.8	256.6

the results of Topo A's four FPGAs under the 10GbE situation. We set up strictly resource constraints to benefit placement and routing, and we set the utilization limitation around 80-90%. We can see that after the dynamic programming partition, every FPGA almost fully utilizes the on-board hardware resources.

The performance comparison between different topologies is shown in Table III. In this table, we assume that Ethernet is connected using 10GbE (1162.24MB/s), and the throughput is smaller than this bandwidth. Another observation is that when the number of FPGAs is increased, the throughput (bandwidth requirement) grows as well. As we can see, the performance improvement is approximately proportional to the number of FPGAs when bandwidth is sufficient. Besides, the number of duplicated hardware component depends on the resource capacity. It is similar to an integer planning problem, and leads to this non-linear increasing in throughput.

The performance comparison between different host-to-host bandwidth is shown in Table IV. Topo A is not affected by the host-to-host bandwidth because it uses single server. When we use GbE network to connect the servers, the performance is limited even if the number of FPGAs is increased. 10GbE is suitable for current design, because the throughput has not exceeded its bandwidth. The results show that limited bandwidth will lead to a hard upper-bound of throughput. The performance comparison between our method and others is shown in Table I. In this table, we add Baseline implementation as a comparison. Baseline uses a single FPGA board for the ResNet-152. Therefore, only $E(1, N, 1)$ is needed for calculation. Besides, we also tried to adopt [5]'s work into our algorithm. We use their framework to evaluate $E(i, j, x)$, and estimate the performance of this mixed method.

V. CONCLUSION

In this paper, we propose a mapping approach to accelerate large-scale DNNs such as ResNet-152 onto asymmetric FPGAs. Specifically, we formulate the mapping problem as a resource allocation problem and design a dynamic programming-based partitioning algorithm to solve it optimally. Experimental results show that multi-FPGA architecture can have great performance, and the bandwidth requirement could be the bottleneck when the scale of FPGAs system grows.

ACKNOWLEDGMENT

This work is partly supported by Beijing Municipal Science and Technology Program under Grant No. Z181100008918015, Beijing Natural Science Foundation under Grant No. L172004, National Natural Science Foundation of China (NSFC) under Grant No. 61433019, No. 61802446, and No. 61520106004, and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant No. 2016ZT06D211.

REFERENCES

[1] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster," in *ISLPED*, 2016, pp. 326–331.

[2] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local Distributed Mobile Computing System for Deep Neural Network," in *DATE*, 2017, pp. 1400–1405.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," *ASPLOS*, pp. 615–629, 2017.

[4] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015, pp. 161–170.

[5] 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.