Application Partitioning on FPGA Clusters: Inference over Decision Tree Ensembles

Muhsen Owaida, Gustavo Alonso
Systems Group, Department of Computer Science, ETH Zurich
{firstname.lastname}@inf.ethz.ch

***Abstract*—In the same way multi-core and CPU clusters are used for large problems, multi-FPGA clusters are needed to tackle applications not fitting within a single FPGA, such as machine learning methods based on large models. Recent FPGA deployments in datacenters offer flexible pools of FPGAs that can be used in different configurations. In addition to those of a typical cluster architecture, FPGA clusters often have added capabilities such as being hosted by large server nodes (e.g., Amazon F1 Instance), or a network backbone directly connecting multiple FPGAs (e.g., Microsoft Catapult). While such designs open up many opportunities, mapping application logic onto a pool of FPGA resources is a non trivial task. It requires partitioning the application across multiple FPGAs, inter-FPGA communication management of multiple data stream classes, and balancing communication-computation bandwidth.**

In this paper, we explore and develop techniques for mapping a resource-intensive machine learning application, namely inference over decision tree ensembles on a datacenter-grade FPGA cluster. The FPGA cluster is built out of 20 Microsoft Catapult FPGA boards with a flexible inter-FPGA network topology. We developed a lightweight inter-FPGA communication protocol and routing layer to facilitate the communication between different parts of the application. Our evaluation provides insights on the overall performance benefits of the design and outlines some of the techniques needed to efficiently map applications onto a pool of distributed FPGAs.

I. Introduction

Machine learning workloads have become significant in datacenters. Recent work has demonstrated high performance gains from using FPGAs to accelerate machine learning, mostly by using single FPGA implementations [14], [19], [8], [2]. However, as the complexity of machine learning algorithms increases (e.g., deep ResNets [7], [13], large decision tree ensembles [5], etc.) and more data becomes available for training and testing, the resources of a single FPGA device are not enough to efficiently implement these complex algorithms. Hence, clustering multiple FPGAs is imperative to tackle today's big data applications.

As FPGAs are becoming commodities in datacenters [1], [4], [12], a broader range of users are considering FPGAs as accelerators for compute and data intensive workloads. FPGA deployments offer a large, flexible pool of reconfigurable resources, allowing users to cluster multiple FPGAs to tackle big data applications. Recent FPGA deployments provide more than a typical CPU-FPGA server node with a single PCIe attached FPGA device. A server node may have multiple PCIe attached FPGA boards to the same host CPU (e.g., Amazon large F1 instances [1]). FPGA devices may have direct network access and share a network switch with other FPGAs in neighbor server nodes (e.g., Microsoft Catapult [4]). These designs allow low latency, high bandwidth inter-FPGA communication without going through the CPU and software network stack. While such configurations open up opportunities, they also raise challenges since mapping application logic onto a pool of FPGA resources is not trivial.

One such challenge is the inter-FPGA communication management needed to support multiple data stream classes. Different communication scenarios arise depending on how the implementation is spread over the cluster nodes. For example, the initial data may be partitioned and distributed from a master node to all other nodes in the cluster, or it can be broadcasted to all nodes from a central node. Intermediate-state generated from one FPGA might need to be transferred to another FPGA node in the cluster. Partial results have to be aggregated from all nodes and written back to the application software. Doing so requires balancing the communication-computation bandwidth between the cluster nodes to maximize resource utilization and performance.

In this work, we study the use of clustered FPGA resources to accelerate a large machine learning application: inference over a decision tree ensemble. Inference is a resource-demanding operation and exhibits different behaviors depending on the ensemble size and data dimensionality. For large tree ensembles it becomes compute-bound, while increasing data dimensionality makes it memory and network bound.

In this paper we explore different ways to partition and map the inference operation onto an FPGA cluster, such that it scales up linearly as we add more FPGA nodes. We decouple the I/O part of the inference operation from the compute part to efficiently manage and map the different communication scenarios on the cluster network. Through the implementation of the inference operation, we have developed a set of software and hardware constructs which can be reused for other complex operations. Our main objective is to draw insights from our case study that can be generalized for other operations and FPGA clusters configurations, which then can be used to outline a framework with software/hardware components and a methodology to partition, balance, and implement large applications on clustered FPGAs.

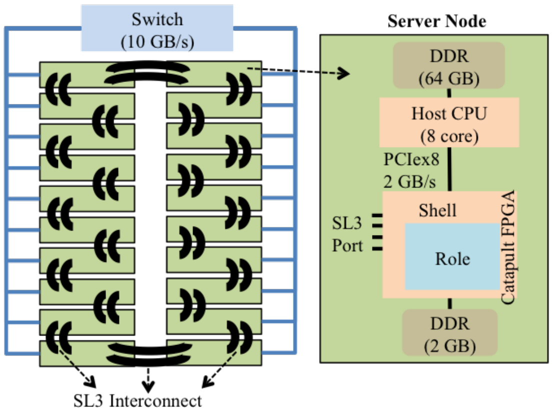
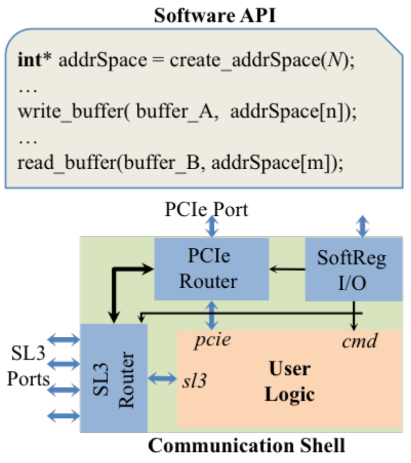


Fig. 1: Target FPGA cluster architecture.



```
int* addrSpace = create_addrSpace(N);
…
write_buffer( buffer_A,  addrSpace[n]);
…
read_buffer(buffer_B, addrSpace[m]);
```

Fig. 2: Communication management shell architecture.

II. Elastic FPGA Cluster Architecture

A. Cluster Architecture

Our FPGA cluster is built out of 20 server nodes each featuring an 8-core Xeon CPU (E5-2620) and a PCIe-attached Microsoft Catapult FPGA board[1](academic version 1.2) as shown in Figure 1. This is the first iteration of Catapult boards used to accelerate the Bing search engine [16]. The Catapult board includes one Intel's Stratix V FPGA (5SGSMD5H2F35I3L) and a 2 GB DDR channel attached to the FPGA. The server node is equipped with a 64 GB DDR3 attached to the CPU. Microsoft Windows Server 2012 R2 is used on the servers.

The server nodes are organized in two racks and all the 20 nodes are connected to a 10 Gbps network switch. In addition to a PCIex8 port, the FPGA has four Serial Lite III (SL3) ports which can be connected directly to other FPGAs. The SL3 ports are used to create a ring network topology connecting all the FPGAs. The FPGA is partitioned into two regions: Shell and Role. The Shell abstracts low level I/O resources (PCIe, SL3, DDR) providing simple streaming interfaces for the Role region where the user logic is implemented. On the software side, a low level driver which facilitates access to the FPGA and implements communication constructs with the FPGA Shell region. It provides simple I/O data structures such as buffers and registers for application developers to move data in and out of the FPGA.

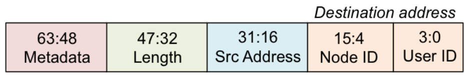
B. Inter-FPGA Communication Framework

To enable communication between the application partitions distributed over the cluster nodes, we have developed a packet-based light-weight communication framework. The framework supports communication only over the inter-FPGA network and does not cover communication over the inter-CPU 10 Gbps network. However, it can be extended easily to combine all networks in the cluster connecting the different compute units into a unified logical network, something we plan to do as part of future work. The communication Shell in the FPGA (Figure 2) implements two features. First, it provides a direct communication path between the PCIe and SL3 ports allowing the CPU to write directly to remote FPGAs. Second, it implements a routing layer determining which SL3 port to send packets over, and routes incoming packets over a SL3 port to its final destination FPGA.

[1]The Catapult boards are a generous donation from Microsoft.

The communication shell includes software interfaces allowing an application to create an address map of all its parts. Then, the application uses this address mapping to specify sources and destinations when reading and writing buffers.

Communication protocol. The communication protocol supports variable length packets with maximum size of 16 KB. A packet consists of one or more 128-bit words and a 128-bit header. Figure 3 shows the format of the packet header lower 64-bits, the upper 64-bits are currently reserved. The header includes four bit-fields; destination address, source address, packet length and metadata. The metadata field is set by the source user module and passed unchanged to the destination user module. The address format consists of a node ID field indicating the destination/source server node, and a user ID indicating the user module running on the server node. The communication shell supports multiple user modules occupying the user logic region running on the FPGA concurrently.

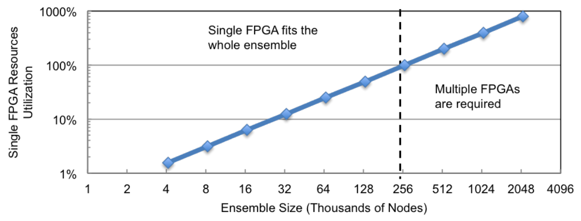


			Destination address	
63:48 Metadata	47:32 Length	31:16 Src Address	15:4 Node ID	3:0 User ID

Fig. 3: Packet header format.

III. Case Study: Decision Tree Ensemble Inference

Inference over decision tree ensembles is a resource-demanding operation in terms of both compute and memory [11], [6], [14]. Moreover, depending on the ensemble size and data dimensionality, it can become either compute-bound or network-bound. Figure 4 depicts how the resources requirements grow with increasing tree ensemble size. A single FPGA has capacity for 256K tree nodes. A larger ensemble,



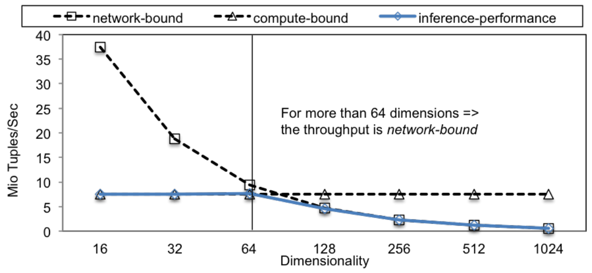


Fig. 4: (Left) Resources requirements with respect to ensemble size. (Right) Performance behavior of decision tree ensemble inference with respect to data dimensions.

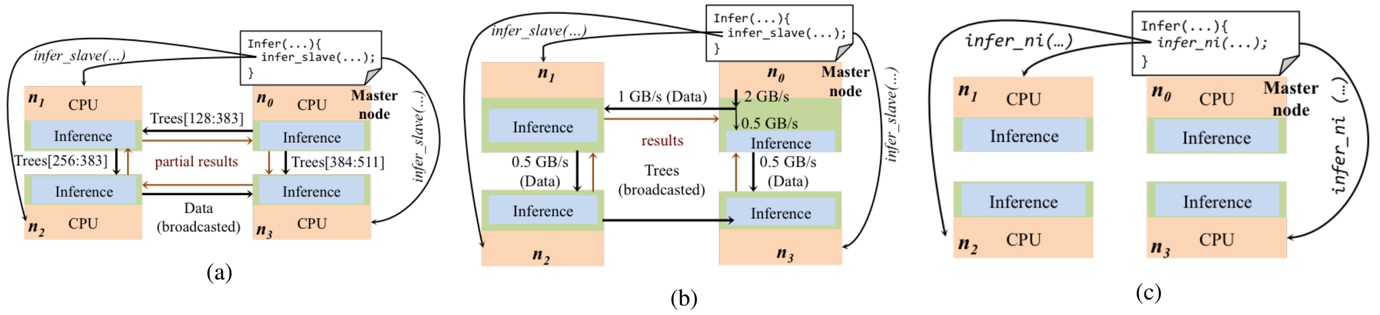


Fig. 5: (a) Partitioning large ensemble over multiple FPGAs. (b) Replicating tree ensemble on all FPGAs and partitioning data over all FPGAs. (c) Distributed data over all server nodes.

for example 512K nodes, needs two FPGAs. As an example, Essen et. al [6] used four Virtex-6 FPGAs to fit an ensemble of 234 trees (6 levels deep).

Figure 4 also shows how the behavior of the inference operation changes according to data dimensionality (i.e., the number of features of a data tuple). Increasing the ensemble size makes the problem compute-bound, while increasing the data dimensionality makes it network-bound. To support different behaviors, different mappings on the FPGA cluster are necessary. In the following section we discuss the different possibilities.

A. Multi-FPGA Implementation

We consider three configurations for mapping the inference operation onto an FPGA cluster. The first mapping targets large tree ensembles which do not fit onto a single FPGA. In this case, we partition the tree ensemble over all the cluster FPGAs as depicted in Figure 5(a). In this mapping, there is a single master node (n_0) where all the data is stored and the inter-FPGA network is used to aggregate the resources of all the FPGAs in the cluster. The master node first distributes the trees to all FPGAs over the SL3 interconnect, then it broadcasts the data tuples to all FPGAs, also using the SL3 interconnect. At the same time, the master node invokes a slave instance of the inference operation on the rest of the server nodes CPUs, which only initializes the FPGA and triggers the decision tree inference in each server node. This communication with remote CPUs occurs over the 10 Gbps Ethernet network. The partial results computed in each FPGA are propagated in one direction through the SL3 interconnect starting from the master node toward the last node in the ring (n_3). The partial results are aggregated while moving from one FPGA to another, until the final results arrive back at the master node, where they are pushed through the PCIe to the application software.

The second mapping scenario in Figure 5(b) is used when the tree ensemble fits into a single FPGA but it is compute bound. In this mapping, the inference operation consumes a very small portion of the PCIe/Network bandwidth, Figure 5(b) shows an example where the inference operations consumes 0.5 GB/s of network bandwidth. One way to scale up performance is by replicating the tree ensemble on multiple FPGAs and partitioning the data between the FPGAs for parallel processing. The master node first broadcast the trees to all FPGA nodes through the SL3 interconnect, then distributes data partitions to all FPGAs over the SL3 interconnects. The example in Figure 5(b) shows how the data is distributed in a bidirectional ring network and how the allocated 4 FPGAs consume the 2 GB/s PCIe bandwidth over which the data is pushed.

In the third mapping shown in Figure 5(c), the inference engine on a single FPGA is network-bound. This occurs for large data dimensionality as we saw in Figure 4. In this case, multiple FPGAs are beneficial if the data is already distributed over the server nodes as a distributed database would do. In this mapping, the aggregate bandwidth of multiple PCIe ports allows moving more data to the FPGAs, increasing performance linearly. In this mapping, no inter-FPGA communication occurs as every FPGA operates on its local copy

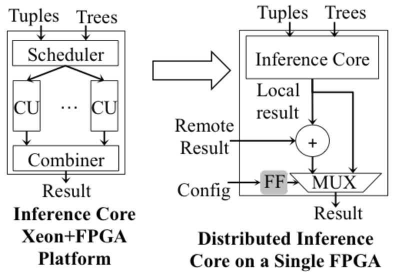


Fig. 6: Adapted inference implementation to a multi-FPGA distributed logic.

of data and the tree ensemble fits in a single FPGA.

Application mapping insights. Inference over decision tree ensemble is an example of algorithms which easily map to multi-FPGA clusters. Since the inference over individual trees is independent from each other, trees can be distributed and processed in parallel on different FPGAs. However, it is important to avoid splitting a tree over multiple FPGAs, as this will produce extra communication overhead on inter-FPGA channels. The aggregation of partial results from individual trees can be pipelined, hence partial results run in one direction over the FPGA ring to minimize communication overhead. A rule of thumb in partitioning an application over multi-FPGA cluster is to make a data stream to run only in one direction (i.e., pipeline fashion), and keep heavily dependent application components close to each other as much as possible. In other words spatial locality is important to minimize communication overhead and maximize throughput.

B. *Inference Engine Implementation*

We adapted our previous implementation [14] on Intel's HARP platform to run on the Catapult cluster. The implementation consists of an inference core as shown in Figure 6 and an I/O unit (not shown in the figure) to read tuples and trees from CPU main memory and write inference results back. We extracted the inference core from the original implementation and added to it a floating point adder to aggregate partial results from remote FPGAs to the local result (if the ensemble partitioned between multiple FPGAs). It is also possible to bypass the adder and output directly the local result if every FPGA is programmed at runtime to process the complete ensemble. To feed tuples/trees to the inference core and route results to the right destination, we replaced the I/O unit in the original implementation with a runtime programmable

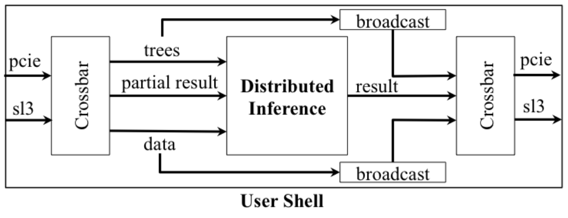


Fig. 7: Components of the User Shell around the user logic.

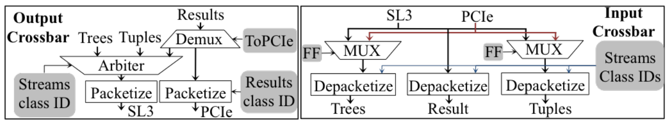


Fig. 8: User Shell crossbars architecture.

I/O user shell (Figure 7). The user shell supports several communication scenarios and the different mappings of the inference operation as discussed in the previous section. In the user shell we implement programmable crossbars (Figure 8) on the transmit and receive channels. The input crossbar is configured with the source interface of each data stream (e.g., trees from SL3, tuples from PCIe). Results always come over SL3 from remote FPGAs. The *De-packetize* module first determines if the incoming stream is of a certain class (tuples, trees, or results), then it eliminates the packet header and passes the data stream. The output crossbar is configured to pass results either over PCIe or SL3, and to arbitrate between the different streams over SL3. It then puts data into packets and communicates it to the SL3 interface. In the user shell, we implement a broadcast capability which can be enabled or disabled from software for either data or trees depending on the target mapping.

The concepts and techniques we use to develop the user shell can be generalized and reused for many other operations in machine learning. The user shell can be extended with extra functionality to support different I/O behaviors.

IV. Evaluation

A. *Experimental Setup*

We run the experiments on 1 million data tuples in all configurations. We used the Scikit-learn [15] machine learning package in Python to generate data sets for our experiments. We generated a training data set of 100K data points and a testing data set of 1 million data tuples. We generated two data sets, one for data with 32 dimensions and another with 256 dimensions. We used XGBoost [5] to generate eight decision tree ensembles each of size 512 trees. The ensembles differ in the maximum depth of their trees starting from 6 to 13 levels. We run the experiments on 1, 2, 4, 8, and 16 FPGAs. In all

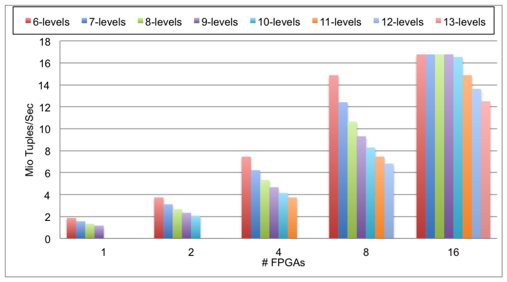


Fig. 9: Performance scaling with number of FPGAs used with low dimensionality.

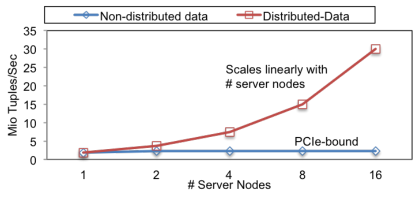


Fig. 10: Performance scaling with number of FPGAs used for data with high dimensionality.

configurations, the FPGAs are connected in a ring network topology.

B. Performance Scale up

In this experiment, we evaluate the performance of the inference operation scalability when adding more FPGA nodes. The data resides on the master node from where it is distributed to the rest of the server nodes through the SL3 ring network. We vary the ensemble size by increasing the tree depth from 6 to 13 levels for 512 trees. Figure 9 shows the experiment result.

Up to tree depth 9, a tree ensemble fits into a single FPGA, hence we apply the second mapping as in Figure 5(b). Doubling the number of FPGA nodes doubles the performance. For 16 FPGAs, the system becomes network-bound which limits the performance to 16.7 Mio Tuples/Second.

For trees deeper than 9 levels, the tree ensemble does not fit into a single FPGA. For these ensembles we apply the first mapping in Figure 5(a). For a tree depth of 10, two FPGAs are required to accommodate the ensemble. Every additional level requires doubling the number of FPGAs to fit the ensemble. This explains the missing bars for different numbers of FPGAs. Linear scalability is also achieved as we distribute the ensemble across multiple FPGAs up to 16 FPGAs, where the network bandwidth becomes the limiting factor.

We run a second experiment for a data set with 256 dimensions. For this number of dimensions, the PCIe and SL3 network bandwidth become the limiting factor. Figure 10 shows that the non-distributed data scenario, where data resides on a master node, does not scale up when adding more FPGA nodes because the PCIe bandwidth becomes the limiting factor. On the other hand, a scenario where the data is already distributed across the server nodes in the cluster achieves linear scalability when more server nodes are added.

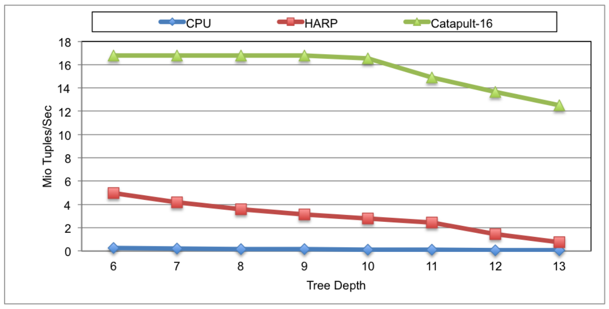


Fig. 11: Performance results for a ring of 16 Catapult boards compared to 10-threaded CPU and HARP Xeon+Stratix V FPGA. Results for 512 trees ensemble.

TABLE I: Consumed FPGA resources by different modules.

Module	ALMs		M20k	
Catapult Shell	33008	19.1%	104	5.2%
Communication Shell	3832	2.2%	113	5.6%
User Shell	1136	0.7%	38	1.9%
Inference Core	33070	19.2%	527	26.2%

C. Performance comparisons

Figure 11 compares our performance results on 16 FPGA nodes with our prior work on Intel's Xeon+FPGA platform [12]. The implementation on the Intel's Xeon+FPGA allocates double the amount of compute resources compared to a single Catapult FPGA implementation. Scaling to 16 FPGAs achieves 3X to 16X speedup over prior work. If a larger FPGA is used, a similar throughput can be reached with less nodes.

D. Network Latency and Resources

Figure 12 shows the round trip time for a packet to travel from the master node FPGA to each other FPGA in the cluster. This FPGA-to-FPGA round trip runs through the SL3 interconnect and does not include the PCIe latency. Since the FPGAs are connected using a ring topology, the latency grows linearly with the remote FPGA distance from the master node. The maximum latency occurs to the FPGA node in the middle of the ring. The inference operation is latency insensitive as long there is sufficient amount of data keeping the network busy the whole runtime.

Table I shows resource utilization by the different modules in a single FPGA. The communication and user shells consume 7.5% of the memory resources and less that 3% logic resources, leaving most of the Role region available for user logic.

V. Related Work

Many researchers have explored combining the resources of multiple FPGA devices to overcome limitations of a single FPGA. A commonly used approach is a ring network topology connecting multiple FPGA devices using direct FPGA-to-FPGA serial interconnect [3], [10], [20].

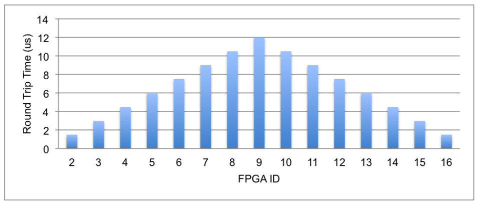


Fig. 12: Round Trip Latency measured on a ring of 16 FPGAs from a master node.

Castillo et. al. [3] proposed a multi-FPGA platform (SMILE) with 32 FPGA devices to accelerate options pricing for financial institutions. The FPGAs are connected in a ring network topology using Aurora serial interconnects and controlled by a host CPU. Mencer et. al. [10] developed a different massively parallel multi-FPGA platform built out of 8 boards each including 64 FPGAs. The FPGAs in each board are connected in a systolic array using FPGA-to-FPGA high speed, on board interconnects. A host CPU is used to configure and run the application on the FPGAs. Zhang et. al. [20] proposed an implementation of the AlexNet CNN which is scaled on a deep pipeline of 6 FPGA devices connected in a ring topology using the Aurora serial interconnects. These designs serve as a practical example of a massive server node architecture with multiple FPGA devices similar to Amazon's large F1 instances [1]. In addition, they highlight the great benefits of having a low overhead, high bandwidth inter-FPGA network. In this paper, we opted to apply similar concepts to scale up the performance of inference over decision tree ensembles. Moreover, we demonstrated the different possible scenarios for inter-FPGA communication together with the host CPU.

Another group of researchers have considered a typical cluster architecture with multiple server nodes connected through an Ethernet network. Kono et. al. [9] studied the scalability of the lattice Boltzmann computation on an FPGA cluster, where each server node includes a host CPU and two PCIe attached FPGAs. All the FPGAs are connected through a ring network of direct FPGA-to-FPGA serial interconnects. Tsoi et. al. [18] proposed a heterogeneous cluster architecture with server nodes combining CPU, FPGA and GPU compute units. High speed Infiniband interconnect is used to connect all the FPGAs in a ring network topology. The authors show how to implement Map-Reduce, developing the necessary software abstractions to facilitate applications development.

Tarafdar et. al. [17] considers a server node architecture with a PCIe attached FPGA board. Multiple FPGAs in the cluster communicate with each other through a 1 Gbps Ethernet network. The authors develop a framework based on OpenStack and Xilinx SDAccel hypervisor allowing developers to specify the different application kernels and how they communicate. The framework then maps logical resources to physical resources. Our work is complementary to Tarafdar et. al. We draw insights on how users should partition and map their application onto multiple FPGA devices. In addition, we demonstrate how to decouple I/O from compute in an application and implement a user shell that decodes data streams into user specified stream categories(e.g., tuples, trees, partial results). These techniques can be combined with the framework proposed by Tarafdar et. al. to facilitate application mapping on FPGA clusters.

VI. Conclusion

In this paper we explore the design space of the inference over decision tree ensembles on a cluster of FPGAs. Through the proposed designs, we draw insights on how to distribute computations over multiple FPGAs to overcome resources limitations and balance network and compute bandwidth to scale up performance with more nodes added. Modeling the target operation I/O behavior is necessary to maximize cluster utilization and scale up performance linearly.

Implementing an algorithm on a cluster of FPGAs is a daunting task, in particular, implementing the hardware and software I/O management components. We highlight the importance of decoupling I/O from the compute part in the algorithm, and developing reusable hardware and software I/O management components to productively map complex operations to an FPGA cluster.

Acknowledgment

We would like to thank Microsoft for the generous donation of the Catapult boards and Intel for the generous donation of the Xeon+Arria 10 platform.

References

[1] Amazon EC2 F1 Instances.

[2] Z. K. Baker and V. K. Prasanna. Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs. In *FCCM'05*.

[3] J. Castillo, J. L. Bosque, E. Castillo, P. Huerta, and J. Martinez. Hardware Accelerated Montecarlo Financial Simulation over Low Cost FPGA Cluster. In *IPDPS'09*.

[4] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, H. Michael, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger. A cloud-scale acceleration architecture. In *MICRO'16*.

[5] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *KDD'16*.

[6] B. V. Essen, C. Macaraeg, M. Gokhale, and R. Prenger. Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA? In *FCCM'12*.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR'16*.

[8] K. Kara, D. Alistarh, C. Zhang, O. Mutlu, and G. Alonso. FPGA accelerated dense linear machine learning: A precision-convergence trade-off. In *FCCM'17*.

[9] Y. Kono, K. Sano, and S. Yamamoto. Scalability Analysis of Tightly-Coupled FPGA-Cluster for Lattice Boltzmann Computation. In *FPL'12*.

[10] O. Mencer, K. H. Tsoi, and S. Craimer. Cube: A 512-FPGA cluster.

[11] J. Oberg, K. Eguro, and R. Bittner. Random decision tree body part recognition using FPGAs. In *FPL'12*.

[12] N. Oliver, R. Sharma, S. Chang, et al. A Reconfigurable Computing System Based on a Cache-Coherent Fabric. In *ReConFig'11*.

[13] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. Chung. Accelerating Deep Convolutional Neural Networks Using Specialized Hardware, 2015.

[14] M. Owaida, H. Zhang, C. Zhang, and G. Alonso. Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms. In *FPL'17*.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[16] A. Putnam, A. M. Caulfield, E. S. Chung, and D. e. a. Chiou. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *ISCA'14*.

[17] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow. Enabling Flexible Network FPGA Clusters in a Heterogeneous Cloud Data Center. In *FPGA'17*.

[18] K. H. Tsoi and W. Luk. Axel: a heterogeneous cluster with FPGAs and GPUs. In *FPGA'10*.

[19] C. Zhang, P. Li2, G. Sun, Y. Guan1, B. Xiao, and J. Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *FPGA'15*.

[20] 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'16*.