Disaggregated FPGAs: Network Performance Comparison against Bare-Metal Servers, Virtual Machines and Linux Containers

Jagath Weerasinghe, Francois Abel, Christoph Hagleitner
IBM Research - Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
Email: {wee,fab,hle}@zurich.ibm.com

Andreas Herkersdorf
Institute for Integrated Systems
Technical University of Munich,
Munich, Germany
Email: herkersdorf@tum.de

CloudCom2016

***Abstract*—FPGAs (Field Programmable Gate Arrays) are making their way into data centers (DC). They are used as accelerators to boost the compute power of individual server nodes and to improve the overall power efficiency. Meanwhile, DC infrastructures are being redesigned to pack ever more compute capacity into the same volume and power envelops. This redesign leads to the disaggregation of the server and its resources into a collection of standalone computing, memory, and storage modules.**

To embrace this evolution, we propose an architecture that decouples the FPGA from the CPU of the server by connecting the FPGA directly to the DC network. This proposal turns the FPGA into a network-attached computing resource that can be incorporated with disaggregated servers into these emerging data centers. We implemented a prototype and compared its network performance with that obtained from bare metal servers (Native), virtual machines (VM), and containers (CT). The results show that standalone network-attached FPGAs outperform them in terms of network latency and throughput by a factor of up to 35x and 73x, respectively. We also observed that the proposed architecture consumes only 14% of the total FPGA resources.

***Keywords*—network-attached FPGA; disaggregated server; data center.**

I. Introduction

In the current era of big data, large-scale applications ranging from business analytics to scientific simulations are moving to the cloud. The compute infrastructure of the DCs, which host these applications, is typically based on bare-metal servers, VMs, and CTs. While computing frameworks, such as Hadoop or Spark, enable distributed computing on large clusters, many applications do not scale well on the current DC infrastructures because of the limited compute power of server nodes and the performance of the network that interconnects them [1] [2]. Meanwhile, FPGAs are making their way into DCs. Traditionally, FPGAs are assembled on a PCIe add-in card, and they are used as accelerators [3] [4] [5] [6] [7] to boost the compute power of individual server nodes and to improve their overall power efficiency. However, this approach limits the number of FPGAs that can be deployed per server and therefore hinders the potential of offloading large-scale applications with this kind of bus-attached accelerators.

At the same time, DC infrastructures are being redesigned to pack ever more compute capacity into the same volume and power envelops. This redesign leads to the disaggregation of the server and its resources into a collection of standalone computing, memory, and storage modules. In practice, it translates into the transmutation of the traditional rack- and blade-servers into sled- and micro-servers [8]. This move is purely driven by the performance-per-dollar metric, which is improved (i) by increasing the density of the servers by sharing resources, such as power supplies, PCB backplanes, cooling, fans, networking uplinks, and other management infrastructures, and (ii) by pruning the extraneous parts to the processor, to the local memory and to the boot storage.

We observed these two trends, and predict that if FPGAs want to continue gaining ground in future DCs, a change of paradigm is required for the FPGA-to-CPU attachment as well as for the form factor of the FPGA-cards. The form factor of the existing cards and their CPU-to-FPGA interface are not compatible with, or even no longer available on, the emerging dense and cost-optimized servers. Instead, in a disaggregated server design, the individual computing, memory, and storage resources are interconnected over a network [9]. There are three main implications for the FPGAs when following this new approach. First, in contrast to the traditional co-processor approach in CPU bus-attachment, a novel architecture is needed to enable such a connection of the FPGA to the network. Second, once dismantled from any server host coupling, an FPGA must be turned into a self-contained standalone appliance capable of managing itself. Third, a resource management framework is needed to manage this new class of independent resources.

While resource disaggregation can bring greater modularity by allowing DC operators to optimize their deployments for improved efficiency and performance [9], the feasibility of such an approach has to be validated for FPGAs. To assess the feasibility of disaggregation, two metrics are important: (i) the network performance and (ii) the FPGA resource consumption for realizing such an architecture. Our contributions in this paper are threefold:

1) we propose an architecture for standalone network-attached FPGA
2) we show the implementation of this architecture on a merchant FPGA
3) we compare the network performance of our prototype against the existing cloud compute resources, namely bare-matal servers, VMs and CTs.

The remainder of this paper is organized as follows: We explain the related work in Section II and we introduce the concept of disaggregated FPGA resource in Section III. Section IV elaborates on the proposed architecture and we discuss the experimental results in Section V and the insights gained from them in Section VI. Section VII explains how we build the DC infrastructure, and we conclude in Section VIII.

II. Related Work

The common approach for deploying FPGAs in a server is by tightly coupling one or two FPGAs to the CPU over the PCIe bus. However, this PCIe-attachment has two major issues in DC deployment. First, the power consumption of a server is order of magnitude higher than that of an FPGA. Hence, the power efficiency that can be gained by offloading tasks from the server to 1 or 2 FPGAs is very limited [10]. Second, in DCs, the workloads are heterogeneous and they run at different scales. Therefore, the scalability and the flexibility of the FPGA infrastructure is vital to meet the dynamic processing demands. With PCIe-attachment, a large number of FPGAs can not be assigned to run a workload independent of the number of CPUs, and also those FPGAs can not be connected in flexible user defined topologies. Some large scale FPGA deployments [3], get around this issue of scalablity and flexibility to a certain extent by having a secondary dedicated network, connecting multiple PCIe-attached FPGAs together. However, a dedicated secondary network breaks the homogeneity of the DC network, and increases the infrastructure management overhead.

The other approach for deploying FPGAs is by attaching them directly over the DC network [4] [5] [11] [12], which significantly improves the scalability and the flexibility of the FPGA infrastructure compared to the PCIe-attachment. Even though previous attempts do provide a network connection, the FPGA always remains physically attached, hosted and controlled by a dedicated server. Instead, we proposed the concept of standalone network-attached FPGA in [13] in order to completely disaggregate the FPGA resource from the server. This approach sets the FPGA free from the traditional CPU-FPGA attachment and is the key enabler for large-scale deployments of FPGAs in DCs. This work is a continuation of [13], and in this work, we show a system architecture and an implementation of a disaggregated FPGA resource, and we evaluate its network performance.

III. Disaggregated FPGA Resource

We set the following two main requirements for the development of an architecture for a disaggregated FPGA. First, because the disaggregated FPGA resource is decoupled from any host, we want it to operate in a self-contained manner by executing tasks that were previously under the control of a host CPU. These tasks include the ability to perform power-up and -down actions, to hook itself up to the network after power-up, and to perform all local health-monitoring and system-management duties. Second, we want infrastructure providers to provision such disaggregated FPGAs in the DC for users to deploy their customized application logic on demand, similarly to VMs.

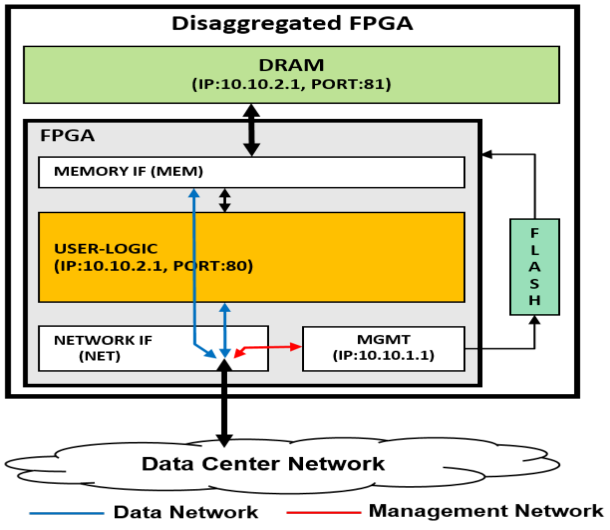

Fig. 1. Disaggregated FPGA Resource

Figure 1 illustrates the system architecture of the proposed disaggregated FPGA. It consists of an FPGA to serve as generic and programmable hardware accelerator, a nonvolatile memory device, such as a serial or parallel flash memory, to store the FPGA's configuration information, and optional static or dynamic scratchpad memories.

To support the above two requirements, we define two regions in the floorplan of the FPGA: a *vendor* (infrastructure vendor) and a *user* region. The *vendor* region is persistent as long as the FPGA is powered up. It implements the minimum functions required for the FPGA to boot and to connect to the DC network. These functions include a memory management layer (MEM) for interfacing with external memories, a management unit (MGMT) for monitoring and controlling the disaggregated FPGA resource, and a network layer (NET) for interfacing with the DC network. These three functions are represented by white boxes in Figure 1. The *user* region is shown in yellow in Figure 1. It represents the larger part of the reconfigurable logic and is dynamically allocated to the user's applications by a resource-provisioning service, which control the programming and erasing of this region. Figure 1 also illustrates the multiplexing and de-multiplexing performed by the network layer of the *vendor* region between the user data traffic and the management traffic.

The *vendor* and *user* regions of the FPGA are configured in two steps. First, the FPGA is booted from a default initial system setup image provided by the DC infrastructure provider.

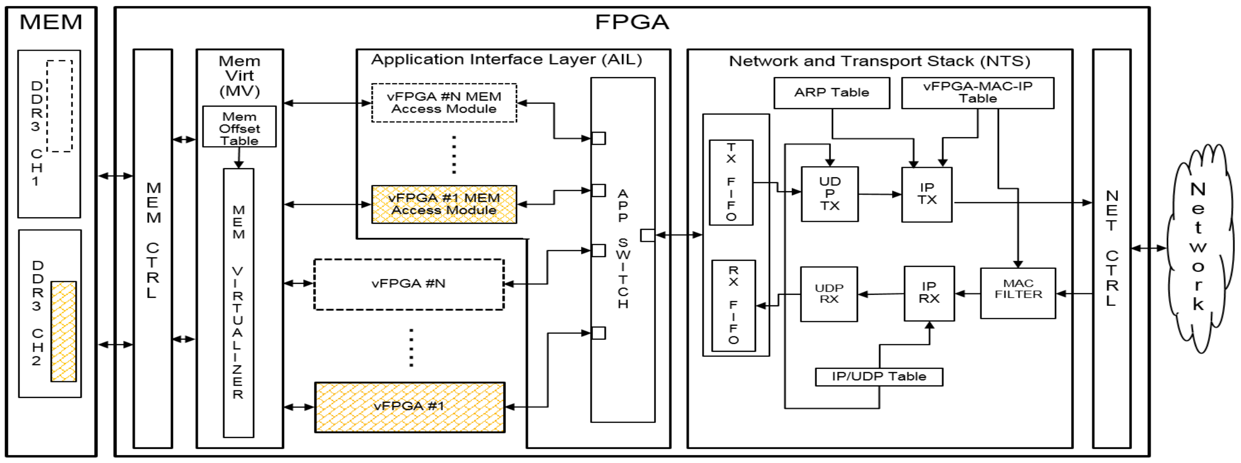

Fig. 2. Architecture of the Disaggregated FPGA Prototype

This image includes the above three functions (MEM, MGMT, NET), which are needed for the board to boot and signal its presence on the DC network by requesting the assignment of a dynamic IP address. Once such a default disaggregated FPGA is up and running in the DC, it can be allocated to a user by a resource-provisioning service as described in [13]. Second, a partial reconfigurable bitstream containing only the customized logic of the user's application is sent over the DC network to the management layer of the FPGA, which triggers the partial reconfiguration of the *user* region, before setting the corresponding unique MAC and IP addresses into the network layer. In the second step, the use of partial reconfiguration is optional. A full bit stream containing both the *vendor* and the *user* regions can be received and stored into the flash device, at the cost of the FPGA being rebooted and therefore disconnected from the network for few seconds.

IV. Implementation

To realize the disaggregated FPGA architecture introduced in Section III, we implemented a prototype on a Xilinx Virtex7 FPGA. This section covers the implementation of the prototype, which is shown in Figure 2.

A. User Logic

The user-logic region of the FPGA can be dedicated to one or multiple independent applications, which we call virtual FPGAs (vFPGA). In our architecture, the vFPGA interfaces are implemented in a customizable way, so that the number of vFPGAs in a particular design can be changed just by setting a parameter in the code. In this prototype, we implemented four vFPGAs by dividing the user-logic region of the physical FPGA.

The vFPGAs operate as independent devices, and each vFPGA consists of a MAC address, an IP address, and a UDP port. For each vFPGA, 4 GB of memory and a 32-bit virtual address space are allocated in the external DRAM. A vFGPA and its external memory can be accessed by a host over the network. The external memory is assigned the same MAC and IP address as the vFPGA, but with a different UDP port.

B. Vendor Logic

The vendor logic consists of an interface to the external memory and an interface to the network.

1) Memory Interface:

a) Memory Controller (MEM CTRL): For the memory controller, we used a Xilinx MIG (Memory Interface Generator) dual-channel memory controller with an AXI interface (Advanced eXtensible Interface).

b) Memory Virtualizer (MV): The virtualizer divides the total DRAM into multiple physical memory chunks. The mem-offset table is used to assign a requested amount of memory capacity to a particular vFPGA. Each physical memory chunk is allocated to a single vFPGA and can be accessed by the associated vFPGA as well as by an external host over the network.

2) Network Interface:

a) Network Controller (NET CTRL): The network controller implements the data link layer (L2) of the Ethernet standard, for which we used a Xilinx 10GbE MAC IP core. A Xilinx 10GbE PHY IP core connects the MAC to the external PHY over the integrated GTX transceivers of the FPGA.

b) Network and Transport Stack (NTS): The NTS provides a MAC address filtering layer, an IP layer, and an UDP layer. We implemented these layers with a 64b bus at 156.25 MHz. The minimum size of the Ethernet frame that travels in the physical layer is 84 bytes, and the UDP payloads of up to 18B make this the smallest Ethernet frame on the wire. Therefore to achieve the line rate, each module in the data path must process the packets of up to 18B of UDP payload in less than 10.5 clock cycles. We designed each module of the data path to satisfy this criterion.

In the NTS, we do not implement control path protocols such as ARP (Address Resolution Protocol). In the current implementation, we use static ARP tables. Instead of implementing a distributed control plane by having the ARP functionality in each FPGA, we plan to have a centralized control plane, in which the ARP tables are programed using a software-defined network (SDN) technique. We implemented static UDP/IP tables, that consist of five tuples: (1) source port, (2) destination port, (3) source IP, (4) destination IP, and (5) buffer ID. We plan to use the same SDN-based approach to program the five tuples that belong to open ports of each vFPGA. The vFPGA-MAC-IP table consists of MAC and IP addresses of each vFPGA. The IP/UDP table and the vFPGA-MAC-IP table are updated when new vFPGAs are started on the physical FPGA.

3) Application Interface Layer (AIL): The AIL has two main functions: (1) It contains the memory access modules corresponding to each vFPGA to enable access to the memory that belongs to each vFPGA over the network by external hosts. (2) It serves as a switch to multiplex and de-multiplex incoming and outgoing packets to and from vFPGAs and memory access modules. Furthermore, it hides the socket programming from vFPGA users by offering a simple IO interface to communicate with other FPGAs and SW applications.

V. Experiments and Results

We evaluated our architecture in terms of network latency, throughput, application predictability, and resource consumption. The disaggregated FPGA architecture presented was implemented and validated on a Alpha Data PCIe card featuring a Xilinx Virtex7 XC7VX690T FPGA. The design uses one 10GbE network interface and two 1333 MHz DDR3 SODIMMs with 8 GB each. In the experimental setup, the Alpha Data FPGA cards are plugged into a PCIe expansion chassis [14], from which only power is taken for the cards. The FPGA cards and two servers are connected to a top-of-rack switch as shown in Figure 3. Each server consists of a 4-core (with 2 CPU threads per each core) Intel i7-3820 clocked at 3.6 GHz and has 32 GB of main memory. The servers run Linux (Fedora 22, Kernel 4.0) and are equipped with a Mellanox ConnectX-3Pro 10G Ethernet Controller on PCIe Gen3.

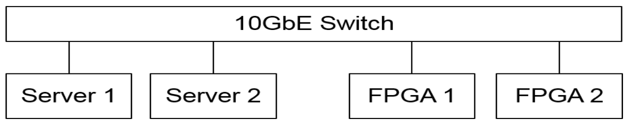

Fig. 3. Experimental Setup

We compared the UDP network latency, throughput, and variance of response time of multiple SW configurations with those of the FPGA. For the SW configurations, we used VMs, CT, and bare metal servers (Native). In the case of VMs, we also used the VM direct IO (VMDIO) network configuration, which is a technology used to assign a NIC directly to the VMs. Altogether we investigated six experimental cases: (a) VM-VM, (b) VMDIO-VMDIO, (c) CT-CT, (d) Native-Native, (e) Native-FPGA, and (f) FPGA-FPGA. In each configuration, the first member acts as the client and the second as the server.

The VMs run the same Linux configuration explained above on two KVM (Kernel-based Virtual Machine) hosts. Each VM has a single core running at 3.6 GHz and 4 GB of memory. For the CTs, we used standard Linux application containers.

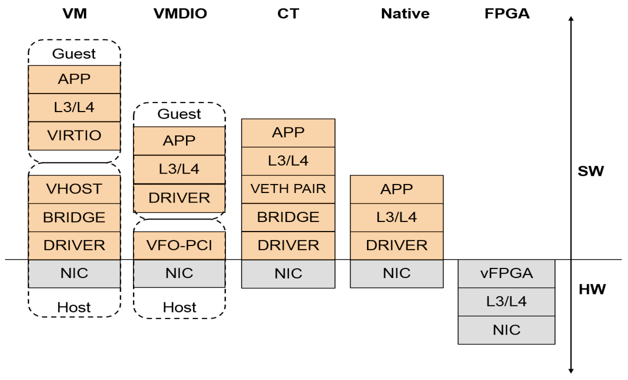

Fig. 4. Network Stack Configurations of the Experimental Cases

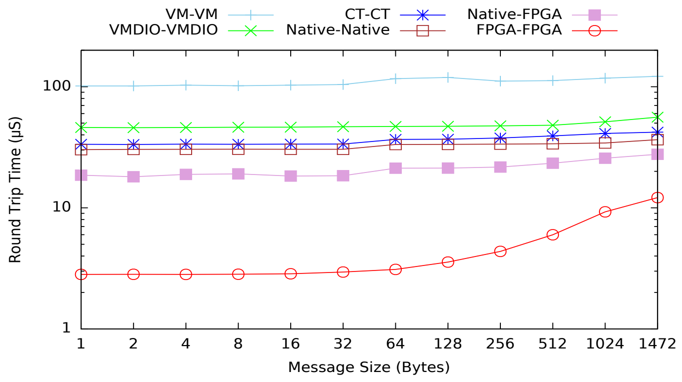

Fig. 5. Latency Comparison (UDP)

The network stack configurations of the experimental cases are shown in Figure 4. In the VM configuration, the VM network stack is connected to the host NIC through virtio and vhost drivers over a Linux bridge, whereas in the VMDIO case, the VM network stack is directly connected to the host NIC. The CT network stack is connected to the host NIC through two virtual Ethernet interfaces (VETH PAIR) over a Linux bridge.

A. Latency

To measure the latency, we used a client-server application in which the client sends a message to the server, and the server sends the same message back to the client. The FPGA sends back the message as soon as it receives it, whereas the NICs in the servers DMA copy the messages to the DRAM. For each message size (ranging from 1 B to 1472 B), the average time of one million such round trips is taken as the final RTT (Round Trip Time).

As shown in Figure 5, moving from VM-VM to FPGA-FPGA, the RTT becomes incrementally better. We see that FPGA-FPGA achieves an impressive RTT of 2.8 μs and 12.1 μs for 1 B and 1472 B messages, respectively. The FPGAs decrease the node-to-node RTT by a factor of 10x to 35x compared with the VMs and by a factor of 5x to 16x compared with VMDIO. The FPGA-FPGA node latency is 3x to 12x better than that of Native-Native and CT-CT and 2x to 7x better than native-FPGA. Also, we observe that the FPGA-FPGA RTT of 3.1 μs for 64 B messages is better than that of kernel bypass networking solutions on native servers. As a comparison, Mellanox VMA (Mellanox Messaging Accelerator) achieves an RTT of 2.86 μs for 64 B messages in a back-to-back configuration without crossing a network switch [15].

B. Throughput

To measure the throughput, we used a client-server application in which the server sends a stream of packets and the measurement is taken at the client. In a single iteration, one million packets are received from the server, and the average of ten such measurements is taken as the final result. In this experiment, we defined the maximum theoretical throughput as follows: Maximum theoretical throughput = Line Rate * (UDP Payload/(UDP Payload + UDP Header(8B) + IP Header(20B) + MAC Header(14B) + FCS(4B) + Preamble(7B) + Start of Frame(1B) + Inter-Frame Gap(12B))).

As shown in Figure 6, FPGAs can achieve the maximum theoretical throughput at all message sizes from 1 B to 1472 B. Irrespective of the experimental case, the throughput performance of SW is poor, particularly for the smaller message sizes. For 1 B messages, the bare-metal servers, CTs and VMs can achieve only 0.32, 0.28 and 0.2 million messages per second, respectively, whereas the FPGA can achieve 14.88 million messages per second, with an impressive improvement of up to 73x. Even though VMDIO's RTT is better than that of VMs, its throughput is extremely poor for small message sizes, and thus we excluded VMDIO from the comparison. The literature shows that the native servers can achieve 3.64 million messages per second for 1 B messages using kernel-bypass networking solutions [15], which is still far below what FPGAs can achieve.

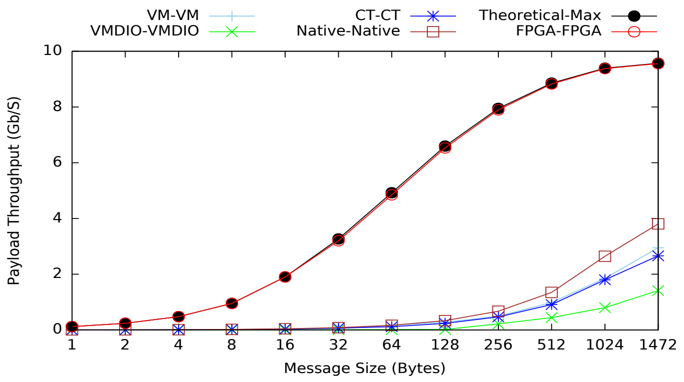

Fig. 6. Throughput Performance (UDP)

C. Response Time Variation

We evaluated the variation of the network response time by considering the RTT of one million iterations for each payload size. The standard deviation of the 99th percentile of the RTT distribution is shown in Figure 7. The standard deviation of the FPGA-FPGA response time ranges between very low values of 0.027 and 0.043, whereas the standard deviation for the VM-VM ranges between 16.731 and 22.154. For CT-CT, native-native, and native-FPGA, we observe a value ranging between 1.1 and 2.3, whereas the standard deviation of VMDIO-VMDIO gradually increases up to 7.8 with the message size.

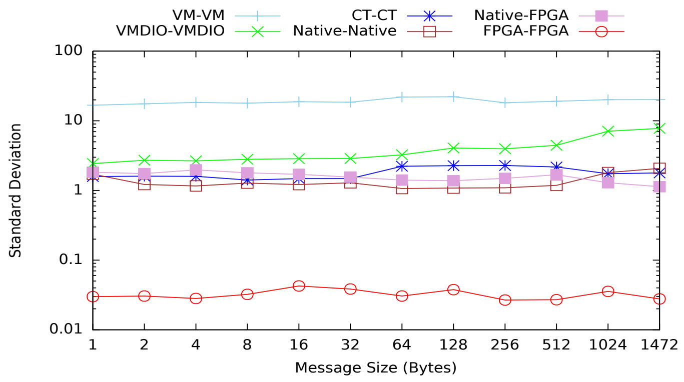

Fig. 7. Variation of Response Time (UDP)

D. HW TCP vs SW TCP and RDMA

The prototype we implemented is based on UDP, which is an unreliable protocol. To investigate reliable protocols, such as TCP, we evaluated the performance of a HW TCP stack from Xilinx [16], and compared its results with those of a SW TCP stack. Also we compared the results with RDMA (Remote Direct Memory Access), as it is considered to be the best networking approach for distributed applications. Here, we used only the best SW configuration from the above experiments, which is the Native-Native configuration. For the RDMA experiments, we used two Chelsio T420-CR iWARP (Internet Wide Area RDMA Protocol) RDMA NICs. The same experimental setup as described above is used.

As expected, we observed that the latency performance of HW TCP is far better than that of SW TCP for all message sizes. Interestingly, the HW TCP stack performs better than the RDMA NIC when the message sizes are less than 512 B (Figure 8). Similarly to the latency performance, when the throughput is considered (Figure 9), HW TCP performs better than RDMA up to message sizes of 512 B, whereas it outperforms SW TCP irrespective of the message size. As shown in Figure 10, when the variation in response time is considered, HW TCP performs better than RDMA and SW TCP for all message sizes. The results from our prototype are included in the Figures 8 - 10 to show that FPGA-FPGA UDP performs better than all other configurations we considered.

E. FPGA Resources

In the prototype shown in Figure 2, the NET CTRL (MAC/PHY) and the MEM CTRL are Xilinx IP cores, whereas the rest is the work covered in this paper. The whole design uses approximately 62K lookup tables (LUT) and 59K Flip-Flops (FF), which is equal to 14% and 7% of the overall resources available. Out of the total resources, the network-related modules (NET CTRL, NTS, AIL) consume less than

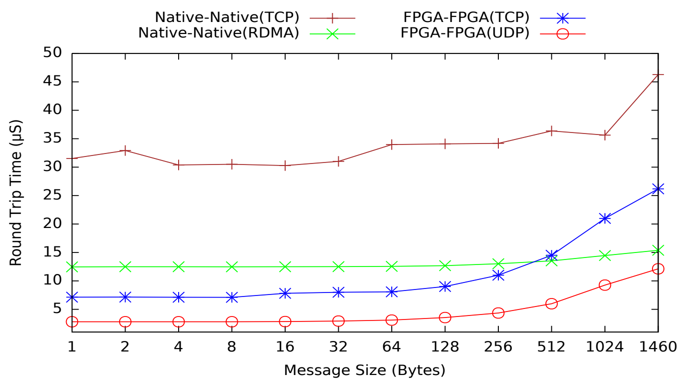

Fig. 8. Latency Comparison (UDP, TCP and RDMA)

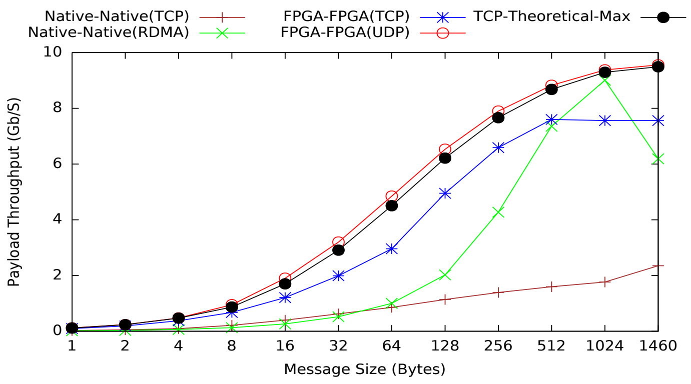

Fig. 9. Throughput Performance (UDP, TCP and RDMA)

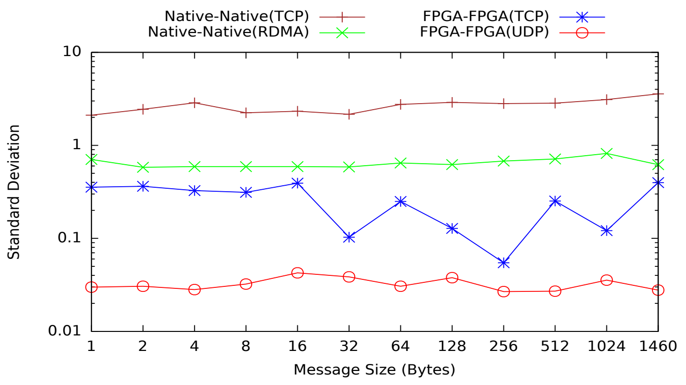

Fig. 10. Variation of Response Time (UDP, TCP and RDMA)

4% of the LUT, FF and BRAM resources. When the L4 transport protocol is compared, our UDP implementation used around 3K LUTs and 3K FFs, whereas the TCP stack we evaluated used 16K LUTs and 17K FFs, consuming 5x more resources to provide the reliability.

VI. Discussion

A. *Feasibility of Disaggregated FPGAs*

1) Performance: Overall, the experimental results show that FPGAs outperform general-purpose servers in network performance by a large margin. We found that the server network throughput is significantly degraded for smaller message sizes, whereas FPGAs achieve the line rate irrespective of the message size. Even though modern discrete NICs support advanced features, such as segmentation offload, small messages do not benefit from them. The impressive FPGA-FPGA latency and throughput open the path for the deployment of network-attached FPGAs in DCs. Also, this eliminates the need to implement RDMA and other kernel-bypass networking mechanisms [15], which are known to provide better latency and throughput than traditional networks, when deploying distributed applications in DCs.

2) FPGA Resource Consumption: We find that it takes less than 4% of the resources of a Xilinx Virtex7 (XC7VX690T) FPGA to hook it up to IPv4/UDP over an Ethernet network, and that the overall architecture consumes only 14% of the total resources. Even if management interfaces are used to connect the device to a centralized management software, we expect the overall resource utilization to be around 20%, which is a comparable amount considering the 23% resource utilization in the Catapult FPGA fabric [3].

B. *Impact on Applications*

Even if the available compute power is sufficient, inefficient networks hamper the scalability of CPU-based applications. For example, when the number of servers is increased beyond a certain threshold, the training of a distributed deep neural network becomes slow, as the network overhead starts to dominate [7]. Similarly, the scalability of Hadoop TeraSort [2] is hampered by the network throughput performance, particularly in the data-shuffling phase, in which an increased amount of DC network traffic is generated.

Moreover, on-line data-intensive applications, such as high-frequency trading (HFT) and web searches, need to obey tight latency constraints. In synchronous cluster applications, a typical cause of degraded performance is variance in processing times across different servers, leading to many servers waiting for the single slowest server to finish a given computation phase [1]. This variance in processing times is caused by unpredictable scheduling of CPU and IO resources.

According to our results, as FPGAs perform far better than SW in throughput, latency, and variation in response time, they largely resolve the network bottlenecks in CPU-based distributed applications.

C. *Network Protocol*

We also found that FPGA UDP throughput reaches the theoretical maximum throughput and its latency stays at very low values, whereas the TCP throughput and latencies are better than those of RDMA for small massage sizes. In contrast to UDP, the TCP throughput stays well below the theoretical maximum throughput for all message sizes. Our UDP implementation used only one fifth of the resources of the TCP stack we evaluated, and moreover the TCP stack required a few hundred connections to achieve the maximum throughput, whereas UDP needed only one.

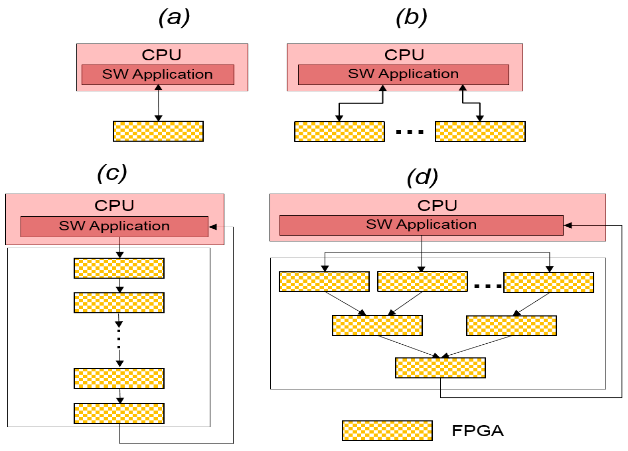

Fig. 11. Use Cases of Disaggregated FPGAs in Applications

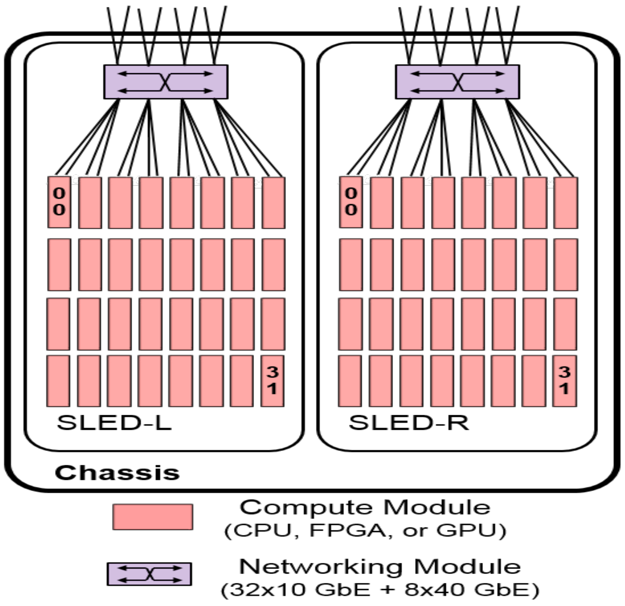
0 0
3 1
SLED-L
0 0
3 1
SLED-R
Chassis
Compute Module (CPU, FPGA, or GPU)
Networking Module (32x10 GbE + 8x40 GbE)

Fig. 12. Chassis Organization with 2 SLEDs.

The insights gained from the results call for a custom protocol that performs like UDP and has the minimum features for reliable inter-FPGA communication, using fewer resources than TCP. Preferably, this protocol must be able to saturate the line rate using a few connections, because large number of connections entails the issue of how to distribute the traffic between connections to achieve the maximum rate.

Even though we have implemented unreliable UDP protocol in the prototype, we want to implement a reliable protocol for inter-FPGA communication. Traditional Ethernet does not guarantee lossless frame reception. Instead, packets are dropped whenever a receive buffer reaches its maximum capacity. The modern CEE (Converged Enhanced Ethernet) networks are designed to prevent these frame losses by using a link-level flow-control mechanism called PFC (Priority Flow Control). As our FPGA network stack has been designed to be lossless, we can build an end-to-end lossless DC network for inter-FPGA communication by deploying FPGAs on modern CEE networks. In those infrastructures, we envision that the level of reliability that must be provided by the L4 protocols, such as TCP, can be further simplified, leading to a wide use of simpler L4 protocols, such as UDP. This reliable protocol can be implemented on top of UDP, like UDT (UDP-based Data Transfer Protocol) [17] and QUIC (Quick UDP Internet Connections) [18] does.

D. How to Use the FPGA

When running an application, the typical flow of using a vFPGA and its memory is as follows. First, a host sends a data set over the network to the vFPGA's external memory. Then, the host triggers the vFPGA over the network to start the application. This trigger message may contain the memory address of the data set copied. Next, the vFPGA accesses the data in the external memory transferred by the host, processes that data, and writes the result to the memory. Once the processing has been done, the vFPGA notifies the host that the task has been finished. Finally, the host reads the result from the vFPGA memory over the network.

E. Large Scale Applications

The provisioning of FPGAs as standalone resources and with direct connections to the DC network is a key enabler for the large-scale deployment of FPGAs in the DCs. The disaggregated FPGAs can be used to build applications that need only one (Figure 11-(a)) or several independent FPGAs (Figure 11-(b)). For applications that need to setup a large number of FPGAs in the cloud, the architecture scales to large multi-FPGA fabrics (Figure 11-(c) and (d)).

VII. Future Work

Two approaches can be envisioned when disaggregating an FPGA. The first one is governed by the historical coupling of FPGAs as slave workers to a host CPU over a PCIe bus. The disaggregation is achieved by clustering multiple such bus-attached FPGAs on a single board. The network connectivity is provided to them through a PCIe switch and/or a multi-host NIC (Network Interface Controller) [19]. The second approach promotes the FPGA to the rank of a peer processor and attaches it directly to the DC network as a standalone computing resource [13]. However, at the time of writing, DC networks are being upgraded to 40-100 GbE. Attaching every FPGA to a 100 GbE network is not justified and too expensive. Therefore, it is preferable to assemble multiple FPGAs on a carrier board, and expose that board as a single network end-point to the DC network.

We are working on an architecture at the intersection of the two approaches by attaching the FPGA to the network (2nd approach), but using an intermediate Ethernet switch instead of connecting directly to the DC network (1st approach). We use the Ethernet network interface rather than the PCIe interface for two reasons. First, the two interfaces use a similar amount of resources (according to the comparison of soft logic IP cores) and have a comparable speed and a power requirement per gigabit (i.e. when the 10GBASE-KR copper backplane version of Ethernet and a single PCIe 3.0 lane are compared). Second, the deployment of FPGAs at large scale is easier to execute and manage with a unified Ethernet network. Therefore, we leverage the first approach and assemble multiple FPGAs onto a passive carrier board, which we call a SLED. This SLED can be considered as a network point-of-delivery (POD) that interconnects a number of FPGAs over an Ethernet switch, which is also used to expose the POD to the DC network with faster up-links.

Figure 12 shows a block diagram of two such SLEDS assembled into a chassis of 19" rack. The SLED is disaggregated into 32 compute modules, each the size of a double-height dual in-line memory module (DIMM - 140 $mm \times 62$ mm). The 32 modules of such a cluster are plugged into the SLED and are interconnected over 10 GbE to the south side of an Intel FM6700 Ethernet switch, for a total of 320 Gb/s of aggregate bandwidth. The north side of the FM6700 switch connects to eight 40 GbE up-links, which expose the SLED to the DC network with another 320 Gb/s. This provides a uniform and balanced (no over-subscription) distribution between the north and south links of the Ethernet switch, which is desirable when building large and scalable fat-tree topologies (a.k.a. folded Clos topology). Two such SLEDs fit a 19" $\times$ 2U chassis, and 16 chassis fit a rack. This amounts to 1024 compute modules in a single rack.

The connector of the compute module is generic and is designed to incorporate compute modules, such as CPUs (64-bit single-socket CPU from ARM or Freescale with 32-48 GB of DRAM), GPUs, and FPGAs. It is therefore possible to assemble uniform SLEDs with 32 CPUs or 32 FPGAs, as well as any combination thereof. This mixture of DC compute resources is key for improving the performance and energy efficiency of the applications running in the cloud. Furthermore, the flexibility in allocating these resources will become vital in the near future, as 86% of all workloads will be cloud based by 2019 according to the predictions from Cisco [20].

VIII. Conclusion

Data center infrastructures are being revolutionized by shrinking and disaggregating data center resources. Riding the wave, we want to investigate how FPGAs can be deployed in such disaggregated data centers. In this paper, we proposed an architecture for disaggregated FPGAs and implemented a prototype. We compared its application interface performance with that of current DC compute resources, namely, bare metal servers (Native), virtual machines (VM), and containers (CT). The results show that FPGAs outperform them in terms of network latency, throughput, and response time variation by big margins. Moreover, we observe that it takes less than 4% of the resources of a Xilinx Virtex7 (XC7VX690T) FPGA to hook it up to IPv4/UDP over an Ethernet network, and that the overall architecture consumes only 14% of the total resources. Our next target is to implement a complete framework, including HW modules for disaggregated FPGAs, FPGA firmware, and resource management SW, to offer disaggregated FPGAs to cloud users.

References

[1] J. Dean *et al.*, "Large scale distributed deep networks," in *Neural Information Processing Systems, NIPS 2012*.

[2] Y. Guo *et al.*, "iShuffle: Improving Hadoop performance with shuffle-on-write," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, San Jose, CA, 2013, pp. 107–117.

[3] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24.

[4] M. Blott *et al.*, "Achieving 10Gbps line-rate key-value stores with FPGAs," in *the 5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013.

[5] J. Lockwood *et al.*, "A low-latency library in FPGA hardware for high-frequency trading (HFT)," in *2012 IEEE 20th Annual Symposium on High-Performance Interconnects (HOTI)*, Aug 2012, pp. 9–16.

[6] J.-A. Mondol, "Cloud security solutions using FPGA," in *2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, Aug 2011, pp. 747–752.

[7] L. Xu *et al.*, "PFC: Privacy preserving FPGA cloud - a case study of MapReduce," in *2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*, June 2014, pp. 280–287.

[8] R. Luijten and A. Doering, "The DOME embedded 64 bit microserver demonstrator," in *2013 International Conference on IC Design Technology (ICICDT)*, May 2013, pp. 203–206.

[9] S. Han *et al.*, "Network support for resource disaggregation in next-generation datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 10:1–10:7.

[10] H. Giefers *et al.*, "Analyzing the energy-efficiency of dense linear algebra kernels by power-profiling a hybrid cpu/fpga system," in *2014 IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, June 2014, pp. 92–99.

[11] S. Byma *et al.*, "FPGAs in the cloud: Booting virtualized hardware accelerators with openstack," in *Proceedings of the 2014 IEEE 22Nd International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '14, 2014, pp. 109–116.

[12] Z. István *et al.*, "Consensus in a box: Inexpensive coordination in hardware," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 425–438.

[13] J. Weerasinghe *et al.*, "Enabling FPGAs in hyperscale data centers," in *2015 IEEE International Conference on Big Data and Cloud Computing (CBDCom)*, August 2015, pp. 1078–1086.

[14] Cyclone, "Pcie2-2711 - PCIe Gen2 eight slot expansion system." [Online]. Available: http://cyclone.com/pdf/600_2711%20datasheet.pdf

[15] HP and Mellanox, "HP Mellanox low latency benchmark report 2012," July 2012. [Online]. Available: www.mellanox.com

[16] D. Sidler *et al.*, "Scalable 10Gbps TCP/IP stack architecture for reconfigurable hardware," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2015, pp. 36–43.

[17] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, May 2007.

[18] R. Hamilton *et al.*, "QUIC: A UDP-based secure and reliable transport for HTTP/2." [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-01

[19] Intel, "Intel Ethernet multi-host controller FM10000 family." [Online]. Available: http://www.intel.com/content/www/us/en/embedded/products/networking/ethernet-multi-host-controller-fm10000-family-overview.html

[20] Cisco, "Cisco global cloud index: Forecast and methodology, 2014 2019," 2015. [Online]. Available: http://www.cisco.com/c/dam/assets/sol/sp/gci/global-cloud-index-infographic.html