FlexForge: Efficient Reconfigurable Cloud Acceleration via Peripheral Resource Disaggregation

Se-Min Lim
Department of Computer Science
University of California, Irvine
Irvine, CA, USA
seminl1@ics.uci.edu

Sang-Woo Jun
Department of Computer Science
University of California, Irvine
Irvine, CA, USA
swjun@ics.uci.edu

***Abstract*—Reconfigurable hardware acceleration in the cloud using Field-Programmable Gate Arrays (FPGAs) is an increasingly popular solution for scaling performance and cost-effectiveness. For efficient utilization of FPGA resources, cloud platforms typically support elastic FPGA resource allocation. However, FPGAs are usually allocated in a homogeneous unit consisting of logic, memory, and PCIe bandwidth. Because user kernels have a wide and varying combination of resource requirements, this can result in high internal fragmentation and underutilization of each resource. To address this issue, we present FlexForge, a platform facilitating high-performance disaggregation of peripheral resources over a network of potentially untrusted FPGAs, aided by a secondary inter-FPGA network. Evaluated on a mix of representative accelerator applications deployed on a prototype cluster, FlexForge improves the overall performance of the cloud by up to 70% and 20% on average across all possible combinations without significant additional hardware resource requirements.**

***Index Terms*—FPGA, Cloud, Disaggregation**

I. Introduction

Cloud FPGA acceleration platforms such as Amazon AWS and Microsoft Azure have made the performance and power efficiency of FPGAs readily available for important applications [1]. In an FPGA cloud, multiple FPGA cards are installed into each server machine [2], [3] so that one or more FPGAs can be elastically allocated to virtual instances on demand. This approach is employed by many commercial cloud FPGAs, including Amazon EC2, Huawei Cloud, and Alibaba Cloud.

However, we noticed that the conventional approach of allocating FPGAs in the unit of physical cards can result in high resource fragmentation due to the high variability of user requirements. Conventionally, all resources on the FPGA card, including reconfigurable fabric, memory, and PCIe bandwidth, are allocated as a homogeneous unit, but different user applications are limited by different resources (e.g., reconfigurable logic for encryption and DRAM bandwidth for FFT [3]), while the rest of the resources are underutilized.

To address this issue, we present **FlexForge**, the first cloud acceleration platform that disaggregates peripheral resources such as on-board memory and host-side PCIe bandwidth from individual FPGA chips, allowing their elastic allocation independently from the reconfigurable fabric. FlexForge takes advantage of inter-FPGA serial communication, which is often available between FPGAs on the same node [4], [5] to handle disaggregated traffic beyond the PCIe bandwidth limitations, while addressing the security hazards of disaggregation over untrusted FPGAs. We note that FlexForge currently does not target multi-FPGA distribution of user logic, leaving it to already existing work mentioned in Section II. We focus on our contribution of disaggregated peripheral resources.

FlexForge implements disaggregation via a secondary mesh network constructed with inter-FPGA serial links between local FPGAs on the same physical machine instead of further burdening the host-side PCIe bandwidth, which is already a critically limited resource [3]. This is a similar approach to the popular NVLink fabric between multiple NVIDIA GPUs [6] and takes advantage of already available inter-FPGA links in commercial cloud FPGAs [4], [5]. While we also demonstrate disaggregation over remote nodes by extending the mesh network, we note that conventional Ethernet connections can also be used to avoid additional inter-node network fabric.

FlexForge also supports disaggregation over a mesh network of untrusted FPGAs, which may maliciously compromise the platform-provided shell to leak information. A compromised shell can leak other users' disaggregated DRAM and network payload contents. Leaking the mesh routing table by itself can also allow side-channel attacks [7]. FlexForge employs a source-routed protocol to hide the routing information from intermediate nodes and encrypts each routing step using a public-key encryption scheme. All memory content, as well as network payload, is also encrypted.

We evaluate FlexForge on a multi-FPGA prototype consisting of PCIe-attached Xilinx VC707 FPGAs, using a mix of representative accelerated applications spanning various computational, memory, and bandwidth requirements. The low-end VC707 cluster is a cost-effective [3] but representative (Section IV-A) evaluation environment, with the benefit of extending the lifetime of legacy community FPGA resources.

Depending on the mix of requested accelerator requirements, the FlexForge deployment improves overall performance by up to **70%** on the same hardware environment and 20% on average across all configurations while considering the shell overhead. This also allows much larger problems to fit on pooled accelerator memory. Compared to implementing disaggregation over the PCIe link, FlexForge improves performance **almost 2**×.

While some existing work has demonstrated networked FP-

979-8-3503-4859-0/DATE24/© 2024 EDAA

GAs for distributed management of resources such as storage [8] and computation [9], we emphasize that FlexForge is the first system to enable disaggregated allocation of peripheral resources across multiple tenants, especially in a potentially untrusted cloud setting.

The rest of this paper is organized as follows: We first present background and existing work in Section II. We present the architecture of the FlexForge system, including details about the design of the mesh network, multi-access memory controller, PCIe controller, and the central composer, in Section III. We evaluate the benefits of FlexForge using a mix of representative accelerators in Section IV and then conclude in Section V.

II. Background and Related Work

A. FPGA Virtualization

Virtualizing elastic allocation of FPGA resources is an important technique for improving resource utilization in the cloud environment, where demands are irregular and unpredictable. One approach is sharing a single FPGA chip across multiple accelerators by dividing it into smaller virtual FPGA instances [10], [11] or time-sharing it [12]. On the other hand, multiple FPGAs can be virtualized to provide an abstraction of a single large FPGA [13], [14]. Some projects have merged the two approaches to divide each chip into smaller units and elastically allocate them across multiple FPGAs [15]. However, these works focus mainly on virtualizing reconfigurable logic resources, not peripheral ones such as memory and PCIe bandwidth. In this work, we do not address reconfigurable logic virtualization, leaving it to such already existing work. Instead, we show that the elastic allocation of peripheral resources can have significant performance benefits.

B. Networked FPGAs

When inter-FPGA traffic is necessary due to reasons such as virtualization, a secondary inter-FPGA network can significantly improve performance without burdening the precious PCIe link, similar to the approach taken by GPU clusters using NVIDIA's NVLink inter-GPU network [6]. Many systems have demonstrated the benefits of inter-FPGA networks, including storage management [8] and machine learning [14].

However, a separate inter-FPGA network comes with the overhead of additional routing resources. A conventional switched routing approach requires additional switches. Mesh networking, where individual nodes participate in routing decisions, removes this requirement, but the routing table overhead becomes expensive as networks become large [16]. Source-routing embeds all routing decisions into the packet itself, which reduces the routing table overhead if the potential number of source-destination pairs is small [16].

III. FlexForge Architecture

Figure 1 illustrates a simple FlexForge deployment consisting of three FPGAs plugged into two host servers via PCIe. More than one FPGA accelerator card can be plugged into each host, and point-to-point inter-FPGA network links connect FPGAs within each host. Between remote FPGAs, either the mesh network can be extended, or the existing Ethernet infrastructure

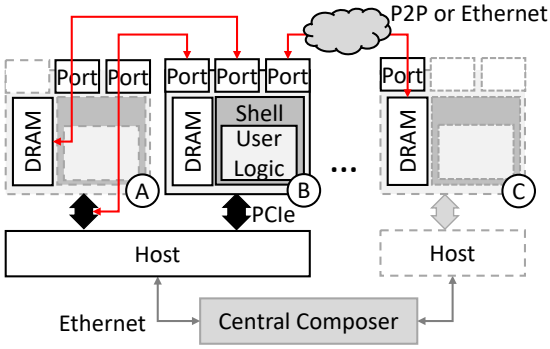

Fig. 1. A FlexForge deployment includes multiple FPGAs networked via P2P connection.

can be used. The network topology does not need to be fully N-to-N connected, and more than one link can exist between each pair of FPGAs for higher bandwidth. A FlexForge deployment also includes a central composer, which is a piece of software responsible for allocating FPGAs and additional sub-resources to users. It manages the network topology to provide source-routing tables to FPGAs, as well as manage and distribute public encryption keys.

Figure 1 shows FPGA B pooling DRAM and PCIe resources of the local FPGA A, as well as the DRAM of the remote FPGA C. In this scenario, FPGA B is the user's deployment target and is called the *"accelerator node"*. FPGAs A and C, which service peripheral resources, are called *"resource nodes"*. The collection of all FPGAs involved in this instance is simply called the *"set"*.

A. Source-Routed Network

The network architecture of FlexForge reflects the expected usage scenario of the system, which is to support pooling peripheral resources among a small set of FPGAs within a larger cloud, without significant additional hardware requirements or sacrificing privacy and security.

FlexForge uses an encrypted source-routed mesh network to minimize hardware requirements. A secondary mesh network can be constructed using the multiple multi-gigabit transceiver cores typically available on cloud FPGAs [4], [5] to form peer-to-peer connections. Source routing removes the requirement for intermediate FPGAs to store the routing table, which can be read by malicious tenants to leak valuable information about network topology. Furthermore, source-routing is a good fit for FlexForge where the number of FPGAs typically allocated for each set is expected to be small, but each node in the set can be allocated anywhere from a large pool of FPGA resources, because a source routing table can be much smaller if the number of potential destinations is small, even if the total size of the network is large.

FlexForge encrypts each field of the routing header, as well as the payload, using separate keys. This ensures that neither the payload nor the set topology is leaked to malicious intermediate nodes. Figure 2 shows an example source-routed packet with encryption, for transporting a packet through two intermediate nodes. Each header field encodes the routing information (i.e.

output port index) of each intermediate node. Each header field is only 8 bits wide, as the number of output ports per FPGA is typically small.

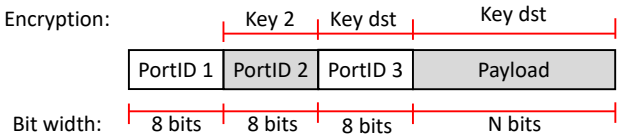

Fig. 2. An encrypted, source-routed packet structure.

Fields of the packet are encrypted using the RSA public-key cryptography scheme. The above example uses two separate encryption keys, for the latter of the two intermediate nodes (PortID 2) and the destination node (PortID 3 and Payload). The routing information for the first intermediate node (PortID 1) is not encrypted because it is used immediately by the local node. All other packet fields are encrypted using the public keys of the intermediate nodes. At every step of routing, an intermediate node decodes the routing information and removes its own portion of the header before forwarding the packet to the next step. This way, the topology of the set's sub-network is kept hidden from potentially malicious intermediate nodes, in addition to keeping the payload data private.

Instead of the source node storing the public keys of intermediate nodes, FlexForge shell stores routing information in a pre-encrypted format. When a set is instantiated, the central composer, who knows all public keys of all FPGA nodes, provides the encrypted header sequence of each path, removing the encryption overhead of the header sequence from the shell. Each shell only needs to store as many paths as the number of FPGAs in the set, which is much smaller than the routing table of a conventional switch-based approach. As a result, the entire source-routing table is stored in on-chip BRAM.

B. Multi-Access Memory

The FlexForge memory controller supports sharing the local memory capacity and bandwidth with remote FPGAs over the source-routed network. The details of this task are split into three components: Memory encryption engine, Memory map, and Access control table. Figure 3 illustrates the path of each memory request. Responses will move in the opposite direction.

To reduce the memory overhead of the memory map and access control tables, the allocation unit is very coarse, in 256 MB *blocks* in the current design. This is in contrast to the small, 4 KB page sizes used by operating system software. Such coarse allocation units allow very small mapping tables to handle 100s of GB of memory resources with small on-chip memory. Such a coarse allocation scheme did not harm flexibility in our experience with driving applications.

Memory encryption: All memory I/O, regardless of whether it is to local or remote memory resources, is encrypted by the memory controller using the shell's private key. Even local memory is encrypted, just in case a vulnerability or bug causes the memory controller to fail in proper access-control.

Memory map: The amount of accessible memory allocated to an accelerator node may be much larger than what is

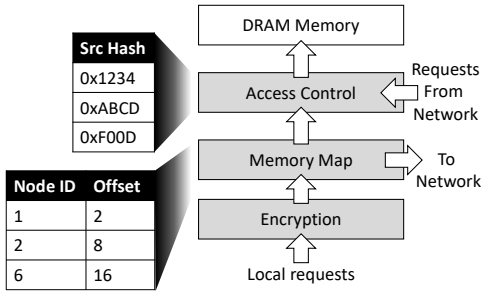

Fig. 3. The memory controller performs memory mapping across the network, as well as access control.

available on the local board. The memory map keeps track of the locations of each block across the network, from the perspective of the accelerator node FPGA. As seen in Figure 3, each row represents a block, and its contents point to the remote (or local) node ID and the memory offset of the block of interest. The node ID used here is the same as the row index of the routing table introduced in Section III-A.

Access control: Access control table makes sure that blocks are read and written only by the FPGA it was allocated to, by comparing each request with the secret *source hash* of each requesting node. One source hash value is created randomly for each FPGA node allocated by the central composer, and its value is notified to all FPGAs in the set.

C. Pooled PCIe Bandwidth

FlexForge also pools PCIe bandwidth resources between FPGA accelerators if they are plugged into the same host server, reducing bandwidth fragmentation. This is achieved by using the mesh network to spread PCIe traffic across multiple FPGAs.

Figure 4 illustrates how one pair of hardware and software (Software 1 and FPGA 1) can take advantage of underutilized bandwidth (if any) of a local FPGA accelerator (FPGA 2). The FlexForge kernel driver maintains a pool of DMA buffers, one for each host software. These kernel-space buffers are mapped to userspace for fast communication. When either initiating or receiving DMA transfer over PCIe, the control path information over memory-mapped I/O (MMIO) includes the source or destination FPGA ID, so the mesh network can route traffic to the intended destination. The hardware-side PCIe controller also includes an on-chip reorder buffer to shuffle the received PCIe traffic back into the correct order if it arrives out-of-order.

D. FlexForge Composer

The FlexForge composer is a central piece of software responsible for correctly managing and allocating FPGA, memory, and PCIe resources to user requests, as well as managing and distributing encryption keys and populating the various mapping tables in hardware including the routing table, memory map, and the memory access control table.

Peripheral resource allocation: When requesting an FPGA accelerator from the cloud, the user can also specify the amount of peripheral resources such as off-chip memory capacity, off-chip memory bandwidth, and host-side PCIe bandwidth.

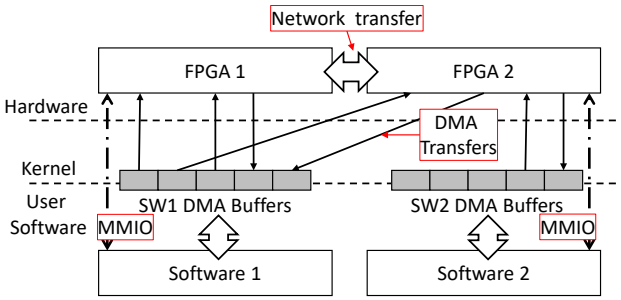

Fig. 4. Kernel driver and mesh network work together to spread PCIe traffic across multiple FPGA accelerators.

To facilitate this, the composer keeps track of the following information:

- For each PCIe, bandwidth requested by each set.
- For each P2P link, bandwidth requested by each set.
- For each DRAM, capacity requested by each set.
- For each DRAM, bandwidth requested by each set.

For each allocation request, the composer performs a simple subgraph matching algorithm to find a satisfying set of resources consisting of one FPGA and other peripheral resources. For example, the target FPGA should have enough unallocated PCIe resources on the host server, as well as enough unallocated DRAM capacity and bandwidth on memory resources, which are reachable via network links with enough unallocated bandwidth. All controllers in the shell, including memory, network, and PCIe, implement throttling to keep each user under allocated limits.

Initializing encryption and access control: Each shell is responsible for generating the asymmetric encryption keys via the standard RSA method and providing the public key to the composer. Public keys for resource nodes in a set are sent to the acceleration node for packet encryption, and routing information between the acceleration node and all of its resource nodes are encrypted at the composer and then sent to all acceleration and resource nodes to populate the source-routing table in a pre-encrypted format. In addition, the composer also generates a *source hash*, which is a randomly generated value used as a secret key for memory-resource nodes to verify that the correct acceleration node is trying to read the corresponding block.

Once the set of devices is determined, the user can provide the bit files to the composer so they can be programmed onto the accelerator node's FPGA chip.

IV. Evaluation

We have implemented and evaluated a prototype FlexForge deployment using an instance of the BlueDBM cluster platform [8], consisting of multiple Intel x86 servers hosting one or more Xilinx VC707 FPGA boards. Each node was equipped with up to two FPGAs, similar to Figure 1. Each FPGA is connected to the host over PCIe Gen2 x8. Each FPGA card is augmented with connectivity expansion cards plugged into the FPGA Mezzanine Card (FMC) to pin out eight 10 Gbps GTX serial connections to SATA ports. In our prototype, the mesh network connections are bundled into groups of four to create a fan-out of two 40 Gbps links per FPGA.

Development was done in Xilinx Vivado, and all hardware was clocked at 250 MHz.

A. Relevance of FlexForge on newer FPGAs

While we use the low-end, more affordable Xilinx VC707 FPGA boards for prototype implementation and evaluation, we argue that the insights obtained are relevant to newer cloud FPGAs because capabilities of the reconfigurable fabric have scaled faster than the performance of peripheral components, making peripheral resource fragmentation a bigger issue on newer FPGAs. Table I compares the individual component scaling between the VC707 and the vu9p used on the Amazon EC2. While most components have scaled at a similar rate, the Look-Up Table (LUT) resources have scaled the quickest ($8.53\times$) and the PCIe slowest ($4\times$). As a result, the relative performance impact of PCIe bandwidth will be more pronounced on the vu9p compared to the VC707.

TABLE I
FPGA RESOURCES HAVE SCALED AT SIMILAR RATES.

	VC707	vu9p	Rate
LUTs (K)	303	2,586	8.53x
Memory performance (GB/s)	11	72	6.54x
PCIe performance (GB/s)	4	16	4x
Network over FMC/+ (Gbps)	120	896	7.47x

Furthermore, low-end FPGAs have shown to have better cost-effectiveness in the cloud environment [3], and we are using FlexForge to pool such existing community FPGA resources into this system to create a sizable cluster for extending the effective lifetime for FPGA resources.

B. System Component Evaluations

The PCIe and memory controller (including compression) consumes 28K LUTs or 9% of the VC707 FPGA, and the network router (including compression) consumes 64K LUTs or 15% of the chip, adding up to 24% of the chip consumed by the FlexForge shell. We note that a single router currently supports up to four mesh network links. Each FPGA node can use up to eight links by instantiating two routers, which will increase chip resource utilization correspondingly.

In terms of effective performance, the shell supports 3.1 GB/s of PCIe bandwidth, 11 GB/s of DRAM bandwidth, as well as 7.5 Gbps of P2P communication per 10 Gbps link.

C. Single-Application Evaluation Results

We first evaluate individual applications and then present a much more detailed analysis of multi-application configurations in Section IV-D. Among the multiple applications we ported to the cluster system, we present three representative ones: N-Body simulation [17], K-Means clustering [18], and LZ4 compression [19]. These were selected as they are popular and important applications for acceleration while displaying varying peripheral resource requirements. Table II compares the resource requirements of the three applications.

TABLE II
VARYING RESOURCE REQUIREMENTS OF APPLICATIONS.

Application	Computation	Memory	PCIe
N-Body	**High**	Low	Low
K-Means	Med	**High**	Low
LZ4	Med	None	**High**

N-Body simulation: N-Body simulates the motions of numerous particles in a 3-dimensional space, where the motion of every particle is affected by every other particle via gravitational forces or otherwise. We organize computation into Processing Elements (PE) and on-chip BRAM caching. N-body is almost entirely processing-bound with very low pressure on both off-chip memory and host-side PCIe. We could fit 20 PEs without the FlexForge shell, and 16 with it. Timing closure failed beyond this.

K-means clustering: K-means clustering is an effective unsupervised machine learning algorithm for high-dimensional data. The accelerator is organized into multiple parallel PEs, each of which is responsible for disjoint blocks of input data. Data type is a dense vector of 1024 single-precision floats and cosine similarity distance metric. Similarly to N-body, 16 PEs fit on FlexForge and 20 PEs without the shell.

LZ4 compression: LZ4 is one of the popular lossless compression algorithms for data analytics due to its simplicity, performance, and compression efficiency. We use the publicly available IP from Xilinx [19], which delivers multi-fold performance over software at 1.6 GB/s compression per core. Four cores fit on FlexForge, and five without the shell.

Performance evaluation: Figure 5 shows the performance evaluations of the three applications on the baseline VC707 without FlexForge (N/A) and the performance of FlexForge as we pool peripheral resources of 1, 2, and 3 FPGAs in the cluster. We emphasize that the accelerator kernel itself is deployed on a single FPGA chip, the *accelerator node*. We are only pooling peripheral resources, i.e., PCIe and memory bandwidth and capacity.

First, N-Body performance actually degrades slightly with FlexForge because the problem is entirely computation-bound, meaning the shell overhead added while peripheral resource pooling adds no benefits. However, even for N-Body FlexForge provides the benefit of pooling larger memory capacities. K-means, on the other hand, is entirely bound by memory

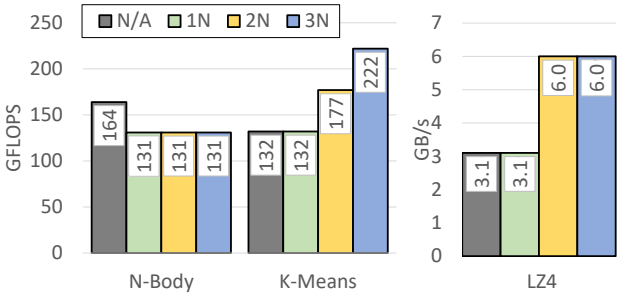

Fig. 5. Performance evaluation of representative applications on FlexForge.

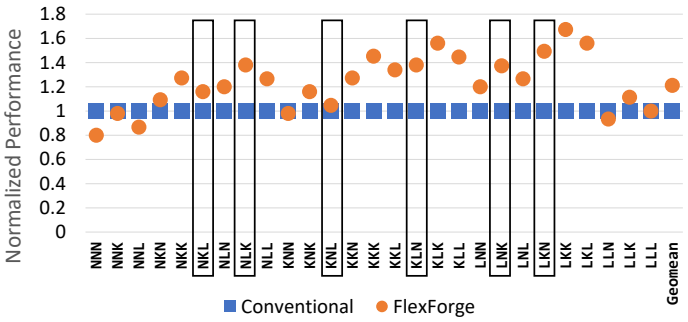

Fig. 6. Multi-Application Disaggregation Benefits.

bandwidth. As a result, we see linear performance improvement as FlexForge pools more bandwidth. The reason pooling two nodes is not exactly twice as fast as one node is because a mesh network link (40 Gbps) is slower than the memory bandwidth (11 GB/s). In the case of LZ4, four LZ4 cores already saturate the PCIe link. As a result, pooling two PCIe links via FlexForge shows significant benefits. However, the third FPGA in our setup is remote, meaning its PCIe bandwidth cannot be pooled.

D. Multi-Application Disaggregation Benefits

Evaluation configurations: We emulate a realistic cloud scenario using a system similar to Figure 1. The local node has two FPGAs, and the remote FPGA is allocated two network hops away. Unlike Figure 1, FPGAs A and B are connected with a single 40 Gbps link. Each FPGA is an identical Xilinx VC707 board.

Figure 6 evaluates FlexForge performance of *all possible combinations* of the three accelerators deployed on the target system, compared against a conventional system without peripheral resource pooling. Each configuration is labeled with a three-letter keyword, where "N" is N-body, "K" is K-means, and "L" is LZ4. For example, "LKN" deploys LZ4 to FPGA A, K-means to B, and N-body to C. LZ4 pools the PCIe bandwidth of FPGAs A and B, and K-means pools the DRAM bandwidth of A and C. N-body does not pool any peripheral resources because it does not benefit from them. Since the three applications have heterogeneous performance metrics (e.g., GFLOPS, GB/s), we present the average of normalized performance numbers. For example, if two applications were deployed in parallel, where one achieved 2x and the other no improvement, the presented normalized performance improvement would be 1.5x. We believe this is a simple yet effective metric for performance comparisons, especially since the accelerators evaluated all consumed similar, non-trivial amounts of FPGA resources.

Results analysis: Figure 6 shows that peripheral resource pooling contributes significant performance benefits, where the geomean of improvements over all combinations is 20%, and the best configurations achieved almost 70% performance improvement ("LKK"). It also shows that performance is sensitive to resource node placement. For example, Figure 6 includes six different mappings for deploying one K-means, one N-body, and one LZ4 (i.e., "NKL", "NLK", "KNL", "KLN", "LNK", and "LKN", highlighted with black boxes), with per-

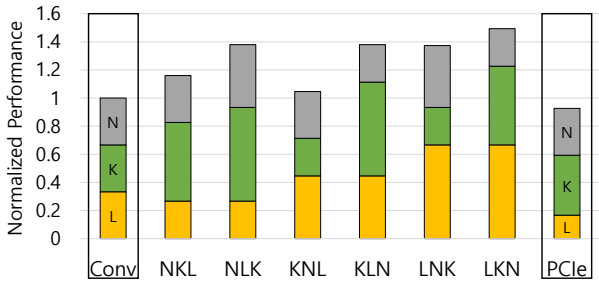

Fig. 7. Disaggregation over PCIe can actually reduce performance due to PCIe bandwidth limitations.

formance improvements varying between 50% ("LKN") and 5% ("KNL"). The worst case configuration is "NNN" (0.8x performance), where all accelerators are computation-intensive and cannot benefit from resource pooling. While we present even these negative results, in a real deployment the composer will simply not include such computation-bound nodes into the pool to avoid the shell overhead.

"LKN" has a very favorable mapping because the LZ4 accelerator could benefit from the combined PCIe bandwidth of FPGA A and B, while K-means could benefit from two 40 Gbps mesh links going to FPGA A and C. On the other hand, "KNL" has a less favorable mapping because K-means on FPGA A only has one mesh link to benefit from, while LZ4 on FPGA C was by itself in its local host and could not pool any additional PCIe resources. To further emphasize this effect, the second-worst performing configuration, "NNL" (0.87x), actually could improve performance to 1.2x with a more favorable mapping ("NLN" and "LNN"). In effect, the composer does a fairly good job of finding favorable mappings by allocating FPGAs with more remaining resources.

Comparison against disaggregation over PCIe: To emphasize the benefit of the inter-FPGA mesh network, we evaluate it against the alternative: disaggregation over host PCIe. For PCIe disaggregation, we assume the best-case setting where all FPGAs are plugged into the same host and assign up to 50% of the PCIe bandwidth for remote disaggregation traffic. We evaluate the performance impact of this approach for the fully heterogeneous scenarios highlighted in Figure 6 and present them in Figure 7. This figure compares the performance between the conventional approach of non-disaggregated, separate FPGAs (*Conv*), disaggregation over PCIe (*PCIe*), and the six FlexForge configurations, with the three applications.

Overall, FlexForge (`LKN`) achieves over 60% higher performance compared to disaggregation over PCIe, which actually achieved *lower* performance compared to the baseline. LZ4 suffered significantly because half of its PCIe bandwidth was consumed by K-means to access pooled memory, while K-means performance did not significantly improve due to slow PCIe bandwidth. Furthermore, the performance of N-body was also slightly negatively impacted due to the shell overhead.

V. Conclusion

We presented our work on FlexForge, an elastic cloud infrastructure for FPGA accelerators, that disaggregates peripheral resources for elastic allocation using a fast and secure network. Using FlexForge, we demonstrated that the benefits of reducing resource fragmentation outweigh the chip area overhead of disaggregation. We are making FlexForge publicly available via an open repository of hardware specifications and code, and we hope it will help improve the utility of cloud FPGA resources.

References

[1] Y.-K. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "In-depth analysis on microarchitectures of modern heterogeneous cpu-fpga platforms," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–20, 2019.

[2] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, *et al.*, "The future of fpga acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 3, pp. 1–42, 2022.

[3] X. Wang, Y. Niu, F. Liu, and Z. Xu, "When fpga meets cloud: A first look at performance," *IEEE Transactions on Cloud Computing*, 2020.

[4] C. Davies, "Fpga as a service in the cloud." https://indico.cern.ch/event/669648/contributions/2838181/attachments/1581893/2500031/Huawei_Cloud_FPGA_as_a_Service_CERN_openlab.pdf.

[5] A. Cloud, "Alibaba cloud fpga as a service product introduction." https://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/download%2Fpdf%2F163522%2FProduct_Introduction_intl_en-US.pdf.

[6] D. Foley and J. Danskin, "Ultra-performance pascal gpu and nvlink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[7] B. Kumar, K. Abhishek, A. Kumar, and M. Singh, "System and method for mitigating cross vm attacks in cloud computing by securing the network traffic," in *2015 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pp. 221–225, IEEE, 2015.

[8] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, *et al.*, "Bluedbm: An appliance for big data analytics," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–13, IEEE, 2015.

[9] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, *et al.*, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*, IEEE, 2016.

[10] J. Zhang, Y. Xiong, N. Xu, R. Shu, B. Li, P. Cheng, G. Chen, and T. Moscibroda, "The feniks fpga operating system for cloud computing," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, 2017.

[11] A. Vaishnav, K. D. Pham, D. Koch, and J. Garside, "Resource elastic virtualization for fpgas using opencl," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 111–1117, IEEE, 2018.

[12] M. Asiatici, N. George, K. Vipin, S. A. Fahmy, and P. Ienne, "Virtualized execution runtime for fpga accelerators in the cloud," *Ieee Access*, vol. 5, pp. 1900–1910, 2017.

[13] A. Iordache, G. Pierre, P. Sanders, J. G. de F. Coutinho, and M. Stillwell, "High performance in the cloud with fpga groups," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, 2016.

[14] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *iEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[15] Y. Zha and J. Li, "Hetero-vital: a virtualization stack for heterogeneous fpga clusters," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 470–483, IEEE, 2021.

[16] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, pp. 43–44, 2012.

[17] S. J. Aarseth and S. J. Aarseth, *Gravitational N-body simulations: tools and algorithms*. Cambridge University Press, 2003.

[18] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[19] X. Inc, "Xilinx lz4 compression." https://xilinx.github.io/Vitis_Libraries/data_compression/2022.1/source/L2/lz4.html.