Characterizing Latency Overheads in the Deployment of FPGA Accelerators

Ryan A. Cooke and Suhaib A. Fahmy
School of Engineering
University of Warwick, Coventry, UK
{ryan.cooke, s.fahmy}@warwick.ac.uk

***Abstract*—FPGA hardware accelerators have recently enjoyed significant attention as platforms for further accelerating computation in the datacenter but they potentially add additional layers of hardware and software interfacing that can further increase communication latency. In this paper, we characterize these overheads for streaming applications where latency can be an important consideration. We examine the latency and throughput characteristics of traditional server-based PCIe connected accelerators, and the more recent approach of network attached FPGA accelerators. We additionally quantify the additional overhead introduced by virtualising accelerators on FPGAs.**

I. Introduction

FPGAs have seen increased deployment within the datacenter to accelerate compute-intensive workloads. The growing complexity and scale of applications and the associated data, alongside the stalled performance scaling of CPU architectures, has resulted in heterogeneity being explored as a way of addressing performance, throughput, and power consumption challenges[1]. FPGA acceleration has been demonstrated to provide considerable benefits in computation latency for a variety of applications [2], [3], [4], [5], and improved performance per Watt compared to GPUs [6]. Virtualisation of FPGA resources is also emerging [7], [8], [9], allowing a single FPGA to be shared by multiple users and applications. These designs comprise a static shell that manages communication and control for multiple partitioned reconfigurable regions. Accelerators are swapped into these regions as needed, hence requiring accelerator allocation middleware to manage data stream sharing at runtime.

Latency reduction is a key challenge within the datacenter, in particular for streaming applications, where data is processed itemwise, and is often time-sensitive. Low latency can be a critical requirement in some applications such as high frequency trading, or customer facing web applications. The performance of distributed applications such as Memcached has been shown to be degraded by fundamental datacenter latency factors [10]. While individual packet latencies are small, they can accumulate in applications where data is spread across multiple packets that travel through multiple switches or servers. Latency variability is also an important consideration. Processing is often distributed across many machines in parallel, and overall completion time is dependent on the slowest response [11]. Some applications rely on large fan-out requests for data across distributed sources, and though average latencies may be low, the effect of small variations can be amplified to cause significant degradation in performance.

While hardware accelerators improve computational latency, they can also introduce additional communication latency. Accelerators are typically attached to a host server through PCI Express (PCIe), which can achieve high throughput communication. Streaming data from the network must then traverse the host NIC and software network stack, subsystems that have been identified to contribute significantly to both average and tail latencies [12]. Inter-accelerator interconnect is also gaining importance for larger applications[13]. More recently, driven by the demand for low latency, there has been a growing interest in network-attached accelerators, where an FPGA is connected directly to the network [14], [15], [16], processing data in-line. While these deployments do reduce communication latency, their impact on overall system latency has not been studied in detail. Finally, the virtualisation required to allow FPGA resources to be shared can contribute further overhead.

Past work has characterised the components that contribute to datacenter latency including the different software components of host machines, the network interconnect, the NIC, PCIe, and switch latencies [12]. An in-depth study of PCIe communication latency and bandwidth as relevant to NICs was presented in [17]. The latency implications of software virtualisation were investigated in [18].

There has thus far been no such study into the communication overheads for FPGA accelerators in the datacenter. In this paper, we present experiments that characterise these important overheads, in particular in the context of streaming applications, where latency is a key performance metric. We characterise the latency characteristics for PCIe-hosted and network-attached FPGA accelerators and isolate the additional delays introduced through virtualisation. We compare these to latency measurements for a typical host server, allowing us to isolate the latency contributions of host networking and management and data transfer to the PCIe accelerators.

Some work has compared the performance of PCIe and network-attached FPGA accelerators for specific applications[15] and minimising streaming data latency for embedded accelerators [19]. We characterise the fundamental delays more generally, to gain insights into the costs of accelerator deployments and inform design decisions for a wide range of scenarios.

II. Background and Related Work

FPGA accelerators connected to a host via PCIe have been commonly deployed in the datacenter for various applications, such as machine learning [20], [21] and database processing [22]. PCIe offers a high throughput interface, existing supporting infrastructure, and a way for the host to control and configure the accelerator. The focus of such integration is high throughput, moving larger batches of data to minimise the impact PCIe transfer overheads. Distributed workloads, however, often comprise streams of data arriving over a network, which must be received over the host's network interface, written to memory via DMA transfers, with file descriptors pointing to packets stored in the driver ring buffer. Packets are then processed by the kernel's network stack before being added to the socket receive queue, which can then be accessed by an application running in user space. In order to transfer data to an accelerator, the user space application typically uses API calls to write data to a memory buffer and issues a command to the FPGA to initiate the transfer. The FPGA then reads the data to be transferred from this buffer. These processes all add to overall data latency when receiving data from the network for processing in an FPGA accelerator.

Factors such as I/O, network, and CPU stress have been demonstrated to have significant impact on the magnitude and variability of packet delays [18], [23], [24]. The various contributors of packet latencies in datacenter environments were examined in detail in [12]. The PCIe interface used by most NICs can be a significant contributor to latency f[17]. It was demonstrated in [18] that virtualisation using Linux Vserver typically added a small delay to packet round trip times, while Xen virtualisation added 3 to 4 times greater latency.

A solution to the latency problem for high data rate streaming applications is for the FPGA accelerator to interface directly with the network, bypassing a host networking stack. This is possible due to the high I/O performance flexibility afforded in modern FPGA architectures and is a model that cannot be considered for accelerators like GPUs that rely on a host CPU for management. This approach has seen use in a variety of scenarios [25], [26], [27], and has been demonstrated to lead to reductions in latency for specific applications compared to PCIe or purely software solutions. There has not been a characterisation of the detailed latency components introduced by this approach however, and there has not been an investigation into the effects of virtualising these devices. This model of accelerator integration poses additional challenges. Without a CPU based host, control of the accelerator and virtualisation logic is more complex. Additionally, in some cases it can be more difficult to operate on large datasets, due to limited available storage.

III. Experiments

In this section, we detail our experimental testbed and the experiments carried out to characterise the communication latencies and throughputs of PCIe and network-attached FPGA accelerators. **Latency:** We use an external device to measure round trip times in order to achieve accurate measurements and ensure fairness between different scenarios. This is a Xilinx KC705 evaluation board programmed with specialised hardware to transmit, receive, and timestamp packets. Packets are sent over 10 Gb/s Ethernet using SFP+ transceivers and an optical cable.

The measurement device sends packets to the platform under test, recording a timestamp as it leaves. The platform under test receives each packet up to the point where the application accelerator would process the data, then returns it to the measurement device, where a second timestamp is recorded on reception. The round trip time is the difference between these two timestamp values. Time values are captured using a free running 64-bit counter implemented in the FPGA measurement device fabric, driven by the 156.25MHz physical board clock, giving a measurement precision of 6.4 ns. Measurements were taken over 20,000 back-to-back transmissions using a closed loop model where the latency of one packet is recorded before the next experiment starts. We subtracted the time taken for the packet to travel out of the measurement FPGA and to the platform under test over an optical fibre from all results (96ns). A 1 m optical cable was used to connect the measurement device to the platform under test in each experiment.

Throughput: To measure throughput, we used the FPGA measurement device to generate traffic over the 10 Gb/s Ethernet link to the platform under test at as high a packet rate as possible, with the minimal 12 frame inter packet gap and fully saturating the AXI4-Stream interface to the Ethernet core IP. All packets received at the platform under test are looped back to the traffic generation board.

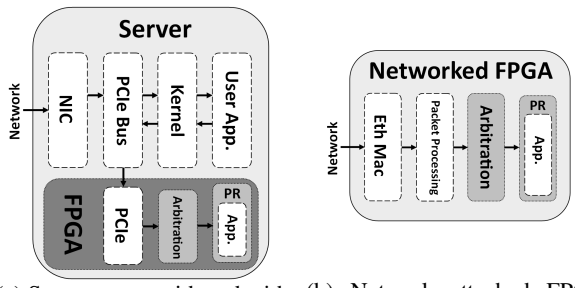
The measurement device generates traffic for a 5 second interval, and counts the number of packets it sends and receives back from the platform under test in this interval. Each packet sent in the interval is of the same size, and the tests were repeated for multiple packet sizes. Using the packets received per second, an average throughput is calculated.

Host Server: We represent the server platforms typically found in a datacenter using a Linux server running CentOS 6.7 on a 12 core 2.20 GHz Intel Xeon E5-2650 v4 CPU with 64GB of RAM. The network card is a 10 Gb/s Mellanox MT26448, using SFP+ transceivers. The latency characteristics for CPU based server platforms have been studied in detail in other work [12]—the measurements detailed in this paper are used to provide a baseline for fair comparison only.

Data is sent over the network to this machine to be processed by a C++ application running in user space. The application is pinned to a processor core to improve performance.

For the throughput experiments, to maximise the number of packets receivable per second, we configured the NIC ring buffer to be as large as possible for our device.

PCIe Attached FPGA: We used a Xilinx VC709 FPGA evaluation board hosting a Virtex 7 XC7V690T FPGA as the accelerator platform. We carried out tests for both virtualised and non-virtualised FPGA accelerators. For both accelerator configurations we used PCIe Gen3$\times$8. For non-virtualised



(a) Server setup, with and without either a virtualised or non-virtualised FPGA accelerator

(b) Network attached FPGA accelerator, with and without virtualisation.

Fig. 1: Configurations used to measure server based and network attached accelerators. Virtualised accelerators have arbitration logic and utilise PR regions for application logic.

accelerators we based the deign on the open source RIFFA framework [28]. It compiles the PCIe communication interface with a fixed accelerator and provides a simple API that abstracts low level transfer mechanics. This static accelerator cannot be replaced without significant interruption to the whole system. The architecture is typical of a fixed function accelerator found in the datacenter, comprising a generic shell of communication logic that simplifies the deployment of applications.

For this scenario, the C++ application running on the host receives the packets from the tester FPGA, writes them to the FPGA accelerator using the RIFFA API, reads them back, then sends them to the measurement FPGA through the network interface. The write to the FPGA is non-blocking. During the throughput experiments packets were transferred to the accelerator packet-by-packet, and not batched.

To test a virtualised FPGA, we used a version of the DyRACT [29] framework allowing for the application logic to be modified at runtime without having to reconfigure the entire FPGA. This allows for multiple accelerators to run independently on the same device, with dynamic swapping of accelerators using partial reconfiguration triggered through the same PCIe interface used for data. This framework thus includes extra logic that contributes to additional communication delays as well as additional software components within the driver. We use the same C++ application as with the non-virtualised FPGA but using the DyRACT API.

Network Attached FPGA: Finally, we consider the proposed direct attachment of accelerators to the network, as might be employed in standalone compute units, smart switches/routers, or smart NICs. We again use the Xilinx VC709 evaluation board to test this. The board sends and receives 10Gb/s Ethernet through SFP+ transceiver modules attached to the FPGA fabric instead via PCIe. This data travels through the physical interface and Xilinx 10G Ethernet subsystem IP, which includes the PHY, PCS/PMA and MAC layers, interfacing inside the FPGA over an AXI4-Stream interface. A pipeline of 3 state machines is used to remove the Ethernet, IP, and UDP headers, with the UDP payload passed to the accelerator. In our experiments we loop this payload back out of the application logic, through the network stack and back out of the device, using the same method in reverse.

We also modified this design to make it more representative of a virtualised platform, by adding an additional stage that checks the destination address and directs the data to the correct accelerator among multiple. An additional FIFO buffer is also placed between this stage and the application logic. An arbiter FIFO is implemented before the transmit side of the network stack. This logic uses round robin arbitration to allow the multiple accelerator slots to share external bandwidth.

TABLE I: Latency results for 20000 back-to-back UDP packets in microseconds.

Scenario	Median	90th perc.	99th perc.	99.9th perc.
Server	6.961	11.300	13.170	21.770
Server+FPGA	13.100	14.910	22.910	29.130
Server+VFPGA	23.290	33.590	41.960	71.350
Net FPGA	0.667	0.673	0.673	0.673
Net VFPGA	0.726	0.737	0.737	0.737

IV. Results

Experiments were conducted using 20,000 back-to-back 80B UDP packet transfers, generating the results shown in Table I. All results are in microseconds. A small packet size, close to the minimum frame size is chosen to give a better indication of the minimum unavoidable latencies in our scenario. Packets were sent using a closed model, where the latency measurement of one packet was taken before the next packet was generated. This was done as uncontrolled transmission from the measurement FPGA to the server based platforms would result in queuing, buffer overflows, and dropped packets.

Median Latency: It is clear that adding an FPGA accelerator to a server approximately doubles the packet return latency. This is due to the additional movement of the data over PCIe to reach the accelerator and back. Enabling virtualisation on the PCIe attached FPGA adds a median delay of around 10μs. Part of this is the additional logic added to the FPGA design, but the change in software running on the host server also significantly contributes to this extra delay. Utilising more of the available virtualised slots on the FPGA could also potentially increase latency further if there is significant saturation of the PCIe interface due to contention. Hence an FPGA accelerator must accelerate an application by a sufficient amount to overcome these additional latencies.

The network attached FPGA has latency an order of magnitude smaller, as packets bypass the PCIe interface of the network card, the server network stack, and the PCIe link to the FPGA. While these delays therefore seem insignificant, network attached accelerators are often deployed for low latency, or high data rate applications, where even the sub-microsecond delay introduced could be relevant.

TABLE II: Breakdown of delays measured from round trip times of a packet travelling through a networked FPGA accelerator design, for 80B UDP packets.

Delay Source	Median Delay(μs)
Packet processing logic	0.135
PHY+MAC	0.532
User logic (simple loop-back)	0.013
Virtualisation logic (for V.FPGA)	0.059

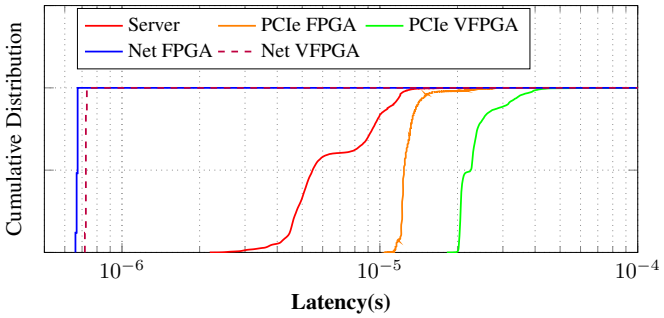


Fig. 2: Cumulative distribution function (CDF) of latencies for each scenario.

FPGA Latency Breakdown: We use hardware counters in the FPGA to isolate sources of the delays in the device; results are shown in Table I.

The packet processing layer, which strips the layer 2/3/4 headers and passes data to the accelerator and user logic in our experiments contributes to the total delay less than the PHY and MAC cores. In more complex designs that may implement more complete layer 3 and 4 functionality, packet processing delay will be higher but a comparable order of magnitude. Additionally, virtualisation logic adds further delay when there are multiple active accelerators present, as network access to different accelerators must be arbitrated. In our design, the accelerator logic loops data back using a FIFO, so adds minimal latency of a few clock cycles. For a real processing task, this latency would depend on accelerator datapath.

Latency Distributions: The CDFs in Figure 2 show the latency distributions for each scenario, demonstrating the relative magnitudes and variations for the alternative platforms (note the logarithmic x-axis).

Tail Latencies: We further examine the tails of these latency distributions. While the median latency is a useful measure, large spikes, even if infrequent, can significantly impact latency-sensitive applications and undermine a system that functions well most of the time. Latencies at the distribution tail have also been acknowledged to have a negative impact on applications where processing is distributed across many machines in parallel, and overall completion time is dependent on the slowest response [11]. To examine these latencies, we measured the 90th, 99th and 99.9th percentile recorded latencies in each scenario, with results shown in Table II.

The server running software has around a 6μs difference between the median and 99th percentile, almost double the delay, while the 99.9th percentile latency is around 3$\times$ the median, meaning that 1 in every 1000 packets may have a latency this high.

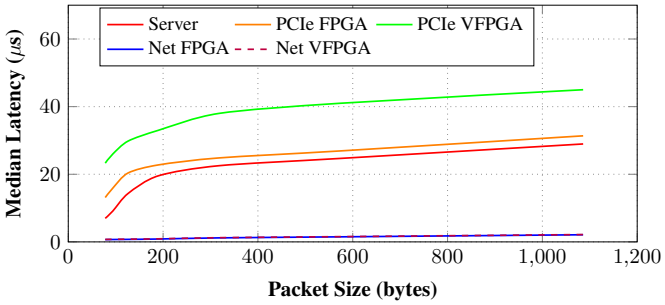


Fig. 3: Median latency for differing packet sizes.

The PCIe FPGA accelerators show less of a latency spread relative to the median. There is around a 2μs difference (14%) between the median and 90th percentile for the non-virtualised FPGA, and a 10μs difference (75%) between the median and 99th percentile, suggesting that this variability is primarily due to the host. There is minimal latency variation for the network attached FPGAs, due to all packet processing being done in dedicated hardware. As soon as software network stacks are introduced, latencies become significantly less deterministic.

Packet Size: We repeated the experiments with different packet sizes, with results shown in Figure 3. Each run of the experiment used 20,000 UDP packets of the same size. All platforms show an increase in median latency as packet size increases. The Server and PCIe platforms show greater sensitivity to packet size, with the virtualisation causing further increases. The PCIe accelerators add a relatively static overhead on top of the host latency, regardless of packet size. Both network attached FPGAs show smaller increases in latency. The initial increase in latency for packet sizes up to around 200 bytes is significant, but reduces as packets grow further.

Throughput: Results are shown in Figure 4, for varying packet sizes. These are values measured with no actual processing and therefore represent the upper limits enforced by the communication infrastructure. These measurements include the reception of the packets at the network interface, and transfer to the accelerator.

Predictably, the network-attached FPGA platforms approach line rate for 10Gb Ethernet, as there is no software involvement or additional interfacing. Adding the extra virtualisation logic has minimal effect on the throughput, as the virtualisation is all hardware based. What is likely to cause reductions in effective throughput for a particular accelerator is when there are multiple accelerators deployed across the other virtualised slots, and this bandwidth must be shared.

The server based platforms suffer considerable throughput penalties in comparison. The host running software was limited to a bandwidth of 153MB/s. While the NIC hardware was capable of receiving data at line rate, the application was not able to fetch the packets from the buffer fast enough, causing

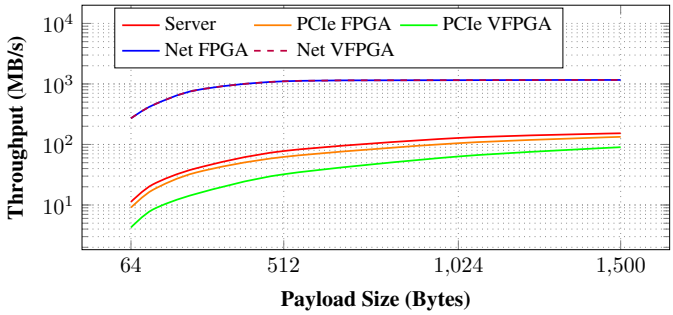


Fig. 4: Average measured throughput in MB/s for varying frame sizes.

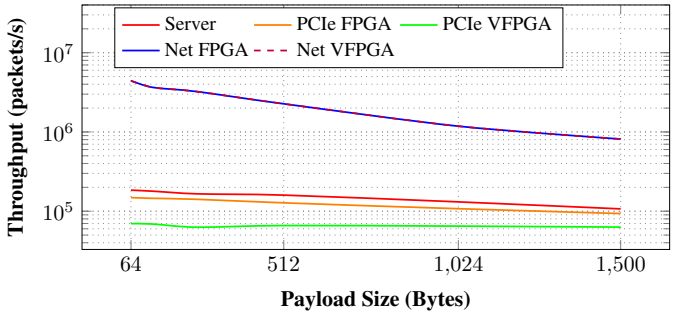


Fig. 5: Average measured packets per second for varying frame sizes.

overflows and thus lost packets. This was despite pinning the application to a CPU core, and maximising the size of the ring buffer allocated to the NIC. Adding the non-virtualised PCIe accelerator into the path resulted in the throughput being reduced to 133MB/s on average, a 13% reduction. The virtualised FPGA accelerator caused an even greater reduction in the total throughput, to around 90MB/s, a 41% decrease. The extra software driver components associated with the virtualisation are likely to be the main cause of this.

The achievable packets per second that can be received and then transmitted back out for each platform are shown in Figure 5. This metric can be important for some streaming applications that rely on packet by packet processing. The network attached FPGA platforms again show the ability to process packets at line rate. The decrease in packet processing rate as packet size increases is due to the maximum number of packets that can be transmitted on a 10Gb Ethernet link decreasing as packet size increases.

The server and non-virtualised PCIe accelerator show a slight decrease in packets processing rate as packet size increases, likely due to the ring buffer allocated to the NIC and the socket buffer using pointers instead of the packet data itself. This means the buffers can hold the same number of packets regardless of the packet size. The slight decrease in the packets per second can then be attributed to the handling of packet data in the user space application.

V. Discussion

PCIe Accelerators: Utilising PCIe FPGA accelerators results in a significant additional latency for distributed streaming applications. Virtualising the PCIe accelerator increases latency, mostly due to the extra software that manages virtualisation. Driver optimisations and improved software control of the virtualised accelerator could significantly improve latency.

While offload to PCIe accelerators has traditionally focused on maximising throughput, especially for large machine learning applications, distributed data across a network introduces a significant throughput bottleneck. Batched can mininmise PCIe overhead and enhance throughput, but this introduces extra latency, undesirable for streaming applications. Even with large batches, total throughput is limited by the host and its networking stack.

Network Attached Accelerators: The communication latency associated with the network attached accelerators is predictably much lower than PCIe hosted accelerators. Virtualisation logic added minimal additional latency - however when there is competition for the shared communication resources from accelerators in different virtualised slots, this is likely to change. The biggest contributor to latency is the physical and MAC layers, implemented in the Xilinx Ethernet core.

With the datacenter moving towards 40Gb/s and 100Gb/s, the need for direct offload from the network interface to the accelerator is increasing, via smart NICs or other networking elements that enable FPGA offload. Network attached FPGA accelerators however pose additional challenges, mainly focused around virtualisation, and managing resources across multiple applications with minimal software involvement.

VI. Conclusion

In this paper we have presented detailed experiments to measure the communication latency characteristics of FPGA accelerators in a datacenter context. We showed the latency overheads inherent in traditional deployments of accelerators hosted on a server through PCIe, and the emerging approach of network attached accelerators. This includes how latency is affected by packet size, and the latency distributions and tail latencies. We additionally detailed the throughput limitations of these deployments in the context of distributed applications where data is received over the network.

Our measurements can be used to aid in design decisions in the deployment of FPGA accelerators. For PCIe deployments, latency and throughput measurements can be used to calculate optimal batch and buffer sizes for different applications. For network attached accelerators, the measured latencies were comparatively small, but these devices are often used for applications that require ultra-low latency, where even sub-microsecond communication delays measured are relevant. These small delays will be even more relevant with emerging 40Gb/s and 100Gb/s networking.

Acknolwedgements

This work was supported in part by The Alan Turing Institute under the UK EPSRC grant EP/N510129/1.

REFERENCES

[1] R. A. Cooke and S. A. Fahmy, "A model for distributed in-network and near-edge computing with heterogeneous hardware," *Future Generation Computer Systems*, vol. 105, pp. 395–409, 2020.

[2] M. Papadonikolakis and C. Bouganis, "A novel FPGA-based SVM classifier," in *International Conference on Field-Programmable Technology*, 2010.

[3] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, "Highly parameterized k-means clustering on FPGAs: Comparative results with GPPs and GPUs," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, 2011, pp. 475–480.

[4] Y. R. Qu, H. H. Zhang, S. Zhou, and V. K. Prasanna, "Optimizing many-field packet classification on FPGA, multi-core general purpose processor, and GPU," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2015.

[5] A. Fiessler, S. Hager, B. Scheuermann, and A. W. Moore, "HyPaFilter: a versatile hybrid FPGA packet filter," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2016.

[6] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?" in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2012, pp. 232–239.

[7] 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.

[8] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualisation," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[9] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA accelerators for efficient cloud computing," in *Proceedings of the International Conference on Cloud Computing Technology and Science CloudCom)*, 2015, pp. 430–435.

[10] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: predictable low latency for data center applications," in *Proceedings of the ACM Symposium on Cloud computing*, 2012.

[11] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, 2013.

[12] N. Zilberman, M. Grosvenor, N. Manihatty-bojan, D. A. Popescu, G. Antichi, S. Galea, A. Moore, R. Watson, and M. Wojcik, "Where has my time gone?" in *Proceedings of the International Conference on Passive and Active Network Measurement*, 2017.

[13] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 94–110, 2020.

[14] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computing is a dumb idea whose time has come," in *Proceedings of HotNets*, 2017.

[15] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner, "Network-attached FPGAs for data center applications," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2016.

[16] R. A. Cooke and S. A. Fahmy, "Quantifying the latency benefits of near-edge and in-network FPGA acceleration," in *International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*, 2020, pp. 7–12.

[17] R. Neugebauer, G. Antichi, J. F. Zazo, S. López-buedo, and A. W. Moore, "Understanding PCIe performance for end host networking," in *Proceedings of SIGCOMM*, 2018, pp. 327–341.

[18] J. Whiteaker, F. Schneider, and R. Teixeira, "Explaining packet delays under virtualization," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 39–44, 2011.

[19] S. Shreejith, R. A. Cooke, and S. A. Fahmy, "A smart network interface approach for distributed applications on Xilinx Zynq SoCs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 186–1864.

[20] J. Yan, N. Y. Xu, X. F. Cai, R. Gao, Y. Wang, R. Luo, and F. H. Hsu, "FPGA-based acceleration of neural network for ranking in web search engine with a streaming architecture," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 662–665.

[21] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using opencl," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2015, pp. 167–170.

[22] B. Sukhwani, H. Min, M. Thoennes, P. Dube, B. Iyer, B. Brezzo, D. Dillenberger, and S. Asaad, "Database analytics acceleration using FPGAs," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[23] R. Shea, F. Wang, H. Wang, and J. Liu, "A deep investigation into network performance in virtual machine based cloud environments," in *Proceedings of INFOCOM*, 2014, pp. 1285–1293.

[24] L. Chen, S. Patel, H. Shen, and Z. Zhou, "Profiling and understanding virtualization overhead in cloud," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, 2015, pp. 31–40.

[25] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, and E. Chung, "Azure accelerated networking : SmartNICs in the public cloud," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018, pp. 51–64.

[26] Y. Tokusashi, F. Pedone, and R. Soulé, "The case for in-network computing on demand," in *Proceedings of the EuroSys Conference*, 2019.

[27] A. Hayashi, Y. Tokusashi, and H. Matsutani, "A line rate outlier filtering FPGA NIC using 10GbE Interface," *SIGARCH Computer Architecture News*, vol. 43, 2015.

[28] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner, "RIFFA 2.1: A reusable integration framework for FPGA accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 4, 2015.

[29] K. Vipin and S. A. Fahmy, "DyRACT: A partial reconfiguration enabled accelerator and test platform," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2014.