Quantifying the Latency Benefits of Near-Edge and In-Network FPGA Acceleration

Ryan A. Cooke
University of Warwick
Coventry, United Kingdom

Suhaib A. Fahmy
University of Warwick
Coventry, United Kingdom

ABSTRACT

Transmitting data to cloud datacenters in distributed IoT applications introduces significant communication latency, but is often the only feasible solution when source nodes are computationally limited. To address latency concerns, cloudlets, in-network computing, and more capable edge nodes are all being explored as a way of moving processing capability towards the edge of the network. Hardware acceleration using Field Programmable Gate Arrays (FPGAs) is also seeing increased interest due to reduced computation latency and improved efficiency. This paper evaluates the the implications of these offloading approaches using a case study neural network based image classification application, quantifying both the computation and communication latency resulting from different platform choices. We consider communication latency including the ingestion of packets for processing on the target platform, showing that this varies significantly with the choice of platform. We demonstrate that emerging in-network accelerator approaches offer much improved and predictable performance as well as better scaling to support multiple data sources.

CCS CONCEPTS

• **Hardware → Hardware accelerators**; • **Computer systems organization → Reconfigurable computing**; • **Networks → In-network processing**;

KEYWORDS

Edge computing, hardware acceleration.

ACM Reference Format:
Ryan A. Cooke and Suhaib A. Fahmy. 2020. Quantifying the Latency Benefits of Near-Edge and In-Network FPGA Acceleration. In *Proceedings of 3rd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '20)*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3378679.3394534

1 INTRODUCTION

With the Internet of Things driving an explosive growth in the connectivity of resource constrained computing platforms, the latency implications of cloud-based computation offload become an important consideration in deciding how to deploy distributed applications. Edge computing represents the broad paradigm of moving processing away from high performance centralised datacenters towards data sources at the periphery of the network. Computing resources are typically less capable at edge nodes but communication is minimised since data need not be moved up the network to be computed on. Communication latency to the cloud can be significant, but for complex applications where the capabilities of datacenter servers offer a significant improvement in computation latency, overall application latency can be improved. This interplay between communication and computation latencies is heavily influenced not just by the network distance but also the choice of processing platform and the complexity of moving packets from a network interface to the processing hardware. As application latency has become more important and hardware at the network edge has improved, the benefits of offloading to centralised computing resources is now heavily impacted by these inherent communication delays.

In [1], the authors demonstrated the significant impact of cloud offload latency on the performance on neural network applications considering different locations of network "edge" processing. We expand upon that work by considering how the choice of computing platform impacts both communication and computation latency.

There has been an increasing trend of adopting heterogeneous specialised hardware in the datacenter to improve computation performance and efficiency. FPGAs in particular have seen increased use in the datacenter due to their flexibility and increased performance per watt compared to CPUs and GPUs in a variety of applications [6, 9, 15, 16].

Cloudlets, also referred to as edge servers, are small-scale datacenters or servers deployed close to data sources in an attempt to provide cloud-like services a few hops away in the network [8, 22]. Data traverses fewer switches over a LAN instead of the Internet, reducing communication delay and improving predictability. Cloudlets may utilise capable hardware comparable to that found in larger cloud datacenters, including hardware accelerators.

Enhanced computation capability at source nodes is an additional emerging trend, where simple microcontrollers are giving way to more capable single board computers like the Raspberry Pi, allowing more complex computation to take place at the data sources. FPGA acceleration is also possible at these nodes, through the use of FPGA SoC platforms such as the Xilinx Zynq, which tightly couples an FPGA fabric with an Arm Cortex A9 processor. These have the advantage of running commodity software for programmability, tightly coupled with custom hardware accelerators for offloaded computation, with the potential for reconfiguration to support dynamic workloads. Application specific accelerators such as the Google Edge TPU allow for computationally intensive machine learning applications to run on specialised hardware at the network edge. These more capable embedded devices can also be used as cluster heads for sensor nodes, or gateways between a collection of data sources and the rest of the network.

The emerging paradigm of in-network computing [17] sees the high performance network switches used to route packets extended

EdgeSys '20, April 27, 2020, Heraklion, Greece
2020. ACM ISBN 978-1-4503-7132-2/20/04…$15.00
https://doi.org/10.1145/3378679.3394534

with the capability of computing on that data in-flight [5]. Application tasks can be deployed to heterogeneous networks comprising both dedicated computing resources in the network and extended network elements [3]. FPGAs are a key candidate architecture for such a paradigm, as they are well suited to packet processing [7], support tight coupling of custom hardware accelerators, resulting in lower latency computation with minimal additional offload latency, and support the dynamic reconfigurability required to support sharing among multiple applications [21].

We use a case study neural network based image classification application to explore the latency implications of different computation offload strategies, assuming a network of edge nodes interacting with a cloudlet server through an FPGA-based layer 2 switch. We explore how edge, in-network, and cloudlet deployments with heterogeneous hardware impact overall performance for a complex application that would traditionally be offloaded to the cloud. We consider how both the communication and computation latencies of different deployment approaches impact overall application performance, including for shared resources. While network round-trip delay contributes significantly to communication latency, we also consider the packet ingestion latency required to process streaming network data, which becomes a more significant component as processing is moved nearer to the data sources, and is heavily impacted by platform choice.

2 DESIGN AND EXPERIMENTS

We consider a distributed application where data is captured at edge nodes, transmitted through network switches via Ethernet, to a server that acts as a cloudlet platform, a common architecture for IoT applications. We use a neural network based image classification application that processes images streamed from the edge of the network. Our evaluation is not focused on the application itself, but rather the relationship between communication and computation latency for such distributed deployments. We have selected this example as it involves the transfer of considerable amounts of data, as well as high enough computational complexity, while also demonstrating a good range of computational scaling on these different architectures. Our investigation focuses on latency – the time taken to receive a result – rather than throughput. This is important in a variety of safety critical applications or where data is time sensitive.

Our network architecture comprises 3 layers, an edge layer, a switching layer, and a cloudlet layer. Edge nodes produce images which they can transmit via 100Mb Ethernet through layer 2 network switches, which can forward the data to a cloudlet server. All layers in the topology can potentially perform the image classification task. The general structure of the setup, showing all the possibilities for application deployment is shown in Figure 1.

2.1 Application

The image classification application uses a Deep Neural Network (DNN) called SqueezeNet [10], that has a small memory footprint, designed for resource constrained platforms. In recent years there has been significant research into optimising DNN architectures for inference at the edge on hardware with less capable resources, trading off accuracy, runtime, and energy consumption [11, 24]. These models aim to reduce complexity and memory footprint while maintaining accuracy. In [1], the authors demonstrated how the performance of such compact DNNs scales on traditional CPU-based edge nodes, while also evaluating network delay based on the location of the edge device. We selected SqueezeNet as it is a compact model with a lower number of operations, leading to comparatively lower inference time, feasible for deployment on constrained platforms, compared to other models [19]. This better allows us to focus on the communication latency considerations, rather than applications where computing latency significantly dominates. It also means the lessons learnt are not tied as tightly to the capabilities of the specific architectures used. SqueezeNet is also representative of the type of DNN designed for use on edge devices such as the Raspberry Pi.

Input images are 224×224 pixels with 32-bit combined channel depth. The neural network has 10 layers, and uses 32-bit floating point weights, totalling 4.7MB. It can be compressed to 0.5MB while retaining the same accuracy [10]. The model is pre-trained and weights pre-loaded on all platforms. Images stream from the edge nodes, mimicking connected camera sources. In our experiments, to avoid saturating the network images are sent one at a time, and the time taken to receive a result measured before the next image is sent. Images are processed to determine the probability that each image belongs to one of the 1000 classes classified by the model.

2.2 Measurements

We measure the total time taken to carry out the classification task on each platform, focusing on latency, rather than throughput. This includes the computation latency, as well as the communication time to transfer images from the edge node to the target computing platform, and to receive the result.

All measurements are taken utilising specialised timing hardware implemented in the FPGA network switch, allowing for consistent measurements across the different platforms and independence from other tasks running on the various platforms. The edge nodes start transmitting an image upon receiving a start command from the FPGA which records the start time for the experiment as this command is sent. The finish time is recorded when the edge node receives the result. A free running counter running at 200MHz gives a resolution of 6.4ns.

2.3 Platforms

Edge Node (1): Edge nodes may act as a gateway or cluster head, or the data source itself. We implement an example of such a node in our experiments using a Raspberry Pi Model 3B running Raspbian OS. Up to 4 edge nodes can be connected to the switch in our experiments. It can carry out the full computation using a Keras implementation of SqueezeNet, or transmit the image through the Linux sockets API over 100Mb Ethernet through the network switch to another platform. Image data is sent using raw Ethernet frames, with no layer 3 or 4 headers. 100Mb Ethernet represents a realistic channel for a lightweight edge node, where LPWAN would be too slow or cellular too costly for video streaming. While network bandwidth continues to scale, the ingestion latency required to process received data at a node does not scale proportionally.

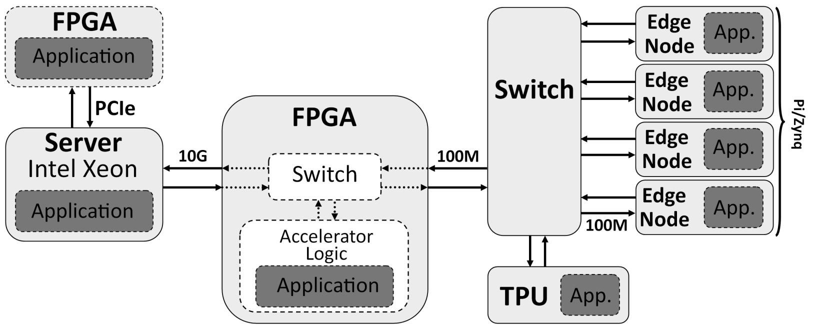


Figure 1: Outline of the experimental setup.

FPGA Accelerated Edge Node (2): Edge nodes may be enhanced with hardware accelerators or co-processors. We build an FPGA accelerated edge node using the Xilinx Zynq based Arty Z7 small form factor development board suited for edge applications. The Zynq consists of a processor subsystem (PS) and programmable logic (PL) and functions similar to the Raspberry Pi but with compute-intensive functions offloaded to a hardware accelerator in the PL.

For our experiments, the SqueezeNet accelerator logic was generated using Vivado HLS, and is implemented in the PL using a heavily modified version of the implementation in [13]. HLS allows for the expression of accelerator logic in annotated C rather than a hardware description language like Verilog. Accelerator parameters such as the memory offsets and sizes of layers are configured through an AXI-Lite register interface, through software running on the PS. Weights and image data are stored in on-board DRAM and data can also be temporarily stored in PL memory while being used.

TPU Accelerated Edge Node (3): An alternative approach is to attach an application specific accelerator to an edge node such as the Google Coral board hosting an ARM processor and tightly coupled Edge Tensor Processing Unit (TPU) coprocessor. The execution of Tensorflow models can be offloaded from the host processor to the TPU for significant improvements to latency and throughput.

Platforms such as the Edge TPU may alternatively be used as shared computing resources available over a network. We model this scenario by connecting a Raspberry Pi edge node to the TPU board through the network switch over 100Mb Ethernet.

Cloudlet Server (4): We use a Linux server running Ubuntu 18.04 on a 12-core 2.2GHz Intel Xeon E5-2650 v4 CPU, with 64GB of RAM. This machine has a 10Gb/s Mellanox Technologies MT26448 network card with SFP+ transceivers. The application runs via Python, with frames sent and received through the sockets API and the SqueezeNet model executed using Keras.

FPGA Accelerated Cloudlet Server (5): Computation can also take place on an FPGA accelerator attached to the Linux server via PCIe. In our experiments we implement this using a Xilinx VC709 evaluation board that has a PCIe Gen 3×8 interface. On the FPGA fabric we use the DyRACT framework [20] to manage communication between accelerator and host. The software application on the host opens a network socket and waits to receive an image from the edge node, triggering the the accelerator when the image is received.

In-Network Processing (6): In our testbed, we use an unmanaged layer 2 switch and a Xilinx KC705 evaluation board, which hosts a Kintex-7 FPGA to connect edge devices. The standalone layer 2 switch allows 4 edge devices to be connected to the single 100Mb port on the FPGA board. The FPGA switch bridges the RJ45 100M Ethernet interface of the edge nodes and the 10Gb Ethernet SFP+ interface of the cloudlet server, and can also host a hardware accelerator. When a packet is received through the 100Mb interface from an edge node, the destination field in the frame header determines where it is sent as part of the switch logic.

If the field contains a specified address, the payload is transferred to the board DRAM, to be used with the accelerator, otherwise it is forwarded to the 10Gb output port. As up to 4 edge devices may be transmitting to the FPGA, the source address field of the inbound frames is used to differentiate between data sent by each device. Image data from each device is written to a different memory address offset in DRAM based on the source address. Upon reception of a full image from any edge device, a request is generated to start the accelerator for that device.

The SqueezeNet accelerator logic was generated using Vivado HLS as for the accelerated edge node, but the expanded resources of the Kintex FPGA on the KC705 allow for greater unrolling of loops and increased parallelisation of the design. The board DRAM also contains the full set of weights for the accelerator, and weights are loaded prior to the start of the experiment, so there is less accelerator management overhead.

3 RESULTS

We measured total application latency, including the time to send the image from the edge node to the platform carrying out the computation, ingestion of the data and computation, and returning of the result back to the edge node. We isolated the communication and computation time for each scenario.

3.1 Isolated edge node measurements

First we consider the measurements for a single active edge device. The computation latencies measured differ from raw execution benchmarks, such as in [4], primarily because our latency measurements include the time to ingest input data packets, rather than raw inference time for data already in memory. This distinction is important in the context of offloading computation where data must be received over the network.

It should be clarified that we do not consider the network communication latency to the location of the edge device beyond the single hop, but rather the network ingestion and computation latencies, as these are platform dependent. The experiments in [1] offer an insight into how locating these platforms may further impact overall latency, but the in-network approach has fundamentally lower latency.

Table 1: Computation and communication latency for offload from a single edge node in milliseconds.

Computing Location	Comp. Latency	Comm. Latency	Total Latency
1) Edge (Pi)	2380	0	2380
2) Edge (Zynq)	1660	0	1660
3) TPU	210	80	290
4) Server	340	50	390
5) Server + FPGA	60	60	120
6) Network Switch FPGA	60	1	61

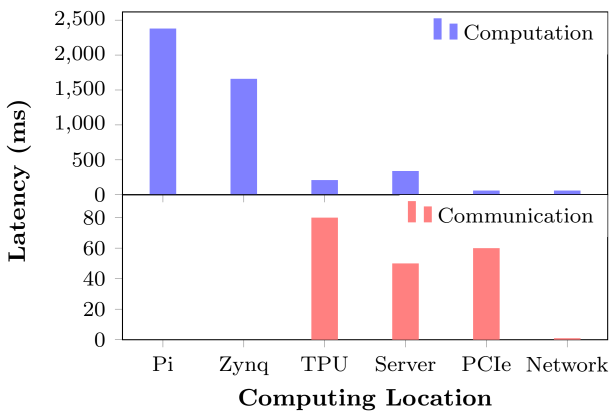


Figure 2: Breakdown of latencies per image for different offload scenarios.

It can be seen in Figure 2 that performing all computation on the edge node with no acceleration has high computation latency, and despite having no communication latency, this scenario performs worst, justifying computation offload. Less computationally intensive applications would reduce this disparity between platforms since the computation latency would not be so dominant, but we have selected a simpler neural network for this study, and others are likely to show even more disparity.

Replacing the edge node with a more capable Xilinx Zynq SoC platform improves latency by 30% due to the hardware acceleration provided by the FPGA. The embedded FPGA SoC, however, is limited in capacity and cannot fully parallelise execution of the neural network, and so numerous iterations of hardware execution are managed by software, adding to the latency. Further optimisation is possible, for example, 8 bit fixed point quantization, which would improve area efficiency, though require further design effort.

The Edge TPU was deployed as a network connected accelerator on account of its cost compared to other options meaning it is more likely to be a shared resource. It demonstrates a significant reduction in computation latency on account of the optimised parallel hardware and native compilation of the model for this architecture that supports 8-bit arithmetic. As an offload engine attached to the network, there is some communication latency due to the software network stack running on the Arm core on the Edge TPU board. The Zynq FPGA SoC platform can also perform this role with lower ingestion latency by diverting packets directly into the PL for processing by the accelerator without the involvement of the PS processor, as has been demonstrated in [18].

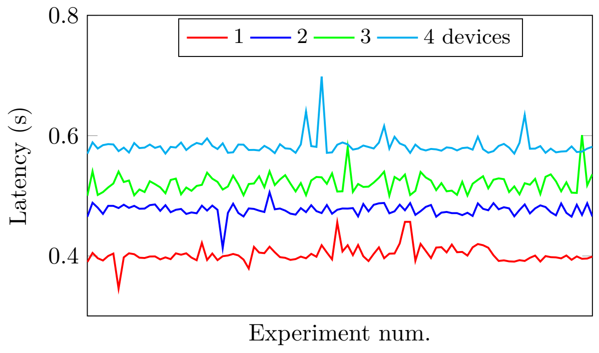
Offloading to the server outperformed both the Raspberry Pi and FPGA SoC platforms due to the more capable server processor. Indeed, it achieves close to the computation latency of the Edge TPU, albeit at much higher power consumption. Improved network stack performance on a server class processor also results in lower communication latency compared to the Edge TPU.

Adding a more capable hardware accelerator using a server class FPGA offers even lower computation latency compared to the other architectures discussed so far, as loops can be fully parallelised and there are fewer movements of data to and from memory. This FPGA accelerator is integrated into a server over a PCIe link which adds a modest communication latency as data must first traverse the network stack, then be moved to the accelerator over PCIe. The software stack that manages this accelerator also adds some management overhead that contributes to the communication latency. However, the significant acceleration of the computation means total latency is significantly reduced.

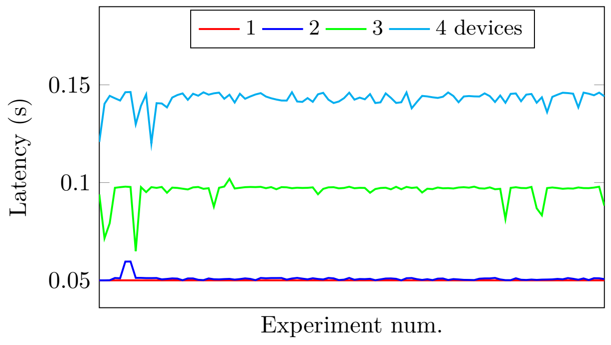
Attaching the FPGA accelerator directly to the switch drastically reduces communication latency as network packets can be directly ingested by the accelerator, and all data movement is managed in hardware. This coupled with the low computation latency of the FPGA accelerator means this deployment has significantly lower latency than those discussed earlier. The communication latency we measured here does not include multiple hops over a network, as characterised in [1], which would further increase the magnitude and variability of communication latency, rather we have isolated the platform-specific components in our experiments.

While the Edge TPU is capable of very low inference latency, measured at just 6ms in isolation, packet ingestion and data transfer from the host processor to the Edge TPU silicon increases latency. This highlights, once again, that the data movement is of paramount importance in determining the end-to-end performance of such a connected application.

We are concerned in this paper with relationship between computation latency and communication (ingestion) latency, considering deep neural network inference as a case study. Further tweaking of neural network model parameters to optimise for latency against accuracy can improve performance on constrained edge platforms [12]. Alternative neural networks also exhibit different scaling of computation latency on different devices. The effect of varying these parameters on the different accelerators is an avenue



(a) Cloudlet



(b) Networked FPGA

Figure 3: Latency for cloudlet server and network switch FPGA servicing multiple edge devices for 100 experiments.

for further work, as is the effect of other factors such as network, processor, and I/O stress.

3.2 Impact of multiple edge devices

The results presented so far consider one edge device with exclusive access to the network and computing resources. In reality, multiple connected devices will stream data, leading to contention for resources and larger, more unpredictable delays. Computing on edge platforms, such as the Raspberry Pi and FPGA SoC, while having a higher latency individually, is not subject to these resource contention issues, as computation is done locally, with no communication cost. So for large numbers of devices, this approach may scale favourably. Cloudlet servers or networked FPGA accelerators will typically share computing resources across multiple streams.

To examine the effects of resource contention, we adapted the previous experiments to support different numbers of streaming edge devices. Our experimental setup allows us to connect up to 4 edge nodes to the network switch and FPGA and take detailed measurements. Experiments were completed with a closed loop traffic model, where one measurement was taken before the next experiment was started to avoid flooding the network with data and causing packet loss. While multiple streams were active in the system, we measure the latency of a single stream.

When using the cloudlet server, all edge devices share the output port of the layer 2 switch, the output port of the FPGA switch, as well as the network pipeline on the FPGA switch, in order to reach the server. At the server, the streams share the network and processor resources. Once a full image for the measured stream is captured, it is processed with the SqueezeNet model in a separate software process. Latency results are shown in Figure 3a. As the number of data sources scales, overall latency increases, up to 600ms with 4 devices connected.

It can be inferred that increasing the number of devices would further increase latency. This could cause the cloudlet to be slower than accelerated edge platforms such as the Zynq, where computation is performed locally and resources are not shared. With a large number of edge devices serviced by a single cloudlet server, standard edge nodes without acceleration might offer lower overall latency due to this network/compute contention. The server-hosted FPGA accelerator would suffer similar sharing costs since its computation is managed in software and the communication latency is similar to the cloudlet server scenario.

For the in-network switch hosted FPGA, the sharing is more fine-grained, so we expect it to better scale with the number of edge nodes. The results shown in Figure 3b demonstrate that an additional edge node has minimal effect on total latency due to the buffering and pipelining of the FPGA design resulting in reduced contention. Adding a third edge node almost doubles total latency, and increases jitter. A fourth edge node adds around 3× the latency compared to a single node, and adds even greater jitter. There are multiple factors than contribute to these increases. The accelerator shares the memory interface with the network pipeline, and to avoid the loss of data, the arbiter gives priority to network data. As the accelerator cannot retrieve image and weight data from memory while the memory interface is busy, latency is higher. Increasing the number of edge node streams means that the memory interface is busy more often. Further increases in latency come from the sharing of a single network interface. More streams means that the data for any given stream is more likely to be in a queue, which also explains the increased variation.

Despite these increases in latency with several connected devices, the network attached FPGA still outperforms the equivalent number of edge nodes performing computation locally by a large margin. The lower computation latency of the FPGA relative to the server software means that it is more likely to scale well with a greater number of edge node data streams.

3.3 Discussion

The interplay between computation latency and communication latency has a significant impact on overall application latency when offloading computation. Network round-trip time is only one aspect of communication latency, and packet ingestion latency also has a noticeable impact. For complex applications, where computation latency dominates, these factors may not be as important. However, with the wider use of hardware acceleration, computation latency is reduced and communication latency, including ingestion latency becomes more important.

Integrating the accelerator into the network switch made communication latency negligible. The TPU hardware, while capable of the fastest inference, had its overall effectiveness reduced by preprocessing and communication time. Going forward, we can see that a complete view of computation and communication latency must be considered when evaluating offload platforms.

While for a single device, offloading from the edge node to any other platform led to large reductions in end-to-end latency, this benefit diminished as the number of edge nodes increased. As systems scale, considerations must be made as to how offload from a large number of devices is handled. In these situations, local accelerators such as the Zynq, while seeming to under-perform in our experiments, will gain value as they are not shared across devices.

4 RELATED WORK

End-to-end latency analysis for an augmented reality application offloaded from mobile devices to the cloud is presented in [23]. Computation is identified as the primary contributor to latency, but alternative platforms are not considered, nor is the impact of a near-edge deployment. The work in [2] examines the latency for a range of cognitive assistance applications, showing that when computation dominates communication latency, the benefits of executing at the edge rather than the cloud are reduced. Network latency is shown to dominate total application latency in [1], in which GPUs and algorithmic optimisations were used to decrease computation latency. Network interface cards, PCIe interconnect, and the software network stack were all shown to contribute to non-deterministic latency in the datacenter in [25]. The PCIe contribution to NIC network packet ingestion latency was evaluated in [14].

5 CONCLUSION AND FUTURE WORK

This paper has explored alternative approaches to accelerated computation near the edge, showing how the benefits of hardware accelerators can be exploited for offloading from lightweight Internet of Things nodes. A case study neural network image classification application was implemented on a variety of platforms for different offloading scenarios. We showed that both communication and computation latency must be considered and that the choice of platform can affect both of these significantly, and that packet ingestion latency is an important factor. In-network acceleration demonstrated significant reductions in communication latency, coupled with the low computing latency of FPGA acceleration. While scaling the number of edge devices sharing these near-edge computing resources leads to increases in total latency due to network and compute contention, the in-network FPGA accelerator approach scaled better than software on a server.

We are interested in exploring the effect of scaling such systems to larger numbers of edge nodes and more complex network topologies, wherein we expect more variability in latency. Experiments with a wider range of applications with differing computation and communication ratios and platform scaling would allow us to make more a more general conclusion about the computation acceleration factors required to offset communication latency for different platforms.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Alejandro Cartas et al. 2019. A reality check on inference at mobile networks edge. In *Proc. EdgeSys*. 54–59.
[2] Zhuo Chen et al. 2017. An emperical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proc. SEC*.
[3] Ryan A. Cooke and Suhaib A. Fahmy. 2020. A model for distributed in-network and near-edge computing with heterogeneous hardware. *Future Generation Computer Systems* 105 (2020), 395–409.
[4] Coral. 2019. Edge TPU performance benchmarks. https://coral.ai/docs/edgetpu/benchmarks/
[5] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. 2016. Paxos made switch-y. *ACM SIGCOMM Computer Communication Review* 46, 2 (2016), 18–24.
[6] Suhaib A. Fahmy, Kizheppatt Vipin, and Shanker Shreejith. 2015. Virtualized FPGA Accelerators for Efficient Cloud Computing. In *Proc. CloudCom*. 430–435.
[7] Andreas Fiessler, Sven Hager, Bjorn Scheuermann, and Andrew W. Moore. 2016. HyPaFilter: a versatile hybrid FPGA packet filter. In *Proc. ANCS*.
[8] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards wearable cognitive assistance. In *Proc. MobiSys*. 68–81.
[9] Hanaa M. Hussain, Khaled Benkrid, Ahmet T. Erdogan, and Huseyin Seker. 2011. Highly parameterized K-means clustering on FPGAs: Comparative results with GPPs and GPUs. In *Proc. ReConFig*. 475–480.
[10] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.
[11] Hyoukjun Kwon, Tushar Krishna, Liangzhen Lai, and Vikas Chandra. 2019. HERALD: Optimizing Heterogeneous DNN Accelerators for Edge Devices. arXiv:cs.CV/1909.07437
[12] Nicholas D. Lane and Pete Warden. 2018. The deep (learning) transformation of mobile and embedded computing. *Computer* 51, 5 (2018), 12–16.
[13] Samyukta Lanka. 2017. Squeezenet HLS implementation. https://github.com/lankas/SqueezeNet
[14] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Sergio López-buedo, and Andrew W. Moore. 2018. Understanding PCIe performance for end host networking. In *Proc. SIGCOMM*. 327–341.
[15] Andrew Putnam et al. 2015. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *IEEE Micro* 35, 3 (March 2015), 10–22.
[16] Yun R Qu, Hao H Zhang, Shijie Zhou, and Viktor K Prasanna. 2015. Optimizing many-field packet classification on FPGA, multi-core general purpose processor, and GPU. In *Proc. ANCS*.
[17] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. 2017. In-network computing is a dumb idea whose time has come. In *Proc. Hotnets*.
[18] Shanker Shreejith, Ryan A. Cooke, and Suhaib A. Fahmy. 2018. A smart network interface approach for distributed applications on Xilinx Zynq SoCs. In *Proc. FPL*. 186–190.
[19] Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galan, and Ángel Rodríguez-Vázquez. 2018. Performance analysis of real-time DNN inference on Raspberry Pi. In *Real-Time Image and Video Processing*. Article 106700F.
[20] Kizheppatt Vipin and Suhaib A. Fahmy. 2014. DyRACT: A partial reconfiguration enabled accelerator and test platform. In *Proc. Field Programmable Logic and Applications*.
[21] Kizheppatt Vipin and Suhaib A. Fahmy. 2018. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. *Comput. Surveys* 51, 4, Article 72 (July 2018).
[22] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. 2016. Fog computing: Platform and applications. In *Proc. HotWeb*. 73–78.
[23] Wenxiao Zhang, Bo Han, and Pan Hui. 2017. On the networking challenges of mobile augmented reality. In *Proc. VR/AR Network*.
[24] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. arXiv:cs.CV/1711.07128
[25] Noa Zilberman et al. 2017. Where Has My Time Gone?. In *Proc. PAM*.