Exploring FPGA Acceleration for Distributed Serverless Computing

Ziyi Yang, Suhaib A. Fahmy
King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia
ziyi.yang@kaust.edu.sa

***Abstract*—Serverless computing has become a popular cloud computing paradigm. However, its deployment abstraction entails significant performance overheads. We explore the potential for enabling serverless computing on FPGAs and present some early results that show the concurrency and scalability benefits on a stream processing workload. The FPGA enables flows of network data to flow directly into accelerators, which is beneficial for scenarios involving large packets and multiple request streams.**

I. Introduction

Serverless computing is a cloud computing execution model in which the cloud provider allocates computing resources to customer applications on demand. It abstracts server management away for application developers. Developers upload their application code, composed of chained function invocations, and the provider determines how to allocate these to computing resources (servers) on demand. Users are then billed only for the time that their functions are running. This has the benefit for the user of not having to manage computing resources, and for the provider of being able to allocate functions to resources in a way that best suits their infrastructure.

However, current serverless computing frameworks have three major overheads:
1) Communication overhead due to clients sending data to the cloud through the network. **2) Computing overhead** due to use of high-level languages and libraries. **3) Abstraction overhead** as these functions are run in a containerized environment and invocations of serverless function go through a full network stack.

FPGA accelerators could address above problems and enable serverless computing with **efficient computation** and **low-overhead composition**. The fine-grained parallelism that is exploitable on FPGAs is well suited to the types of stateless functions used in serverless, which would reduce the computing cost. The serverless programming model breaks applications down into compact functions. FPGAs can be directly interfaced over the network, leading to significant reductions in communication overhead for these small functions. A serverless abstraction can be built in the FPGA hardware to minimise the abstraction overhead while still offering the benefits of the FPGA such as fast reconfiguration (function invocation). Finally, FPGAs can be deployed outside of the datacenter for a more distributed, and hence lower latency, response.

The metrics considered when evaluating serverless computing systems are **communication performance**, **startup latency**, and **resource efficiency and performance isolation** [1]. By leveraging specific inherent characteristics of FPGAs, we can target these metrics. FPGAs are well suited to packet processing, and open source implementations of hardware network stacks are available that can be extended, e.g. Easynet [2] for TCP/IP. Direct ingestion of network data for processing can dramatically reduce the overhead of the FPGA as an accelerator [3]. FPGAs also have a power profile and form factor that can support a wide range of deployment scenarios when networked [4], [5]. Suitable high level abstractions can be built to manage the FPGA [6]. Partial reconfiguration (PR) can be exploited to support multi-tenancy with low startup time in the range of tens of milliseconds. Multiple PR regions can be chained together for low latency function chaining, and this can be managed via the network [7]. Lastly, in terms of multi-tenant efficiency and isolation, it is possible to initialize multiple TCP connections and deliver isolated performance at the hardware level, as demonstrated in Multes [8].

In this paper, we evaluate concurrency and scalability of a stream processing workload and discussed the potential benefits that can be provided by mapping serverless functions to network attached FPGA. In addition, we suggest future research directions in this area.

A. Experimental Setup

We deploy a Streaming Top-K workload in three settings: *serverless*, which uses a serverless framework; *baremetal*, which runs the same code on a dedicated server; and *FPGA*, which integrates a hardware accelerator with the open source HLS TCP/IP stack in [2].

Our server is based on an AMD Ryzen Threadripper PRO 3975WX (32 Core @ 3.5 GHz) processor with 512 GB DDR4 3200 DRAM running Ubuntu Linux 18.04. For the *serverless* platform, we deploy version of 0.6.23 of the 'fn' framework. In the *baremetal* setup, we directly execute the function invocation through a web server (using the default HTTP Python package). In both cases, we use a reverse proxy (request interceptor) to measure the latency of each request invocation from an external client that serves as a closed-loop load generator as shown at the top of Fig. 1.

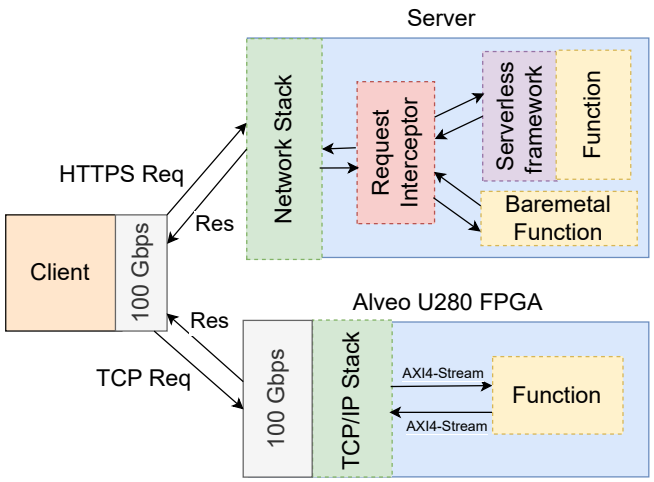

Fig. 1: Experimental setup.

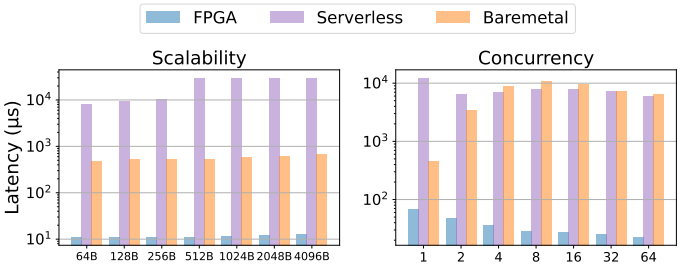

Fig. 2: Top-K workload comparison on three platforms.

The *FPGA* setup is evaluated using an NVIDIA-Mellanox-ConnectX-6 SmartNIC directly connected to a Xilinx Alveo U280 accelerator board using a 100Gbps DAC cable in the same server machine. Single request latency measurement is done by extracting timestamps from the headers of TCP packets that are captured on the NIC interface using Wireshark as shown at the bottom of Fig. 1.

B. Scalability with Packet Size

Sequences of 1000 requests with sizes of 64B, 128B, 256B, 1024B, 2048B, and 4096B are generated by the client. The cold start effect of the serverless platform is filtered out. The left plot in Fig. 2 shows average latency (on a log scale) for all packet sizes. Mapping Top-K on FPGA improves latency by $42\times$ to $53\times$ and $736\times$ to $2473\times$ compared to baremetal and serverless, respectively. The serverless platform shows a significant overhead when the invocation size increases from 256B to 512B. The latency slightly increases on the baremetal platform as the invocation size increases, while the FPGA platform maintains stable latency for all sizes of invocation.

C. Concurrent Requests

The client now generates N concurrent invocations (TCP requests in the FPGA case). For baremetal and serverless, the latency measurement starts when the first function request reaches the request interceptor and stops when the Nth response is collected. For FPGA, the latency measurement starts when the first TCP packet is sent by the NIC and ends when the Nth response is collected, which includes the three-way handshake, data transmission, but excludes the four-way handshake closing stage of the TCP connection. All experiments are repeated 10 times. The right plot in Fig. 2 shows the overall batch latencies divided by N. The first result of each batch is filtered out for serverless due to the cold start effect. The results show that the baremetal platform is not capable of handling multiple requests, since the latency per request increases $75\times$ from single to double requests. On the other hand, both the FPGA and serverless platforms demonstrate a decreasing trend in latency per request as the number of requests increases, indicating good scaling with parallel requests. The FPGA shows $0.31\times$ latency per request for 64 parallel requests compared to a single request, suggesting the FPGA has better concurrency performance.

II. Future Work

Our preliminary experiments focus on single function offloading. Function to function communication is essential in serverless computing. Investigating the performance of offloading a function chain across multiple FPGAs, as well as combining sub-chains into a single FPGA, will be explored.

To support this, we are evaluating and developing a number of commonly used stream processing functions to be able to evaluate larger application benchmarks from the serverless and Internet of Things areas.

Secondly, other performance metrics, such as energy consumption, will be evaluated to further justify the use of FPGAs. Apart from saving energy on computing, the benefit of direct network ingestion should result in significant energy gains.

Finally, we will develop a management interface that automates function invocation, chaining, as well as providing instrumentation for evaluation. This will ensure isolation, while also managing multi-tenancy through partial reconfiguration.

We expect this project to enable the idea of efficient serverless computing outside the datacenter.

References

[1] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen, "Characterizing serverless platforms with ServerlessBench," in *ACM SoCC*, 2020, pp. 30–44.
[2] Z. He, D. Korolija, and G. Alonso, "Easynet: 100 Gbps network for HLS," in *FPL*, 2021, pp. 197–203.
[3] R. A. Cooke and S. A. Fahmy, "Characterizing latency overheads in the deployment of FPGA accelerators," in *FPL*, 2020, pp. 347–352.
[4] C. Bobda *et al.*, "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, 2022.
[5] R. A. Cooke and S. A. Fahmy, "Quantifying the latency benefits of near-edge and in-network FPGA acceleration," in *EdgeSys*, 2020, pp. 7–12.
[6] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on FPGAs?" in *OSDI*, 2020, pp. 991–1010.
[7] A. R. Bucknall, S. Shreejith, and S. A. Fahmy, "Network enabled partial reconfiguration for distributed FPGA edge acceleration," in *FPT*, 2019, pp. 259–262.
[8] Z. István, G. Alonso, and A. Singla, "Providing multi-tenant services with FPGAs: Case study on a key-value store," in *FPL*, 2018, pp. 119–1195.