# Hardware-Accelerated FaaS for the Edge-Cloud Continuum

Anastasios Nanos *, Aristotelis Kretsis†, Charalampos Mainas*, George Ntouskos*,
Aggelos Ferikoglou‡, Dimitrios Danopoulos‡, Argyris Kokkinis‡, Dimosthenis Masouros‡, Kostas Siozios‡,
Polyzois Soumplis†, Panagiotis Kokkinos†, Juan Jose Vegas Olmos§ and Emmanouel Varvarigos†
*Nubificus Ltd, United Kingdom,
‡Aristotle University of Thessaloniki, Greece,
§NVIDIA Corporation, Israel,
†Institute of Communication & Computer Systems, NTUA, Greece.

***Abstract*—We present an end-to-end solution to facilitate the seamless execution of hardware-accelerated compute-intensive tasks on heterogeneous hardware platforms spanning the Cloud-Edge continuum. Our approach includes a programming interface, orchestration, application management components, the vAccel framework, and a library of hardware-accelerated kernels. These components enable a Function-as-a-Service (FaaS) based operational flow that supports numerous diverse use cases while minimizing the time required for the developer to integrate their code and for the vendor to provide hardware acceleration capabilities to end users. Experimental results showcase the merits of our approach.**

***Index Terms*—Cloud-Edge Continuum, Serverless, Function-as-a-Service, Hardware Acceleration**

## I. Introduction

The cloud computing paradigm appears ideal to deploy and manage the Internet of Things and other applications' execution at scale. However, several use cases require data processing and decision-making closer to the data sources due to their mission-critical, low-latency, and near-real-time requirements. To this end, multi-layered computing architectures are considered, where computing resources and applications are distributed from the network's edge to the cloud, realizing the edge-cloud computing continuum.

Even though edge-cloud architectures expand the computing capacity of the traditional cloud paradigm by introducing an additional massive pool of computing resources, the inefficiency of conventional CPUs to provide fast, near real-time executions has also led to the introduction of hardware accelerators, such as GPUs and FPGAs, that accelerate computation in applications traditionally handled only by the CPU. Typical examples of accelerators include power-efficient devices at the edge (e.g., NVIDIA Jetson and Xilinx MPSoC) to high-performance, massively-parallel devices in the cloud (e.g., NVIDIA Ampere GPUs and Xilinx Alveo).

In addition, in the emerging serverless computing execution model, the provider takes care of the hardware and software resources to execute an application or a service [1]. Function as a service (FaaS) is a form of serverless computing where a function (a snippet of code) is provided by the user [2], while the cloud provider facilitates the loading and execution of the code when triggered by an event. Functions are designed to be short-lived. To enable execution isolation and minimal interference in multi-tenant environments, serverless vendors often execute these functions in containers or virtual machines.

979-8-3503-0322-3/23/$31.00 ©2023 IEEE

In our work, we present the mechanisms in the SERRANO platform [3] that enable the seamless execution of accelerated compute-intensive tasks, referred to as kernels, across the Edge-Cloud continuum. The proposed solution integrates and utilizes heterogeneous hardware platforms for the edge and the cloud in a unified way. More specifically, it includes (i) the SERRANO orchestration and application deployment components, (ii) the vAccel [26] framework that enables workloads to access diverse hardware accelerators securely and efficiently, and (iii) a library of accelerated kernels that leverage hardware and software acceleration techniques to improve the performance and energy consumption. These mechanisms extend the serverless computing paradigm across a hardware-accelerated Edge-Cloud continuum that can accommodate numerous demanding use cases.

The rest of this paper is organized as follows. Section II discusses the programmability of accelerators and serverless deployments. Section III presents an overview of the SERRANO platform. Section IV describes the proposed end-to-end solution for hardware-accelerated FaaS operation across the Edge-Cloud continuum. Section V provides an overview of the vAccel framework, while Section VI outlines the accelerated kernels. Section VII evaluates the efficiency of the proposed mechanisms. Finally, Section VIII concludes the work.

## II. Accelerators' programmability and serverless deployments

The advent of micro-services, supporting the trend of porting monolithic applications to distributed designs, and the constantly rising popularity of serverless computing have created the need for finer access granularity to hardware accelerators. Approaches like device passthrough and hardware-assisted virtualization are extremely restricting for these setups. Interestingly, popular serverless offerings, like AWS lambda, currently lack support for hardware acceleration due to flexibility limitations and security implications [4].

Edge devices are typically resource constrained. Low-power devices with decent hardware accelerators are already in the market [5], but, despite impressive recent improvements in CPU performance, these devices present a challenge for typical deployments due to limitations in local storage, memory, and CPU cores. On top of that, they cannot accommodate multiple isolated tenants, as they only support bare-metal container-based setups or standard device passthrough for VMs. Containers are notorious [6], [7] for their security implications, which could sometimes cause severe incidents, especially when ML inference is critical [8], [9].

Moreover, the diversity of current accelerator software stacks hinders the process of writing accelerate-able code targeting the multitude of the aforementioned environments. Building *generic* applications that use diverse hardware accelerator types and deploying them efficiently on diverse, multi-tenant, heterogeneous environments is a challenge, especially with respect to the software porting effort.

Transparently deploying serverless workloads at the edge of the network is challenging due to the variety of platforms available and the different ways they can be programmed. Hardware development for FPGAs can be performed at different levels of abstraction. The most commonly used are: (i) the Register Transfer Level (RTL), where circuit-level programming is done, and (ii) algorithmic-level methods such as High-Level Synthesis (HLS), which describe designs more abstractly. In addition, GPU programmability has greatly improved over the years with high-level languages such as CUDA and OpenCL, making GPU programming increasingly common in both academia and industry.

To accommodate multiple tenants on cloud infrastructure, vendors must address hardware accelerator device sharing. As isolation is crucial in such environments, vendors often use virtualization mechanisms to separate the execution environment between tenants. However, this presents a challenge for accelerator device sharing. Typical ways to expose hardware accelerator capabilities inside a virtualized guest involve (i) device pass-through, (ii) paravirtualization, and (iii) API remoting [10]. Device pass-through allows direct access to the physical device, whereas it achieves performance identical to running on bare metal. However, problems arise with sharing a single hardware accelerator across multiple guests executing on the same host. Another line of research focuses on implementing para-virtual drivers for GPUs [11] [12] or even FPGAs [13]. Para-virtual devices uniformly abstract the physical devices, exposing a single API inside the guest in the form of specialized kernel modules and guest libraries. However, it is difficult to efficiently abstract and unify a widely diverse set of hardware and acceleration framework APIs and their evolution. Finally, API remoting provides mechanisms to intercept calls to hardware accelerator drivers inside the guest and offload them to a host with direct access to the physical device. Several frameworks have been developed [14]–[17] that are typically hypervisor-agnostic but tend to suffer from overhead due to network latencies. AvA [18] identifies these shortcomings and suggests a framework for automatically generating hardware-specific and agnostic components through user-provided descriptions of the acceleration API they consume. Frameworks that provide acceleration functionality to VMs (or remote clients) tend to abstract operations at the lowest possible level (e.g., CUDA or OpenCL). Using vAccel, we can determine the layer of the acceleration stack that is to be abstracted and, thus, gain greater flexibility in the development of user code (function) as well as hardware-specific code (acceleration kernel).

Also, many serverless platforms have been developed and used by academia and the industry. Authors in [19] evaluated four known open-source serverless computing frameworks in an edge environment: Apache Openwhisk, Knative, OpenFaaS, and Kubeless. These platforms are used to develop serverless systems for heterogeneous processing units further. Authors in [20] presented a distributed FPGA sharing system to accelerate microservices and serverless applications in cloud environments while including a remote OpenCL library to access the shared devices transparently. [21] developed a FaaS deployment service, supporting heterogeneous computing platforms and heterogeneous functions, enabling the delivery of functions to the right platform. In the evaluation performed, four different target platforms are considered: (i) Google Cloud Functions (GCF), (ii) AWS Lambda, (iii) a private-cloud-cluster consisting of embedded Nvidia Jetson Nano devices and (iv) a hpc-node-cluster representing compute nodes from HPC systems. The HiveMind platform [22] enables programmable execution of task workflows in cloud and edge resources, instantiating serverless functions on shared nodes and allocating resources appropriately. In [23] authors presented $\lambda$-NIC, an open-source framework, for the development and deployment of serverless functions in SmartNICs (Data Processing Units - DPUs). Also, Molecule [24] is a serverless system supporting heterogeneous processing devices, including general-purpose ones (e.g., Nvidia DPU) and domain-specific accelerators (e.g., Xilinx FPGAs). Molecule abstracts hardware distribution and heterogeneity with a shim layer (XPU-Shim) and vectorized sandbox abstraction.

In our work, we use Xilinx Inc. Vitis framework, which provides a unified OpenCL interface for programming edge (e.g., MPSoC ZCU104) and cloud (e.g., Alveo U200) Xilinx FPGAs and the CUDA programming model for using NVIDIA GPUs. We abstract the acceleration operations to function granularity using vAccel. This approach may limit flexibility in terms of generic acceleration primitives, but provides lower data/operation transport overhead and enables interoperability when using different accelerators.

## III. SERRANO Platform

The SERRANO platform combines seamlessly, autonomously, and efficiently heterogeneous resources from various technology domains, including (i) edge platforms to bring adequate resources close to the end users, (ii) multiple clouds (federated operation) to increase robustness and scaling while reducing dependencies on a single cloud provider, and (iii) HPC infrastructures to provide enormous capacity. SERRANO

provides an abstraction layer that fully exploits the available heterogeneous resources and enables their seamless use.

SERRANO facilitates seamless application deployment and management across the heterogeneous continuum by implementing a distributed closed-loop control system that maintains the desired state through self-driven continuous adaptations. The deployment starts with the provision of the application description along with a high-level infrastructure agnostic description of deployment requirements (Step A). Next, SERRANO decomposes the high-level requirements into specific service goals (Step B).Next cognitive orchestration mechanisms assign the application's microservices to resources (Step C). SERRANO also automatically coordinates the application deployment and efficient data movement to the selected resources. Finally, service assurance mechanisms based on real-time telemetry and machine reasoning techniques safeguard that applications perform as intended while triggering any required re-optimization (Step D).

SERRANO implements a robust and flexible platform that caters to diverse application needs while maximizing resource utilization and security through a layered architecture (Fig. 1). The Resource Layer incorporates diverse edge, cloud, and HPC SERRANO-enhanced resources. The Infrastructure, Platform, and Application Telemetry stack captures and records telemetry metrics regarding the infrastructure and deployed applications. The Infrastructure Abstraction Layer abstracts the intricacies of managing and interacting with individual resources. The Secure Data Layer provides secure and efficient data access and transfer across individual platforms. The Orchestration Layer ensures efficient service orchestration and resource allocation within the disaggregated SERRANO infrastructure. Finally, the Service Layer analyses applications to determine the most suitable platform for deployment.

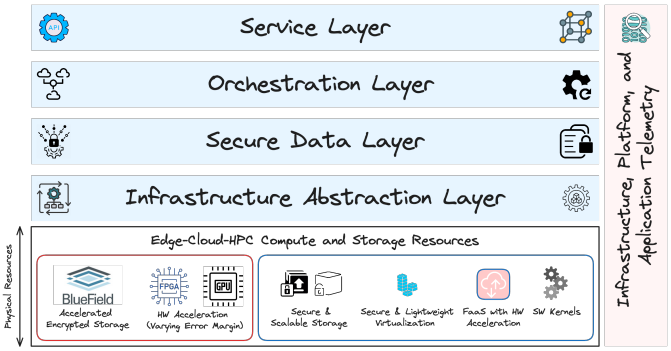


Fig. 1. SERRANO's layered architecture.

Moreover, SERRANO offers a comprehensive Service Development Kit (SDK) that enhances developer productivity in building, deploying, and managing applications that fully leverage the provided functionalities. The SERRANO SDK is developed in Python and includes a set of well-defined APIs.

## IV. Transparent Execution of Accelerated Kernels

The SERRANO platform supports two deployment methods for executing the available kernels on heterogeneous edge, cloud, and HPC resources. The first method involves deploying the kernels alongside the application services. This method is suitable when the application services have a specific set of kernels that must be executed repeatedly. The second method is based on the FaaS execution model, which allows on-demand deployment and scaling of accelerated kernels. In this case, an application service running in the SERRANO platform through the SERRANO SDK can request from the SERRANO platform the execution of a specific kernel. The SERRANO orchestration and deployment mechanisms promptly handle all required operations and provide results. The on-demand (FaaS-like) deployment in SERRANO ensures that users receive transparent access to accelerated kernels without the need to manage the deployment and execution process. This approach also optimizes the use of resources, allowing kernels to be executed on the most appropriate resources only when required by the application services.

To use either deployment method, data provisioning must be automated and abstracted into kernels, and results must be seamlessly delivered to users. From an end-user perspective, the overall process involves the following three steps (Fig. 2):

- *Step 1*: Push input data to SERRANO secure storage services and obtain the corresponding description for the execution request. This description includes a set of identifiers for SERRANO deployment mechanisms, enabling data retrieval and kernel execution preparation.
- *Step 2*: Submit the execution request to the SERRANO platform, specifying the desired kernel, and providing the input data description from Step 1.
- *Step 3*: Retrieve the results using the SERRANO SDK provided methods.

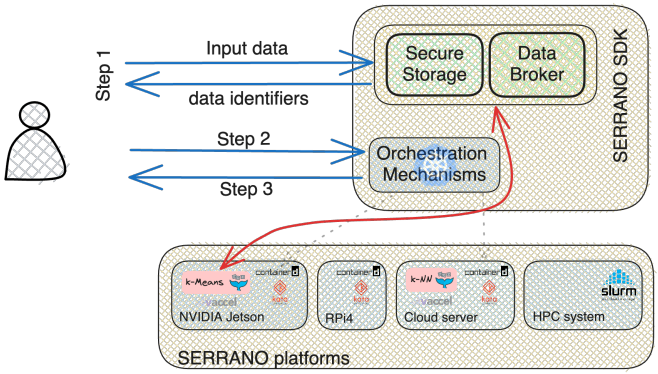


Fig. 2. Hardware-accelerated kernel execution from the end user's perspective, common approach for all supported modes and platforms

The SERRANO SDK provides appropriate APIs abstracting the interaction with the various SERRANO platform services to facilitate these operations. Through the SDK,

it is possible to push the required input data for kernel execution in the SERRANO platform. For instance, in Fig. 2, providing the data input is abstracted as a generic `pack_data()` function, which, in turn, is translated to `pack_data_ss()` or `pack_data_db()`, depending on the kind of data and the application requirements. All arguments are grouped into a generic structure, provided as `JSON` dicts, to facilitate portability. Then, the user requests the on-demand execution of the accelerated kernel using the `serrano_faas_kernel()` method with the appropriate arguments. Finally, the user retrieves results by invoking the `serrano_kernel_results()` method with the corresponding identifier from the first step as a parameter.

## V. vAccel: Hardware Acceleration Virtualization

The edge-cloud computing paradigm of application execution has favoured CPU-intensive applications, shifting the ownership of the hardware resources on which applications are being deployed to large providers (e.g., Google, Microsoft, Amazon, etc.). In these environments, applications run inside Virtual Machines or containerized systems providing increasing flexibility in resource allocation, security, and isolation for multi-tenant operation in the shared edge-cloud resources. However, neither of those systems can control the access to hardware acceleration devices with the same granularity or isolation guarantees as they can with other resources such as CPU, network, or storage.

The problem is exacerbated by the way we program hardware accelerators nowadays. Such devices typically provide hardware drivers and expose APIs to application developers. These APIs are device-specific, and sometimes they are incompatible even across devices of the same vendor. This has two significant side effects. On the one hand, user application implementations end up being device-specific, hindering portability and programmability, whereas, on the other hand, the lack of uniform APIs across devices renders it extremely difficult to virtualize them abstractly and efficiently.

In SERRANO, we use and extend vAccel [26] framework that enables virtualized workloads to access hardware accelerators securely and efficiently while running on environments that do not have direct access to acceleration devices.

The core idea of vAccel is decoupling the user application from hardware-specific code. To this end, it exposes to a user applications functions that can be accelerated. The hardware-specific code implementing these functions for a particular hardware accelerator device is provided as a *plugin*, loaded at runtime. This way, the development of hardware-independent applications is enabled, logically separating an application into two parts (i) the user code, which is part of the application logic itself, and (ii) the hardware-specific code, which is part of the application that runs on a hardware accelerator. At the same time, vAccel's modular design (Fig. 3) eliminates user code running on shared accelerators. Only the code in a vAccel plugin executes on the hardware accelerator, reducing the effective attack surface. Moreover, vAccel enables hardware acceleration within virtualized guests by employing an efficient API remoting approach at the granularity of function calls to delegate "accelerate-able code" to the host system.

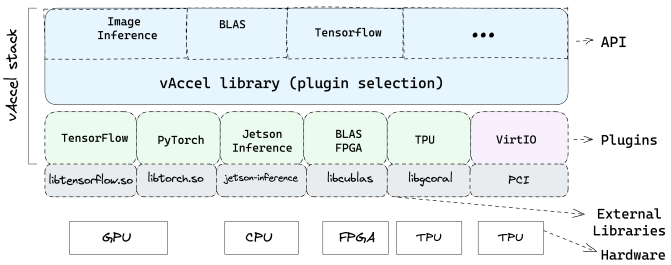


Fig. 3. High-level view of the modular vAccel software stack

To facilitate the deployment of vAccel-enabled applications, we integrate vAccel into a popular container runtime, Kata Containers [25]. Kata Containers enable containers to be executed in a Virtual Machine sandbox. They are as light and fast as containers and integrate with the container management layers while also delivering the security advantages of VMs. vAccel offers bindings for C, Python, and Rust. In SERRANO, the serverless implementation for all kernels uses the vAccel Python bindings. Additionally, we have implemented a subset of Tensorflow and PyTorch APIs so that the user can execute an application written for those frameworks over vAccel with minimal and/or no changes.

One of the key merits of the vAccel framework is the fact that users write their code using the vAccel API and the underlying plugin executes this code in the respective accelerator device. This enables hardware interoperability as the user does not need to rewrite, or even re-compile their code if they want to run on a different hardware accelerator. This greatly facilitates the scaling of hardware-accelerated applications across the cloud-edge continuum, as the user builds a container image with their vAccel API code, deploy it in the SERRANO platform and this code can use hardware accelerators in the Cloud (Generic, NVIDIA GPUs), at the Edge (Jetson GPUs, Orin/Xavier/Nano), or even CPUs when there is no hardware accelerator available (eg. on a RPi4).

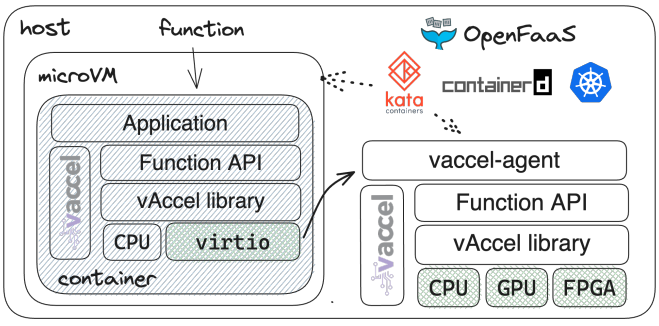


Fig. 4. Example of a Serverless function spawned in the SERRANO platform, with OpenFaaS, containerd, kata-containers and vAccel

## VI. Hardware Accelerated Kernels

SERRANO has identified computationally intensive functions [27] that can be executed much faster, utilizing their inherent parallelism and implementing them in hardware accelerators such as FPGAs and GPUs. Compared to traditional software implementations running on CPUs, these implementations can provide significant performance improvements and power efficiency gains. The accelerated kernels include the Savitzky-Golay filter, the Discrete-Time Wavelet Transform, and the K-Nearest Neighbor algorithm.

The Savitzky-Golay (SAVGOL) filter is a powerful tool in digital signal processing utilised for smoothing experimental data sets and reducing signal noise. By applying polynomial functions and considering neighbouring data points, it effectively removes high-frequency components from signals while preserving their overall shape and features. Different accelerators for the execution of the Savitzky-Golay filter were implemented for its deployment at cloud and edge FPGA and GPU devices, along with a design methodology [28].

The K-Means is a clustering algorithm that can classify time series signals. The basic idea of K-Means is to group data points into a specified number of clusters based on their similarity. The signals are classified into clusters that are close to each other. Typically, the Euclidean metric is used to measure distance in K-Means. However, we use Dynamic Time Warping (DTW) as it offers greater flexibility when matching time series signals with varying shapes, lengths, and alignments. Different accelerators were implemented for its deployment at cloud and edge FPGA and GPU devices.

The K-Nearest Neighbor (k-NN) algorithm is a supervised machine learning technique used for classification, with applications in image processing and generative models. In SERRANO, k-NN is employed to classify time series signals based on labelled training datasets. The k-NN method for time series signals calculates the distance between the inference and training signals using the Dynamic Time Warping (DTW) metric and identifies the k-nearest neighbors. The majority label of these k-nearest neighbors determines the class label of the inference signal. Different accelerators were implemented for its deployment at cloud and edge FPGA and GPU devices.

## VII. Evaluation Experiments

### A. Kernels and vAccel

To evaluate the overhead imposed by integrating the components mentioned in the previous sections, we performed an initial evaluation on a Jetson Xavier AGX system (CPU and GPU execution). We measured the execution time of individual kernel execution as well as end-to-end function execution, with input provided by the partners that developed the kernels.

Fig. 5 illustrates the absolute execution time (in ms) for the GPU version of the three studied kernels: k-NN, k-Means, and SAVGOL. The blue bars represent the execution time of the stock kernels provided by the partners, and the red vAccel-ported ones. We can observe that running the kernels through vAccel on the same host imposes negligible overhead.

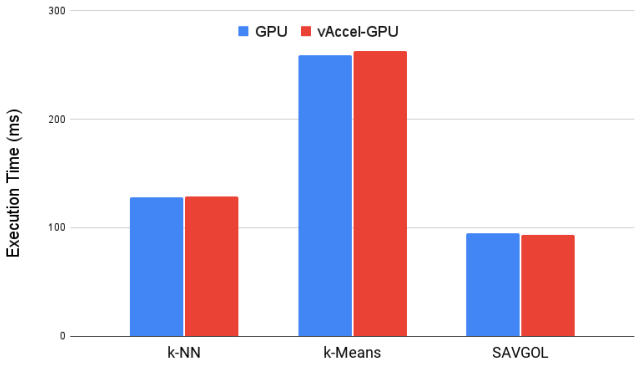


Fig. 5. vAccel - GPU execution

Fig. 6 shows the normalised execution time of the k-Means kernel to the respective native execution when running on a microVM, deployed using the process described in Section V. Specifically, we perform this experiment using two versions of the time-series data, `k-Means-110` and `k-Means-770`. The difference lies in the time spent processing each case (either on the CPU or the GPU). For instance, the execution time of `k-Means-110` on the CPU is almost 4.340 s per iteration, whereas, on the GPU, it is less than 160 ms. The same stands for `k-Means-770`: on the CPU, each iteration takes almost 30 s, whereas on, the GPU, each iteration takes almost 1 s.

Moreover, Fig. 6 shows the overhead associated with vAccel execution for the k-Means algorithm: for the GPU execution (red bars), where time spent processing is not significantly much, vAccel exhibits high overhead (almost 25% for `k-Means-110` and 8% for `k-Means-110`). For CPU execution, vAccel exhibits less than 3% overhead, as time spent during processing alleviates transport and control overheads.

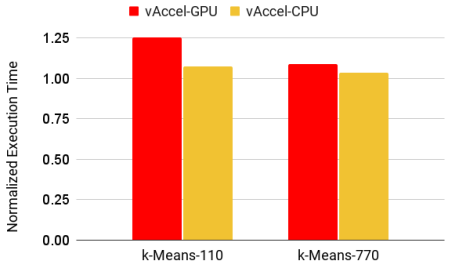


Fig. 6. vAccel - GPU execution

### B. SERRANO Kernels to Serverless Functions

Fig. 7 provides a detailed breakdown of the end-to-end execution time (in ms) for k-NN when executed as a FaaS function. The vertical axis shows the execution time of each step, normalized to the respective CPU-only execution (representing a generic serverless function without vAccel or

acceleration functionality, e.g., CPU only). On the horizontal axis, each step is depicted, whereas the bars (red and orange) represent the GPU and CPU execution when using vAccel[1].

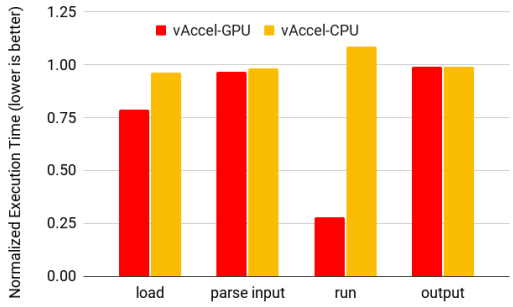


Fig. 7. FaaS Execution

The breakdown of the time spent includes the following basic steps during the function execution:

- instantiating the software stack (load binaries, libraries etc.) (`load`)
- fetching and parsing the input (`parse input`)
- running the compute-intensive operation (`run`)
- pushing the output to the relevant storage backend (`output`)

Fig. 7 shows that compared to the current approach of serverless computing, where each function is instantiated as an isolated microVM instance (e.g., AWS lambda) without access to acceleration functionality, our approach presents significant benefits with almost a 75% reduction in execution time compared to a non-accelerated function instance.

## VIII. Conclusions

In this paper, we present our work on enhancing the Cloud- Edge continuum by enabling interoperable hardware acceleration for workloads deployed in such environments. We present our design of the SERRANO architecture, focusing on hardware acceleration merits brought into the context of serverless computing in the Cloud-Edge continuum. We focus on optimizing the end-to-end execution process, enhancing the SERRANO SDK and deployment mechanisms with additional acceleration options. Our work also includes optimizations in the low-level parts of the software stack to facilitate the seamless execution of hardware-accelerated kernels across the continuum and achieve even faster execution times for compute-intensive workloads from numerous demanding use cases.

## Acknowledgment

The work presented in this paper is supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. 101017168 in the context of the SERRANO project.

[1]CPU execution over vAccel is only presented as an example. The execution flow for this operation should be identical to the generic CPU case, with the only difference being that the actual CPU process for the computation is running on the host and not on the container/microVM)

## References

[1] I. Baldini, et al., "Serverless computing: Current trends and open problems." Research advances in cloud computing (2017): 1-20.
[2] M. Shahrad, et al., "Architectural implications of function-as-a-service computing", Annual IEEE/ACM international symposium on microarchitecture, 2019.
[3] A. Kretsis, et al., "SERRANO: transparent application deployment in a secure, accelerated and cognitive cloud continuum", IEEE International Mediterranean Conference on Communications and Networking, 2021.
[4] Amazon Firecracker Github Issues, https://github.com/firecracker-microvm/firecracker/issues/1179
[5] NVIDIA, Embedded Systems for Next-Generation Autonomous Machines, 2020, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[6] CVE-2019-5736, Available from MITRE, CVE-ID CVE-2019-5736, 2019, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5736
[7] CVE-2019-14271, Available from MITRE, CVE-ID CVE-2019-14271, 2019, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14271
[8] M. Jagielski, et al, "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning", IEEE Symposium on Security and Privacy, 2018
[9] AI Data poisoning attack: Manipulating game AI of Go, Junli Shen and Maocai Xia, 2007.11820
[10] C.-H. Hong, et al., "GPU Virtualization and Scheduling Methods: A Comprehensive Survey", ACM Comput. Surv., vol. 3, no. 50, 2017.
[11] Y. Suzuki, et al., "GPUvm: GPU Virtualization at the Hypervisor", IEEE Transactions on Computers, vol. 65, no. 9, 2016.
[12] D. Vasilas, et al., "VGVM: Efficient GPU capabilities in virtual machines", International Conference on High Performance Computing and Simulation, 2016.
[13] W. Wang, et al., "pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment", International Conference on Hardware/Software Codesign and System Synthesis, 2013.
[14] H. A. Lagar-Cavilla, et al., "VMM-independent graphics acceleration", International Conference on Virtual execution environments, 2007.
[15] J. Hansen, "Blink: Advanced display multiplexing for virtualized applications", 2007.
[16] J. Duato, et al., "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters", International Conference on High Performance Computing and Simulation, 2010.
[17] L. Shi, et al., "vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines", IEEE Transactions on Computers, vol. 61, no. 6, 2012.
[18] H. Yu, et al., "AvA: Accelerated Virtualization of Accelerators", International Conference on Architectural Support for Programming Languages and Operating Systems, 2020.
[19] A. Palade, et al., "An evaluation of open source serverless computing frameworks support at the edge", IEEE World Congress on Services (SERVICES) (Vol. 2642, pp. 206-211), 2019.
[20] M. Bacis, et al., "BlastFunction: an FPGA-as-a-service system for accelerated serverless computing", IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE), 2020.
[21] A. Jindal, et al., "Poster: Function delivery network: Extending serverless to heterogeneous computing", IEEE International conference on distributed computing systems (ICDCS), 2021.
[22] L. Patterson, et al., "HiveMind: a hardware-software system stack for serverless edge swarms", Annual International Symposium on Computer Architecture, 2022.
[23] S. Choi, et al., "$\lambda$-nic: Interactive serverless compute on programmable smartnics", IEEE International Conference on Distributed Computing Systems (ICDCS), 2020.
[24] D. Du, et al., "Serverless computing on heterogeneous computers", ACM International conference on architectural support for programming languages and operating systems, 2022.
[25] The speed of containers, the security of VMs, https://katacontainers.io/
[26] Hardware Acceleration for Serverless Computing: https://vaccel.org
[27] SERRANO D4.2, https://ict-serrano.eu/wp-content/uploads/2022/08/D4.1-HWSW-IPs-for-workload-acceleration-in-disaggregated-DCs.pdf
[28] I.Oroutzoglou, et al., "Optimizing Savitzky-Golay Filter on GPU and FPGA Accelerators for Financial Applications". 11th International Conference on Modern Circuits and Systems Technologies (MOCAST) 2022