OpenCV hardware acceleration with vAccel

Maria Goutha, Ilias Lagomatis, Anastassios Nanos
{mgouth,ilago,ananos}@nubis-pc.eu
NUBIS PC
Greece

Alexandros Patras
patras@uth.gr
University of Thessaly
Greece

Abstract

The rise of data from low-power devices has increased the demand for Edge processing capabilities; specialized processing units and general-purpose hardware accelerators offer notable performance gains for Cloud workloads. Virtualization techniques are employed for user and data isolation, but sharing generic hardware accelerators among untrusted tenants presents challenges. vAccel addresses these challenges by decoupling function calls from hardware-specific implementations through plugins. In this work, we present preliminary findings from the integration of OpenCV to vAccel , initial evaluation results, and discussions on challenges and future steps.

1 Introduction & Motivation

Fine-tuning the execution of a Computer Vision (CV) application presents a number of challenges, mostly associated with I/O arguments and the underlying systems stack that supports it. The benefits from being able to customize the execution of a multi-component CV application are numerous, mostly related to *Performance Optimization*, *Scalability*, *Adaptability to Hardware Constraints*, *Customization*, *Ease of Development* and *Future-proofing*.

The ability to switch between GPU and CPU implementations [1] based on real-time requirements can significantly improve overall execution speed. For tasks that demand quick processing, utilizing the parallel processing power of GPUs can lead to substantial time savings. vAccel [1] adapts to the available hardware resources, ensuring that applications can run on a variety of devices with different capabilities. Developers can fine-tune the execution based on the specific needs of their applications. For instance, tasks that can tolerate lower accuracy might prioritize faster execution, while others might require a more accurate but slower approach.

vAccel decouples the process of building applications with accelerated functions from the implementation of the accelerator code. Applications based on vAccel are compatible with any existing or future acceleration framework. vAccel 's user-facing, user-extendable API is not tied to any vendor/hardware-specific, low-level or high-level software stack. By embedding accelerator code into *plugins*, an application using vAccel can select the necessary implementation at runtime, without any source modifications. Additionally, vAccel provides a modular architecture to transport data in virtualized or remote environments. This design simplifies application-execution paths and, thus, increases execution security by reducing the attack surface, while at the same time ensures minimal footprint for each tenant.

[1] https://docs.vaccel.org

2 vAccel OpenCV Bindings

We introduce language specific bindings, tailored to OpenCV unique function definitions and arguments. In addition, we provide helper functions to facilitate the serialization / deserialization of non-scalar arguments. In the specific case of OpenCV, due to its C++ base, we need to directly link existing applications with the external bindings library. As a result, minimal code changes are required[2]. We plan to investigate this further, so that we can overload OpenCV functions at runtime, without any user-code modifications.

3 Use Case: Obstacle Avoidance

The fundamental idea behind obstacle avoidance involves assessing whether an object's distance from a sensor, such as a stereo camera or ultrasonic sensor, is closer than a predefined threshold. To build an obstacle avoidance system with a stereo camera, the process involves obtaining a depth map, segmenting regions based on a depth threshold, detecting contours, identifying the largest contour and, finally, calculating the average depth [2].

4 Initial Results & Plan

Initial results seem promising, allowing workloads confined in AWS Firecracker microVMs to enjoy hardware acceleration (≈ 60% better performance), while adding ≈ 30% overhead compared to native execution for CPU-only cases (related to transport and SerDe[3] overheads). The challenges we face moving forward include minimizing the SerDe overheads, optimizing the transport layer and generalizing the approach to include diverse OpenCV-related workflows.

References

[1] Batuhan Hangün and Önder Eyecioglu. 2019. Performance Comparison Between OpenCV Built in CPU and GPU Functions on Image Processing Operations. *CoRR* abs/1906.08819 (2019). arXiv:1906.08819 http://arxiv.org/abs/1906.08819

[2] Jianying Yuan, Tao Jiang, Xi He, Sidong Wu, Jiajia Liu, and Dequan Guo. 2023. Dynamic obstacle detection method based on U–V disparity and residual optical flow for autonomous driving. *Scientific Reports* 13 (05 2023). https://doi.org/10.1038/s41598-023-34777-6

[2] an `extern "C"` definition of the function prototype – no user logic change

[3] Serializer/Deserializer