|  |
| --- |
| Interception Inception: Analyzing CUDA call interception methods for Serverless Environments**Manos Pavlidakis**1#\*, Anargyros Argyros1,2, Stelios Mavridis1, Giorgos Vasiliadis1,3, and Angelos Bilas1,21 Institute of Computer Science (ICS), Foundation for Research and Technology - Hellas (FORTH), Greece2 Computer Science Department, University of Crete, Greece3 Department of Management Science & Technology, Hellenic Mediterranean University, Greece# Presenting author: Manos Pavlidakis, email: manos\_pavl@ics.forth.gr\* Corresponding author: Manos Pavlidakis, email: manos\_pavl@ics.forth.gr |

abstract

GPUs are necessary for accelerating applications. However they often remain underutilized [1] because solo executions cannot always fully utilize their resources. To share a GPU to multiple applications, previous work try to virtualize accelerators [3] or to provide Infrastructure as a Service (IaaS) [2]. Other virtualization techniques, such as full- and para virtualization, have limited applicability due to the requirement for custom drivers. API remoting [2] is the only stable and efficient technique for accelerator abstraction. Existing API remoting approaches [2, 3] intercept partially the CUDA driver and runtime API, as well as the high-level calls of CUDA accelerated libraries (e.g., cuBLAS and cuDNN).

This three-level interception approach requires handling more than 2000 calls though, most of which have complex semantics that change often. For example, just to support cuBLAS, cuRAND, and cuFFT, we need to handle more than 1600 high-level calls, a process that is usually performed offline by hand. As a consequence, previous work [2, 3] offer limited support for complex frameworks, such as PyTorch, and Tensorflow. To make matters worse, high level calls of CUDA accelerated libraries (e.g., cublasIsamax) perform implicit CUDA calls that are hidden from the developer. These calls execute both host and device code, which does not scale in client-server setups where the CPUs are not designed to perform computations.

In this work, we propose CUInterposer, a fine-grain interception mechanism at the CUDA driver and runtime library. The design of CUInterposer provides a clear boundary between CPU and GPU code, that allows to fully support popular frameworks, such as PyTorch. More specifically, CUInterposer intercepts the whole CUDA driver and runtime library that consists of 400 relatively simple calls. Due to the simplicity of these calls, the interception process is automated, leading to zero manual effort as opposed to previous works. Finally, we can distinguish host and device calls because our approach intercepts the implicit calls performed from closed-source high-level function calls of CUDA accelerated libraries. As a result, with CUInterposer, only the device code is forwarded to the server, whereas the host code is executed in the client.

**REFERENCES**

[1] Rachata Ausavarungnirun, Vance Miller, Joshua Landgraf, Saugata Ghose, Jayneel Gandhi, Adwait Jog, Christopher J. Rossbach, and Onur Mutlu. 2018. MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency. In ASPLOS ’18.

[2] Jose Duato, Antonio J. Pena, Federico Silla, Juan C. Fernandez, Rafael Mayo, and Enrique S. Quintana-Orti. 2011. Enabling CUDA acceleration within virtual machines using rCUDA. In HiPC ’11.

[3] Niklas Eiling, Jonas Baude, Stefan Lankes, and Antonello Monti. 2022. Cricket: A virtualization layer for distributed execution of CUDA applications with checkpoint/restart support. In Concurrency and Computation: Practice and Experience.