MIGnificient: Fast, Isolated, and GPU-Enabled Serverless Functions

1st Marcin Copik *ETH Zürich*
2nd Alexandru Calotoiu *ETH Zürich*
3rd Pengyu Zhou *University of Toronto*
4th Lukas Tobler *AYES*
5th Torsten Hoefler *ETH Zürich*

I. Introduction

The security of High-Performance Computing is becoming more important with new applications in machine learning and medical data processing. At the same time, the convergence of HPC and cloud computing brings a demand for workload co-location and resource sharing. Instead of providing security guarantees through exclusive resource locations and physical isolation, HPC systems must offer new methods that retain high utilization. These have to include GPUs that have become essential in HPC systems: accelerators are used by 68% of the first 50 systems on the TOP500 list. However, GPUs are often underutilized, as even workloads like machine learning training spend a significant fraction of their time on communication and CPU tasks. The growing capabilities of each new GPU generation make it even more difficult to saturate the device with a single application. These devices could be shared in Function-as-a-Service (FaaS), a new cloud programming model designed around fine-grained functions. There, functions execute on resources assigned by the load balancer, allowing system operators to boost utilization through dynamic scheduling. While traditional functions use containers and microVMs to share CPUs, they need a new model to securely and efficiently co-locate computations on GPUs.

II. Why a New Approach?

While functions can be co-located on NVIDIA GPUs through time sharing and software-based spatial sharing, both methods are ill-suited for serverless workloads. On NVIDIA GPUs, CUDA runtime allows two processes to share a GPU through **time slicing**. Two processes have isolated memory address spaces and kernel co-location is not possible: GPU runtime switches users on the device and uses compute preemption to evict GPU context. The frequent context switches add performance overheads, and functions can overallocate memory. **NVIDIA Multi-Processing Service (MPS)** provides spatial sharing through a dedicated server that implements resource multiplexing, limiting memory allocation and use of computing resources. MPS has been commonly used to share a GPU by workloads from a single user, but it is not designed for multi-tenant applications. Thus, this solution is ill-suited for serverless functions: concurrent execution creates an opportunity for security attacks, and MPS offers no performance isolation. Furthermore, it lacks error containment - a GPU error caused by a single kernel forces all co-located functions to restart from scratch or implement complex checkpoint/restart mechanisms.

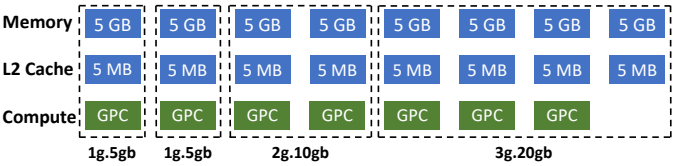

Fig. 1. An example of partitioning the A100 GPU into four independent slices: 3g, 2g, 1g, and 1g.

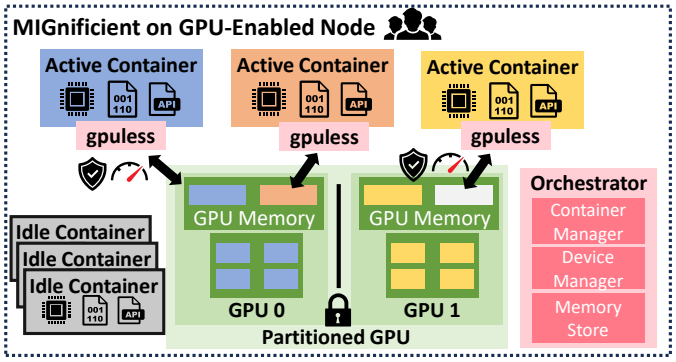

Fig. 2. MIGnificient design: multi-tenant (👥) serverless functions share a GPU device with fault tolerance (🛡), high performance (🏎), and security (🔒).

Instead, we turn to **NVIDIA Multi-Instance GPU (MIG)**, where the entire device is composed of seven computing units, each one with independent caches and memory systems (Fig. 1). However, to achieve high utilization, a single static device partition should be securely shared between functions.

III. MIGnificient Design

We propose a new approach that combines the elasticity of Function-as-a-Service (FaaS) with the isolation of physical GPU partitioning. In **MIGnificient**, we provide spatial isolation through concurrent execution on different device partitions, preventing side-channel attacks and performance interference (Fig. 2). We limit the use of computing resources of a single GPU partition to one function at a time, while permitting CPU phases and data transfers of functions to execute concurrently. We employ **local API remoting** to control kernel invocations and memory transfers, preventing memory overuse and improving resource management in virtualized API. Thanks to the co-location of CPU and GPU parts on the same machine and using shared memory for communication,

MIGnificient is not limited by network performance and does not rely on insecure NVIDIA MPS.

To fully utilize the device, we propose faster function switching on a single MIG partition: we permit overlapping CPU tasks and memory transfers while forbidding simultaneous kernel execution. Thus, functions receive exclusive access to GPU computing without performance interference but do not occupy expensive hardware when not needed.

MIGnificient provides a unified model for optimized serverless GPU functions. Through virtualization of API calls, the system could support swapping GPU pages (*memory store*) and load balancing between MIG partitions through migration (*device manager*). MIGnificient is independent of the CPU sandbox type, as API remoting removes the need for direct GPU access. We expect the device partitioning trend to stay in the future, as the devices grow in size. MIG is supported in the Grace Hopper and upcoming Blackwell architectures [1], and Intel Data Center Max GPUs can be divided programmatically into independent sub-devices [2].

IV. Evaluation

Function Time Distribution First, we execute Rodinia benchmarks on a V100 GPU and measure the total distribution of time spent across CPU and GPU parts. These estimate the improvements that can be obtained through fast switching. We split the runtime into active GPU usage through kernels and memory transfers, and compute the total GPU time, which includes memory allocations. While some applications can spend up to 25% of their time actively using GPU, many spend most of their time on the CPU and I/O operations. While initial CUDA memory allocations can be a major source of overhead, function containers can avoid this problem by reusing warm resources.

Local API Remoting To offer competitive performance, the shared memory-based API remoting of MIGnificient needs to execute calls to the CUDA API with negligible overhead. We evaluate the performance of memory copying as data movement tends to be the denominating bottleneck of GPU-based applications. We execute a pair of host-to-device and device-to-host CUDA copy operations on MIG partitions 1g and 7g of A100 GPU. We compare the native CUDA execution against MIGnificient when using `mignificient_malloc` to allocate host-based pages in shared memory. The comparison against MIGnificient shows increased overhead when using a container, motivating the need for containers customized for HPC environments, as proposed previously in the XaaS model [3].

References

[1] NVIDIA, "Nvidia multi-instance gpu," 2024, MIG in Blackwell GPUs, Accessed: 2024-03-23. [Online]. Available: https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[2] Intel, "Multi-stack gpu architecture," 2024, oneAPI GPU Optimization Guide, Accessed: 2024-03-23. [Online]. Available: https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2024-0/multi-stack-gpu-architecture.html

[3] T. Hoefler, M. Copik, P. Beckman, A. Jones, I. Foster, M. Parashar, D. Reed, M. Troyer, T. Schulthess, D. Ernst, and J. Dongarra, "Xaas: Acceleration as a service to enable productive high-performance cloud computing," *Computing in Science and Engineering*, no. 01, pp. 1–11, apr 2024.