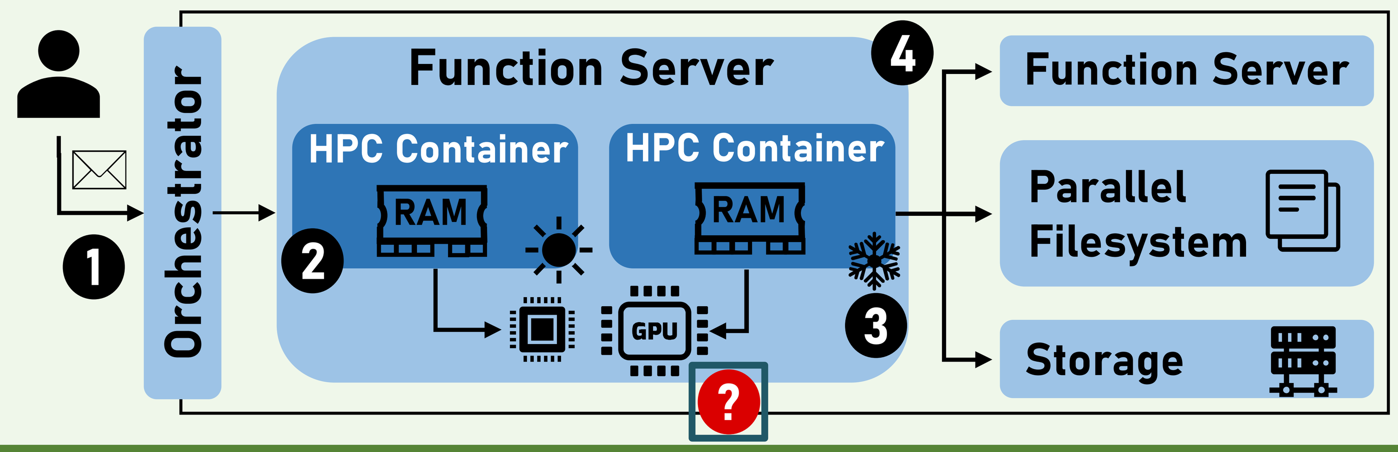
Marcin Copik, Alexandru Calotoiu, Pengyu Zhou, Lukas Tobler, Torsten Hoefler

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

SPCL

High-Performance Serverless: Tailoring Function-as-a-Service to HPC Needs



HPC serverless combines the high performance with cloud elasticity. While fine-grained functions have been improved with fast networks and filesystems, they lack GPU computing.

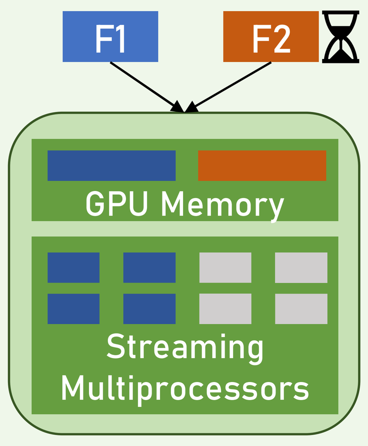
1. Function placement with microsecond-scale allocations.
2. Isolated execution with HPC-native containers, e.g, Sarus.
3. Reducing cold startups latency with warm containers.
4. Distributed communication and high-performance I/O.

? **How to efficiently share the GPU device between functions while keeping the multi-tenant isolation?**

Approaches to Sharing GPUs Between Co-located Functions

Time-Sharing

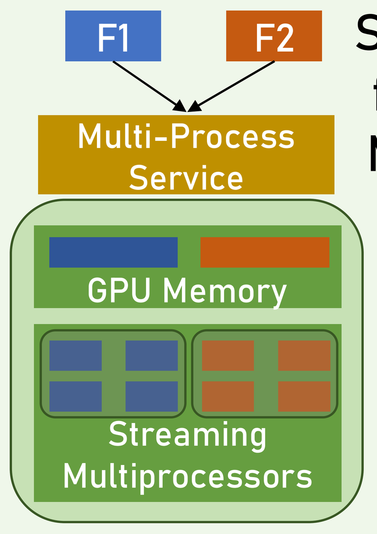
Device switches between kernels from different processes.



✓ Performance and security isolation.

✗ Wasted compute power due to lack of device sharing

Space-Sharing with Multiprocessing (NVIDIA MPS)



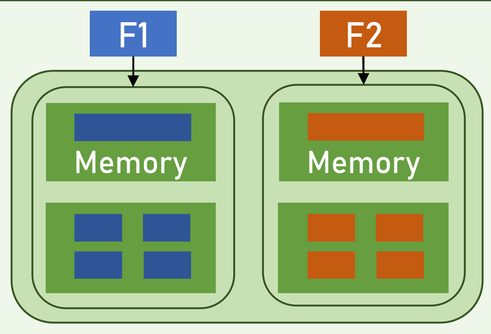
Spatial multiplexing of kernels from different applications in Multi-Process Service (MPS).

✓ Increased utilization and lower overhead of context switch

✗ No multitenancy, fault tolerance, and performance isolation

Device Partitioning (Virtualization, NVIDIA MIG)

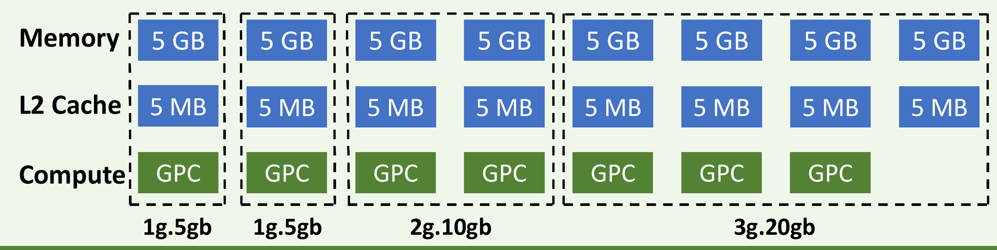
Many device instances with virtualization or physical partitioning in Multi-Instance GPU (MIG).



✓ Multi-tenancy and performance isolation.

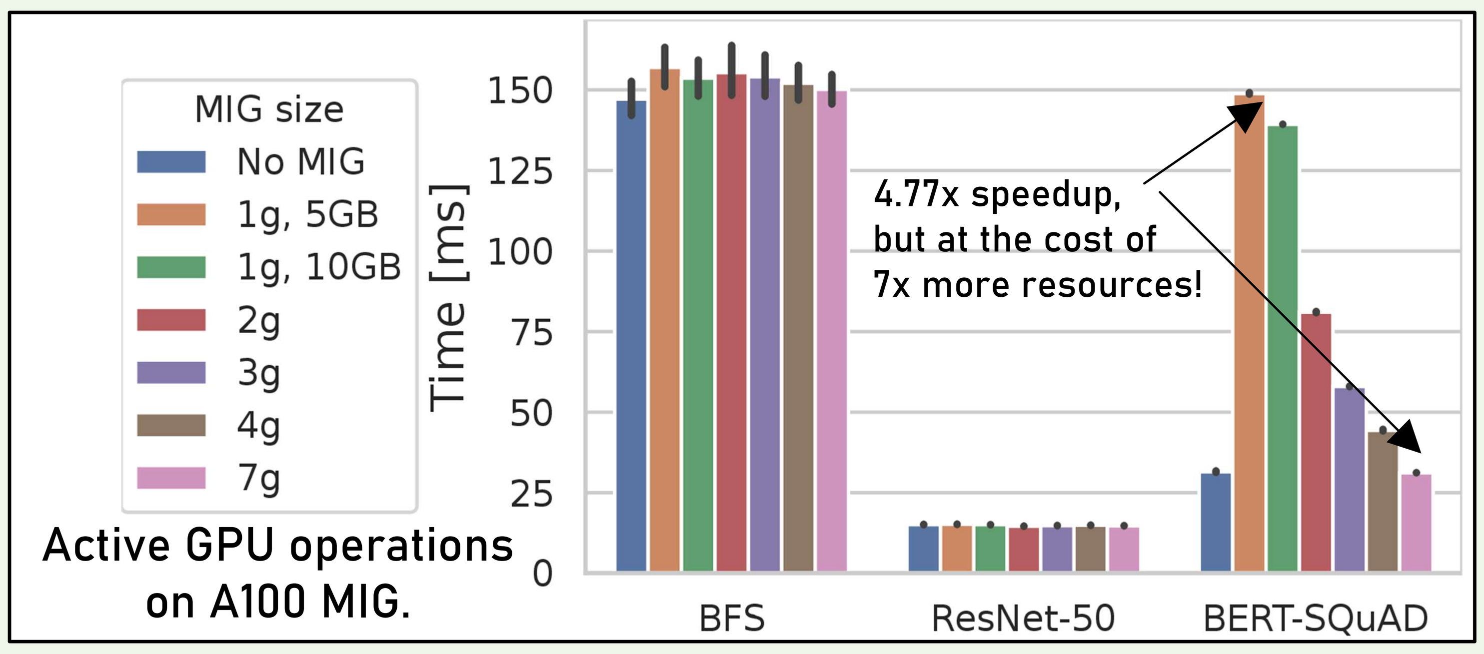
✗ Lower utilization when processes gain exclusive access.

Example MIG partitioning: four slices of sizes 1, 1, 2, and 3.



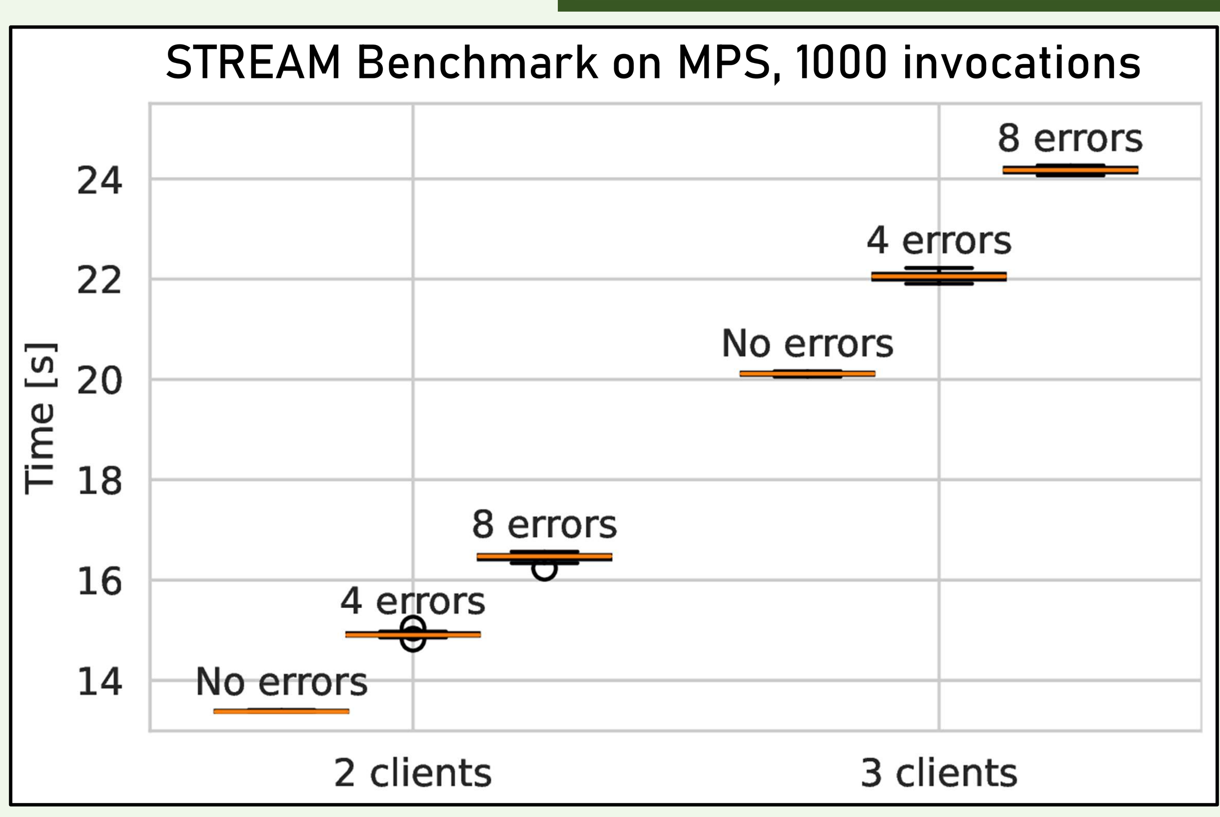
Why do we need to share GPUs?

GPUs are growing larger: from A100 to H100, memory and FLOPs increased by 2x and 2.6x. Fine-grained functions cannot fully utilize large device.



Active GPU operations on A100 MIG.

Why not use NVIDIA MPS?



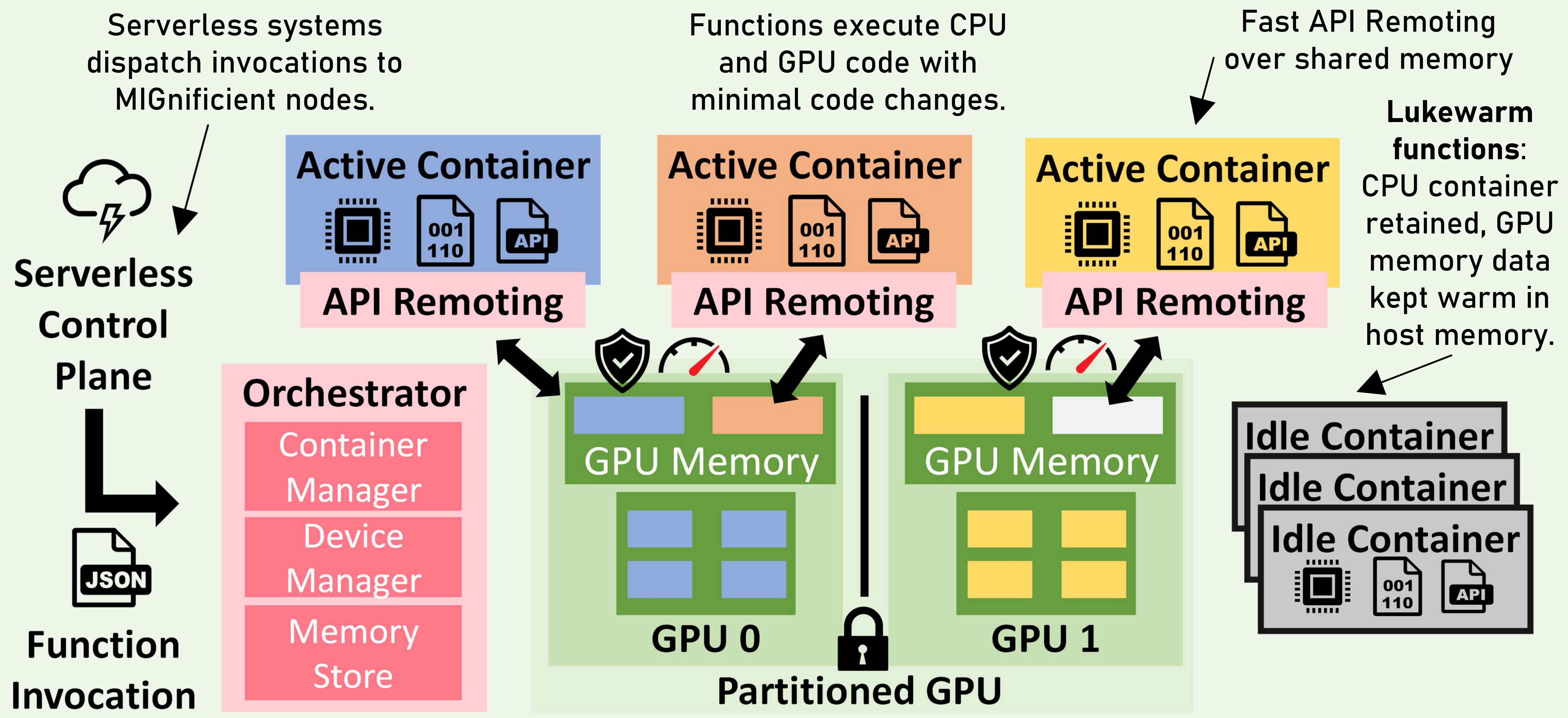
NVIDIA MPS is a common choice for GPU sharing, but it does not satisfy multi-tenancy requirements of serverless functions.

✗ MPS is not designed for multi-tenant workloads. Side-channel attacks are possible on co-located kernels.

✗ MPS has no performance isolation, and co-located kernels compete for memory bandwidth.

✗ MPS has no error containment: a GPU failure on a single tenant is propagated to everyone, requiring reinitialization of CUDA context and restart of kernels.

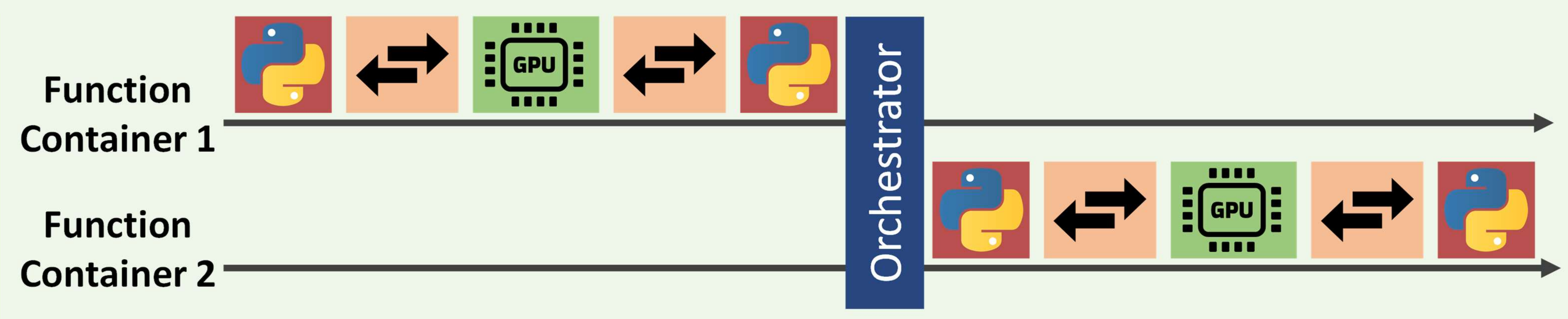
MIGnificient: Secure Functions on Partitioned GPUs



MIGnificient is a new GPU-accelerated serverless design that provides multi-tenant isolation and high performance. Functions are executed on the same node as GPU and isolated by CPU containers. MIG is used to partition the device into smaller units for high efficiency and concurrent execution. Thanks to local API remoting, we limit function access to the device, preventing side-channel attacks in concurrent execution.

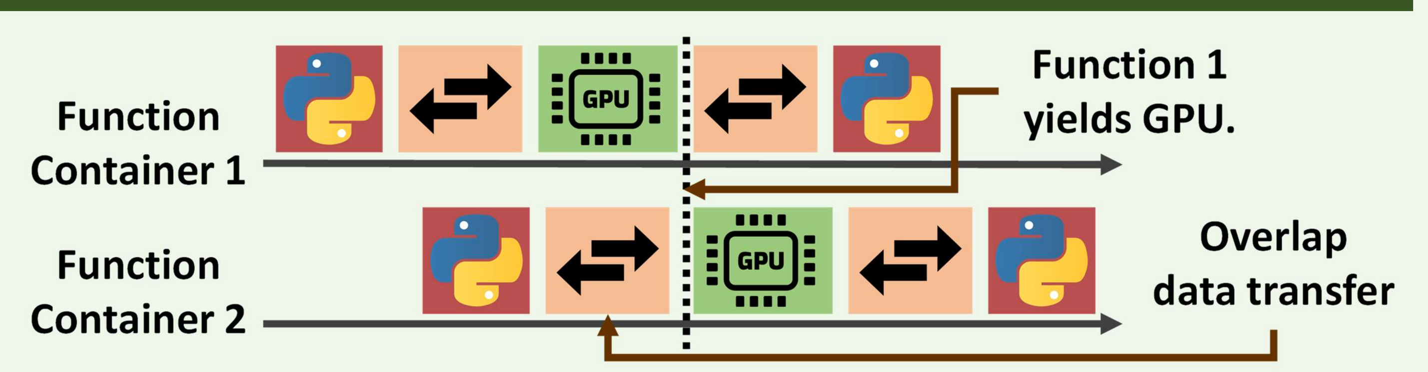
Fast Function Switching

Traditional Functions with Exclusive GPU Access



Functions have to process the request, transfer data to the device, and schedule GPU kernels. When using high-level languages like Python, they can spend more time on CPU than on GPU.

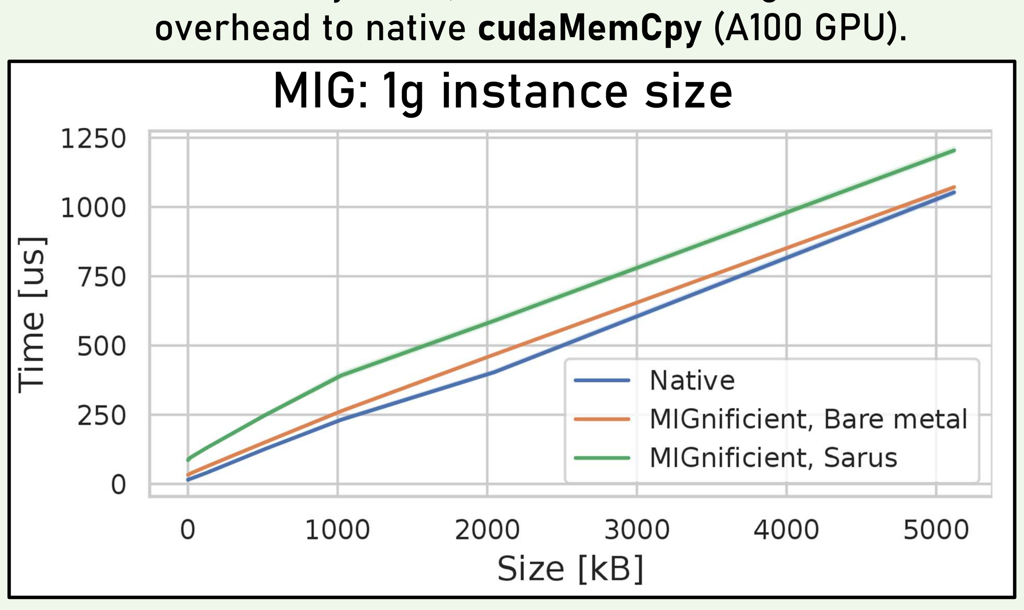
MIGnificient Functions with CPU and Data Movement Overlap



MIGnificient quickly transitions to the next tenant after the current one finishes GPU computations, and overlaps data transfers to increase utilization.

Data Movement in API Remoting

Data movement adds large overheads in network-based API remoting. We add **mignificient_malloc** to allocate host data in shared memory. Then, local API remoting adds minor overhead to native **cudaMemCpy** (A100 GPU).



CPU vs GPU Time Distribution

Rodinia applications spend time primarily on CPU and memory management (V100 GPU). By limiting the exclusive GPU access to kernels only, we can improve system throughput and GPU utilization.

Benchmark	Execution Time (s)	GPU Kernels & Memcpy (%)	All GPU Ops (%)
BFS	1.79	0.46	9.20
Gaussian	0.52	24.28	54.56
Hotspot	0.49	0.26	30.48
Pathfinder	0.32	1.65	30.29
srad_v1	0.21	3.67	75.77

Lukewarm Functions

To estimate benefits of lukewarm functions, we compare just the cost of initializing ML model in PyTorch with time of swapping a warm model data from host memory (RTX 4070 GPU). A cold container needs to additionally initialize CPU container with Python, CUDA context, and PyTorch.

	ResNet-50	BERT
Model Size	142 MB	1332 MB
Load Time	107 ms	730.2 ms
Swap Time	23.45 ms	214.47 ms

Fast Function Switching

We evaluated our system with 2 concurrent clients sending 10 requests to the same function. We deployed MIGnificient orchestrator with HTTP gateway and bare-metal executors on an RTX 4070 GPU. We compare our function switching approach against native CUDA execution (no isolation) and sequential execution (exclusive GPU access). With two clients, we increase throughput of isolated execution up to 1.9x.

Benchmark	Native	MIGnificient		
	Time Sharing	Sequential	Overlap CPU	Overlap CPU + Data Transfer
BFS	505.3 ± 2.5	990.5 ± 112	515.9 ± 18.2	528.8 ± 14.6
hotspot	92.1 ± 0.8	195.1 ± 22.2	102.9 ± 10.1	103 ± 9.9
ResNet-50	18 ± 0.3	53.3 ± 6	28.3 ± 1.9	27.5 ± 0.7
AlexNet	15.4 ± 0.5	49.2 ± 5.5	26.2 ± 0.7	26.4 ± 0.9
Vgg19	23.6 ± 1	54.5 ± 6.5	28.5 ± 1.2	27.8 ± 1
BERT-SQuaD	40.2 ± 2.5	65.8 ± 7.5	46.5 ± 3.9	41.4 ± 3.1