i am hpc.

# Fine-grained Accelerator Partitioning In Serverless Computing


Aditya Dhakal[1], Philipp Raith[1], Logan Ward[2], Rolando P. Hong Enriquez[1], Gourav Rattihalli[1]
Kyle Chard[2,3], Ian Foster[2,3] and Dejan Milojicic[1]

1 Hewlett Packard Labs
2 Argonne National Laboratory
3 University of Chicago


Hewlett Packard Enterprise

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

# Outline

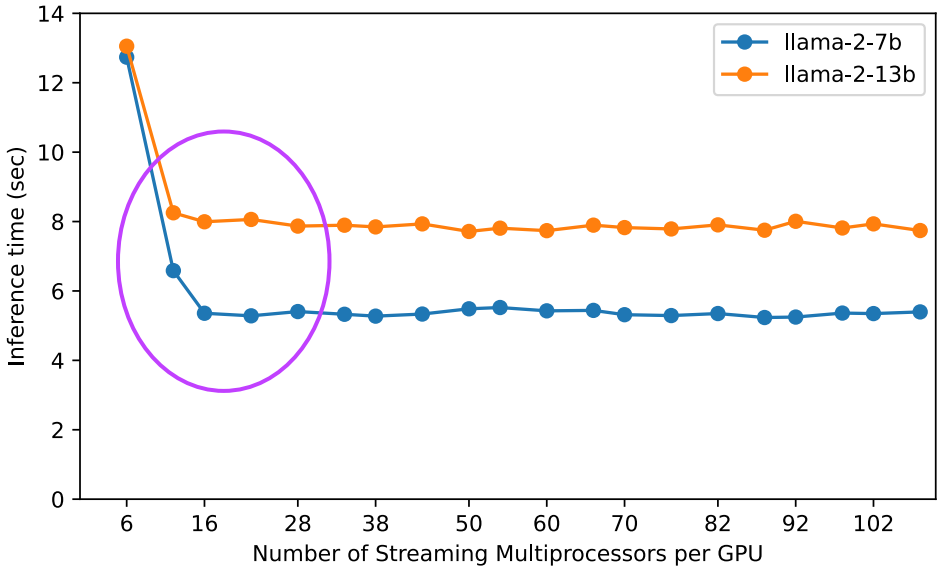
- Accelerators with Function as a Service (FaaS)
- GPU utilization of workflows
- GPU multiplexing techniques
- Parsl FaaS platform
- Experimental Results
- Conclusion

# Accelerators for Function-as-a-Service (FaaS)

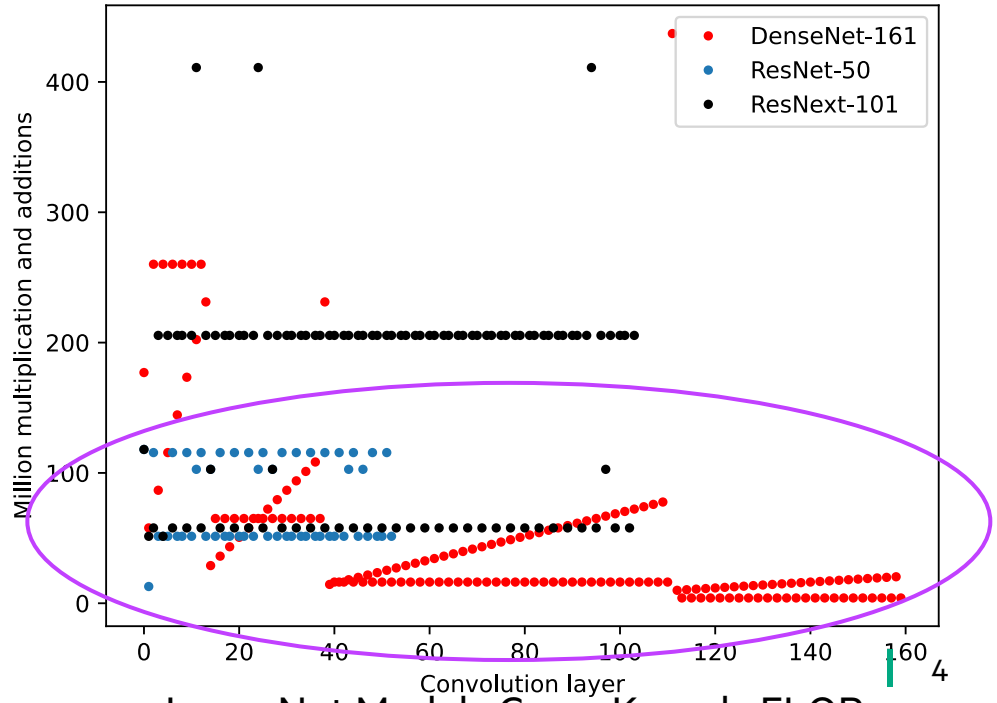
- FaaS functions: stateless and short-lived
  - Can be easily scaled-out when number of invocation increases
  - Resources released as invocation decrease
- FaaS can effectively utilize hardware when compute resources can be fine-grained partitioned
- Finer-grained accelerators partitioning is often not present in FaaS frameworks out of the box
  - Many FaaS systems allocate accelerators as a whole for a particular function
- We enable fine-grained GPU partitioning in a popular Parsl FaaS framework

# Low GPU Utilization of Some Applications

- Workflows do not fully utilize available GPU compute
  - Many kernels of a workflow are small
- LLaMa2 does not improve inference time of text completion when the GPU resources increase
- Some image classification models (convolutional DNNs) have few kernels that utilize a lot of compute
  - Most kernels for these DNN inference only require small amount of compute



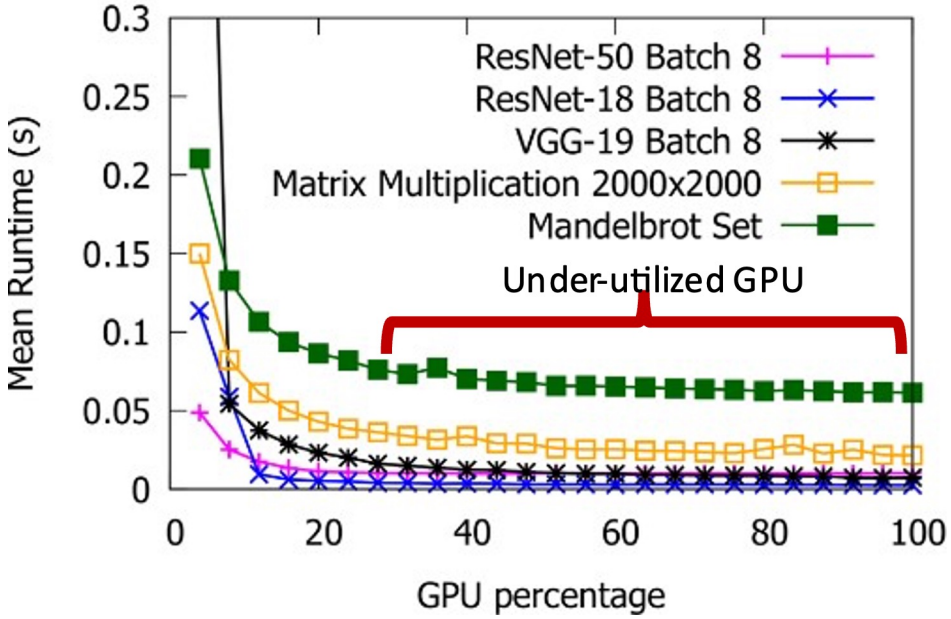

LLaMA2 runtime vs. GPU SM count



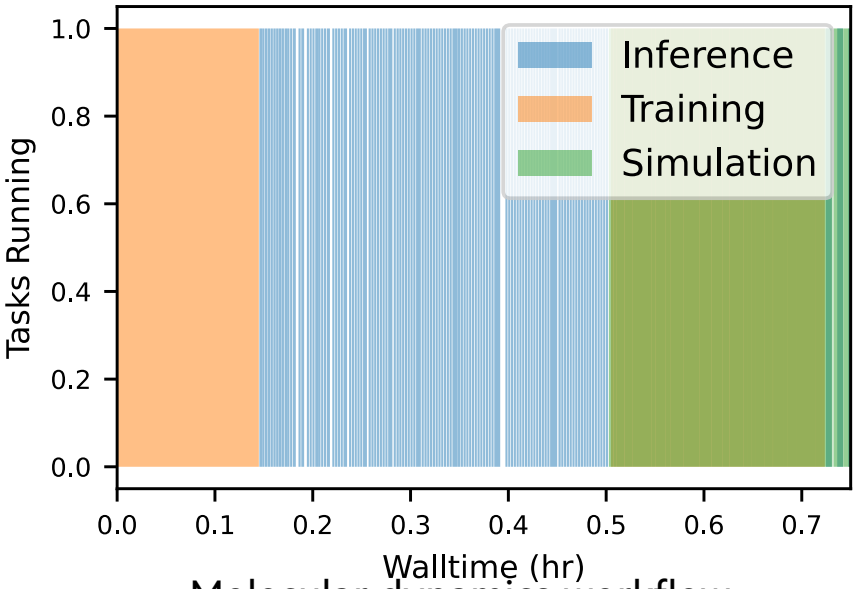

ImageNet Models Conv. Kernels FLOPs

# Lower Utilization in Other Applications

- We had similar “Low utilization” observation in other applications and DNN models




- Other form of underutilization
  - Workflows where accelerator is idle for large amount of time

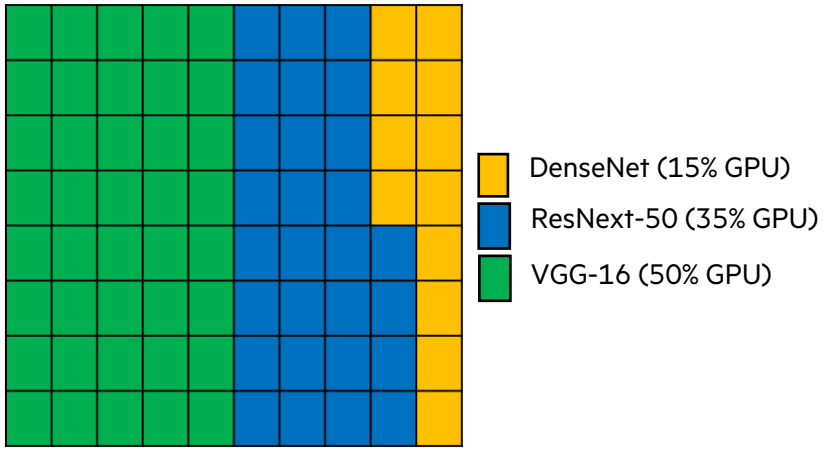



Molecular dynamics workflow

Training and Inference uses GPU but simulation does not

# Multiplexing the GPUs: Current Tools

- A solution to low GPU utilization is to run multiple workloads in GPU concurrently
  - Providing entire GPU for a single function is not cost-effective
- NVIDIA GPUs have Multi-process Service (MPS) that lets user partition GPUs
  - MPS allows user to fix maximum number of streaming multiprocessors
  - Users can choose GPU percentage metric (e.g. 50% of V100 means process will get 40 SMs)
- Timesharing: Default for NVIDIA GPUs.
  - multiple kernels running concurrently will time multiplex the GPU




Example of MPS dividing GPU SMs with MPS

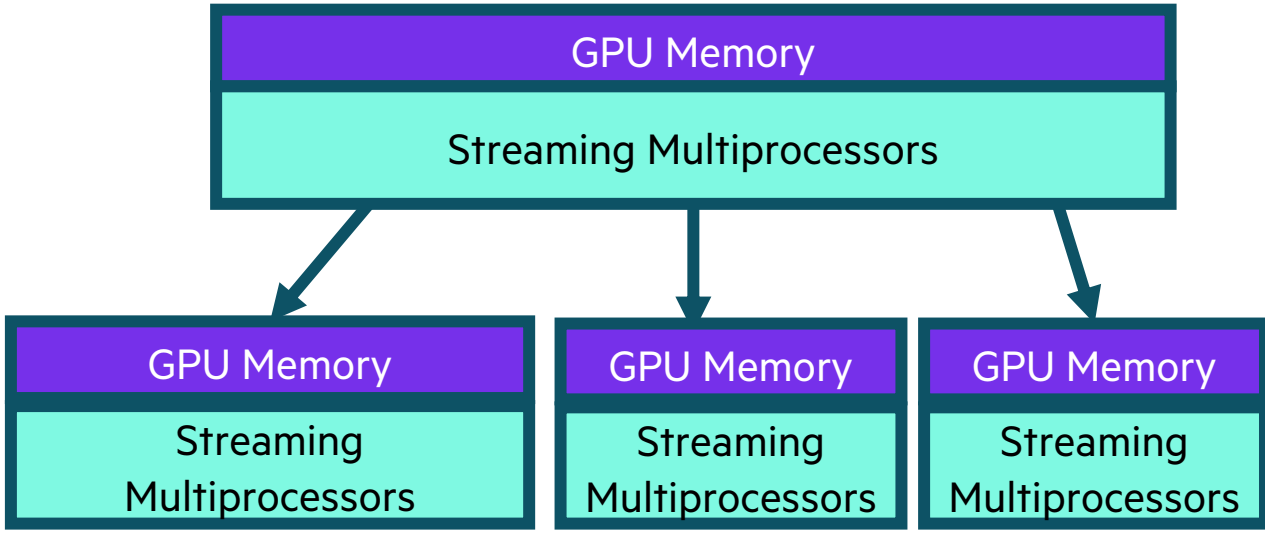
# NVIDIA Multi-Instance GPU

- Multi-instance GPU (MIG) creates pre-defined smaller instances of a GPU and provide isolation for multiple process to utilize GPU

- MIG divides both SMs and Memory of a GPU

| | H100 | A100 |
|---|---|---|
| Available MIG Instances | 7x 10GB<br>4x 20GB<br>2x 40GB<br>1x 80GB | 7x 10GB<br>3x 20GB<br>2x 40GB<br>1x 80GB |

MIG instances available for H100 and A100



# NVIDIA: MPS and MIG

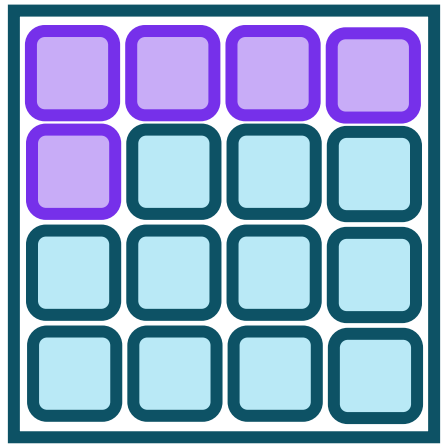
- MPS allows to partition the resources for a process before the process starts
- MIG is bit more static: Partitions are first allocated then applications can launch their kernels on each partition

- MPS does not provide memory isolation. All processes access single global memory
- MIG enforces memory isolation

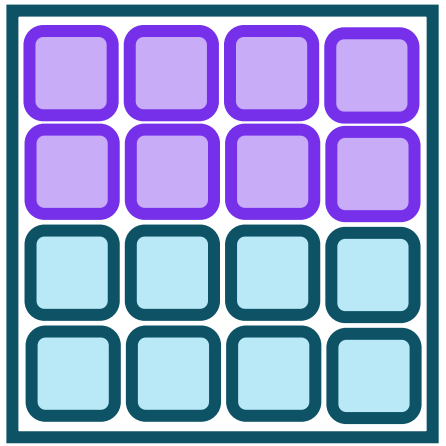
- MIG currently requires GPU reset to change allocations
- MPS does not require GPU reset

# AMD GPUs Multiplexing

- AMD GPUs can also limit application to a specified resource with Compute unit (CU)-masking approach
- With CU-masking a specific set of compute unit per shader can be allocated to each kernel
- A bitmask of CU per shader is provided to kernel



Shader 0: 111110000...0



Shader 1: 111111110...0

# Parsl

- Parallelization framework for python code
- Developed by University of Chicago and Argonne National Labs
- Scripts independent of execution environment
  - Can run in different environments (laptops, supercomputer) based on executors
- Wide variety of executors suitable for different kind of environment

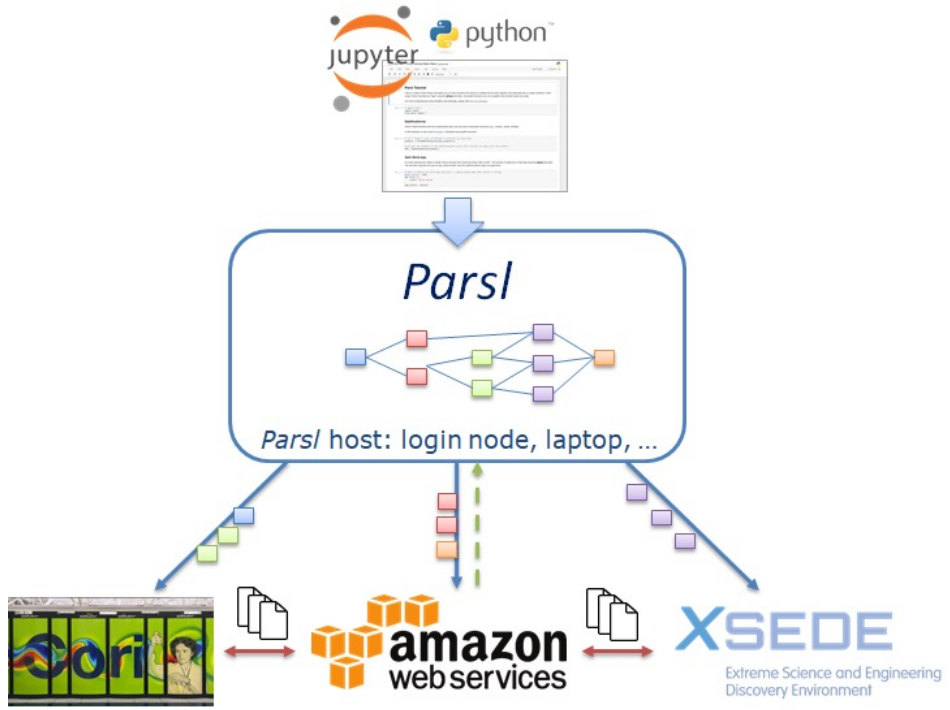




Fig source: https://parsl-project.org/

# Parsl: HighThroughputExecutor

- Parsl Executors are abstractions to represent available compute resources that can be used to process a task

- HighThrougputExecutor is executor designed for cluster-scale use
  - We use “local” provider for single node experiments

- HighThroughputExecutor uses multiprocessing based worker pool to co-ordinate tasks

- Utilizes arguments for use/disuse of a particular accelerator
  - Enforced using environment variable such as: CUDA_VISIBLE_DEVICES

- We use environment variable so a function runs in a certain GPU% bin

# GPU Multiplexing in Parsl (MPS)

- Parsl offers an easy way to insert the environmental variable required for multiplexing the NVIDIA GPUs

- We modified the *HighThroughputExecutor* to start the functions with desired GPU percentage

```
# Highthroughput executor with GPU percentage example
HighThroughputExecutor(
                address='localhost',
                label="gpu",
                available_accelerators=[1,2,4,0,0],
                gpu_percentage=[50,25,30,40,40],
            ),
```

GPU ID

Corresponding GPU%

- The GPU percentage are enforced by populating the CUDA_MPS_ACTIVE_THREAD_PERCENTAGE environment variable for the target function

# GPU Multiplexing in Parsl (MIG)

- An application can be launched in a particular MIG but updating the CUDA_VISIBLE_DEVICES=MIG-ID

- A code snipped for *HighThroughputExecutors* show how to put the MIG ID

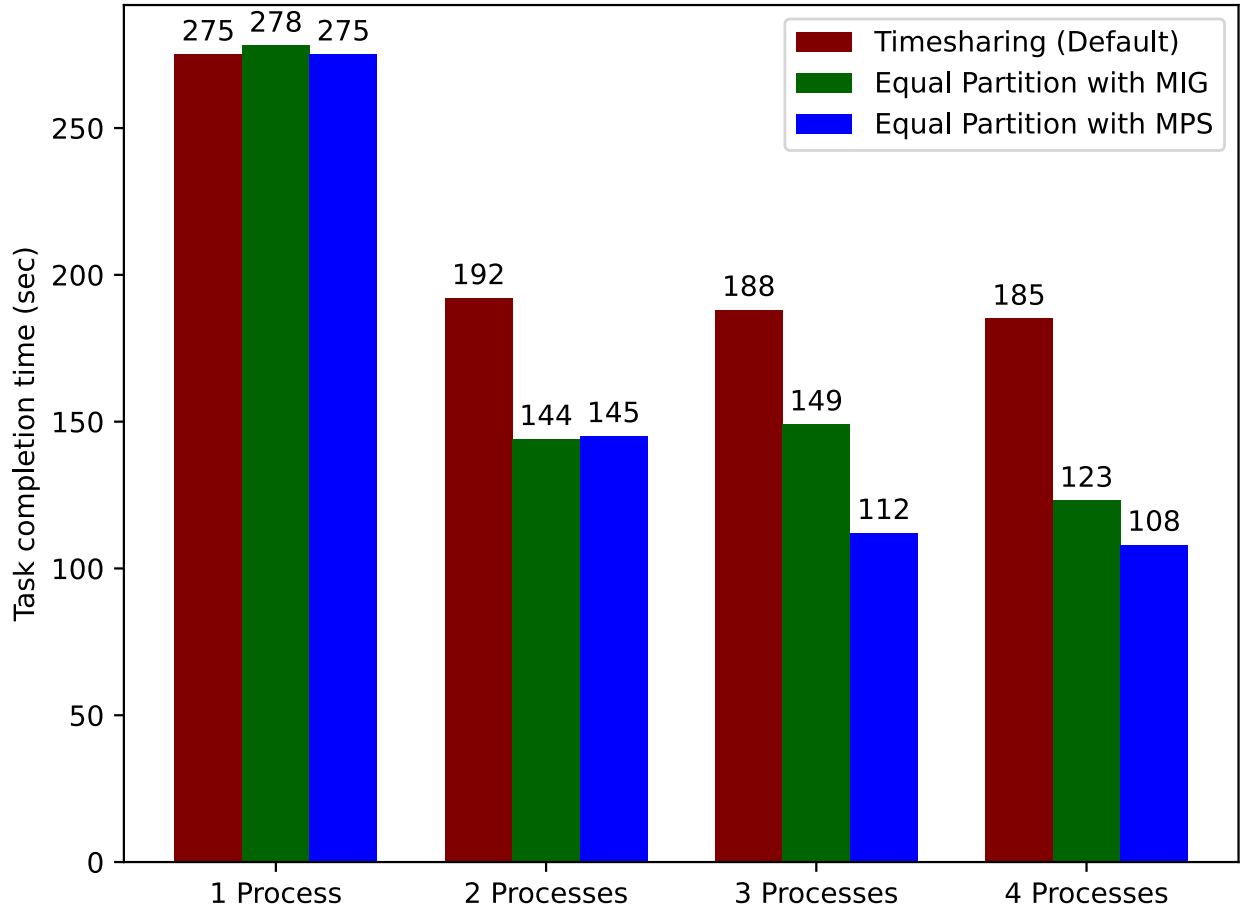
```
            address='localhost',
            label="gpu",
            available_accelerators=[MIG-1-UUID,
            MIG-2-UUID, MIG-3-UUID],
        ),
```

# Performance LLaMA2 Setup

- 1 NVIDIA A100 GPU with 80 GB memory
- CUDA 11.8
- Experiment: Text completion with LLaMA2 (7 billion parameter version)

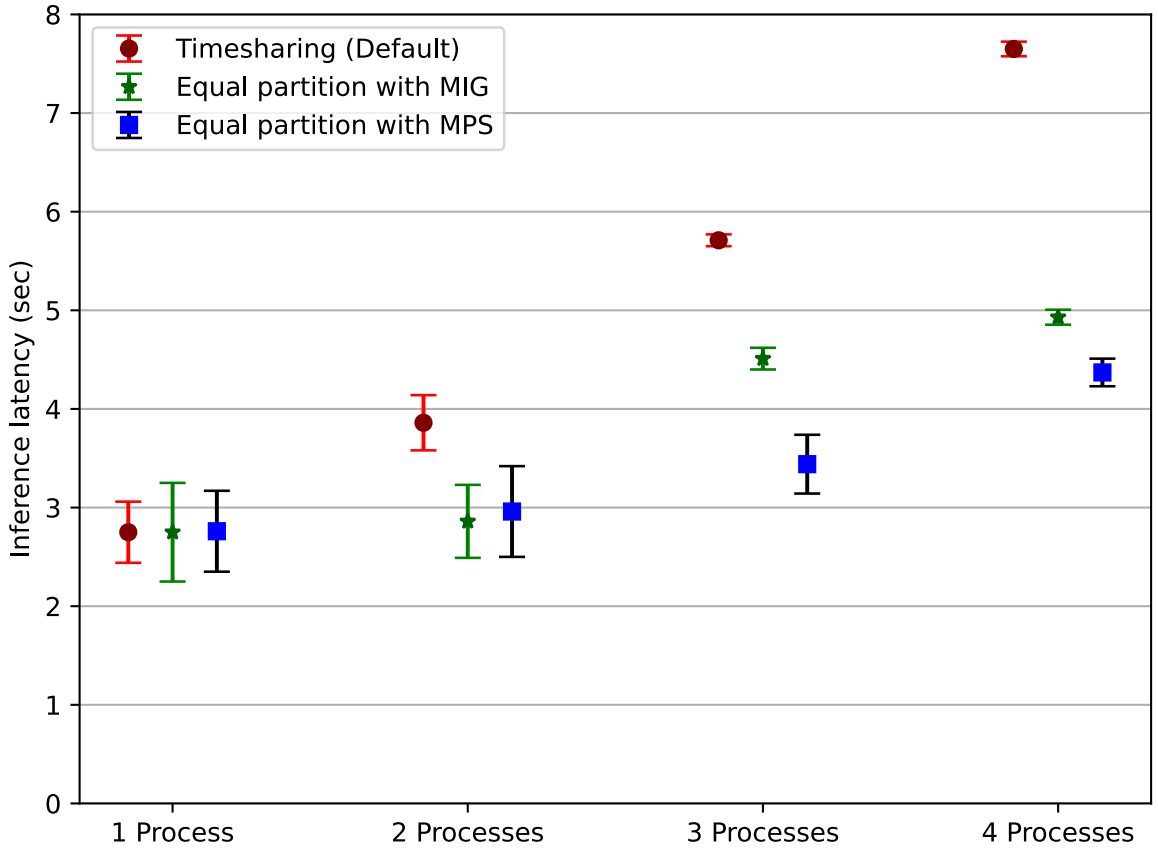
- Total Task: 100 text completion
- When multiple LLaMA2 processes were running, each process got fraction of 100 text completion task

- 60% lower task completion time with GPU partitioning when running 4 processes concurrently



# Latency Measurement

- We measure the time taken to complete one inference
- With default timesharing, adding more processes interferes and increases the time of each inference
- With 5 process, MPS and MIG have 40% lower latency than default timesharing method



# Next Steps

- Environment variable is a simple fix to assign GPU resources to a function
  - Getting a dynamic input from scheduler specifying the GPU% to use
- Changing GPU percentage and MIG attributes is onerous. It requires restarting the processes that are accessing the GPU
  - DNN models with huge weights and parameters are a challenge when changing GPU%
- Implementation beyond single compute node
- Multiplexing where pipelining makes more sense than concurrent execution (e.g. Molecular dynamics workflow)
- Multiplexing strategy. Memory vs. Compute balance

# Conclusion

- Accelerators are usually allocated as a single unit in FaaS platforms
- GPUs are not fully utilized for many workflows
- With multiplexing techniques, the utilization and throughput of GPUs can be increased
- We utilize GPU partitioning and multiplexing with Parsl, a Faas Framework
- With GPU partitioning and multiplexing, we get 60% lower task completion time and 40% lower latency on individual inference while performing inference on a large language model

# Thank you

aditya.dhakal@hpe.com