Hewlett Packard
Enterprise

Spatial Sharing of GPU with Parsl

Aditya Dhakal (aditya.dhakal@hpe.com)
Research Scientist, Hewlett Packard Labs

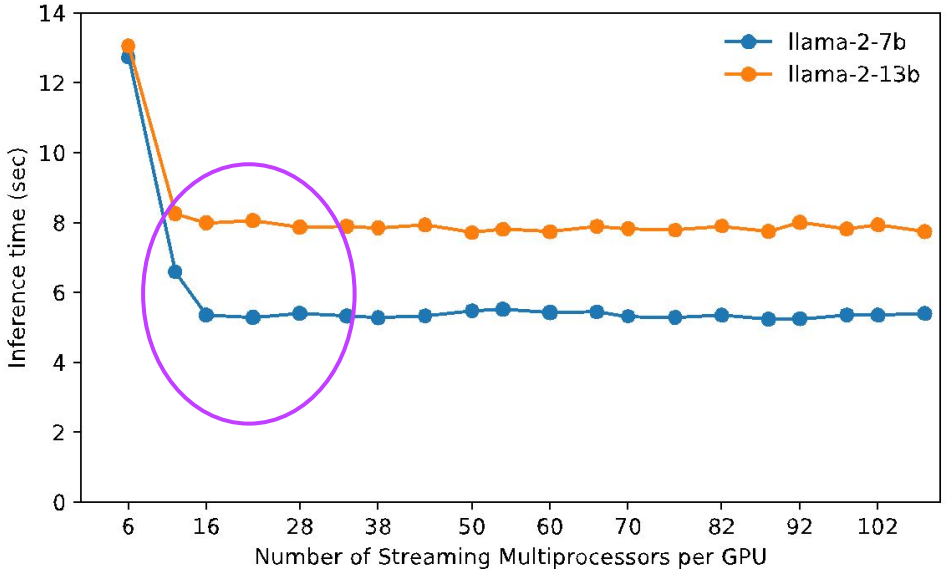
Collaborators:

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

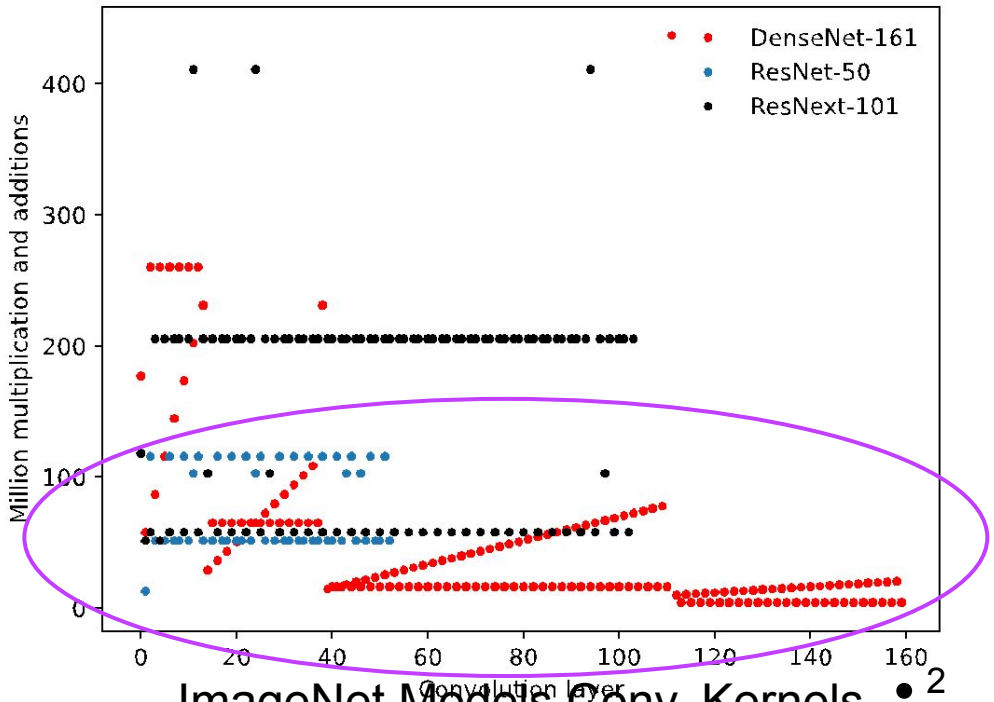
1 Hewlett Packard Enterprise
2 Argonne National Laboratory
3 University of Chicago

Low GPU Utilization of Some Applicatic

- Parts of applications/workflows do not fully utilize available GPU compute
- Many constituent kernels of a workflow are small and/or memory bound
- We show different LLaMa versions do not improve inference time when the GPU
- We also saw some image classification models (convolutional DNNs) have few kernels that utilize a lot of compute



LLaMA runtime vs. GPU SM count

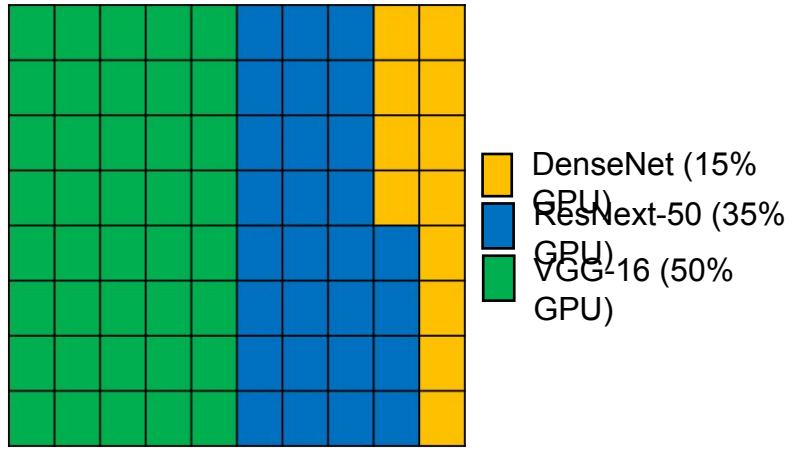


ImageNet Models Conv. Kernels FLOPs

Multiplexing the GPUs

- A solution to low GPU utilization is to run multiple things in GPU concurrently
  - Providing entire GPU for a single function is not cost-effective

- NVIDIA GPUs have Multi-process Service (MPS) and Multi-Instance GPUs (MIGs) that lets user spatially share GPUs
  - MPS allows user to fix maximum number of streaming multiprocessors a process can use
    - Users can choose GPU percentage metric (e.g. 50% of V100 means process will get 40 SMs)
  - MIG creates pre-defined smaller instances of a GPU and provide isolation for multiple process to utilize GPU
  - Other GPU vendors also provide multiplexing solution



An example of MPS dividing GPU SMs

	H100	A100
Available MIG Instances	7x 10GB 4x 20GB 2x 40GB 1x 80GB	7x 10GB 3x 20GB 2x 40GB 1x 80GB

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 percentag

```
# 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

Correspondin g 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

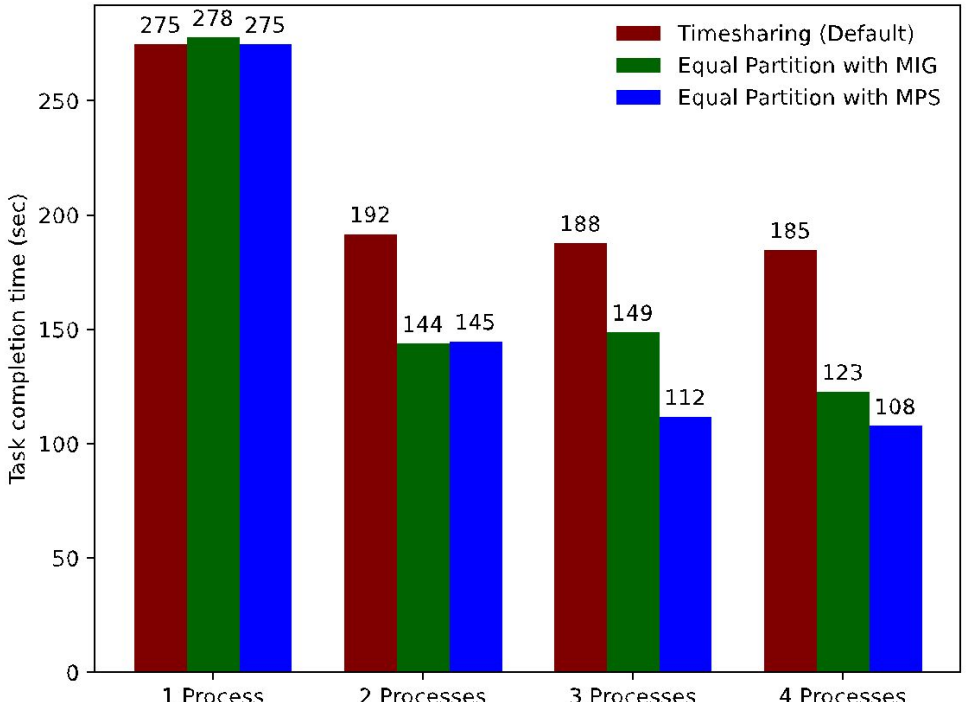
```
            address='localhost',
            label="gpu",
            available_accelerators=[MIG-1-UUID,
            MIG-2-UUID, MIG-3-UUID],
        ),
```

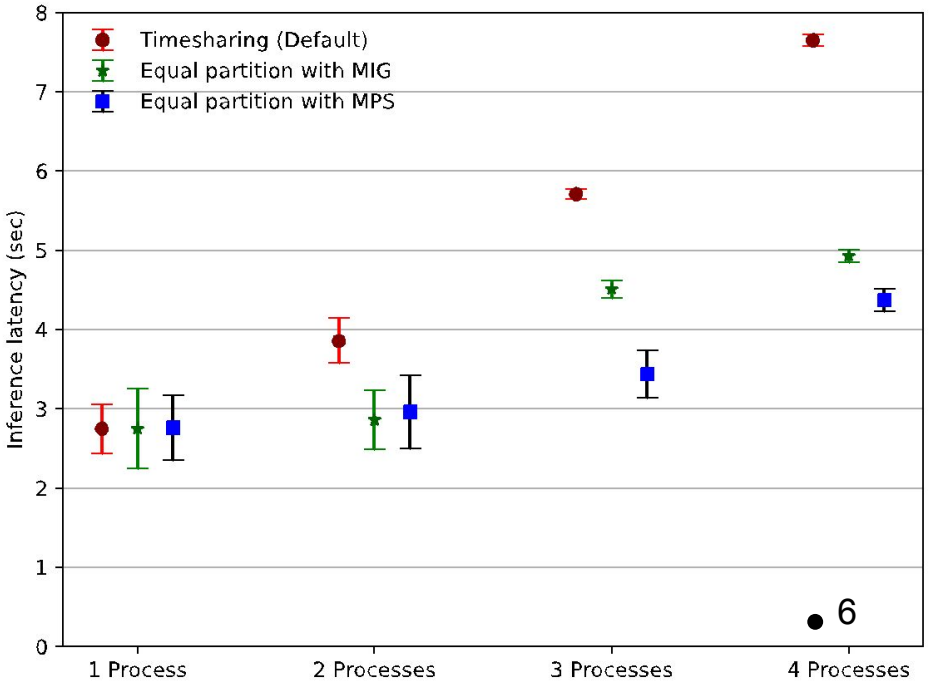
Performance LLaMA2 Setup

- 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
- Still 40% lower latency than default timesharing method





Next Steps

- While environment variable is a simple fix to assign GPU resources to a function, it is not dynamic.
  - 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

Thank you

aditya.dhakal@hpe.com