# Lab 5-part A



## Problem 1 - *PALEO, FLOPs, Platform Percent of Peak (PPP)* 15 points

This question is based on modeling the execution time of deep learning networks by calculating the floating point operations required at each layer. We looked at two papers in the class, one by Lu et al. and the other by Qi et al.

1. Why achieving peak FLOPs from hardware devices like GPUs is a difficult propostion in real systems ? How does PPP help in capturing this inefficiency captured in Paleo model. (3)
2. Lu et al. showed that FLOPs consumed by convolution layers in VG16 account for about 99% of the total FLOPS in the forward pass. We will do a similar analysis for VGG19. Calculate FLOPs for different layers in VGG19 and then calculate fraction of the total FLOPs attributed by convolution layers. (5)
3. Study the tables showing timing benchmarks from Alexnet (Table 2), VGG16 (Table 3), Googlenet (Table 5), and Resnet50 (Table 6). Why the measured time and sum of layerwise timings for forward pass did not match on GPUs ? What approach was adopted in Sec. 5 of the paper to mitigate the measurement overhead in GPUs. (2+2)
4. In Lu et al. FLOPs for different layers of a DNN are calculated. Use FLOPs numbers for VGG16 (Table 3), Googlenet (Table 5), and Resnet50 (Table 6), calculate the inference time (time to have a forward pass with one image) using published Tflops number for K80 (Refer to NVIDIA TESLA GPU Accelerators) both for single-precision and double-precision calculations. (3)

*References*


- Qi et al. PALEO: A Performance model for Deep Neural Networks. ICLR 2017. Available at https://openreview.net/pdf?id=SyVVJ85lg
- Lu et al. Modeling the Resource Requirements of Convolutional Neural Networks on Mobile Devices. 2017 Available at https://arxiv.org/pdf/1709.09503.pdf


## Problem 2 - TTA metric, Stability, Generalization 15 points

This question is based on efforts around DL benchmarking and standardization of performance metrics. We will study TTA metric properties using CIFAR10 dataset. Observe that, MLPerf and DAWNBench both use Imagenet1K for benchmarking but we will use CIFAR10, purely for convenience. In particular, we want to study (i) TTA stability, and (ii) Generalization performance of models optimized for TTA. A similar study was done in Coleman et al. using MLPerf and DAWNBench models. You should read this paper (especially Section 4.1 and 4.2) to have a better understanding of what you will be doing in this question. To study stability we will compute the coefficient of variation of TTA on different hardware by running several times the same training job. You will train Resnet-50 model in Pytorch with 2 different hardware types (V100, RTX8000) using CIFAR10. You will use a batch size of 128 and train the model till you get 92% validation accuracy. From the training logs get the total (wall-clock) time to reach 92% validation accuracy for each of the runs. **We need data from at least 5 different training runs for the same configuration. This data collection can be done in a group of 5 students. Each of the student (in a group of 5) can run the same two training jobs, one with V100 and one with RTX8000. If you decide to collaborate in the data collection please clearly mention the name of students involved in your submission.**

1. Calculate the coefficient of variation of TTA for both the hardware configurations. Compare the value you obtain with that reported in Table 3(a) in the paper by Coleman et al for Resnet-50, 1xTPU. (3)

# Lab 5-part A

2. Collect 5 images from the wild for each of the 10 categories in CIFAR10 and manually label them. **Again in a group of 5 students, each one of you can collect 5 images for 2 categories. In this way each one of you need to collect only 10 images (5 for each of the 2 categories that you choose)**. These images should not be from CIFAR10 dataset. Next test the trained models for these 50 images. What is the accuracy obtained from each of the 10 trained models ? Quantify the mean and standard deviation of accuracy obtained using the 10 models for RTX8000 and the 10 models for V100. (4+4)

3. Measure the GPU utilization using nvidia-smi while training with V100 and report both the time series of GPU core and memory utilization and their average over a period of 3 mins of training. Is the GPU utilization close to 100% ? (2)

4. If the GPU utilization is low, what can you do to increase the GPU utilization? Try your trick(s) and report if you are successful or not in driving GPU utilization close to 100%. (2)

*References*

- Coleman et al. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. Available at https://cs.stanford.edu/ matei/papers/2019/sigops_osr_dawnbench_analysis.pdf

## Problem 3 - *C++, CUDA, Unified Memory* 20 points

In this problem we will compare vector operations executed on host vs on GPU to quantify the speed-up.

1. Write a C$^{++}$ program that adds the elements of two arrays with a $K$ million elements each. Here $K$ is a parameter. Profile and get the time to execute this program for K=1,5,10,50,100. Use `free()` to free the memory at the end of the program. (4)

2. Next using CUDA execute the add operation as a kernel on GPU. Use `cudaMalloc()` to allocate memory on GPU for storing the arrays and `cudaMemcpy()` to copy data to and from the GPUs. Note that host and GPU memory is not shared in this case. You will be running three scenarios for this part:

   (a) Using one block with 1 thread

   (b) Using one block with 256 threads

   (c) Using multiple blocks with 256 threads per block with the total number of threads across all the blocks equal to the size of arrays.

   Again, profile and get the time to execute this program for $K = 1, 5, 10, 50, 100$. So for each of the three scenarios above you will get profile time for five different values of $K$ Use `cudaFree()` and `free()` to free the memory on the device and host at the end of the program. (6)

3. This time you will repeat Step 2 using CUDA Unified Memory and providing a single memory space accessible by all GPUs and CPUs in your system. Instead of `cudaMalloc()` you will use `cudaMallocManaged()` to allocate data in unified memory, which returns a pointer that you can access from host (CPU) code or device (GPU) code. To free the data, just pass the pointer to `cudaFree()`. Again, profile and get the time to execute this program for $K = 1, 5, 10, 50, 100$ for each of the three scenarios. (6)

4. Plot two charts one for Step 2 (without Unified Memory) and one for Step 3 (with Unified Memory). The x-axis is the value of $K$ (in million) and y-axis is the time to execute the program (one chart for each of the three scenarios). In both the charts also plot the time to execute on CPU only. Since the execution time scale on CPU and GPU may be orders of magnitude different, you may want to use log-log scale for y-axis when plotting. (4)

# Lab 5-part A

For this problem you may want to refer to slide deck **HPML-07-CUDA-basics-Spring2022** which has a similar example and the following blogpost:
*An Even Easier Introduction to CUDA*, available at https://developer.nvidia.com/blog/even-easier-introduction-cuda/