Assignment-1

Sai Teja Kuchi (21317, kuchisai@iisc.ac.in)

Configuration Details:-

GPU Machine1 :-

Name	NVIDIA GeForce GTX 1080 Ti
Number of Worker(s)	2
Memory	10.91GB
Driver Version	520.61.05
Cuda Version	11.8

GPU Machine2 :-

Name	NVIDIA A100-SXM4
Number of Worker(s)	8
Memory	39.45GB
Driver Version	515.86.01
Cuda Version	11.7

Packages :-

python	3.8.10
torch	1.13.1
torch-tb-profiler	0.4.1
torchsummary	1.5.1
pandas	1.5.3
matplotlib	3.6.3
seaborn	0.12.2

Task-1

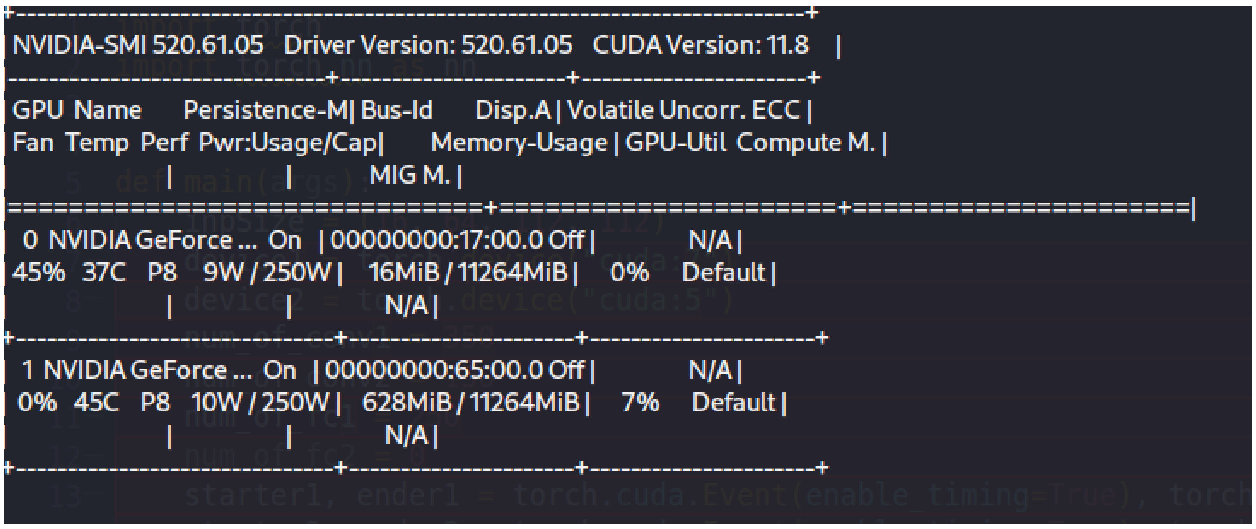
- task1.py contains the code for Part 1 and 2, while task1_part.py contains the code for Part 3 and 4.
- This task is completely run on **Machine1** with inpSize = (16, 64, 112, 112)
- Below is the table which has the details on when a model can no longer fit on a single GPU (GPU0 in this case) along with the timing data if it was successful.
- Before recording the time, an warmup of 5 iterations is done and while calculating the elapsed time an average of 10 iterations is taken.

num_of_conv	num_of_fc	out_of_memory?	Average Time (ms)
61	62	No	109.2050903320312 5
62	61	No	109.8805297851562 6
62	62	Yes	-

- The partition of layers into 2 GPUs (in **Machine1**) and their respective timing data with an warmup of 5 iterations and average of 10 iterations is shown below.

Device	num_of_conv	num_of_fc	Time Taken (ms)
GPU0	84	0	110.233056640625
GPU1	54	62	109.9095703125

- It has to be noted that, there is still memory present in GPU0 for adding additional layers but there is no longer enough memory in GPU1 due to some preallocated memory or some other unknown reason which the below image illustrates.



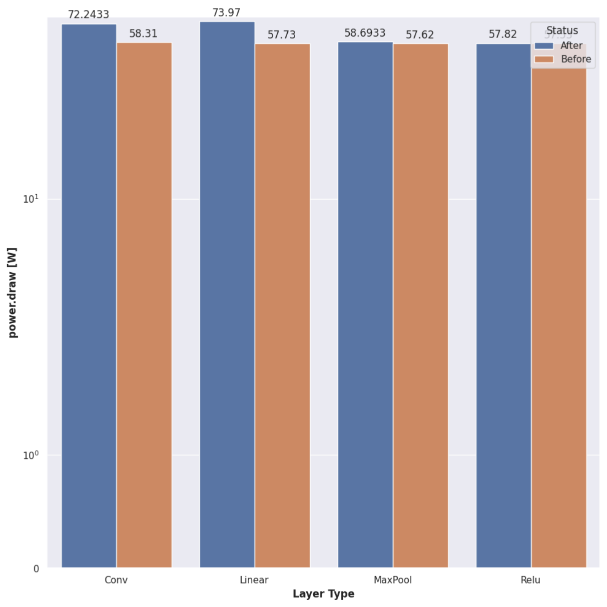
```
+-----------------------------------------------------------------------------+
 NVIDIA-SMI 520.61.05   Driver Version: 520.61.05   CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
 GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
 Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
                               |                      |               MIG M. |
===============================+======================+======================|
   0  NVIDIA GeForce ...  On   | 00000000:17:00.0 Off |                  N/A |
 45%   37C    P8     9W / 250W |     16MiB / 11264MiB |      0%      Default |
                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
   1  NVIDIA GeForce ...  On   | 00000000:65:00.0 Off |                  N/A |
  0%   45C    P8    10W / 250W |    628MiB / 11264MiB |      7%      Default |
                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

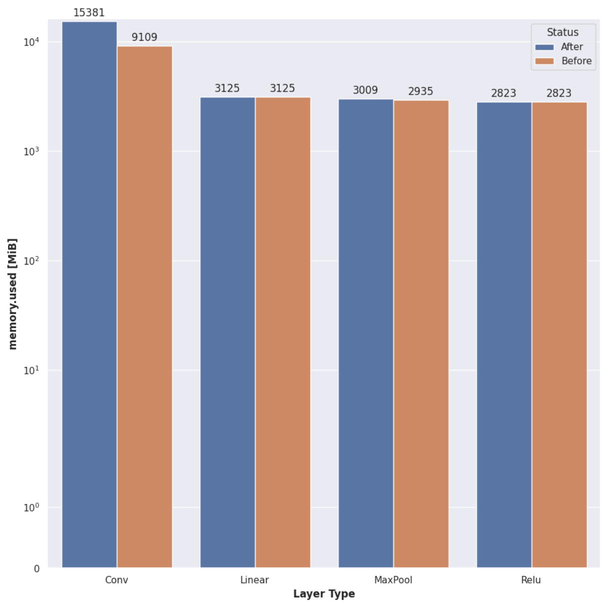
Task-2

- Following are the layers and their respective configurations that are used.

Layer Type	Input size	Code
ReLu	(256, 3, 224, 224)	nn.ReLU(inplace=True)
MaxPooling2D	(256, 3, 224, 224)	nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
Convolution	(256, 3, 224, 224)	nn.Conv2d(3, 128, kernel_size=3, padding=1)
Linear	(256, 512 * 7 * 7)	nn.Linear(512 * 7 * 7, 4096)

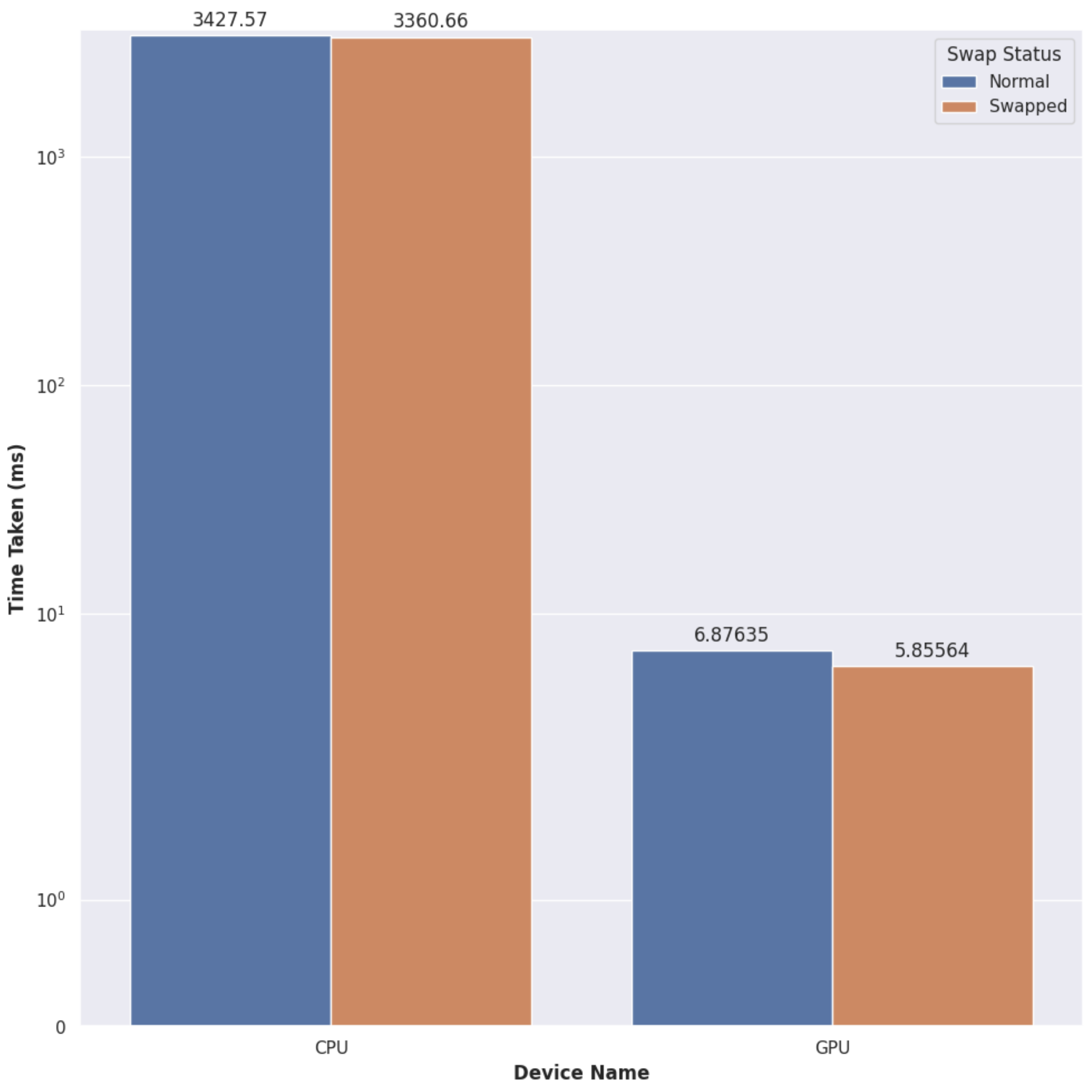
- Each layer has their own python file designated with its layer name, task2.py is called to get the required data for each layer which internally calls the layers python file.
- Below images show the memory utilization (MB) and power drawn used by different layers that is run on **Machine2**.
- Warmup iteration followed by average of 3 iterations is used for calculation of below data.





Task-3

- vgg.py contains the code for the vgg-16 model along with the swap logic, while task3.py contains the code for calling the vgg model with and without the swap logic.
- Below image shows the timing data of both CPU and GPU with inputSize = (128, 3, 224, 224) that is run on **Machine2**.
- Swap Status normal indicates that VGG architecture is not modified, while swapped indicates that ReLu is swapped with that of MaxPooling2D.
- While measuring time, an warmup of 5 iterations and average of 10 iterations are used for calculation of elapsed time.



- As seen from the above image, while the performance improvement is minutus when an swap is performed this is due to the shallow layers in the models. Current DNN models (Resnet, Transformers etc) are having a deeper number of layers, thus if such technique can be used there can be a good amount of improvement both in time and space aswell because when MaxPooling2D is performed before ReLu, the input activation to the ReLu layer is smaller when compared to the other case. Hence the size required in both the forward and backward pass for the model with swapped layers will also be lesser than the original architecture incase the model is very deep.

Task4

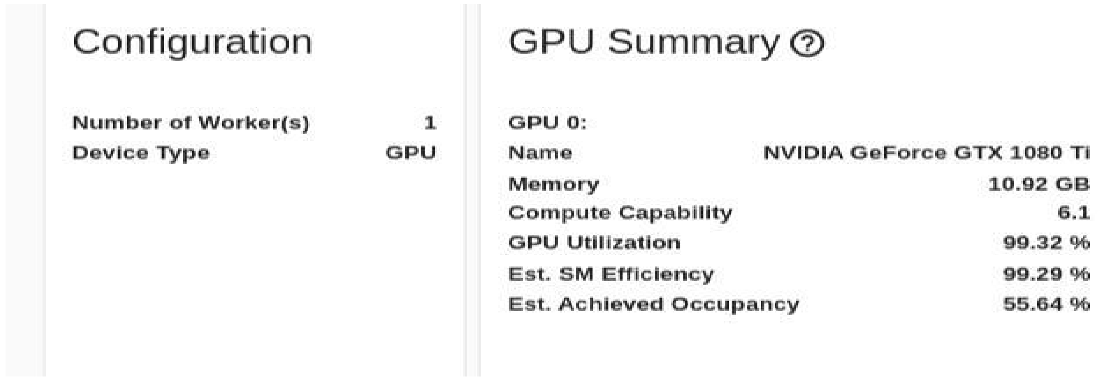
- vgg_normal.py contains the code for executing the vgg-16 model on a single GPU, while vgg_distriubted.py contains the code for executing the vgg-16 model on multiple GPUs along with profiling.

inputSize	(8192,3,224,224)
total_epochs	1
output_classes	1000
batch_size	64
Profiler infomation	wait=1, warmup=2, active=5, repeat=2

- Below are the results of timing data obtained on a single GPU and Distributed GPU for VGG16 model. For the Single GPU, an average of 3 iterations is taken to calculate the run time value.

Machine Type	Single/Distributed	Number of workers used	Average Run Time (ms)
Machine1	Single	1	84090.46875
Machine1	Distributed	2	44416.57923

- Average run time in case of distributed/multi node gpu is taken as max(gpu0, gpu1) time.
- Speedup = Time taken in single GPU/Time taken in Multiple GPUs = 84090.46875/44416.57923 = 1.89.
- From the below images, it can be observed that incase of distributed GPUs one of a GPU has 98.8% and 99.15%, while the incase of Single GPU it has 99.32%. So this along with the communication overhead costs resulted in less than 2x speedup is what I think.

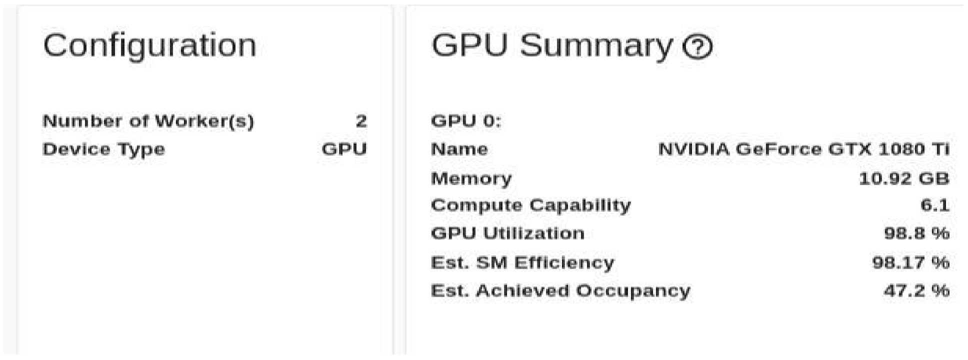


Configuration

Number of Worker(s)	1
Device Type	GPU

GPU Summary

GPU 0:	
Name	NVIDIA GeForce GTX 1080 Ti
Memory	10.92 GB
Compute Capability	6.1
GPU Utilization	99.32 %
Est. SM Efficiency	99.29 %
Est. Achieved Occupancy	55.64 %

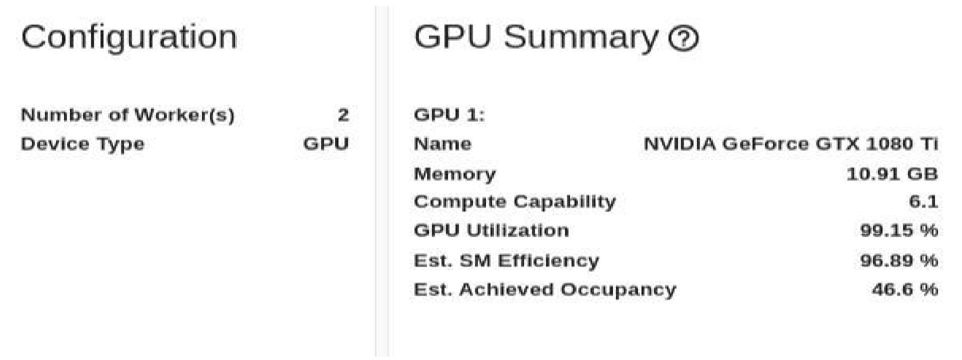


Configuration

Number of Worker(s)	2
Device Type	GPU

GPU Summary

GPU 0:	
Name	NVIDIA GeForce GTX 1080 Ti
Memory	10.92 GB
Compute Capability	6.1
GPU Utilization	98.8 %
Est. SM Efficiency	98.17 %
Est. Achieved Occupancy	47.2 %



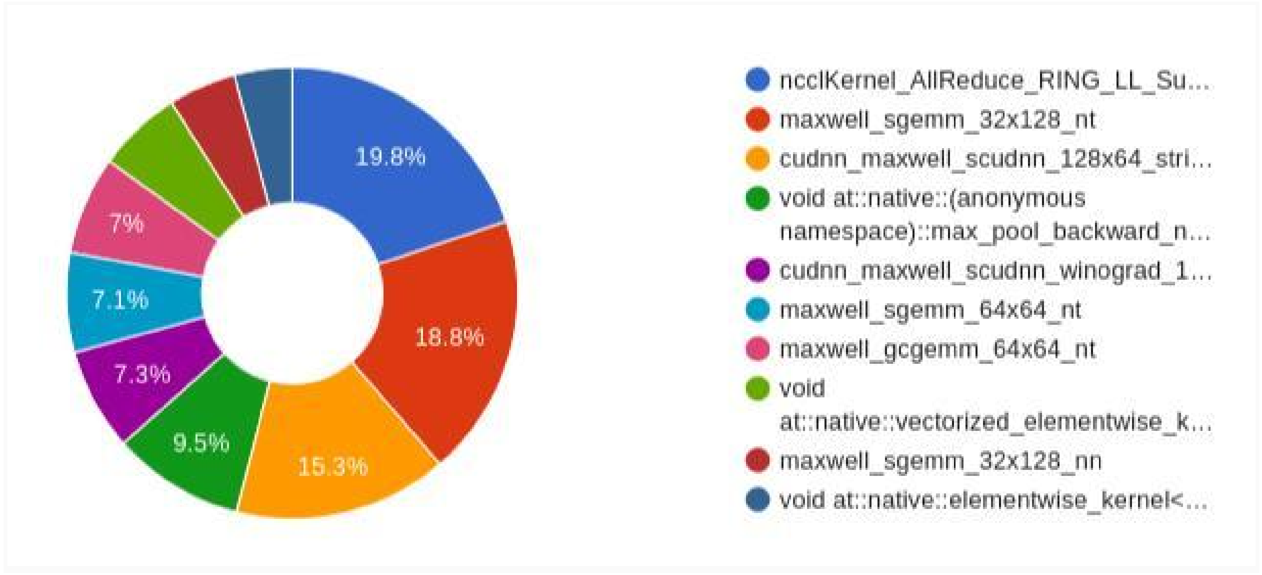
Configuration

Number of Worker(s)	2
Device Type	GPU

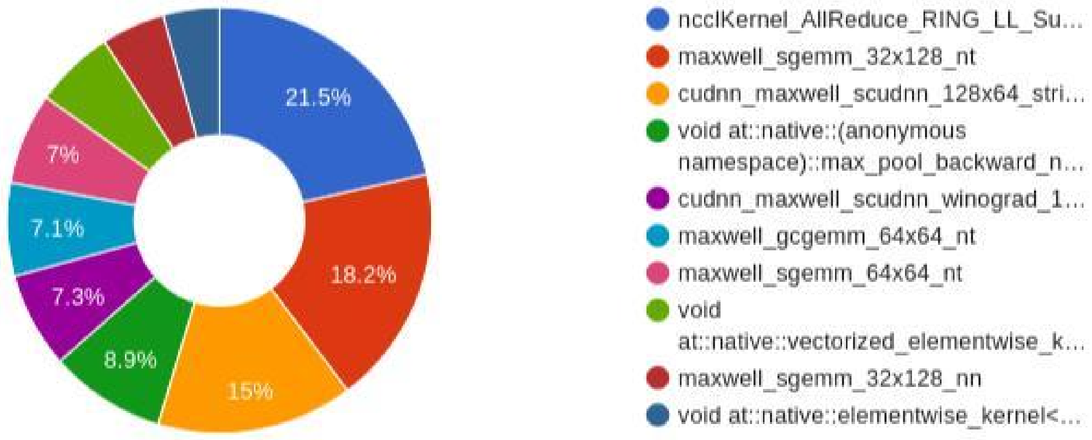
GPU Summary

GPU 1:	
Name	NVIDIA GeForce GTX 1080 Ti
Memory	10.91 GB
Compute Capability	6.1
GPU Utilization	99.15 %
Est. SM Efficiency	96.89 %
Est. Achieved Occupancy	46.6 %

- Because of unavailability of the Distributed View [2], ncclKernel_AllReduce which is the only available kernel command of distributed communication in GPU kernel View is shown below.
- In GPU0 of distributed setting, the ncclKernel is taking a total duration of 624626 us which contributes to 19.8% as shown in below image.



- In GPU1 of distributed setting, the ncclKernel is taking 562991us of total duration which accounts to 21.5% as shown in below image.



References

1. Geifman, Amnon, and Nave Assaf. n.d. “The Correct Way to Measure Inference Time of Deep Neural Networks - Deci.” Deci AI. Accessed February 15, 2023. https://deci.ai/blog/measure-inference-time-deep-neural-networks/.
2. “GitHub Issue raised for Distriubted View not working.” 2022. YouTube. https://github.com/pytorch/kineto/issues/640#issuecomment-1407049257.
3. “PyTorch Benchmark.” 2021. Lei Mao. https://leimao.github.io/blog/PyTorch-Benchmark/.
4. “PyTorch Profiler With TensorBoard — PyTorch Tutorials 1.13.1+cu117 documentation.” n.d. PyTorch. Accessed February 15, 2023. https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html?highlight=tensorboard.
5. “torch.profiler — PyTorch 1.13 documentation.” n.d. PyTorch. Accessed February 15, 2023. https://pytorch.org/docs/stable/profiler.html#torch.profiler.profile.
6. “Useful nvidia-smi Queries.” 2021. NVIDIA. https://nvidia.custhelp.com/app/answers/detail/a_id/3751/~/useful-nvidia-smi-queries.
7. “Writing VGG from Scratch in PyTorch.” n.d. Paperspace Blog. Accessed February 15, 2023. https://blog.paperspace.com/vgg-from-scratch-pytorch/.