COMS 6998 - Practical Deep Learning Sys. Performance (Spring 2021)

Project Report

Feroz Ahmad, Li Cai
fa2581@columbia.edu, lc2928@columbia.edu

April 21, 2021

1 Objective

To analyze and compare training speed while training Generative adversarial networks (GANs) on single and multiple GPUs (2, 4, 8) on a single node instance.

The application under focus is text to image generation whose state of the art methods involve Generative Adversarial Network based architectures. We study the percentage increase in speed up with increase in batch size, scaling efficiency and communication overhead with increase in GPU count.

2 Motivation

There have been some studies [1] about GPU performance scaling of networks such as Inception, VGG, ResNet etc. However, to the best of our knowledge we could not find the performance scaling study on Generative Adversarial Networks. Hence, we decided to put a GAN based application under evaluation.

Given text to image generation is an interesting problem that has gained a lot attention in past years and its solutions have been lead by Generative Adversarial Networks on benchmark rankings, we decided to pick this problem. GANs in this setting operate over text and images making the network highly computational, made it a suitable candidate for the speed up study.

3 Related Work

3.1 Image to Text Generation

We wish to explore state of the art methods in the current times. Current state of the art methods are GANs models on several benchmarks such as COCO, CUB and Oxford 102

Flowers. [2]

One of good implementations currently available is StackGAN++ (or StackGAN++) [5] which is built upon StackGAN [4]. However, due to computational expense we chose StackGAN for our performance analysis due to its feasibility to train on cheaper resources.

3.1.1 StackGAN for Image Generation

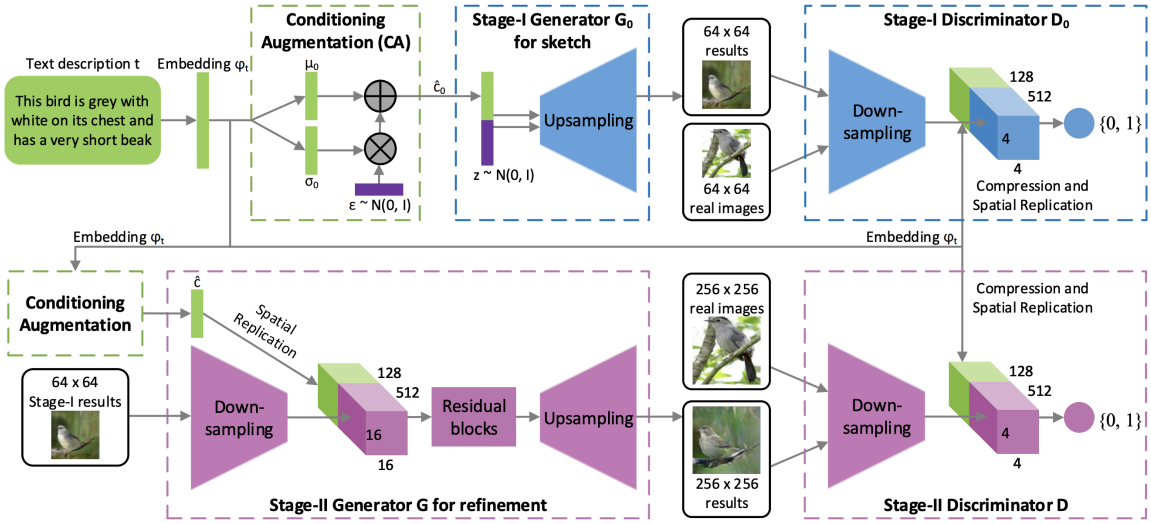


Figure 1: Stack GAN Architecture

The StackGAN architecture consists of 3 main components:

1. Stage-I
2. Stage-II
3. Conditioning Augmentation

Stage-I

Instead of directly generating a high-resolution image conditioned on the text description, StackGAN's architecture simplifies the task to first generate a low-resolution image with the Stage-I GAN, which focuses on drawing only rough shape and correct colors for the object.

Stage-II

Low-resolution images generated by Stage-I GAN usually lack vivid object parts and might contain shape distortions. Some details in the textual description might also be omitted in the first stage, which is vital for generating images. The Stage-II GAN is built

upon Stage-I GAN results to generate high-resolution images. It is conditioned on low-resolution images and also the text embedding again to correct defects in Stage-I results. The Stage-II GAN completes previously ignored text information to generate the final output.

Conditioning Augmentation

The limited number of training text-image pairs often results in sparsity in the text conditioning space and such sparsity makes it difficult to train GAN. Thus, authors propose Conditioning Augmentation technique to encourage smoothness in the latent conditioning manifold. It allows small random perturbations in the conditioning manifold and increases the diversity of synthesized images. The StackGAN for the first time generated images of 256×256 resolution with photo-realistic details from text descriptions and authors give this credit to Conditioning Augmentation.

3.2 Multi GPU Communication

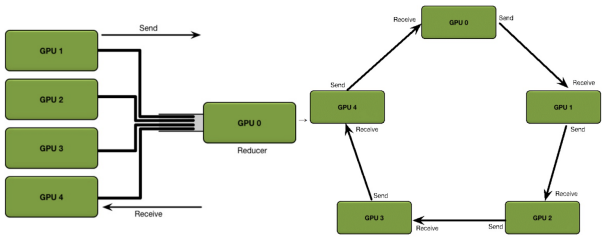


Figure 2: Single Reducer vs Ring-based Strategies

In this project we will be comparing two communication algorithms: single reducer vs ring-based. The single reducer strategy has a leader GPU who is responsible for receiving the updates and broadcasting the reduced values to other GPUs. On the other hand, the ring-based strategy allows all GPUs to communicate to each other for updates. Due to the clear bottleneck present in single reducer, the ring-based strategy is considered to be more efficient.

4 Challenges

4.1 Hardware Challenges

Our original proposal was to compare performance on different GPU architectures available by cloud providers, such as K80, P100, V100. However, it was extremely difficult to get cloud account quotas increased for 8 GPUs even with an upgraded account, specifically with GCP. Eventually, we got 8 V100 GPUs in a AWS EC2 P3 (p3.16xlarge) instance with an upgraded account.

Since the educational credits provided in the course do not cover multiple GPUs from the cloud provider. It was very expensive for us to keep the p3.16xlarge instance running for long periods of time.

4.2 Training Challenges

1. Using Caltech-UCSD Birds 200 dataset (CUB) [3] of 11,788 images (60 per species) Training an epoch at StackGAN took about 1 hour 40 minutes on a single V100 GPU. Due to this we decided to reduce the dataset by downsampling the number of images (in count). We randomly selected 20 images of each bird species (to maintain diversity) from the CUB-200 dataset resulting 4,000 images for training.

2. Training required high CPU memory ($>$ 16 GB). Our VM would crash in the middle of training causes to lose the progress and added unnecessary turn around time for our experiment completion.

3. Even after down-sampling, the training took too long. Due to financial resource constraint, we limited our training epochs to attain either specific loss or we trained for 5 epochs as per the requirement of the experiment. After this decision, we had to trade convergence for time and resources.

4. Since we did not train till convergence, we employed loss as our performance metric instead of the Inception Scores as used by the authors of StackGAN.

5 Experimentation

5.1 StackGAN model results

(a) A white bird with grey feathers (b) A brown bird with thin legs

Top: Output from StackGAN Stage 1, Bottom: Output from StackGAN Stage 2

The above figure indicate the efficacy of the architecture. In both the examples, we can see the network is able to generate images with matching textual description.

Additionally, we also observe the difference between Stage I and Stage II outputs. As mentioned by the authors, we can clearly see how the Stage I outputs lack the fine detailing in the generated image. However, it is evident that the skeleton on the images has been setup by the Stage I output, and the stage II output builds over the low resolution image to achieve a coherent high resolution image.

5.2 Experimental Setup for studying performance of GAN

The platform we used was AWS EC2 P3 instance with 8 V100 GPUs. The deep learning framework used was tensorflow 2.4 and keras. As previously mentioned, the data set we ended up using was the reduced CUB data set with 4,000 images. We updated an existing StackGAN implementation to be compatible with tensorflow 2.4 and added distributed training functionality. Multiple experiments were run with different combinations of number of GPUs, batch sizes, and communication algorithms. Training time and loss value were recorded for each epoch.

For distributed training we used MirroredStrategy in tensorflow that does synchronous multiple GPU training in one local machine. In this strategy all the information is replicated in each GPU. Eventually the gradients are synced across GPUs and the same update is sent to each GPU. To specify the number GPUs we simply suppled the list of GPUs as an argument: tf.distribute.MirroredStrategy(devices=[“/gpu:0”, “/gpu:1”]). Last but not least, as the number of GPUs increases, the global batch size is also scaled up accordingly.

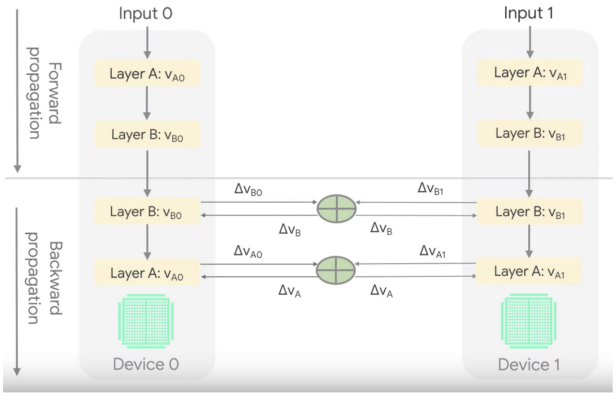


Figure 4: Mirrored Strategy with 2 GPUs

For cross device communications we chose to study NcclAllReduce and ReductionToOneDevice in tensorflow. NcclAllReduce is very similar to the ring-based strategy. On the other hand, ReducetionToOneDevice is very similar to the single reducer strategy. We compared these two strategies in terms of scaling efficiency in distributed training.

5.3 Experiment Results

First we studied the impact of batch sizes. We trained with 2 V100 GPUs, NcclAllReduce, for 16, 32, 64 batch sizes and recorded time taken to reach the same target loss value. As

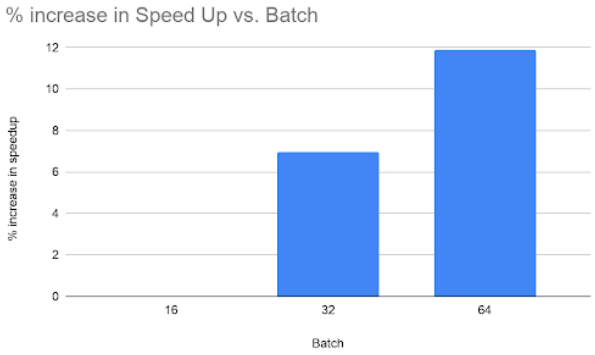
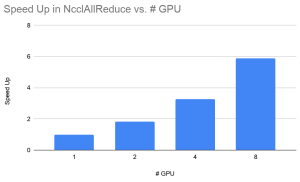


Figure 5: Batch Size Speedup

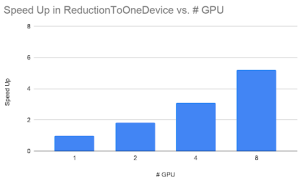
you can see in the chart above, a speedup of around 6% was achieved for doubling the batch size. We were not able to get to 4 and 8 GPUs training due to resource constraints.



(a) NcclAllReduce Speedup

# GPUs	Communication Overhead	Scaling Efficiency
1	0	1.00
2	8.70%	0.92
4	21.74%	0.82
8	35.65%	0.74

(b) NcclAllReduce Speedup



(a) ReductionToOneDevice Speedup

# GPUs	Communication Overhead	Scaling Efficiency
1	0	1.00
2	9.13%	0.92
4	28.70%	0.78
8	53.04%	0.65

(b) ReductionToOneDevice Speedup

Next we ran the same experiments for NcclAllReduce and ReductionToOneDevice with 64 batch size, 5 epochs, and 1, 2, 4, 8 GPUs. It's obvious that StackGAN benefited from multiple GPUs training and gained significant speedup as the number of GPUs increases. In addition, we also observed the pattern of increasing communication overhead and decreasing scaling efficiency as the number of GPUs increases. This is expected because communication with large amount of GPUs takes longer and impacts the scaling efficincy.

Now lets compare ReductionToOneDevice and NcclAllReduce side by side and find the winner. In the case of 2 GPUs, both of the communication strategies had similar scaling efficiency. However, as the number of GPUs increases, NcclAllReduce starts to outperform ReductionToOneDevice. The larger the number of GPUs, the bigger the gap between the two. In the end, there's enough evidence to confirm that NcclAllReduce is indeed better than ReductionToOneDevice.

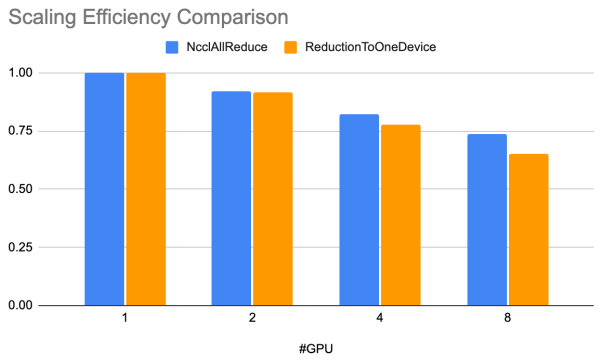


Figure 8: ReductionToOneDevice vs NcclAllReduce

6 Conclusion

StackGAN is very computationally intensive and can achieve good speedup with distributed training over multiple GPUs. We also found out that increasing batch size in distributed training helped with stable convergence and provided speedup over small batch size with noisy gradients. Last but not least, we proved that NcclAllReduce is more efficient than REductionToOneDevice with higher scaling efficiency and lower communication overhead.

GitHub Code Repository: github.com/fz-29/PractDLSysPerformanceProject

References

[1] Alexander Sergeev and Mike Del Balso. "Horovod: fast and easy distributed deep learning in TensorFlow". In: *CoRR* abs/1802.05799 (2018). arXiv: 1802.05799. URL: http://arxiv.org/abs/1802.05799.

[2] *Text-to-Image Generation*. URL: https://paperswithcode.com/task/text-to-image-generation/codeless.

[3] P. Welinder et al. *Caltech-UCSD Birds 200*. Tech. rep. CNS-TR-2010-001. California Institute of Technology, 2010.

[4] Han Zhang et al. "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks". In: *CoRR* abs/1612.03242 (2016). arXiv: 1612.03242. URL: http://arxiv.org/abs/1612.03242.

[5] Han Zhang et al. "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks". In: *CoRR* abs/1710.10916 (2017). arXiv: 1710.10916. URL: http://arxiv.org/abs/1710.10916.