USE OF GPUS IN DEEP LEARNING

1.Introduction:

Graphics processing unit (GPU) is a processor that is good at handling specialized computation, this is contrast to CPU, which handles general computation. CPU's have usually around 4 to 16 cores whereas, GPU's have 1000's of cores. GPU processing is fast if we have a task that can be parallelized but here, we have to note that if the task is small then the time to move the task from host to device can be costly. So, this is why we need only that tasks that can actually benefit from the GPU computations. The main reason why GPUs are faster for computation than the CPU's is also because of bandwidth.

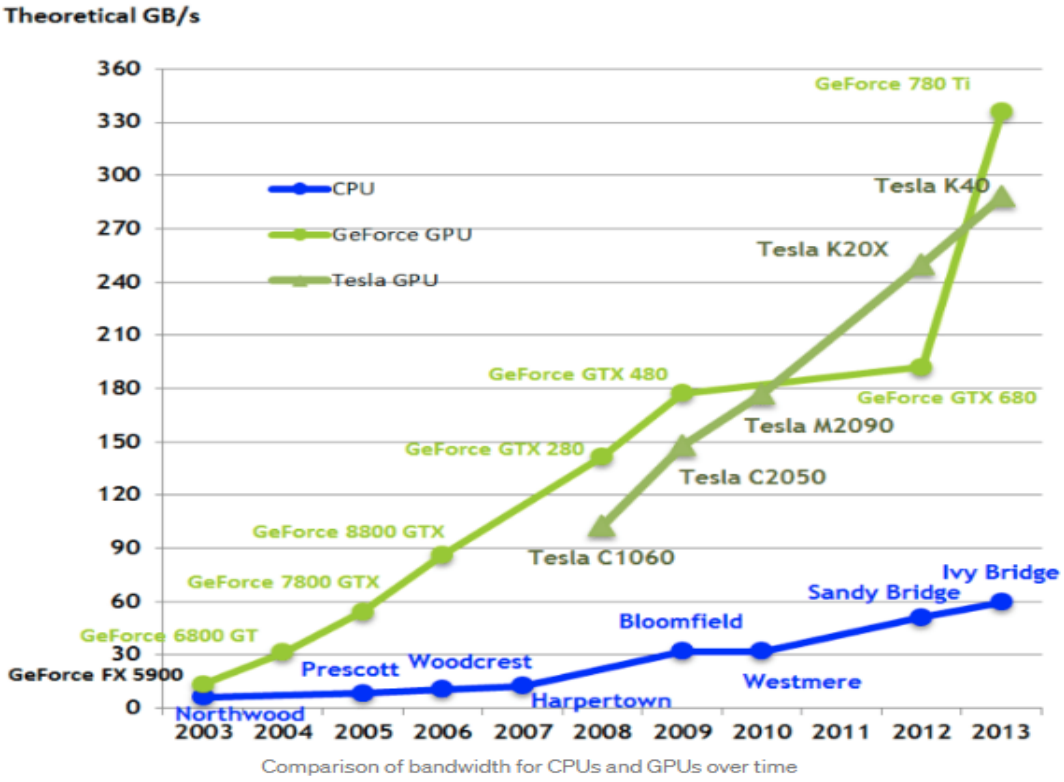

Figure 1: the above figure shows the bandwidth of CPU vs GPU's

With larger datasets size CPU will take up lot of memory while training the model whereas, GPU has a dedicated memory called VRAM memory (Video RAM memory). This gives CPU ability to work on different tasks.

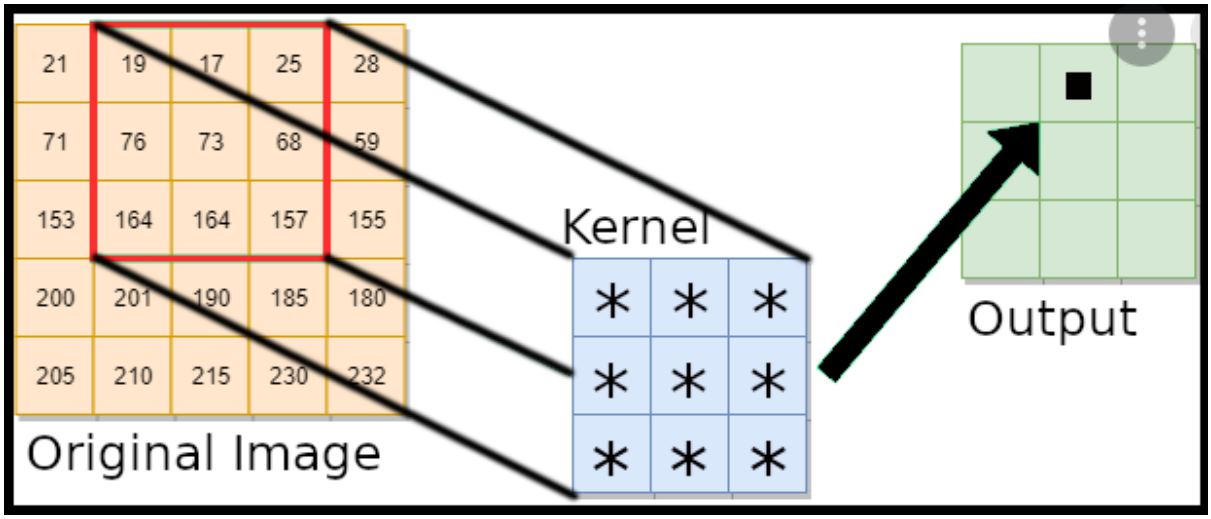

Figure 2. The above figure shows the convolution process which is described below in detail.

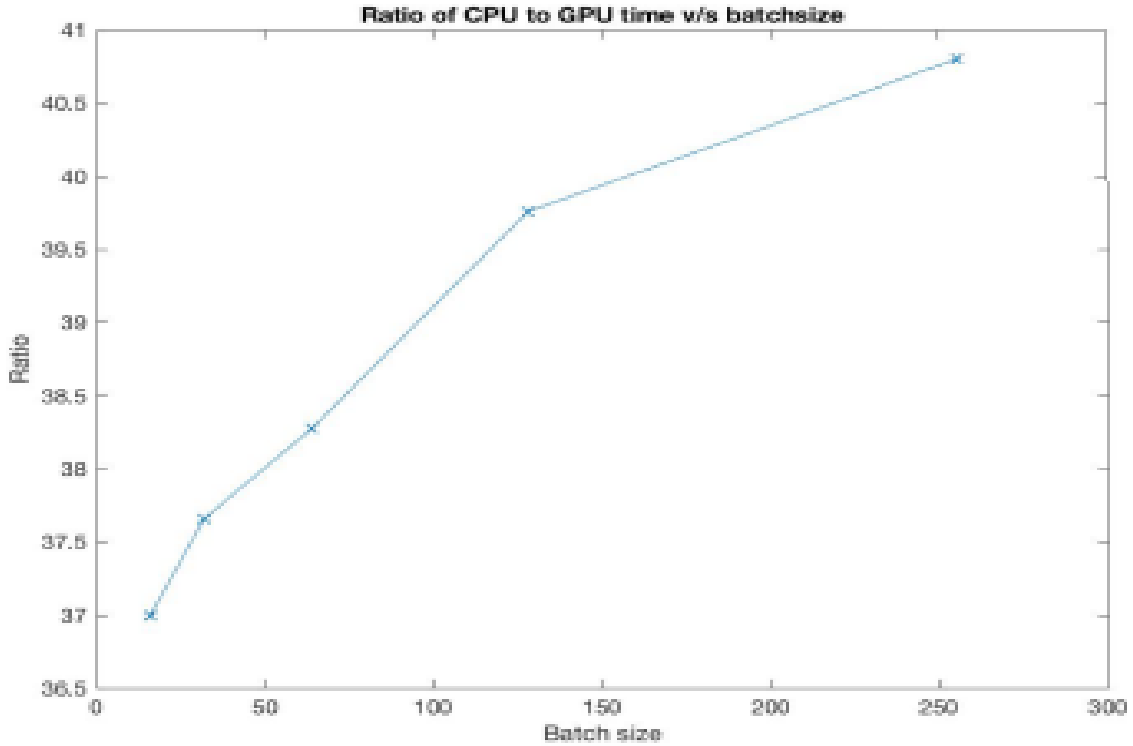

Figure 3. Ratio of CPU vs GPU speedup as a function of batch size on the convolution layer

In Neural Networks GPUs are well suited for parallel computation. There is a term called embarrassingly parallel which tells the problem is very basic and easy

to parallelize. Embarrassingly parallel is the case when there is little to no dependency or need for communication between the parallel task according to Wikipedia. The main operation that is used in the deep learning frameworks such as CNN, GAN, Encoders etc. is the use convolution layer. Convolution is the basic averaging of image pixels in the nearby area as seen in the figure 2 above. This is done so that the noise in the image can be reduced and then the image can be more interpretable by the Neural Network. The singular elemental averaging (i.e., averaging a single element) can be done in parallel as it is not dependent on the output of other averaging elements. By doing this we can reduce the computation time of the convolution operation of the image. This particular operation is often done by GPUs in the background. The frameworks such as Pytorch and TensorFlow support this kind of computation on the GPU in the background.

2.Types of parallelism in the GPUS:

2.1 Data parallelism:

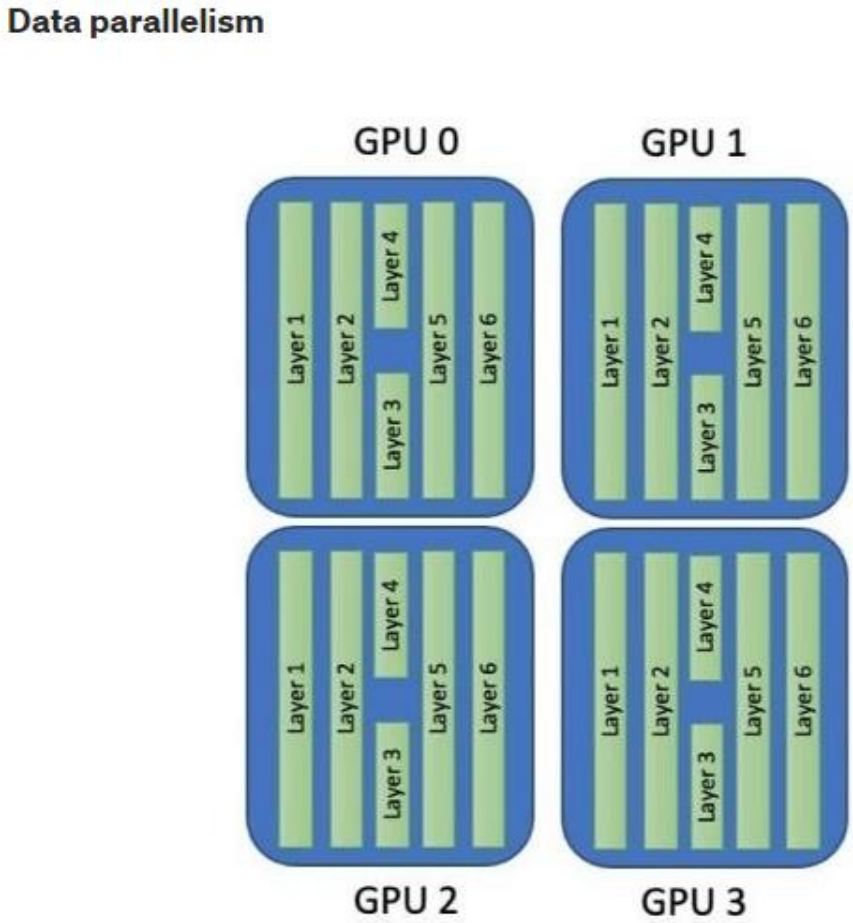

Figure 4. The above figure shows the data parallelism on GPU's

Data parallelism is when we use the same model for every thread i.e. The dataset is divided into N parts (N=number of GPUs) then a copy of the model is placed on each GPU and trained on the corresponding data chunk. We either train that in synchronous or asynchronous manner.

After this happens the gradients are averaged. In this also there are two implementations and they are parameter serving and ring-all reduce. Parameter server is a node that will be responsible for synchronizing the gradients.

When we have clusters of GPUS, we assign different role to each Cluster. In this some devices act as parameter server and the other act as training workers.

2.1.1 Parameter servers:

These are the servers that hold the parameter of the model and they are responsible for updating them on the global state of the model. Whereas the training worker(device) are the ones that are doing computations and calculating the loss in the model. In this case the model is again replicated on each training worker and then each training fetches the parameters from the parameter server and updates the model weights.

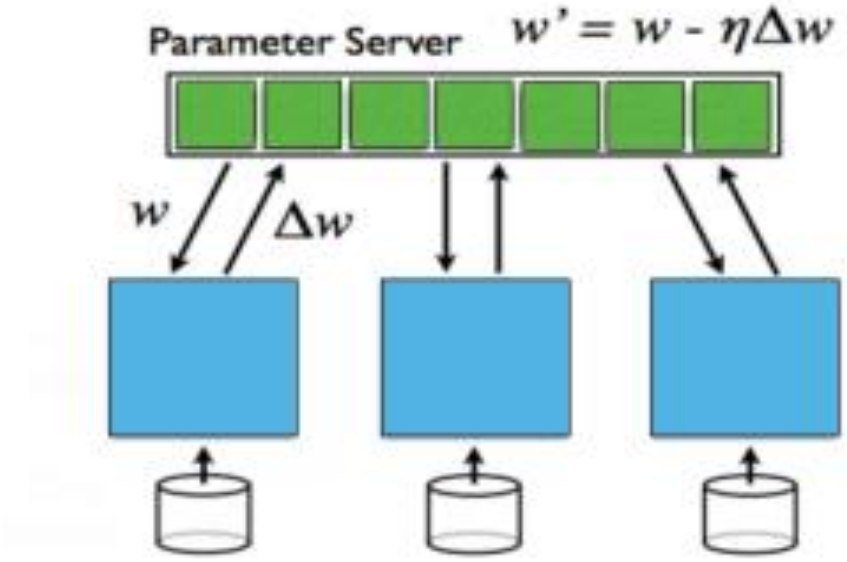

Figure 5. The above figure shows the process of parameter serving for updating the parameters

The problem in this method is that the channel bandwidth capacity of parameter server is dependent on the number of parameters that are involved in the training. Whereas ring all reduce is designed to solve this problem because in this approach

the channel bandwidth capacity does not depend on the number of agents participating in the training.

2.1.2 Ring all reduce:

As we can see in the below figure that the GPUs are arranged in a logical ring format and this can reduce the time spent on sending the data. But majority of the time is spent in the communication of the GPUS locally. Usually, the way training was done was using a single reducer GPU and this GPU will be connected to multiple GPU and the data is sent and the reducer GPU through network channel and this will create a problem as we don't want the bandwidth to become a bottleneck in this situation. So, Ring all reduce can solve this problem. Baidu recently announced a faster version of the ring all reduce algorithm recently.

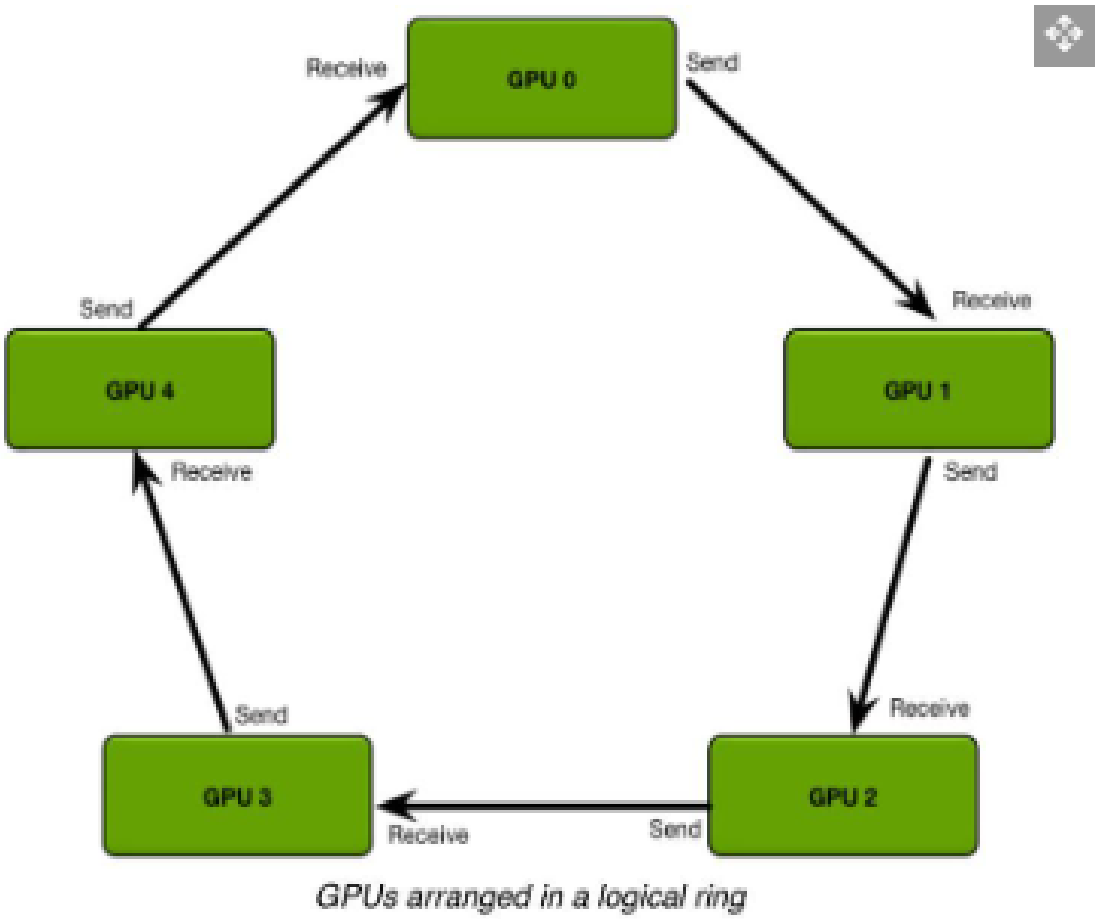

Figure 6. The above figure shows the ring all reduce Algorithm

2.1.3 Synchronous training:

In synchronous training there are many strategies and the main ones are Mirrored strategy, Multi worker Mirrored strategy and Central storage strategies

2.1.3.1 Mirrored Strategy:

In synchronous training we send the slices of data into each GPU. And each device has the replica of the full model and it is trained on a particular part of the data. They all compute a different output and gradients.

At this moment the devices are communicating with each other and they are aggregating the gradients using the all-reduce algorithm. And they do the backward pass. Then they wait for the next forward pass until the weights are updated again. It is called synchronous because at each point the weights, they trained on are similar. This strategy is also called mirrored strategy.

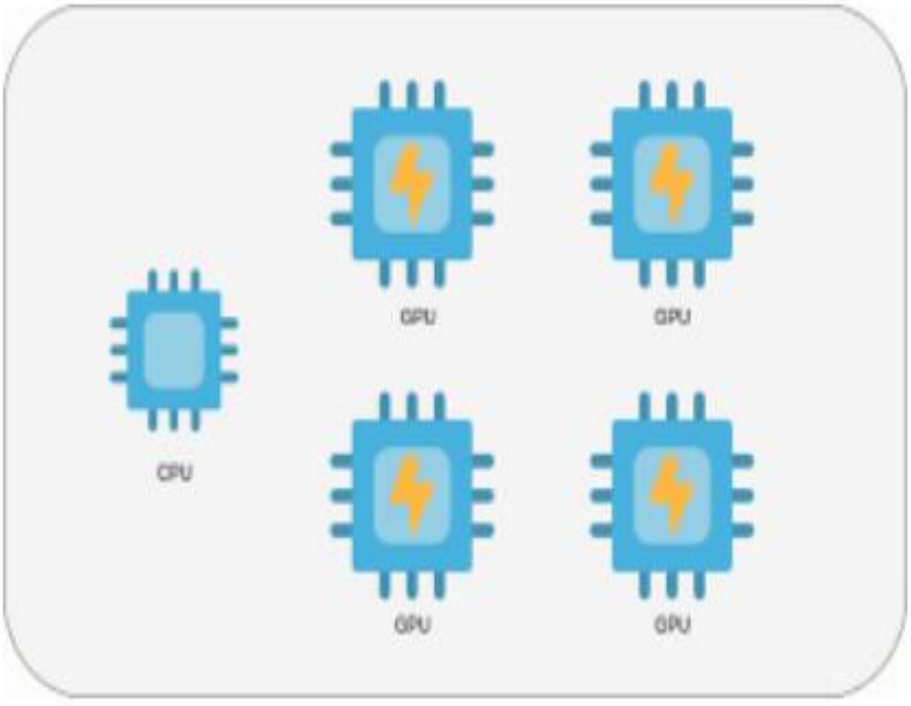

Figure 5. the above figure shows the mirrored strategy in synchronous training

2.1.3.2 Multi worker Mirrored strategy:

Similar to mirrored strategy, this implements the same thing in multiple workers/accelerator (GPUS clusters) i.e., it creates copies of all the variables across all the clusters of GPUS and then run them in a synchronous manner.

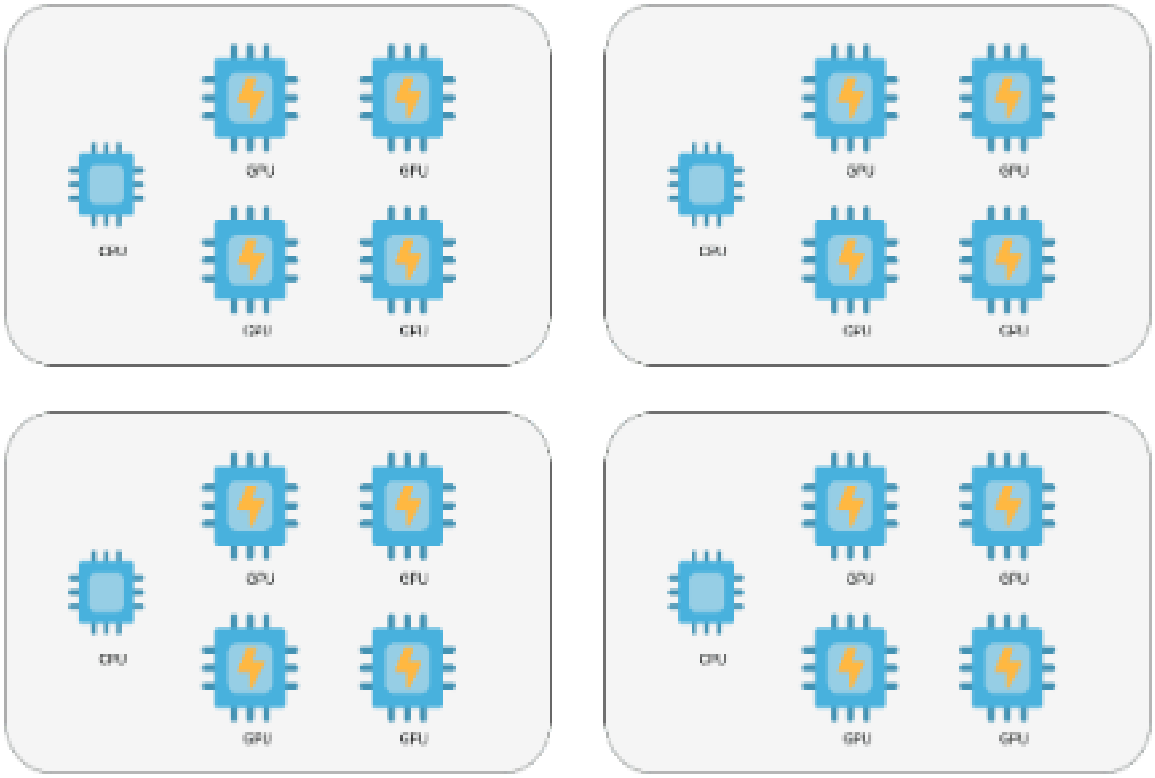

Figure 6. the above figure shows the Multi worker mirrored strategy in synchronous training

2.1.3.3 Central storage strategy:

This comes into effect when we have single machine with many GPUS. And in this case our model does not fit into the GPUS. Due to this we might not be able to store the entire model and we make the CPU as our central storage unit which stores the global state of the model. In this approach as I mentioned earlier the variables are not replicated and stored in the CPU. In this method the GPU performs the training, computes the gradient and update the weights and send them back to the CPU which again uses all reduce operation to combine them.

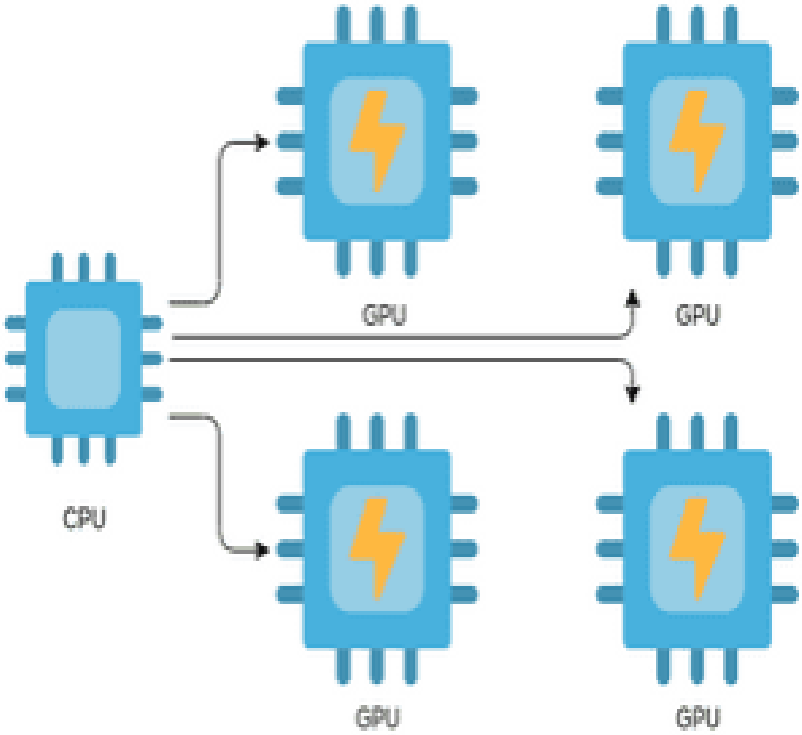

Figure 7. In the above figure we can see the Central storage strategy.

2.1.4 Asynchronous Training:

Asynchronous training can be advantages because synchronous training can have scaling issues and it can be hard to scale. And as we can see there is a possibility that any cluster or a particular GPU can finish the computation faster and it can be idle. Due to this asynchronous technique can be better as we don't have to wait for any other devices to compute. Asynchronous training can be better if we have limited capability and small unreliable devices and whereas if we have powerful communication links a synchronous approach might be better. To accomplish asynchronous training, we use the parameter serving strategy which I defined earlier in the paper. To sum it up here in the Asynchronous Training approach we aggregate the gradients and update the weights before waiting for other workers, whereas in the Synchronous Training we wait for other workers to compute and then we update the weights by calculating the gradients

2.2 Model Parallelism:

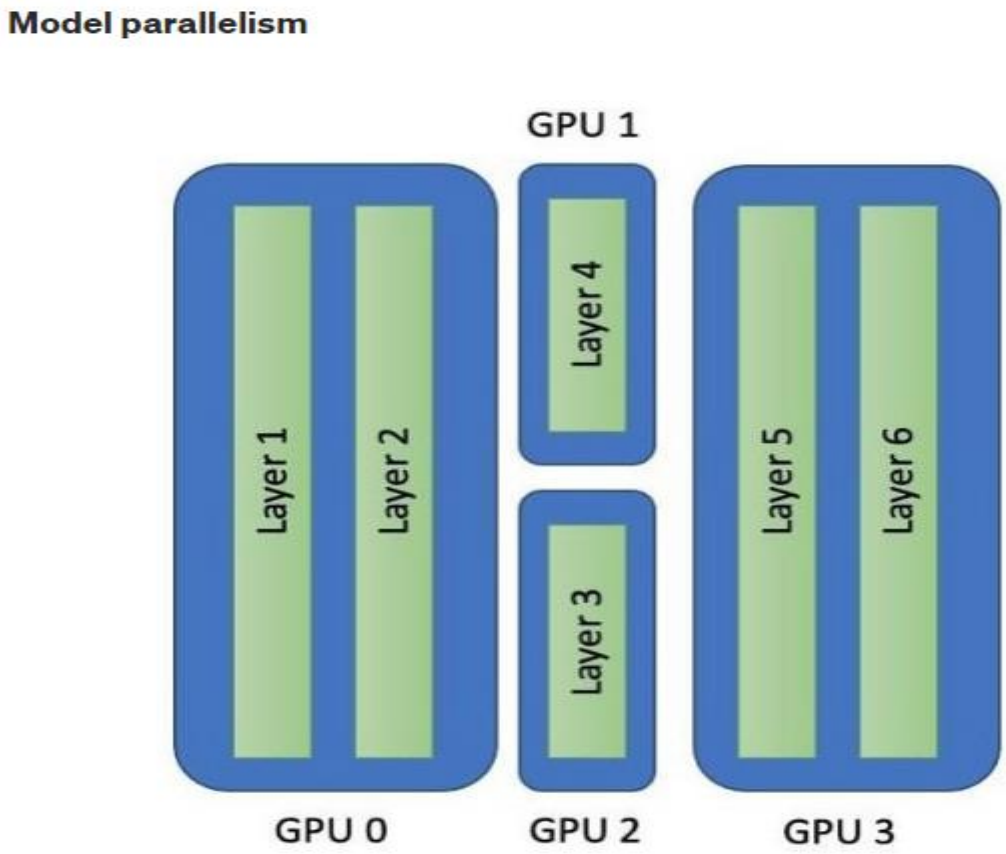

Figure 8. The above figure shows the model parallelism

In this approach the model is divided into N parts where N being number of the GPU's and each part of the model is placed on a separate GPU as we can see in the figure 8. As we can see that we are feeding the model with same data and we train each part of the model separately and we combine all the gradients using ring all reduce algorithm that I defined earlier and we do forward and backward pass and then we update the weights based on all the aggregated weights This can be beneficial for the model that does not fit in one GPU. Whereas, the major disadvantage of this model would be that it also takes different time for separate tasks to finish, this will lead to a situation where a particular process finishes fast and the GPU can be idle for some time.

In both schemes some kind of synchronization between workers is required. In model parallelism neuron activities need to be communicated, while in data parallelism model parameters (weights and biases) are communicated to ensure all models are trained evenly.

One of the biggest examples in the usage of model parallelism are the new Deep learning architectures (for Natural language processing) such as GPT-2, BERT, Transformers. Etc Although these models. These are the models that have more than 100 billion parameters (open AI GPT-3 has 175 billion parameters) and can take the leverage of model parallelism to compute things fast.

3. Comparing the results of data parallelism vs model parallelism on distributed GPUS:

To compare the performance of the model in a distributed Machine learning environment, I am choosing the U-net architecture, which is used to do segmentation (a task of founding boundaries in the images in computer vison) and it is trained in both ways and they are data parallelism and model parallelism. The results are shown below.

The Configuration of the machines are:

- Two GPU NVIDIA RTX 2070 SUPER (8 GB each)
- Intel Core i9–9900K CPU (8 cores)
- Ubuntu 18.04
- CUDA 10.2

Approach	Mean epoch time, s	Speedup, %
Single gpu	24,76	-
Distributed Data Parallel	18,976	30,5
Model Parallel	23,7	4,5

Table 2. Speedup of the learning epoch for Model Parallelism and Data Parallelism

Figure 9. The above figure shows the data and model parallelism speedup from 1 GPU to 2 GPU

As we can see with in the above figure that the task of doing data parallelism gives a greater speedup for U-net architecture than the model parallelism. Although, here we are getting a greater speedup for U-Net architecture in terms of data parallelism, we can assume that the U-Net model is not very complex and can fit into a single GPU and this makes the model parallelism not that efficient compared to data parallelism. But if the model is complex as a transformer or a complex U-NET ten we can be sure that the model parallelism can be the only way to go.

4.Conclusion:

In this paper I discussed the use of GPUS in deep learning and also the importance and use cases of distributed Machine learning (On multiple GPUS across clusters) data and model parallelism and how they can be implemented using parameter serving and ring all reduce algorithms. I also gave the examples of data and model parallelism in the modern world and with U-NET model architecture.

As the world is progressing the world is moving towards creating state of the art models by utilizing the use of Distributed GPUS by using the data and model parallelism techniques as using these techniques models such as GPT-3, GPT-2 and BERT are made by Open AI and Google Respectively and in future, this might even increase(number of training parameters) as currently we have seen the increase in the number of parameters is giving rise to more accuracy and more intelligent models and this is what Google recently did, they created a new Model called PaLM that has 540 billion parameters and it uses TPU v4 Pods(a faster version of GPU that can’t be coded using CUDA and are used for machine learning tasks(Neural Networks)). The training is done using data parallelism at the pod level across two pods and it uses model parallelism and again data parallelism in each pod. It seems to have received around 57.8% hardware FLOPs utilization which is known to be the highest for large language models at this scale.

5.Refrences:

These are the websites and research papers I have referred, to learn and write the contents on this survey.

[1] https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

[2]https://aws.amazon.com/blogs/machine-learning/parallelizing-across-multiple-cpu-gpus-to-speed-up-deep-learning-inference-at-the-edge/

[3] https://theaisummer.com/distributed-training/

[4] https://medium.com/deelvin-machine-learning/model-parallelism-vs-data-parallelism-in-unet-speedup-1341bc74ff9e

[5] https://leimao.github.io/blog/Data-Parallelism-vs-Model-Paralelism/

[6]https://www.tomshardware.com/news/baidu-svail-ring-allreduce-library,33691.html

[7]https://ai.googleblog.com/2019/03/measuring-limits-of-data-parallel.html#:~:text=When%20training%20neural%20networks%2C%20the,the%20neural%20network%20in%20parallel.

[8]https://timdettmers.com/2014/10/09/deep-learning-data-parallelism/#:~:text=Data%20parallelism%20is%20when%20you,split%20the%20model%20among%20threads.

[9] https://towardsdatascience.com/why-deep-learning-uses-gpus-c61b399e93a0

[10]https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d

[11]https://web.stanford.edu/~rezab/classes/cme323/S16/projects_reports/hedge_usmani.pdf