Applied Accelerated Artificial Intelligence
Prof. Bharatkumar Sharma
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 38
Distributed Deep Learning using TensorFlow and Horovod

(Refer Slide Time: 00:15)

Welcome back everyone. This is the 3rd lecture on Distributed Deep Learning. So, far what we have covered is the necessity of using deep learning, we also covered the part of what is the impact of different kinds of system topology. And, we saw a demo of on a machine which is called a DGXV and which is consisting of 8 volta cards.

And, we saw the impact of latency and throughput and the effect of using peer to peer transfers using NVLink versus the non-NVLink based system and how the it can get impacted. We briefly covered the usage of NGC and how TensorFlow and or what kind of support distributed deep learning.

Today, we will go slightly deeper into two frameworks. One is TensorFlow, another is Horovod and we will look at a demo of it. Followed by in the end, we will look at what kind of a problems that you can face when you start using distributed deep learning and what convergence problems can come, how the accuracy get effected and how you can solve some of those problems as well. So, let us get started and with this we are going to start with distributed training and TensorFlow.

(Refer Slide Time: 02:01)

So, as we said all of the frameworks support distributed deep learning. This one is particularly referring to the TensorFlow APIs. If I look at from a point of view of this support, basically it has multiple implementation. And, these implementations are based on two types or two axes. One is synchronous versus asynchronous and the second one is based on the hardware type that you are going to run this particular strategy on.

So, in the synchronous versus asynchronous, we have touched this in the previous lecture. In the synchronous training all the workers train over different slices in a synchronous fashion and the gradient aggregation happens at every step. While, in asynchronous all the workers work independently training over the input data and updating the variables asynchronously.

Again, the second axes is hardware platform. TensorFlow supports multiple hardware. It supports GPU, it also supports multi-distributed training across CPUs and also on Cloud TPUs which is a Tensor Processing Units by Google.

(Refer Slide Time: 03:22)

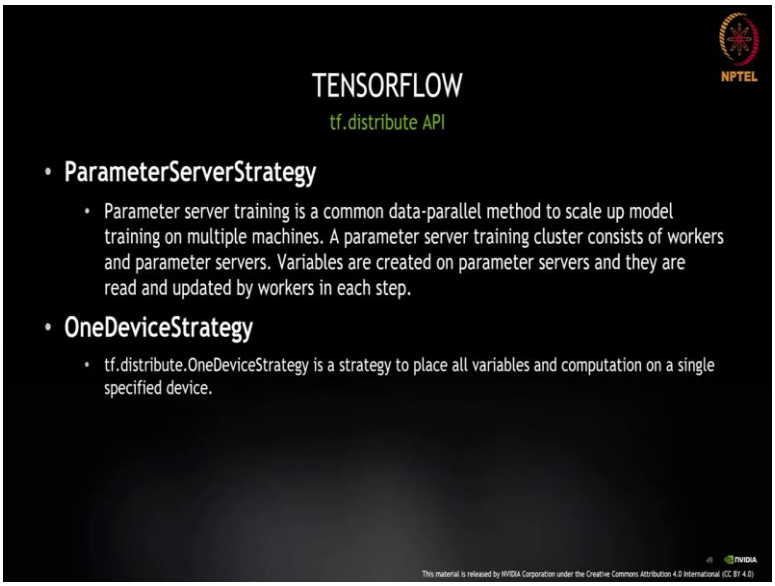
So, TensorFlow supports total of 5 strategies that we can use on the GPUs. There are more strategies once that once can be used for accelerator more like for Google TPUs and all. But, we are not going to touch upon them in this particular session, but let me go through the ones which are useful for the accelerators.

So, the first one is the MirroredStrategy, I have kind of highlighted some of the keywords. First, is it is synchronous in nature. It creates the replica per GPU. So, one of the key things its part of the data parallel and you can see it will create a replica per GPU. And, each variable in the model is mirrored and the variables are always kept in sync and that is what mirrored strategy works on.

MultiWorkerMirroredStrategy as the name says, the keyword there is multiple workers. So, it is very similar to the MirroredStrategy, but it creates multiple worker each with potentially multiple GPUs. It creates similarly to mirrored strategic copy of all the variables in the model on each device and across all the workers. There is another strategy which can be used which is CentralStorageStrategy, where the variables are not mirrored.

The updation of the variables or the main copy of it is kept or placed on the CPU and the operations are replicated across local GPUs. So, all so, if there was suppose only one GPU, in that case it will reside only on that particular one GPU.

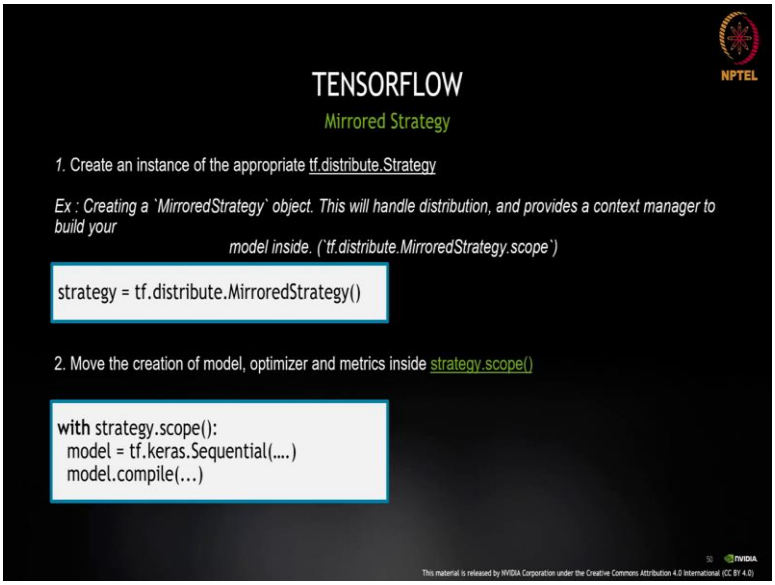
(Refer Slide Time: 05:13)



ParameterServerStrategy, we had covered to certain extent last time and it is common data parallel method to scale up. And, it kind of helps because all the workers are not talking to each other, while they fundamentally talk to a parameter server and it consist of multiple workers. So, the variables are created on the parameter servers and they are read and updated by the workers in each and every step.

Another one is OneDeviceStrategy, where you can define where to place all the variables, on which particular single device you would like to put them on.

(Refer Slide Time: 05:52)



So, let us look at the steps which are required inside TensorFlow to convert your normal training to a distributed training. So, the first step in this particular example that we are showing is first of all we are using a mirrored strategy and here you can see the first thing is we are defining the strategy ourself.

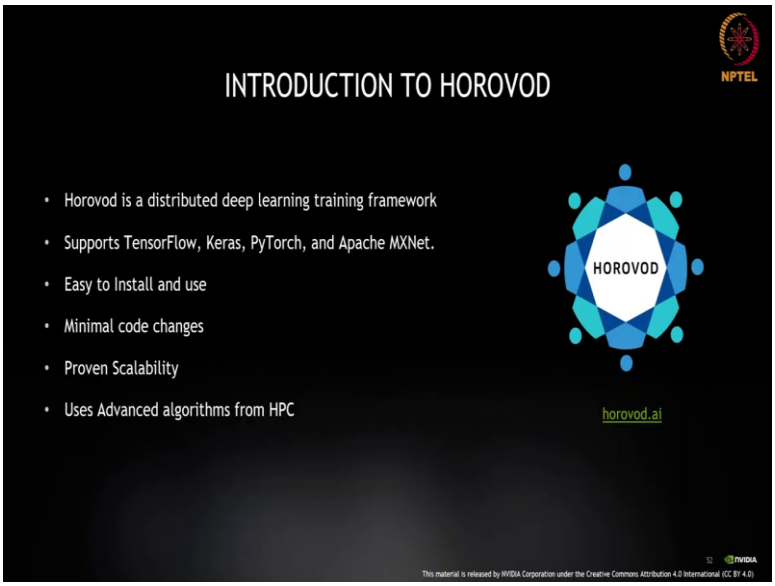
The second step in this particular session, in this particular approach is to bundle your existing model within the scope of that strategy. So, you can see here that we are saying within the strategy dot scope, I am going to define my model. So, that model is defined within the scope of that particular strategy, that you have chosen.

If you are not using the distributed strategy, if you are doing it on a normal one GPU; this step is not required right. You would directly start creating your model layers. So, we can say it is quite simple to do this. And, the TensorFlow kind of makes the scaling very very easier by only a few lines of code. We will look at the code into more details later also.

(Refer Slide Time: 07:08)

But, before showing you the hands on part of it, let me also introduce you to another framework which is Horovod. And, then we will look at the demo of the Horovod and TensorFlow both of them.

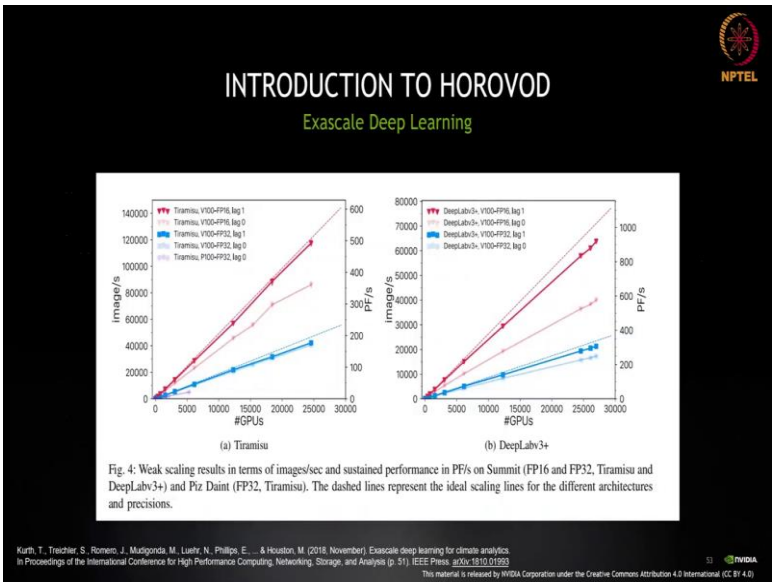
(Refer Slide Time: 07:25)



So, Horovod basically is a library for distributed deep learning. It was initially a tool for internal use at Uber which was later open source for the community. Apart from the features which we have, it also have additional tools such as Horovod timeline which can be used to analyse the time taken for each operation and optimize accordingly. So, we can see what all features it has. It supports in the back end, it supports TensorFlow, it supports Keras, PyTorch, Apache's.

So, you can think of Horovod as a meta framework over existing frameworks like TensorFlow, PyTorch and all. It is very easy to install. There is almost minimal code changes that you have to do. And, Horovod has been used for multiple exercises and for even doing exascale kind of computing. It uses some advanced techniques to do really good scaling or it has different types of optimizers which are used in those scenarios.

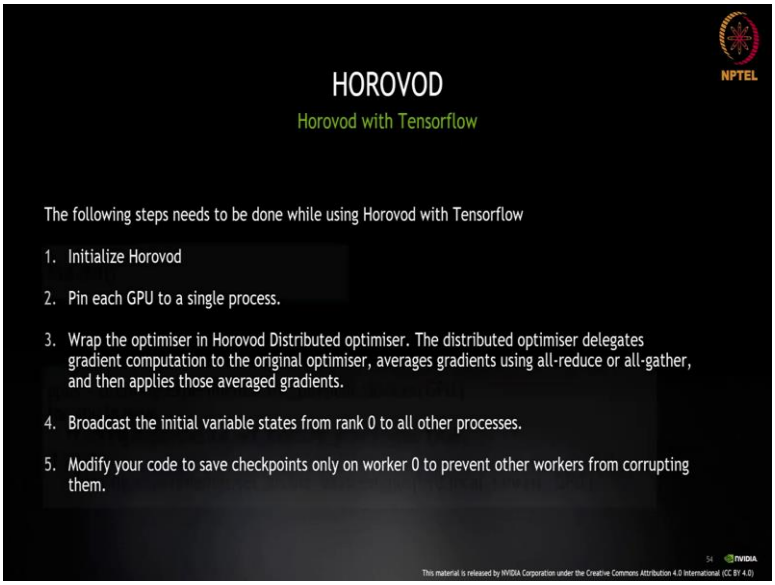
(Refer Slide Time: 08:30)



So, Horovod has been used widely and is proven to be scalable across as I said 1000s of GPUs and nodes. This is from a paper Exascale Deep Learning for Climate Analytics and the model was trained on the summit supercomputer, with that let us now move to scaling a training script using Horovod.

But, the key point again that I would like to have here is that this is like a petaflop simulations and the number of GPUs across which it was run. And, you can see it is scaling pretty well.

(Refer Slide Time: 09:07)



Horovod actually takes it its I would say motivation from something called as MPI. MPI stands for Message Passing Interface. It is one of the most prevalent distributed computing model in the high performance computing environment. Who are working in molecular dynamics, computational fluid dynamics and all those kinds of high-performance computing domain, if you want to scale or if you want to distribute your work across multiple nodes; the model which is used is message passing interface.

And, Horovod uses the MPI model where each multi-CPU thread would be launched, each of them corresponding to one GPU which is very different from the TensorFlow case, where one CPU thread was used to handle all the GPUs. This was the design choice by the Horovod team.

And, it is very very critical to understand this because this is one of the fundamental reason why Horovod can scale really well. The steps which are required in the code to make use of Horovod is; obviously, like MPI we need to do initialization. Initialization is key to finding out different aspects behind the scenes and Horovod kind of sets the context there. Then, you pin each GPU to a single process.

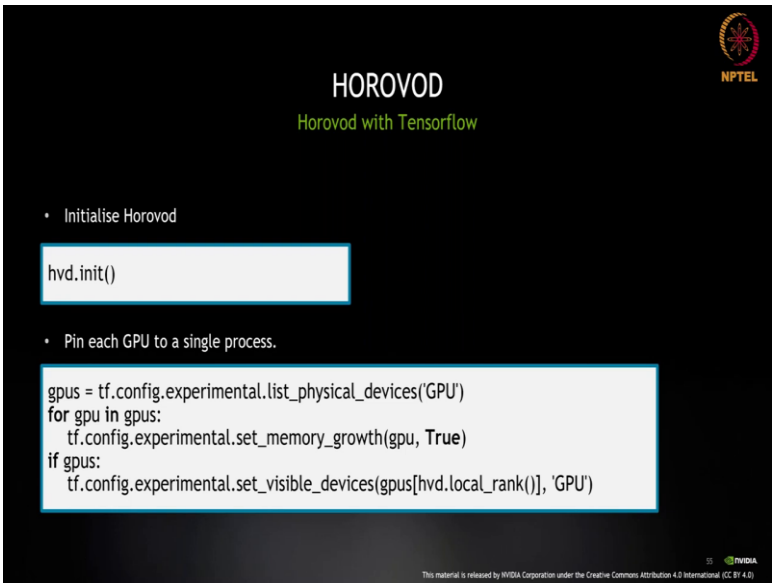
So, you would define multiple you at runtime you will define how many processes you want and then for each process you will pin a particular GPU. So, you can pin one or more GPUs to a single process. Like TensorFlow again you are going to wrap the optimizer also inside Horovod distributed optimizer.

So, you are going to wrap the existing optimizer onto the Horovod distributed optimizer which will be responsible for calculating or delegating the gradient computation to the original optimizer. So, it does the gradient calculation to original optimizer whichever is there, but it takes the responsibility of averaging the gradient across multiple processes using MPI features, like all reduced, all gathered which are very well known in the MPI community.

And, it is proven to be working really well with different kinds of topologies using InfiniBand like structure and all. And, then applying those average gradients and once it is done with that. So, it is basically broadcasts the initial variable states from rank 0 to all the other processes. And, if you want you the code, you will have to change certain code to safety checkpoints and other things only by worker 0.

So, you do not want all of the workers or all the processes to save the checkpoint because the same state is maintained by everyone. So, there is no need of everyone writing to the same writing the same checkpointing values and corrupting them in short.

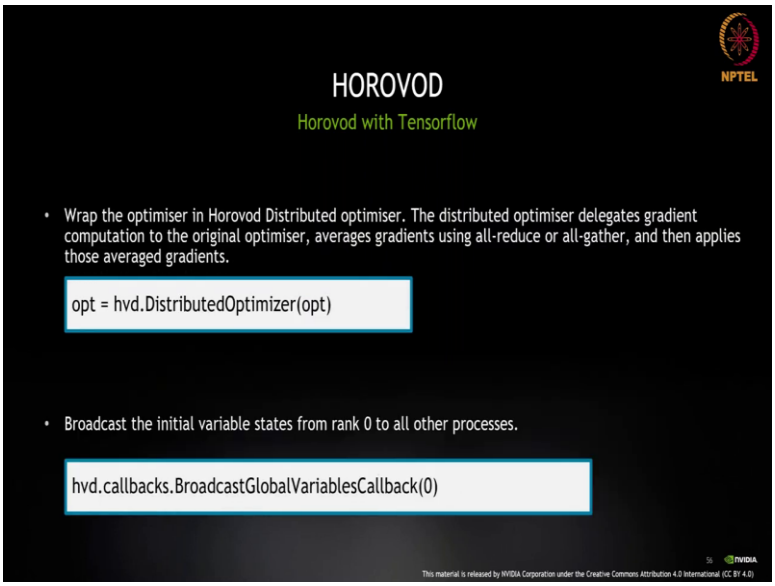
(Refer Slide Time: 12:19)



So, from a code point of view this is how it will look like. We first import and then we will initialize the Horovod for the entire script. We then are going to pin the GPUs to a single process and you can see here the code which is we are first of all getting all the listed devices available. We get that particular list and for every GPU we are basically based on the rank of that particular process, like the process when you are launching said Horovod.

If you are launching two process, there will be process 1 will get a rank of 0, process 1 will get a rank of a process 2 will get a rank of 1. So, based on the rank which every Horovod process gets, you would do set visible device which means every process will get its own GPU. In TensorFlow, the strategy API would handle the updates, reading of the weights and making identical updates.

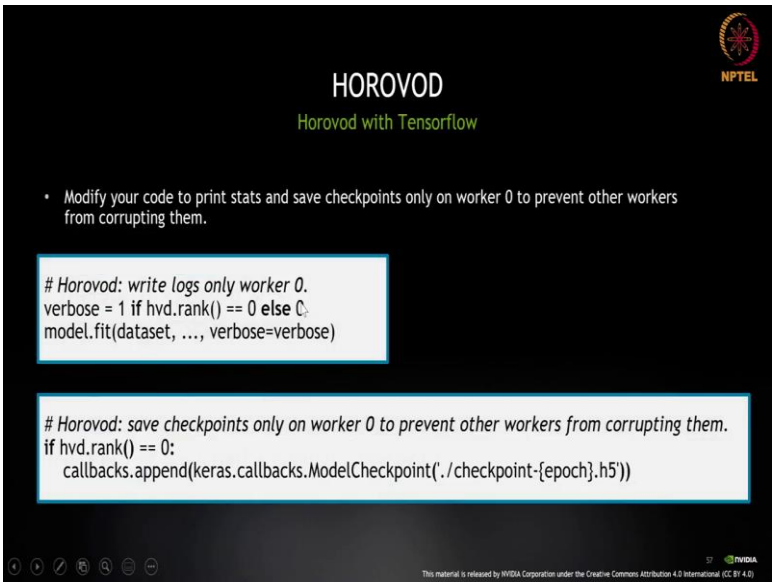
(Refer Slide Time: 13:37)



But, in Horovod we can use the optimizer of a distributed optimizer method, updates are identical and are kept in sync. So, if you see here, we can choose our own optimizer and then wrap that optimizer inside the distributed optimizer. So, the current optimizer will continue to work, but the distributed optimizer as we said before, responsibility there for it is to make sure that it will basically take over and do the all reduce and all gathered part of it.

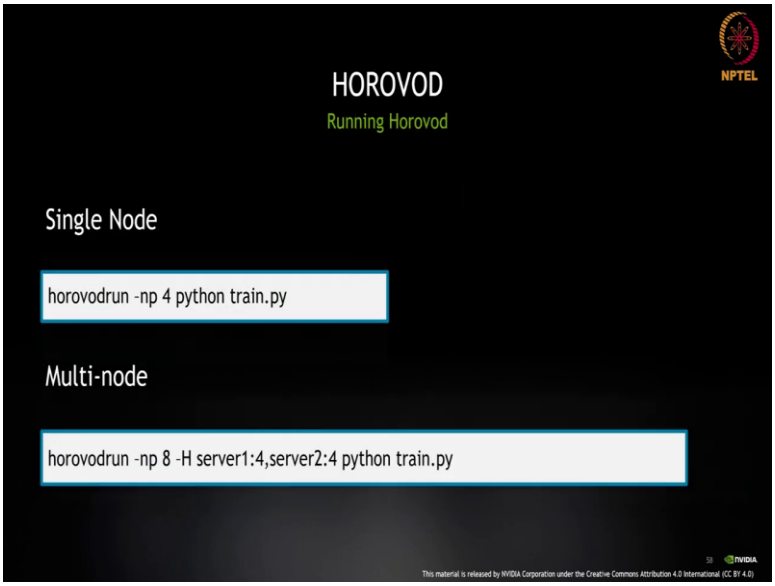
And, then do the averaging of the gradients and then you can also define callbacks from rank 0 to all the other processes.

(Refer Slide Time: 14:18)



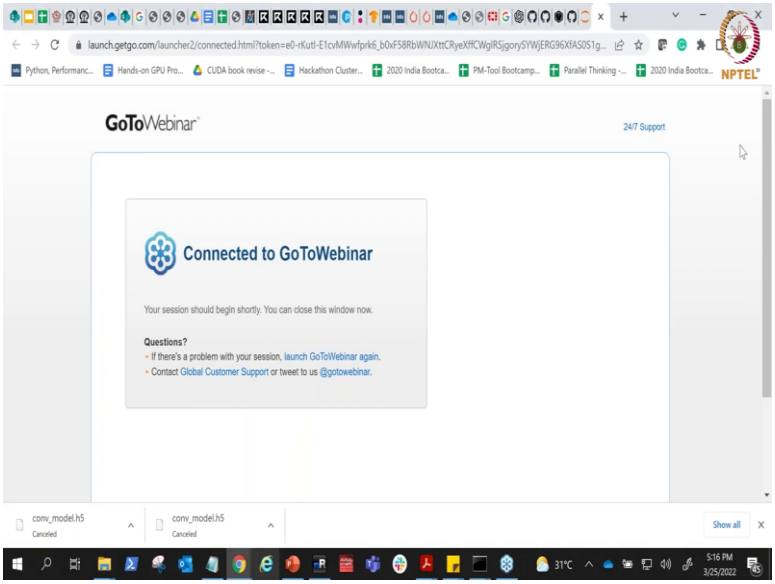
The last, but not the least we already said to you that you can save checkpoints and we would ideally like only one worker and worker 0. So, that they are to prevent the other workers from corrupt corrupting them. So, you can say only if my rank is 0, then only I am going to basically save my checkpoints via basically a hook.

(Refer Slide Time: 14:49)



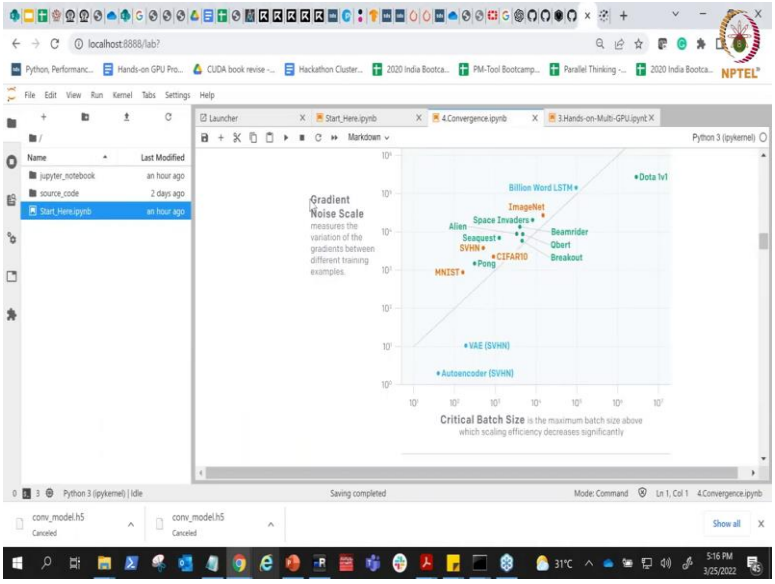
So, this is what kind of completes the flow of the Horovod part of it.

(Refer Slide Time: 14:57)



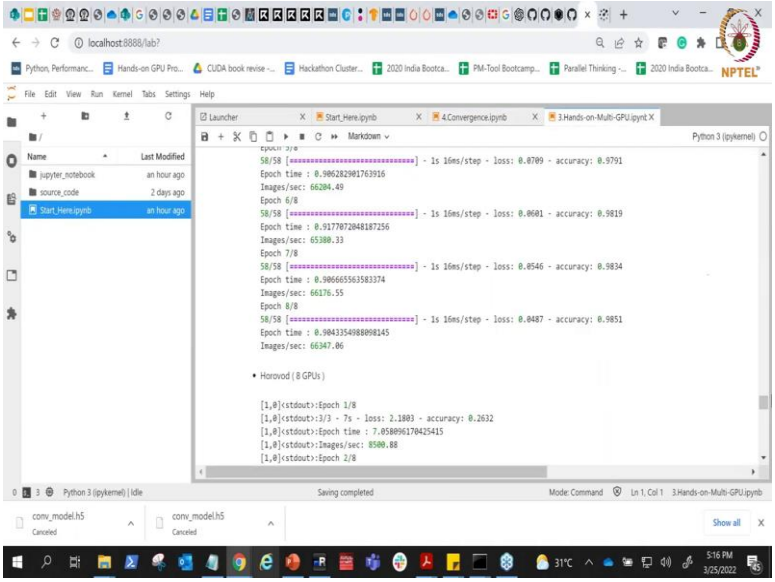
Let me take you to the live as into the code.

(Refer Slide Time: 14:59)



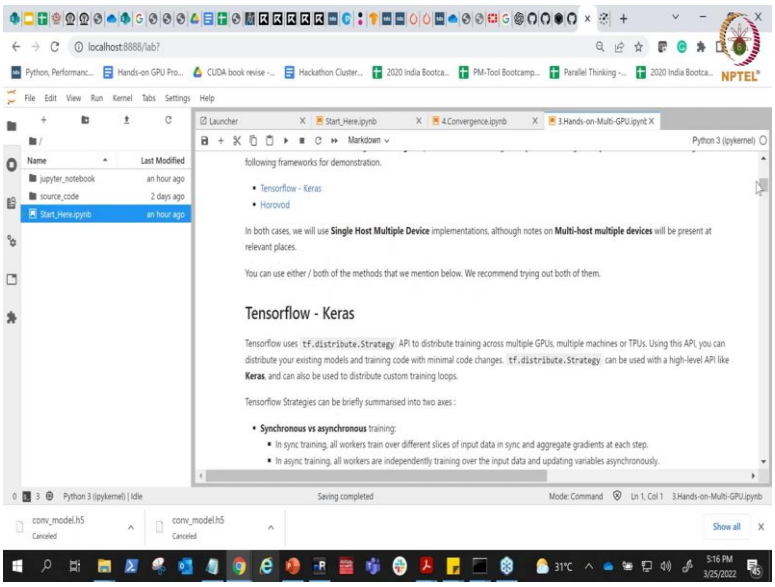
So, that you get an idea of how it looks like.

(Refer Slide Time: 15:01)

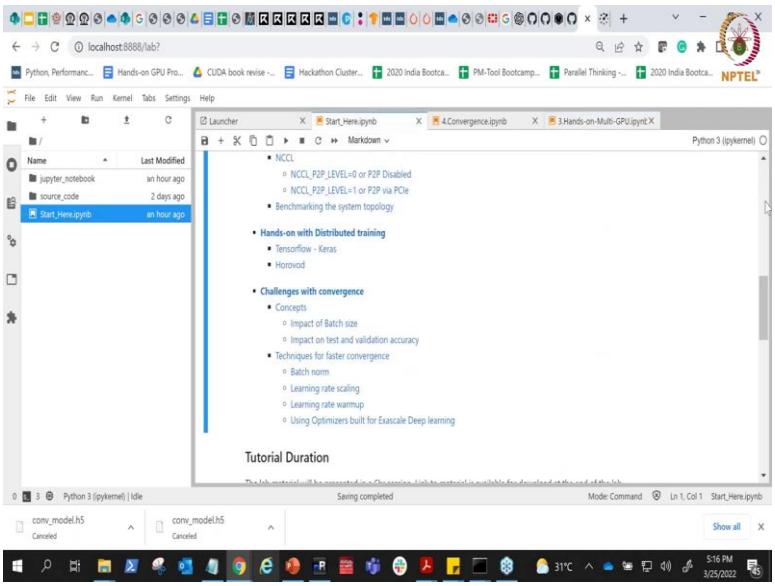


So, we are going to continue from our session which we did last time also.

(Refer Slide Time: 15:08)

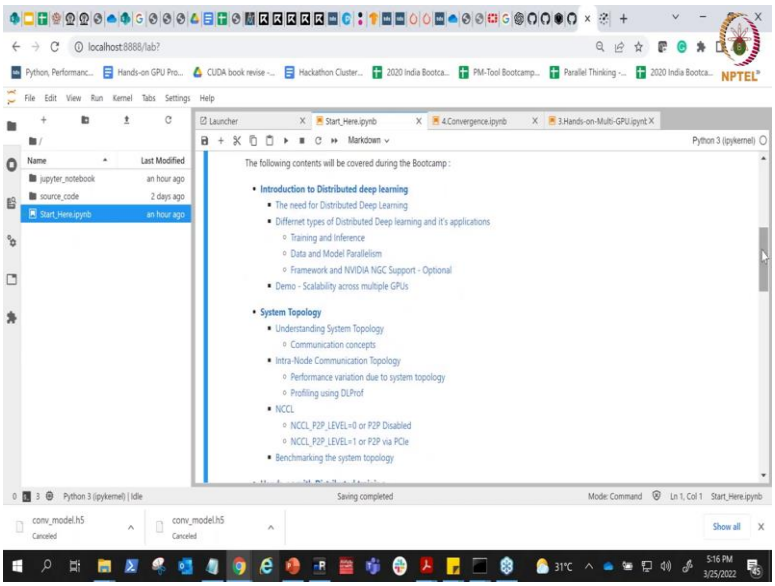


(Refer Slide Time: 15:11)



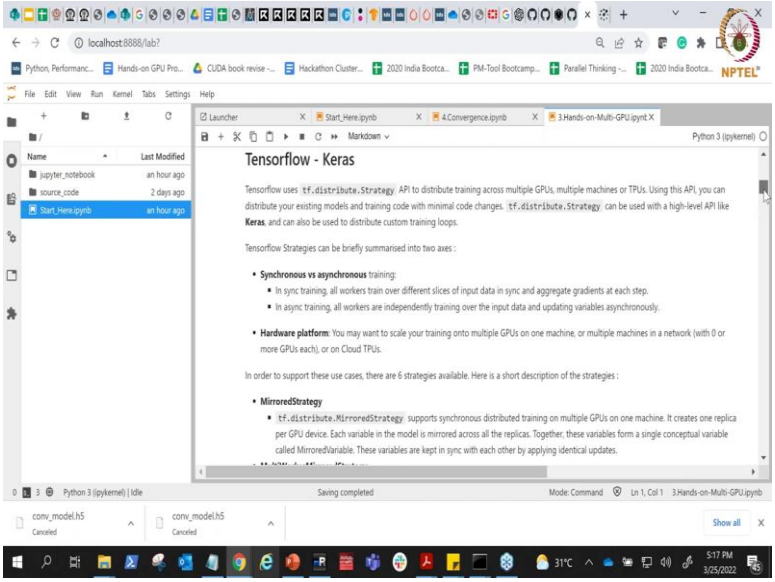
So, if you remember I launched the start here notebook.

(Refer Slide Time: 15:16)



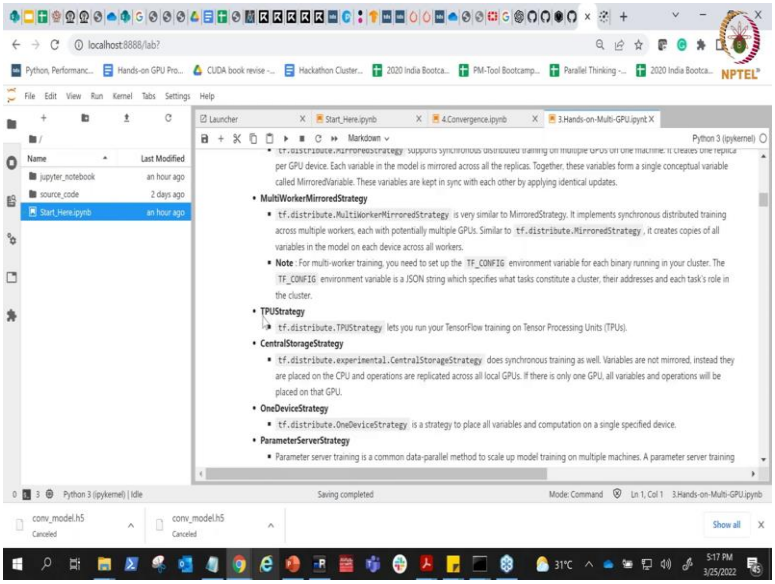
In the start here notebook, we basically went through the part of the concept or the necessity of distributed deep learning. Then, we saw the system topology and what was NCCL, how it was impacting the performance. And, today we are looking at the TensorFlow and Horovod implementation. So, again I am not going to repeat the theory part of it which I just explained in the method.

(Refer Slide Time: 15:39)



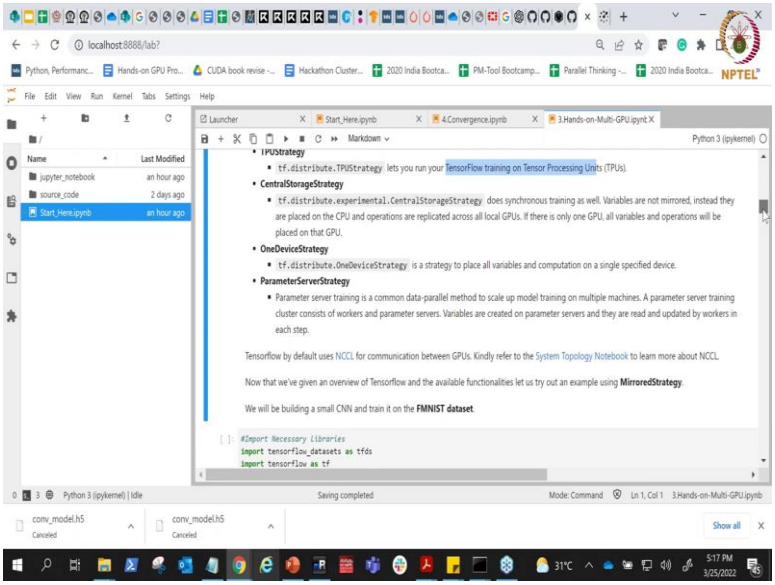
So, the all of these notebooks where they exist would be provided to you on the slack channel. The link would be provided. These are all open source notebooks. You can go ahead download them and try them in a multi GPU environment, you would require at least two GPUs to get a feeling of how you can distribute the work across multiple GPUs; within a single GPU environment these labs will be of almost of no use.

(Refer Slide Time: 16:05)

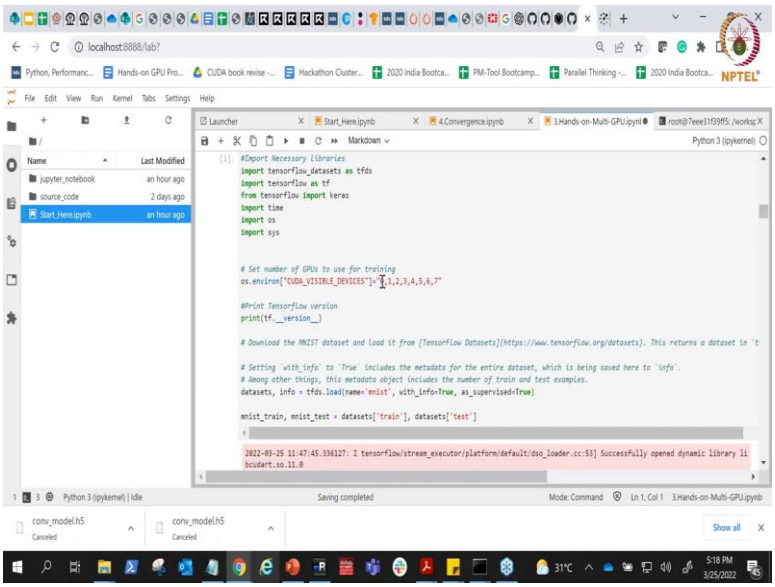


As we showed you earlier there are different kinds of strategies, like what we did not show you was there is a different strategy for TPU which is Google's TPU. And, you can run it and set it in case you are using any other kind of a hardware as well.

(Refer Slide Time: 16:21)

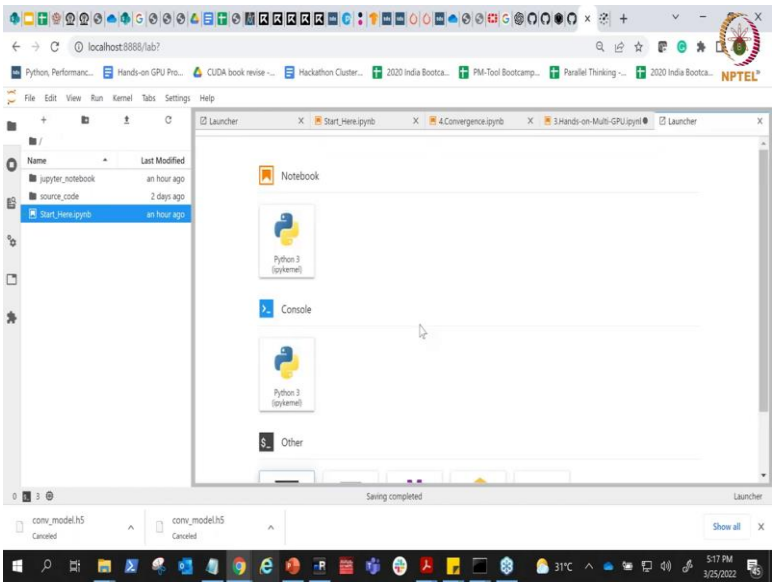


(Refer Slide Time: 16:23)



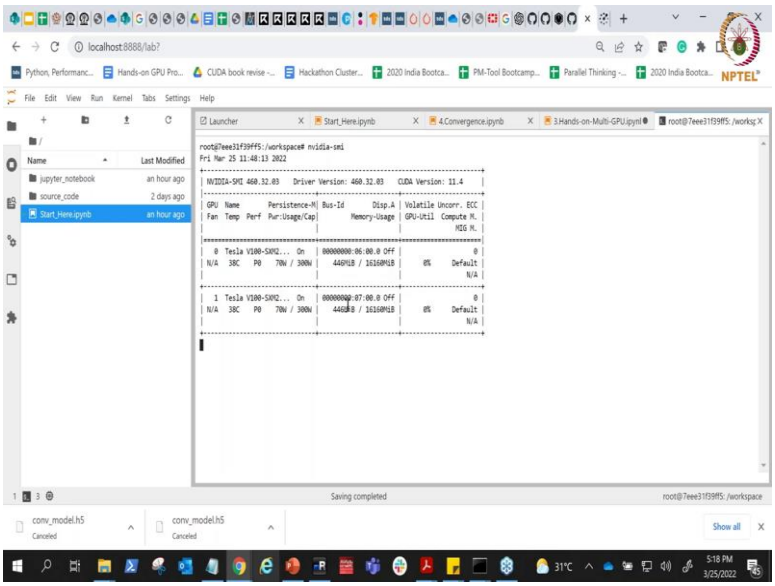
So, let me just start running the code one by one. So, like in the previously we are importing the tensorflow, then the keras. If you remember last time if I will show you the terminal ones.

(Refer Slide Time: 16:39)



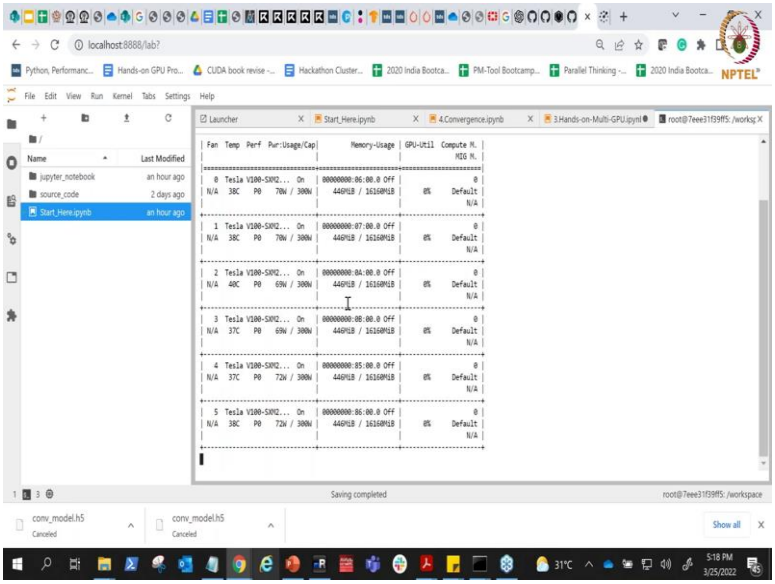
This particular yeah.

(Refer Slide Time: 16:49)

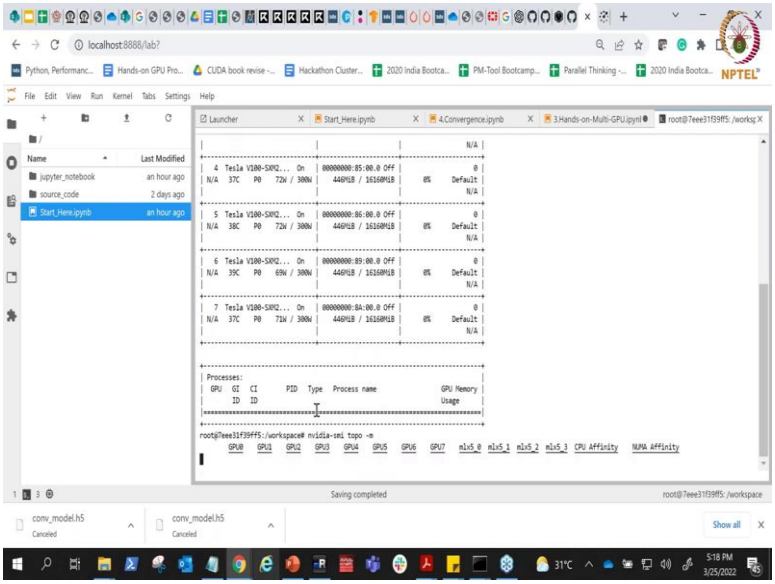


So, if you remember this particular machine, it has basically 8 GPUs.

(Refer Slide Time: 16:58)

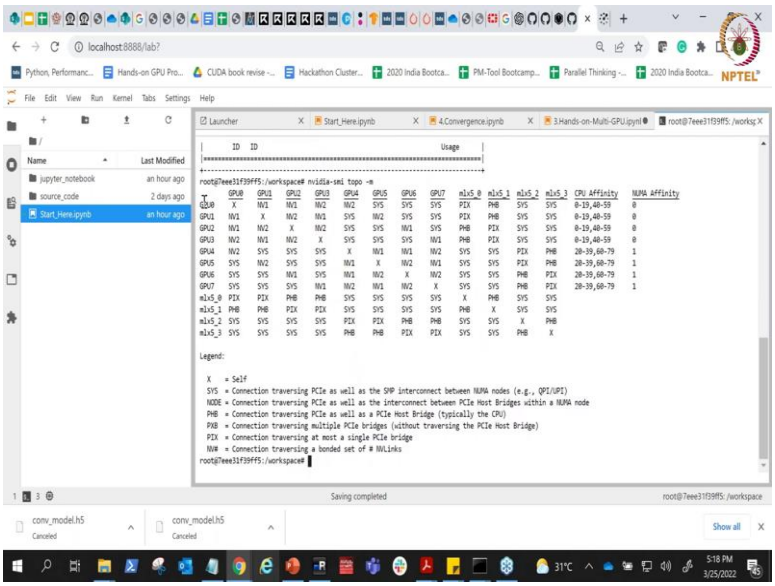


(Refer Slide Time: 16:59)



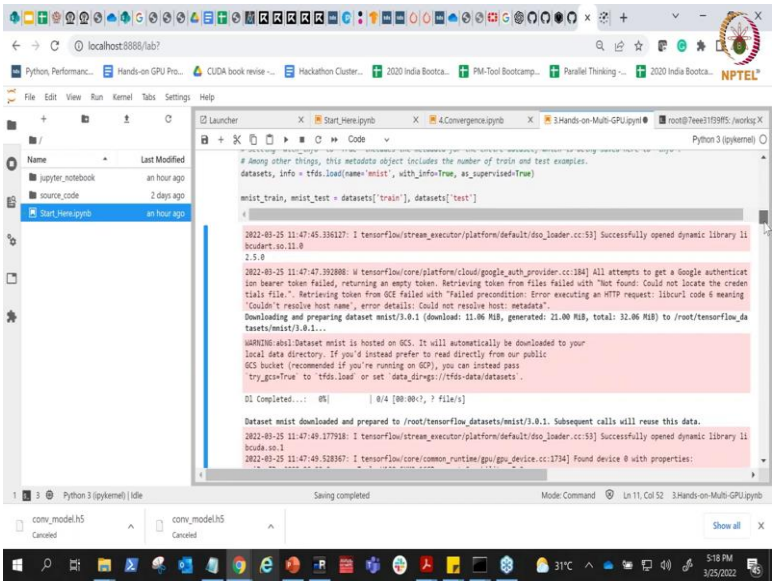
And, you can see here 1 0 1 2 3 4 5 6 7 and we also talk about how to see ok. So, there is a lag which is happening nvidia-smi topo -m will basically show you the system topology as well.

(Refer Slide Time: 17:20)



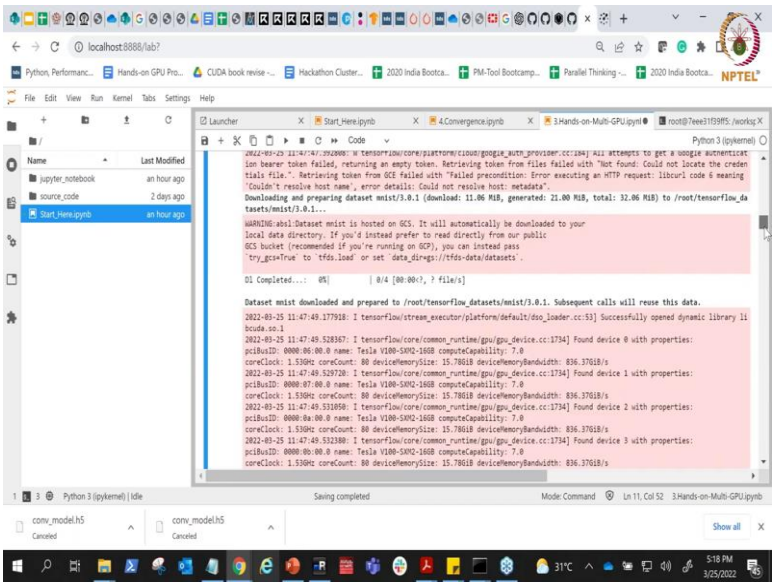
So, we are in a machine where we have 8 GPUs. So, here you can see that I have set the CUDA_VISIBLE_DEVICES to all of the available GPUs.

(Refer Slide Time: 17:37)



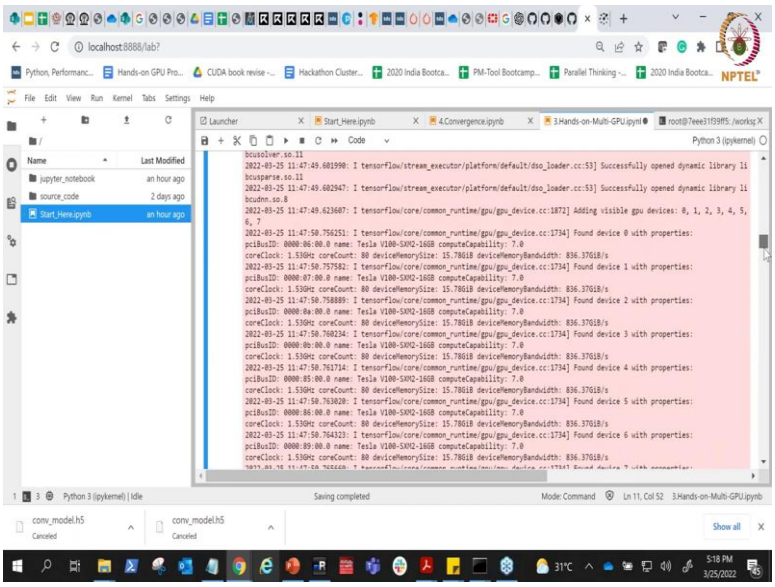
And, we are again running the fashion mnist data set.

(Refer Slide Time: 17:40)

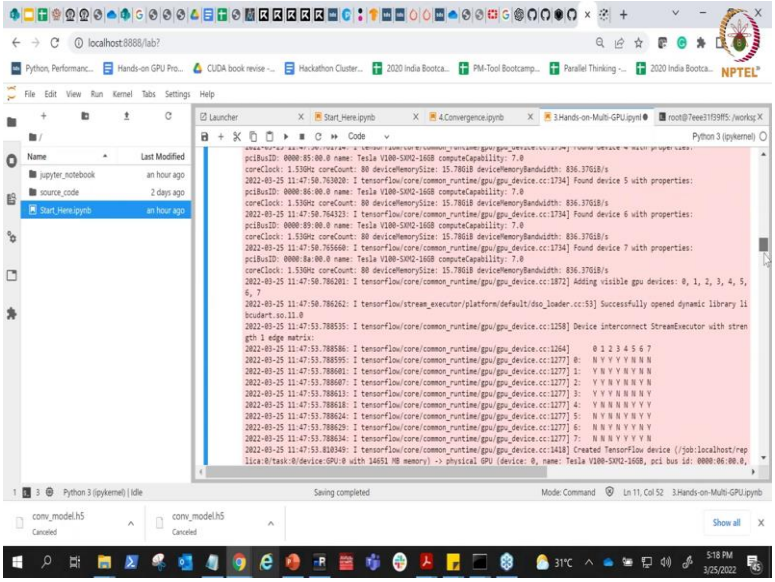


So, we do not need to worry about that part.

(Refer Slide Time: 17:41)

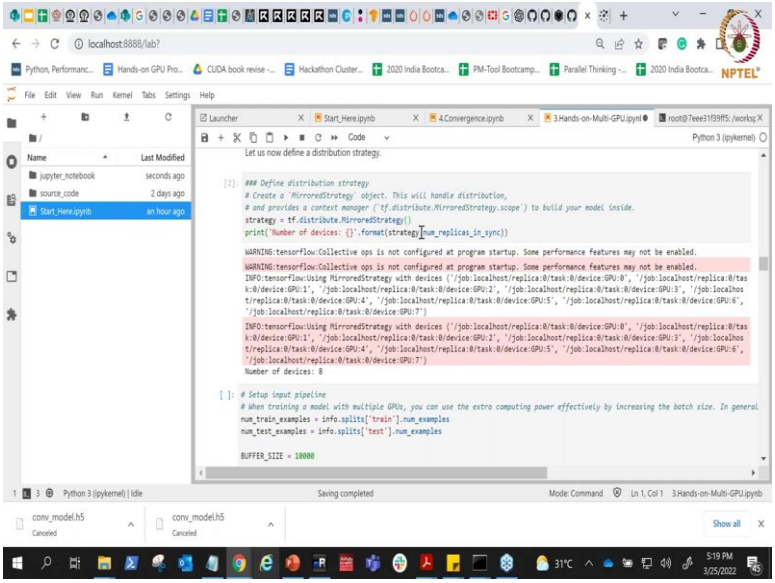


(Refer Slide Time: 17:42)



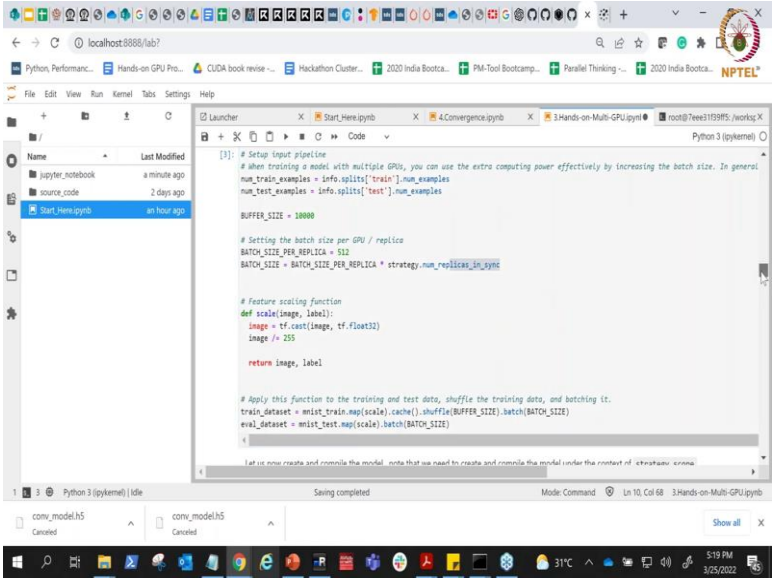
So, it just downloads the fashion mnist data set.

(Refer Slide Time: 17:46)



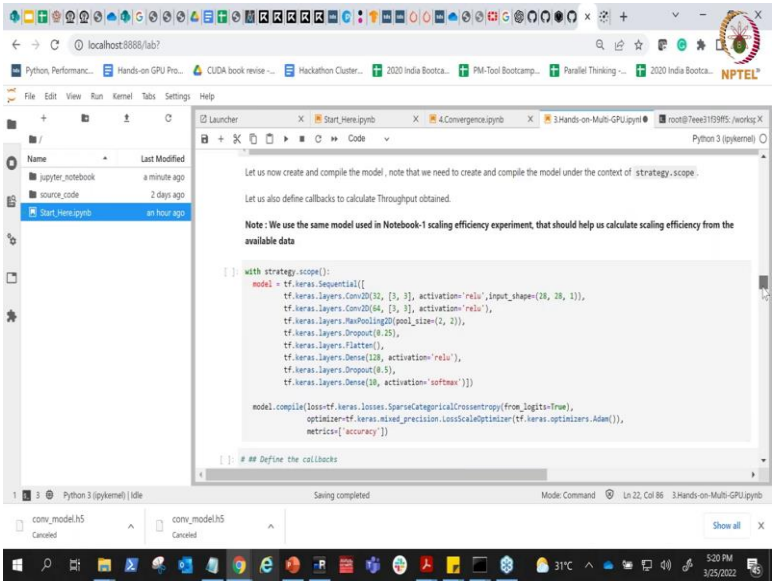
So, the first step that we discussed in the in the TensorFlow was to define the strategy. Here, as you can see here we are seeing tf dot distributed MirroredStrategies. We are adopting the MirroredStrategy here. And, you can see what it will do is that once you run it, it will try to you can use number of replicas in sync. You would what all GPUs devices, how many of them are visible to it can be seen. By this you can see here it can see almost a all of the devices which is the 8 devices which are available.

(Refer Slide Time: 18:22)



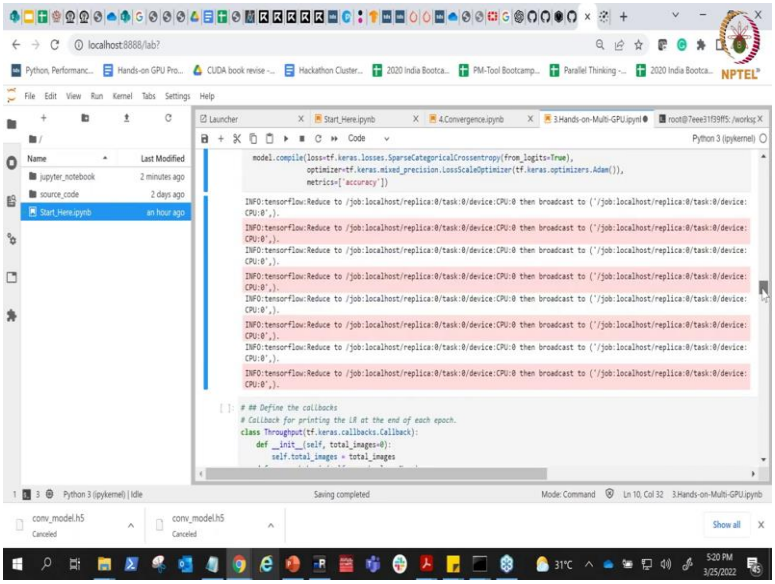
After this, let us run the code again. So, this particular code is basically a code to just do some additional task of doing defining the batch size. For each and every replica you are going to have your batch size and then you are going to basically just give it. So, there is nothing so, great about this code.

(Refer Slide Time: 18:43)



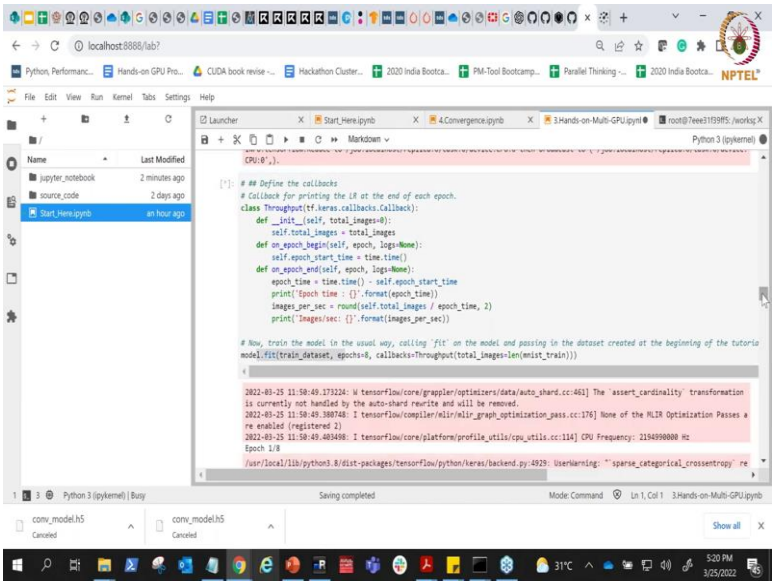
You can just look at the code and just, here we are trying to shuffle the data for the training and the batch set.

(Refer Slide Time: 18:53)



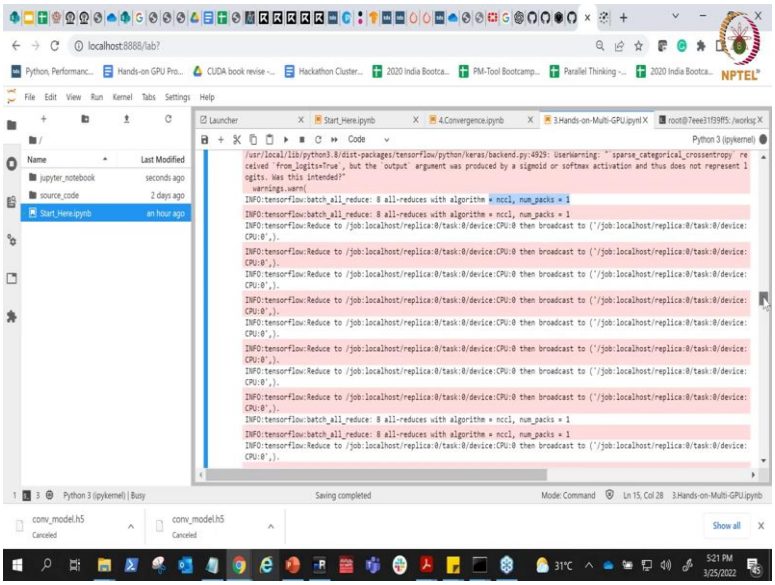
After you are done with this, you would see that we have to define the scope and you can see here within the scope here we are creating the model. So, the model is a traditional convolution neural network model. And, the main idea here is to get it inside the scope and then within you can define your hyper parameters like what symmetric and what is the optimizer and all those things here. So, here we are using the Adam optimizer in this particular case.

(Refer Slide Time: 19:25)



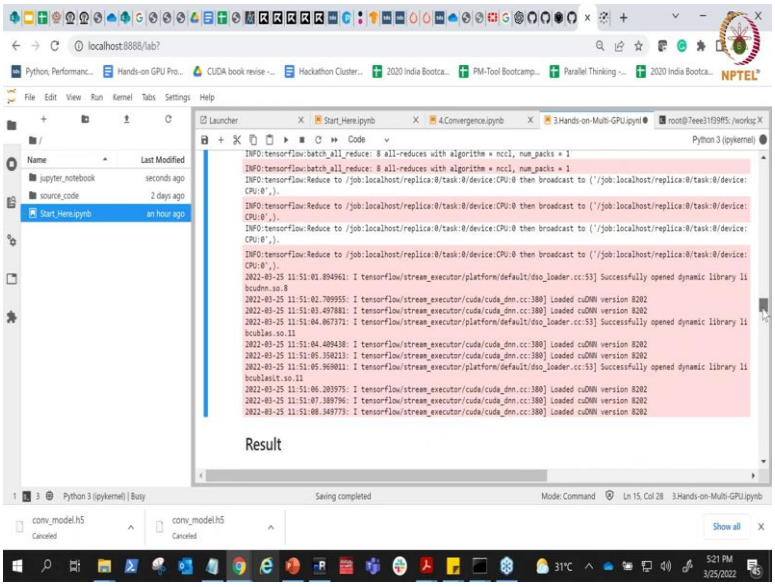
Once you are done with that, you can basically call the model dot fit function and it is ideally supposed to run across all of the GPUs.

(Refer Slide Time: 19:34)

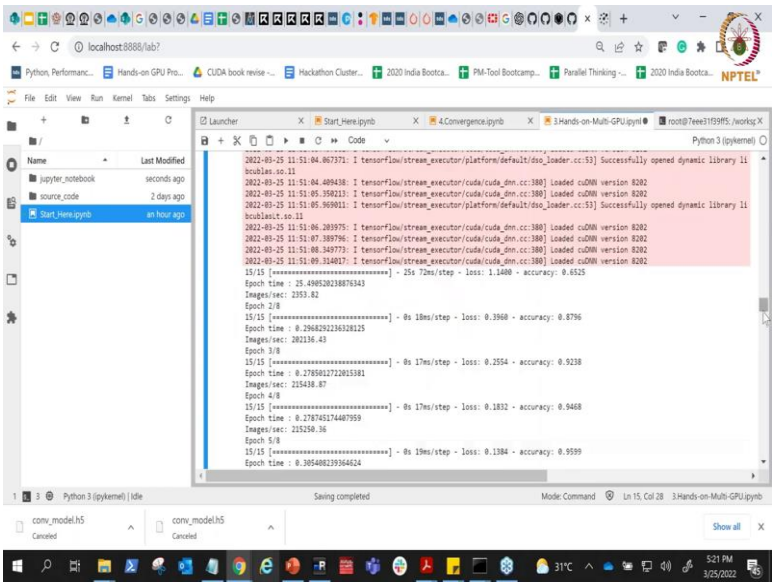


So, again you can see here it is using nccl. So, it will still use nccl as we have shown you last time.

(Refer Slide Time: 19:43)

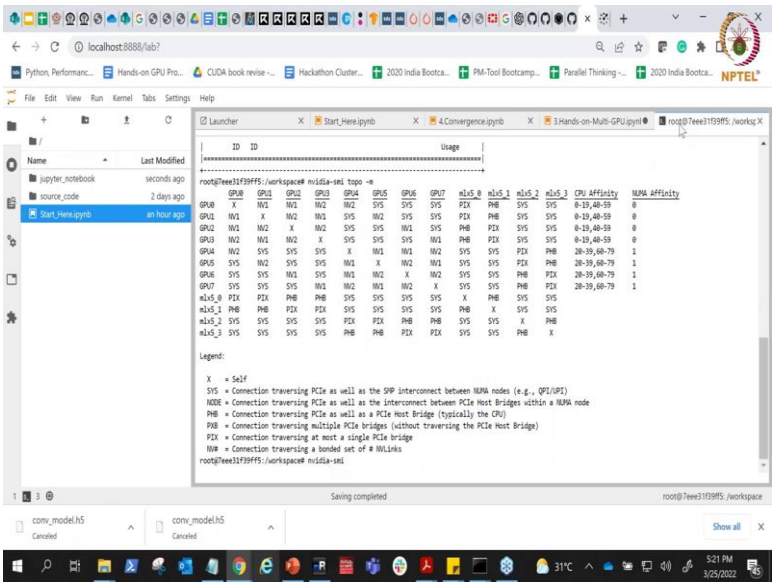


(Refer Slide Time: 19:53)

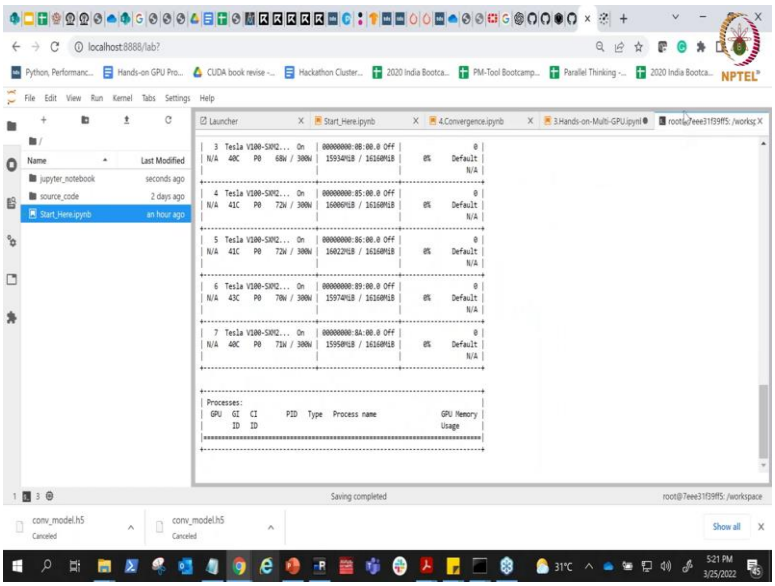


And, it will start the job and.

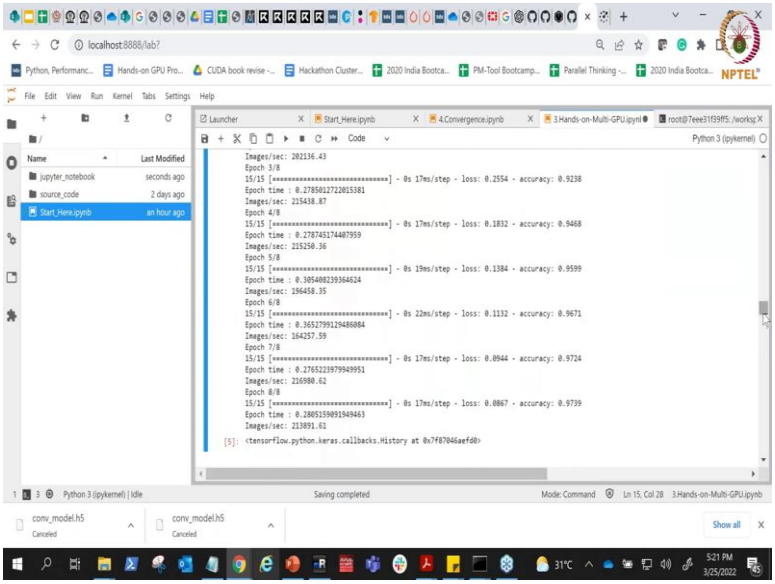
(Refer Slide Time: 20:01)



(Refer Slide Time: 20:06)

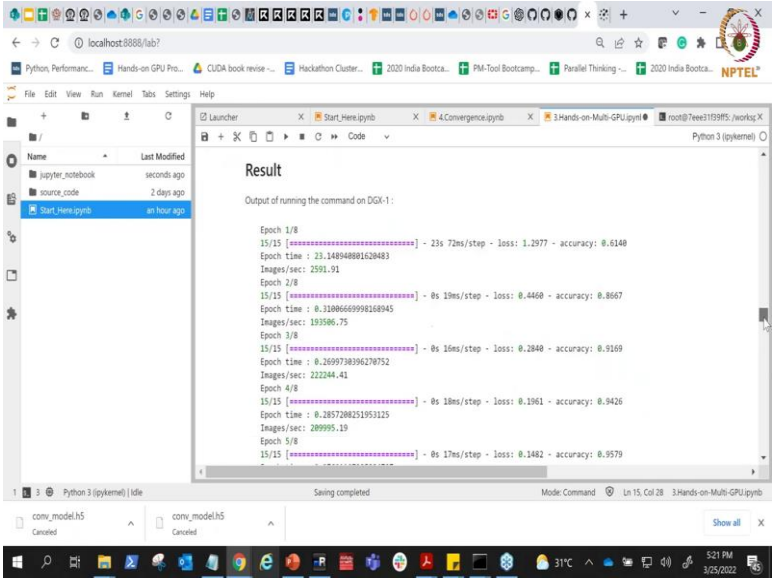


(Refer Slide Time: 20:13)



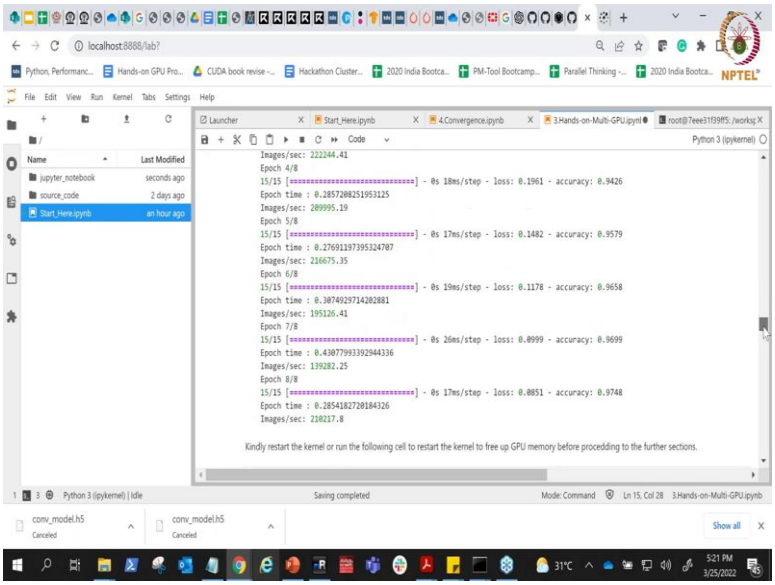
So, you can see here it has already finished and the images per second is almost 213891.

(Refer Slide Time: 20:23)



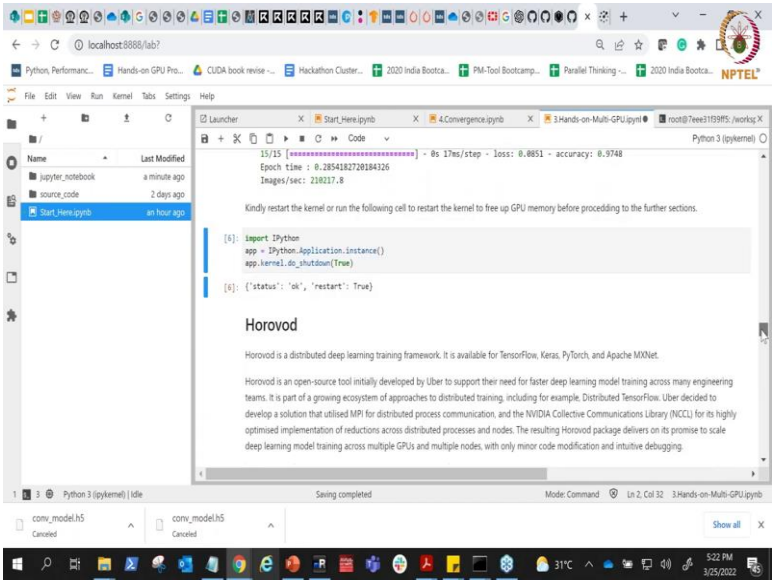
So, it was quite easy to run it on a TensorFlow environment, practically a code does not change so much.

(Refer Slide Time: 20:29)



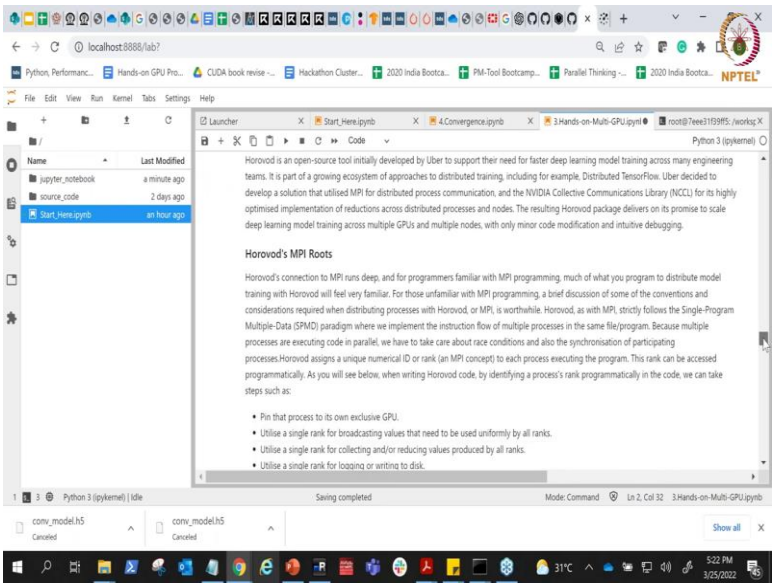
I am going to just run this particular cell.

(Refer Slide Time: 20:31)



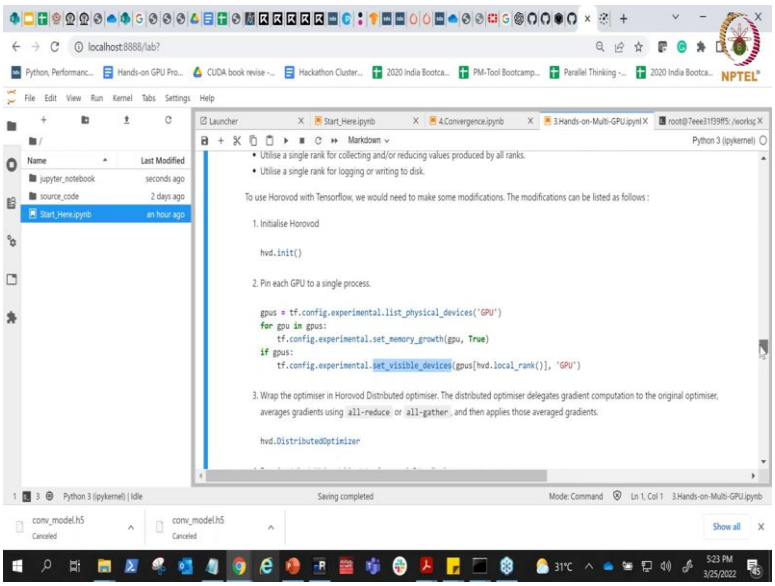
Because, TensorFlow kind of does not read the allocation and it will not allow us to run the Horovod simulation. So, I am just shutting down the kernel, just to make sure we have freed up all these space.

(Refer Slide Time: 20:47)



So, so this was a very quick demo of how to use TensorFlow. Now, we are going to move towards Horovod. So, Horovod again I have told you that it takes its motivation from MPI which has been the most scalable distributed framework, that we have seen in the high performance computing domains.

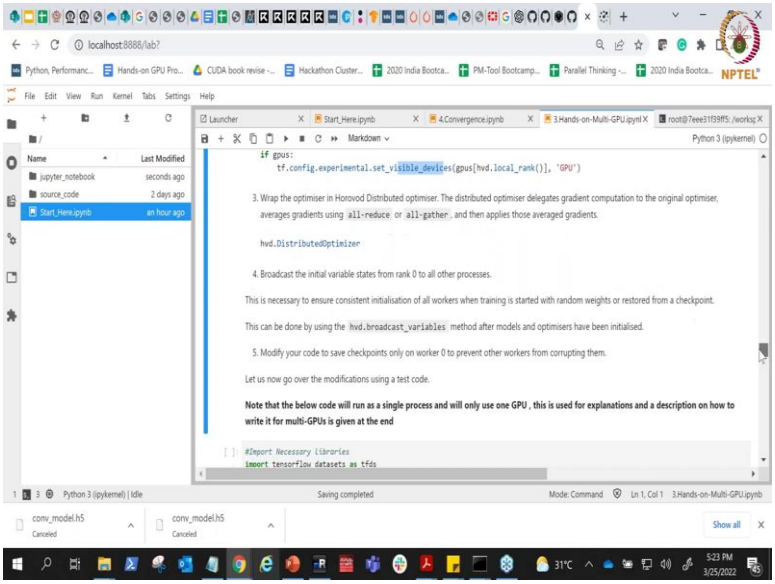
(Refer Slide Time: 21:09)



But, it basically has to make sure that it kind of adheres to certain principles like pin the process to its own exclusive GPU. So, every process is going to pin itself to a particular GPU. It will utilize a single rank for broadcasting values that needs to be used uniformly. So, a single rank needs to be used for broadcasting the values. Utilize a single rank for collecting the values and reducing, because without this there would be a problem in terms of the accuracy.

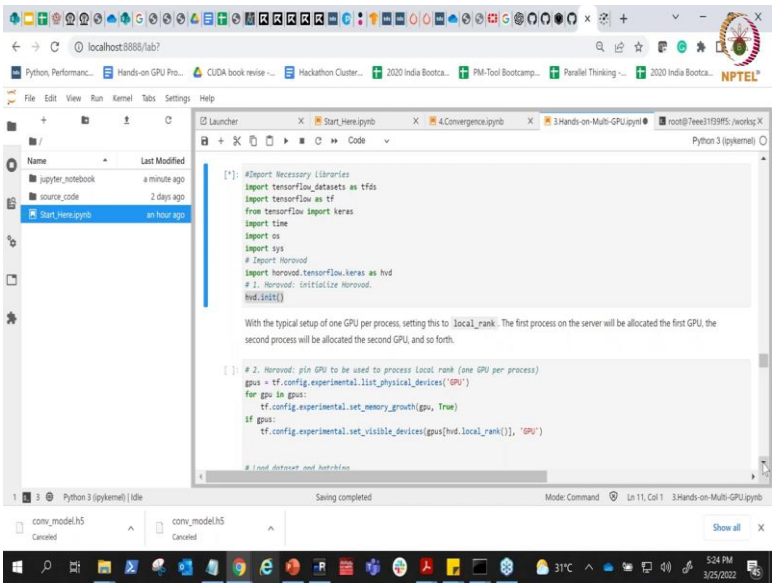
So, utilize a single rank for also logging and writing to the disk. So, generally this is done by your a primary process or the rank 0. And, I will show you what do I mean by that right. So, you can see here you are first doing the initialization. After the initialization basically, the second thing which we told is you have to set the set visible device.

(Refer Slide Time: 22:07)



And, it is based on the rank. So, the rank 0 will set the g will use the GPU 0, rank 1 will use GPU 1, that is how it uses. And, then you will define your own distributed, you will wrap the optimizer onto distributed optimizer.

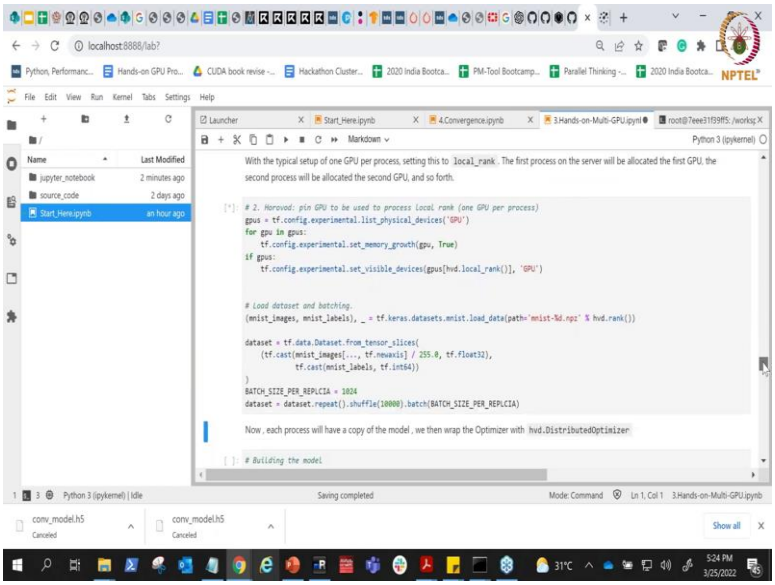
(Refer Slide Time: 22:26)



So, let us look at the actual code here. So, again you can see we are using Horovod, but we are using within horovods still tensorflow. As I said that Horovod is basically a higher level framework. It is it can work on different backends like PyTorch and other methods also. Here, we are still using horovod, but over tensorflow. But, if you write

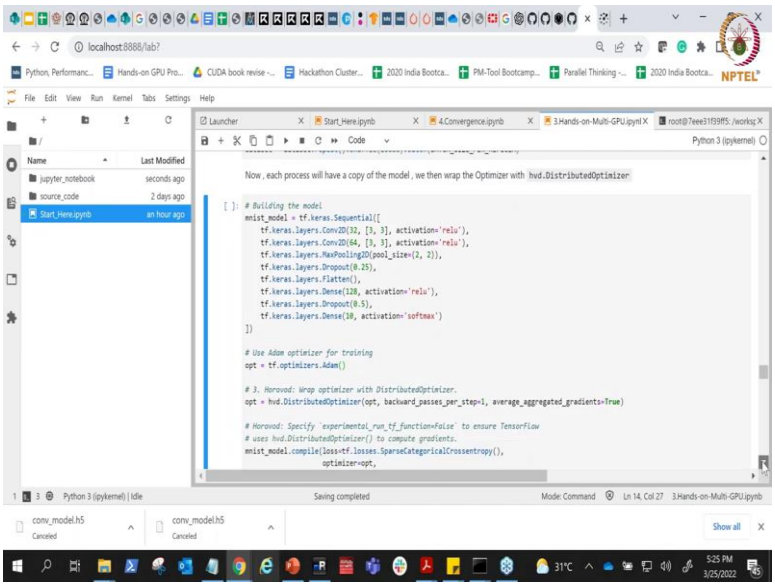
code in Horovod, it can use any of the back ends. So, let us use initialize, the first thing that we are doing here is to initialize the Horovod. So, till the time it is running anyhow.

(Refer Slide Time: 23:14)



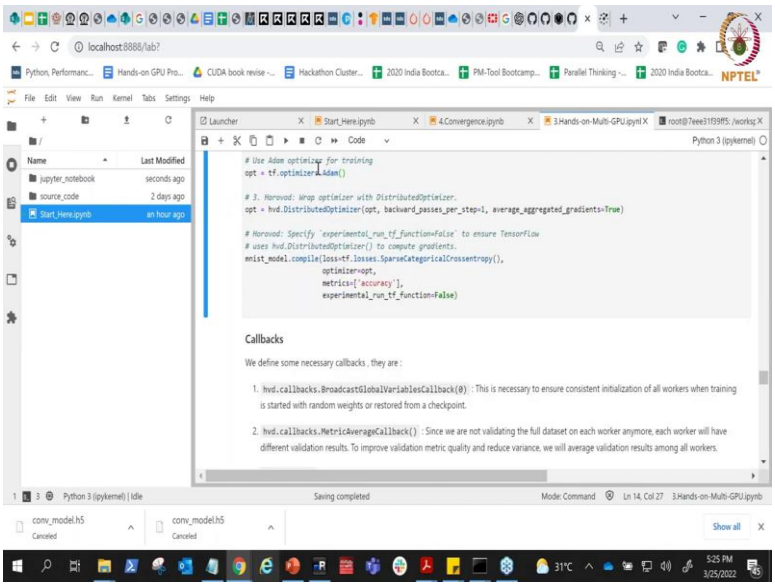
The second thing which we told was to set the visible device. This is very very critical, otherwise you will all end up using one single GPU. And, we are doing the same thing, we are creating the data sets and creating a replica of it to. So, let it run, it is taking an certain amount of time.

(Refer Slide Time: 23:35)



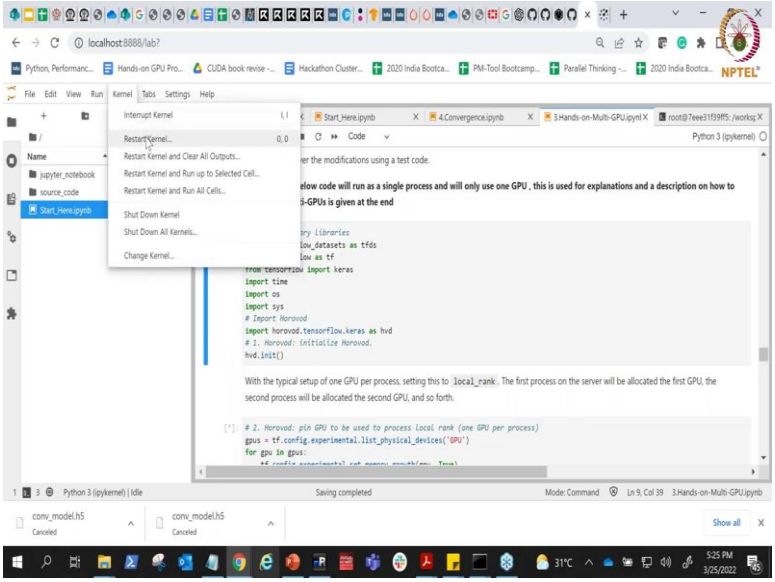
After you are done with this, as you can see you are creating your normal model which is in this particular case we are building the model with couple of layers convolution, MaxPooling, Dropout. And, then we are also defining a normal optimizer, here in this case we are using the Adam optimizer for doing this.

(Refer Slide Time: 24:04)

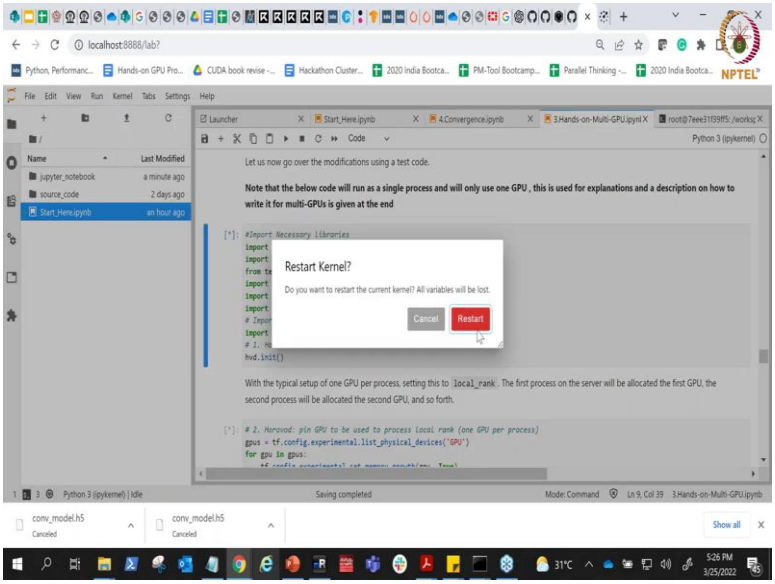


After we are done with the Adam optimizer, we are doing Horovod dot DistributedOptimizer. So, we are wrapping the existing optimizer within the DistributedOptimizer and it takes certain flags like how many backward passes you need per step or average aggregated gradients. Do you want average gradients or not and we have set this to True obviously. And, then we compile the overall code. Let me check, it is still not finished.

(Refer Slide Time: 24:40)

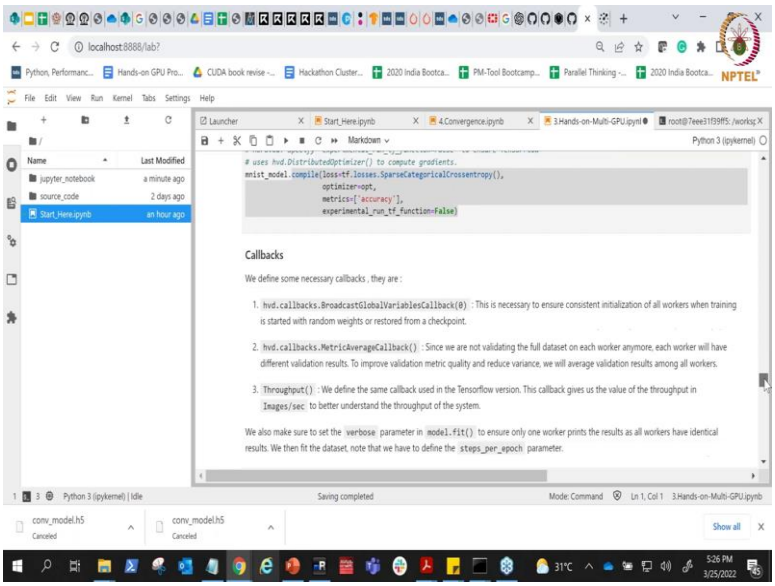


(Refer Slide Time: 24:42)

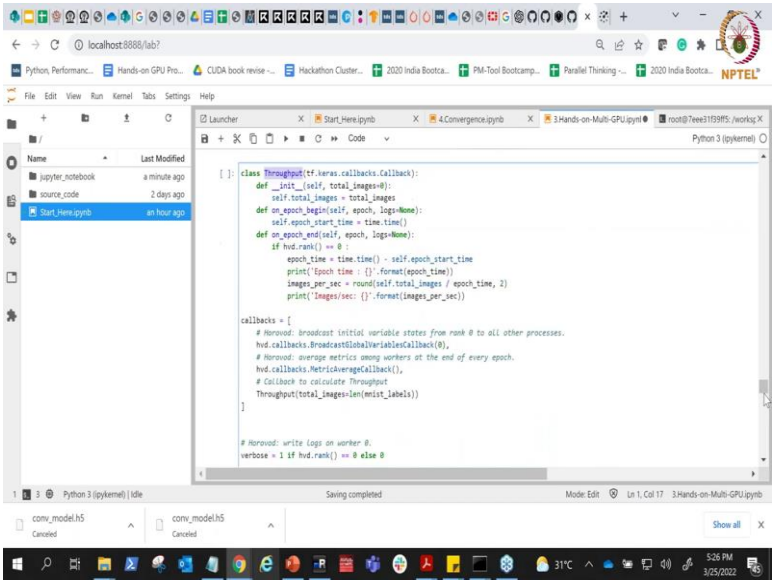


Let me just restart the kernel, after you are done with this.

(Refer Slide Time: 24:54)

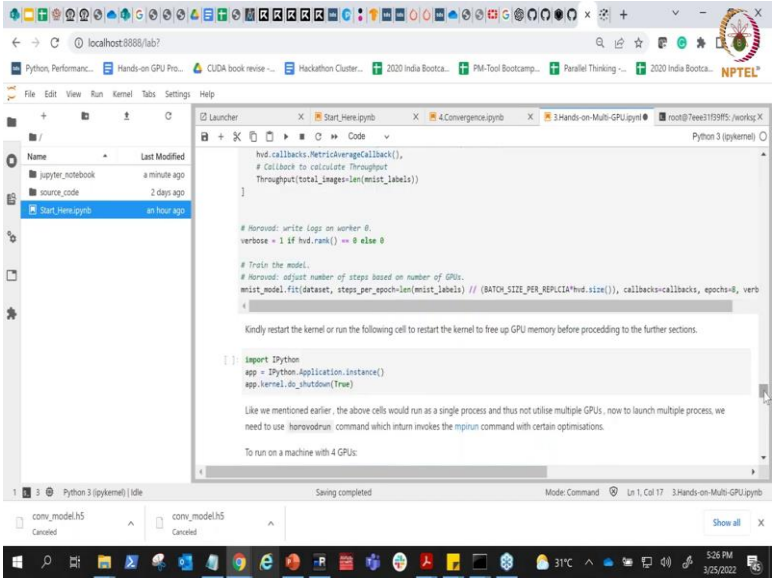


(Refer Slide Time: 24:55)



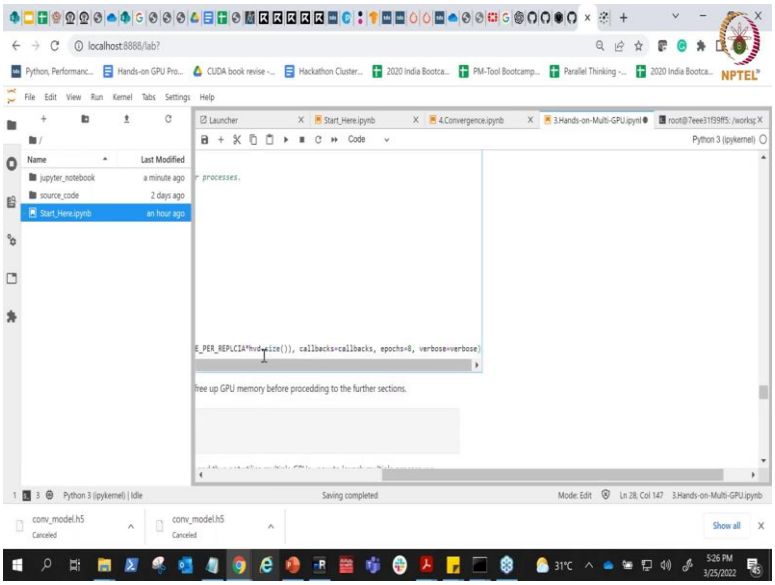
After that I think we have just written some wrapper to figure out how what is how many images per second, we are able to calculate using the throughput.

(Refer Slide Time: 25:05)



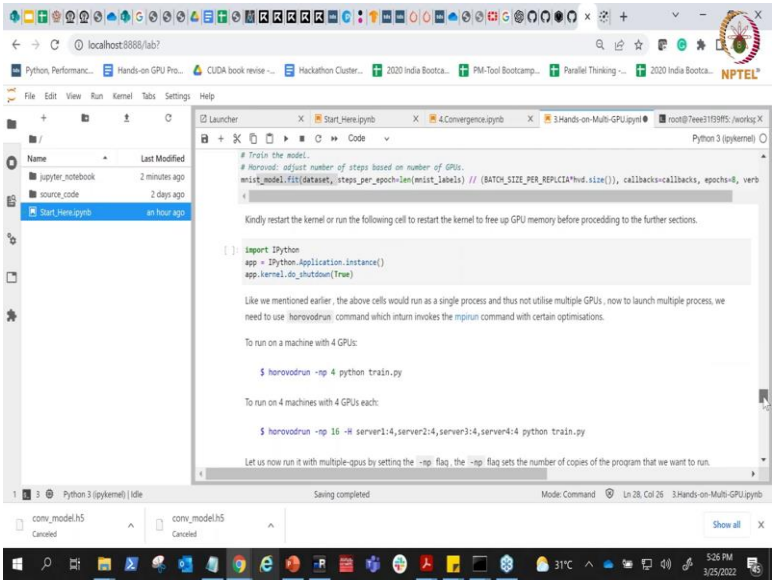
And, the callback that we are going to provide it with. So, this is the callback that we are creating to calculate the how many images per second, we are able to work on.

(Refer Slide Time: 25:19)



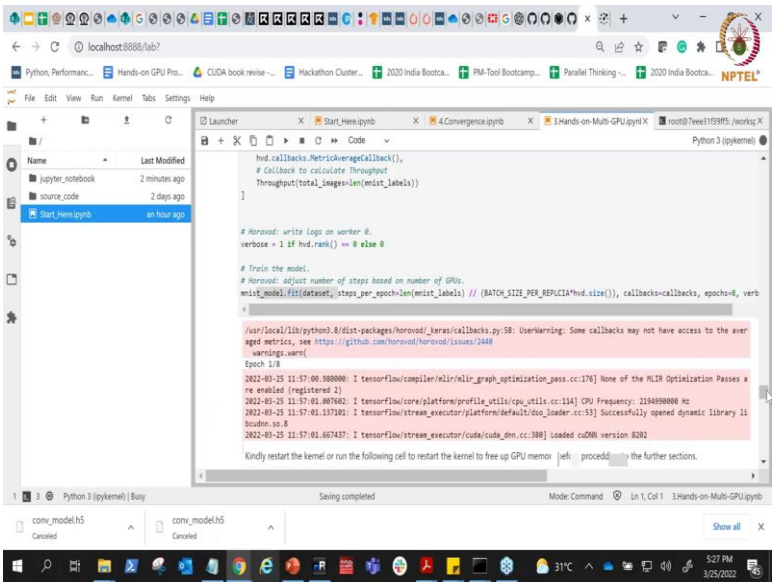
And in the end yes again, you would do model dot fit. So, you say model dot fit and it you basically provide it with the same thing. So, you basically are saying this.

(Refer Slide Time: 25:31)



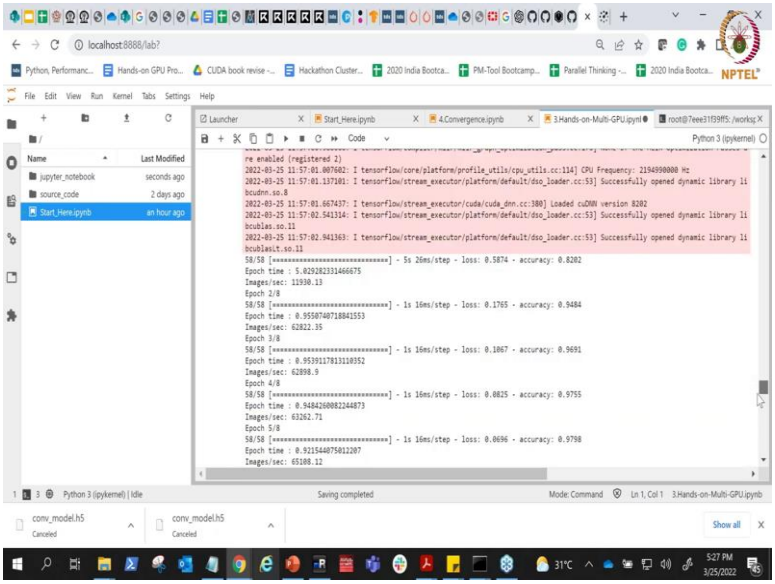
So, let us see if anything has happened.

(Refer Slide Time: 25:43)



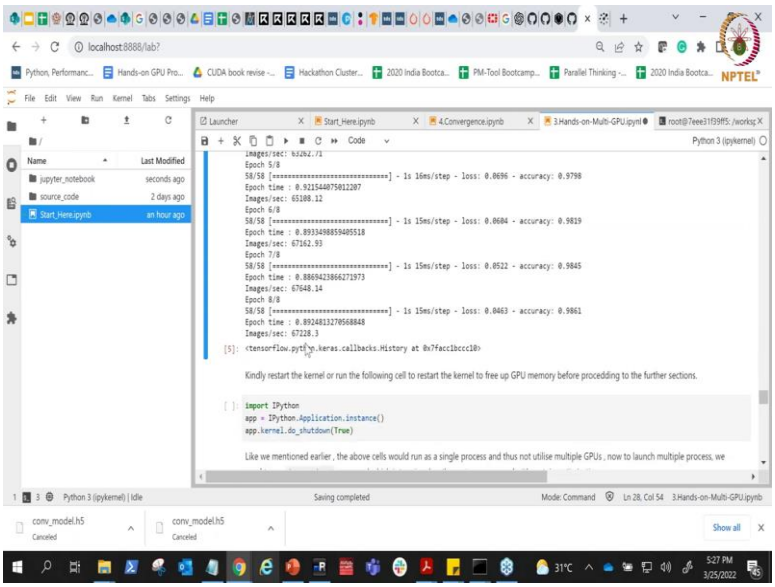
It seems to having some problem to run output ok. So, it started working now great. I do not know what is happening.

(Refer Slide Time: 25:54)



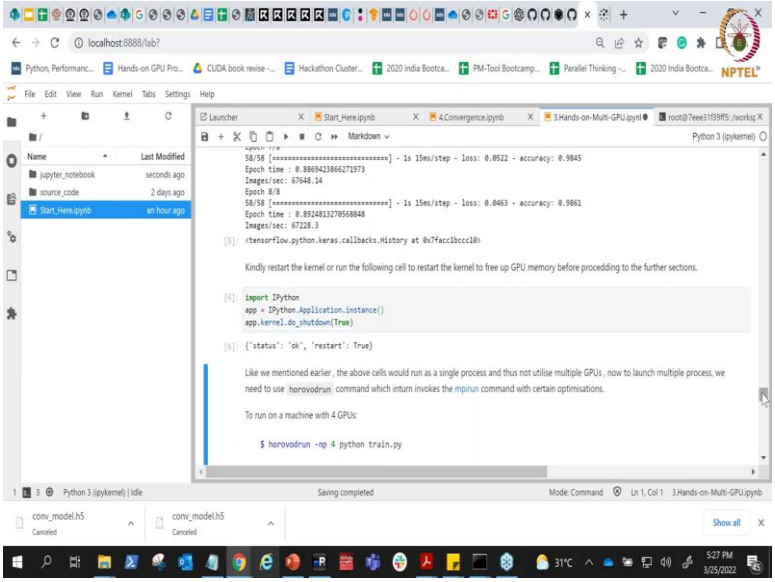
So, there is some refresh problem actually, it is not the problem of a this thing. So, I hope the code was kind of clear with respect to a model dot fit here. This is going to run basically a one optimizer, here is one process only.

(Refer Slide Time: 26:10)



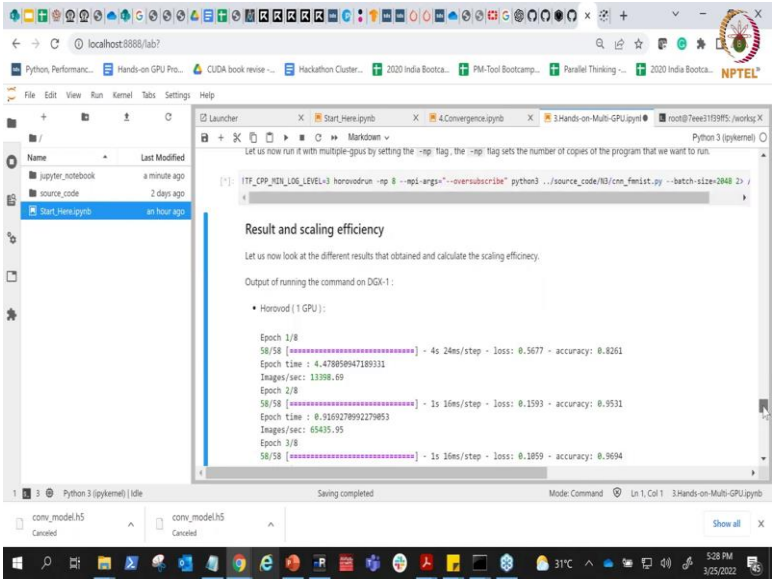
And, you can see it is like 67000 something right.

(Refer Slide Time: 26:15)



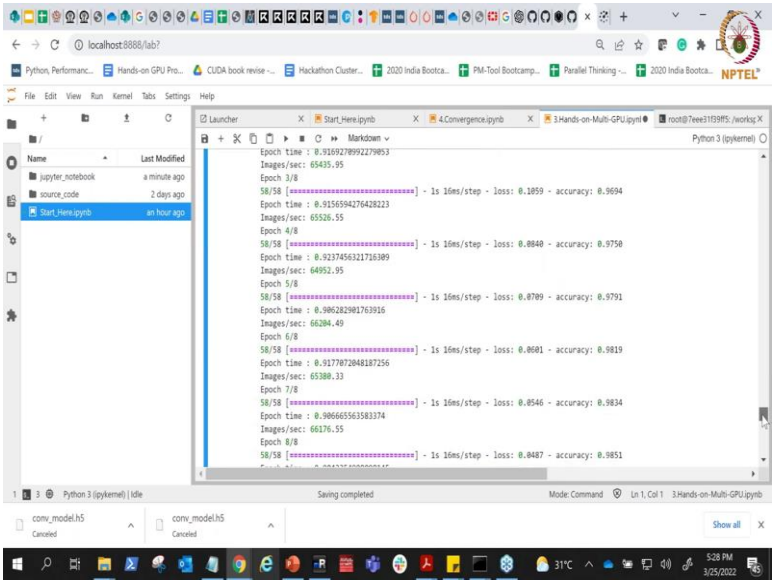
So, let us stop this as well. So, we have just restarted it.

(Refer Slide Time: 26:27)

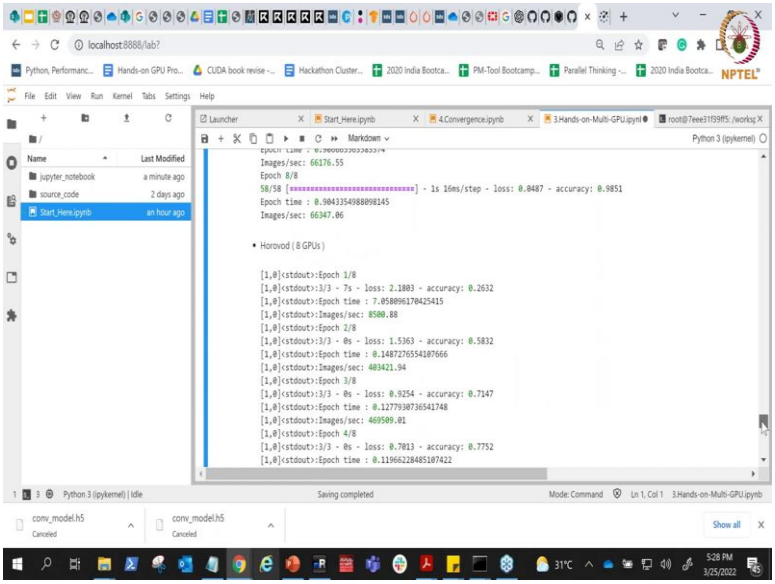


And, we are going to now run the horovod with multiple number of processes. You can see here, I am running the same code, but this code I can define the number of processes. So, if I have 2 GPUs, I can set -np 2, if I have 4 GPUs, I can set -np 4, if I have 8 GPUs, I can set -np 8.

(Refer Slide Time: 26:56)



(Refer Slide Time: 26:56)



So, based on the number of GPUs that you have, you can basically run this and, ideally it is supposed to give you the scaling efficiency that you are expecting out of it. So, there is some problem running this thing on this machine. But, hopefully it will get through in a bit. So, let it run, till the time we will go back to the theory part and hopefully we have got an idea of how to use both Horovod and TensorFlow.