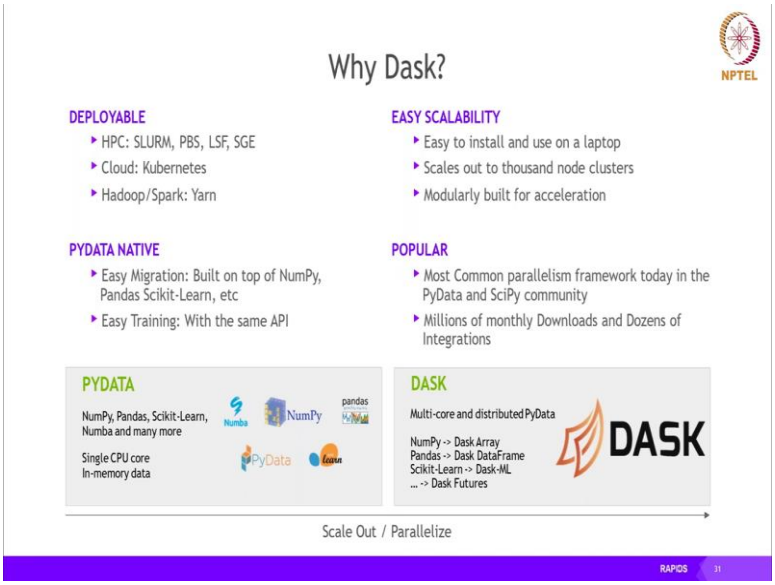
Applied Accelerated Artificial Intelligence
Prof. Bharatkumar Sharma
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 49
Scale Out with DASK

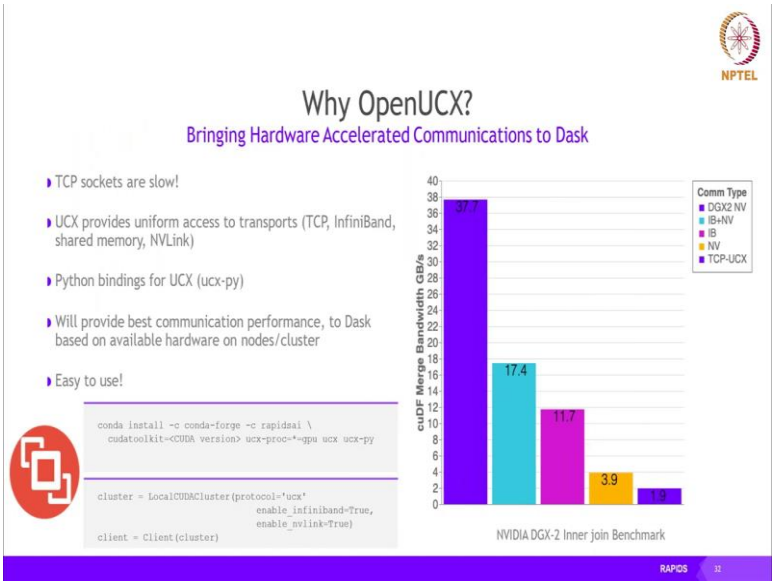
(Refer Slide Time: 00:14)



We have covered this in our first lecture that the Dask basically helps us in scaling scale out, not scale-up, but scale out across multiple nodes and it has all the right components, because it takes its motivation from the existing HPC scenarios. It supports a cloud or Hadoop-based deployable scenarios or the HPC traditional schedulers like SLURM, PBS and all.

And, it is it supports all the built-in types which have been traditionally done on NumPy and so, it practically uses the same API and we are going to look at the demo of it as well. It is very easy to scale. So, you can use Dask on your laptop or you can scale across thousands of nodes in a cluster environment also. And it has, with because as we said we have also supporting Dask for accelerating it on the GPU.

(Refer Slide Time: 01:23)



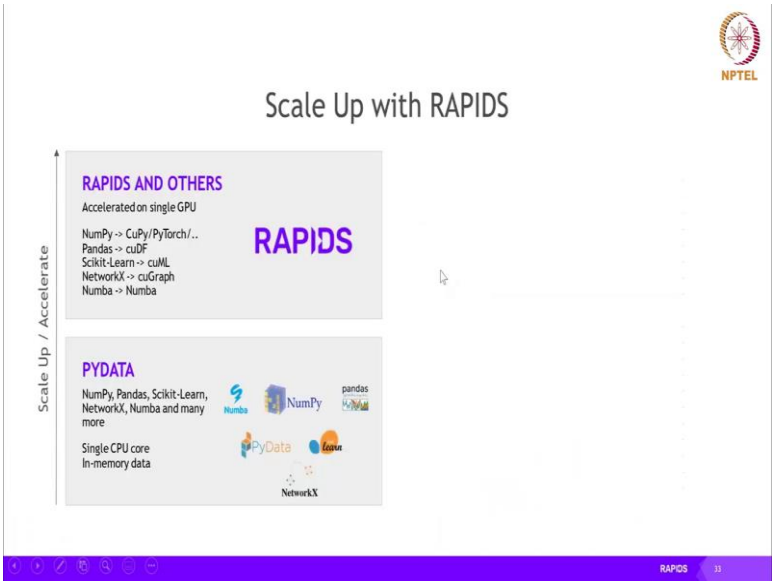
So, behind the scene actually Dask supports high-performance stack and it uses something called as OpenUCX which kind of is standard for getting really really high performance.

So, because, we did a session on networking as well and TCP sockets if you use traditional methods, they might be slow. So, UCX provides a uniform access to not just one kind of a transport layer, but it supports different ones like TCP, if you are having an InfiniBand network then it will support InfiniBand.

And if within a network if you are using NVLink base and it supports NVLink as well. And, there is a Python binding for UCX and you can very easily so far install ucx-py. So, it kind of supports and you can see here that if you are running it on a normal system versus a InfiniBand plus NVLink based system plus a UCX base system.

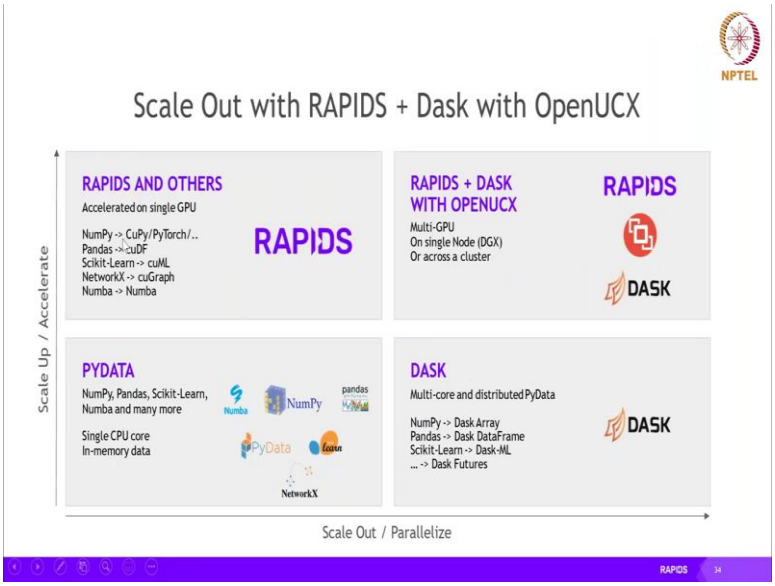
And you can see here how much what kind of performance jump that you can see when you keep on improving and start using some of these features. So, the performance would again be dependent on the hardware characteristics that we had seen earlier also. And also, what kind of a package you have installed on that system and if it is able to use those features like NVLink and all those other things also as much as possible.

(Refer Slide Time: 02:55)



But, with RAPIDS as we said, if you are using traditional environment with Numba, Pandas and all and Scikit-Learn and all you can scale up by running it on a single GPU and by using its equivalent RAPIDS line using cuDF or using cuML and all for cuGraph for NetworkX and is and so and so forth.

(Refer Slide Time: 03:23)

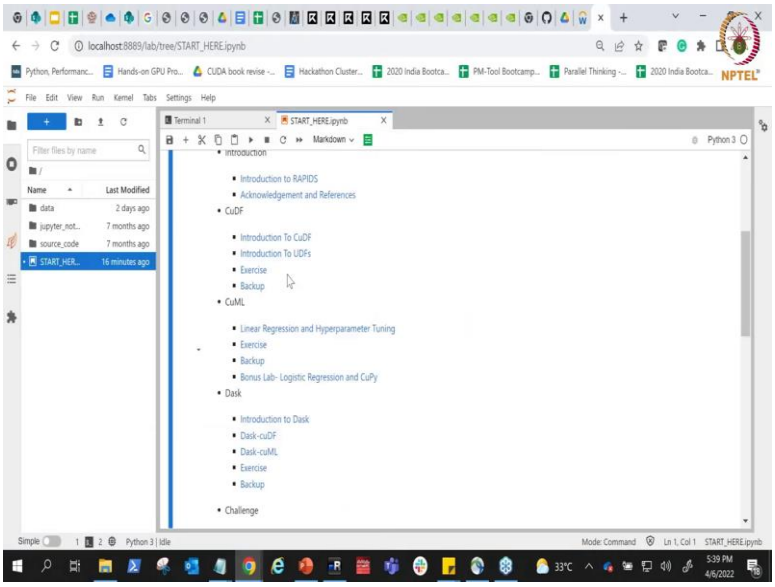


But if your data set requires a higher number of efforts and it is much more larger then you can scale out. So, within the sequential world as in within even the CPU-World you

can scale out using Dask which supports multi core and distributed PyData features. So, you can scale across multiple CPUs across multiple nodes.

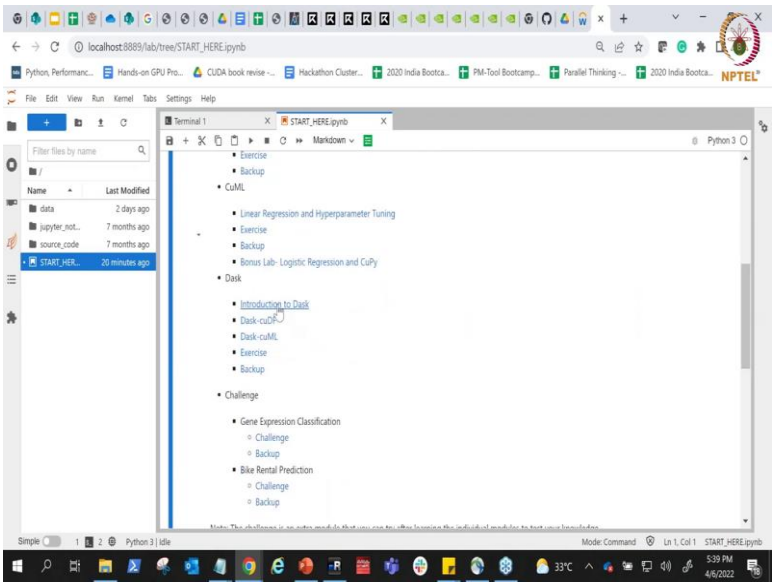
But the same Dask, if you use in a RAPIDS environment you can run it across multiple GPUs or within multiple GPUs within the single node or across the whole cluster as well. So, the best performance that you can get is here in this particular window where you have taken care of both scale up and scale out together, which can give you really really good speed ups.

(Refer Slide Time: 04:14)



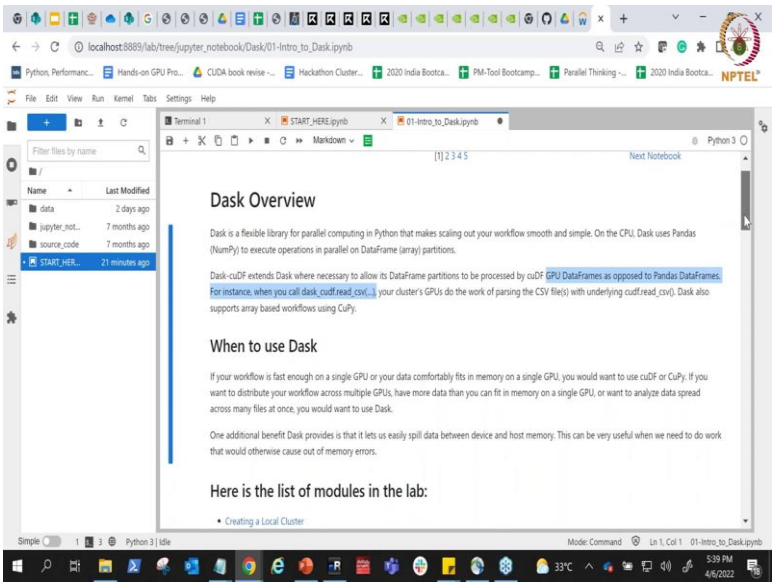
So, with that let us move on to another demo that, I would like to show you here. So, so, far we have seen the cuML part.

(Refer Slide Time: 04:19)



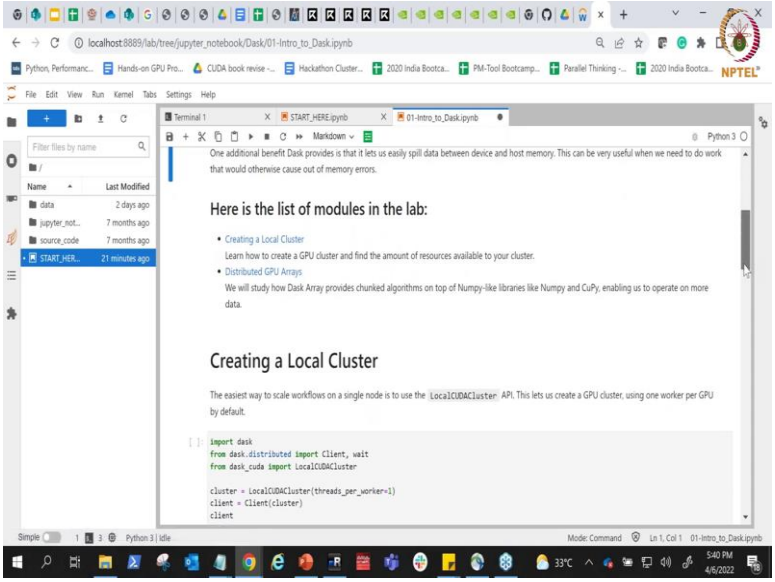
We are going to now look at Dask.

(Refer Slide Time: 04:22)



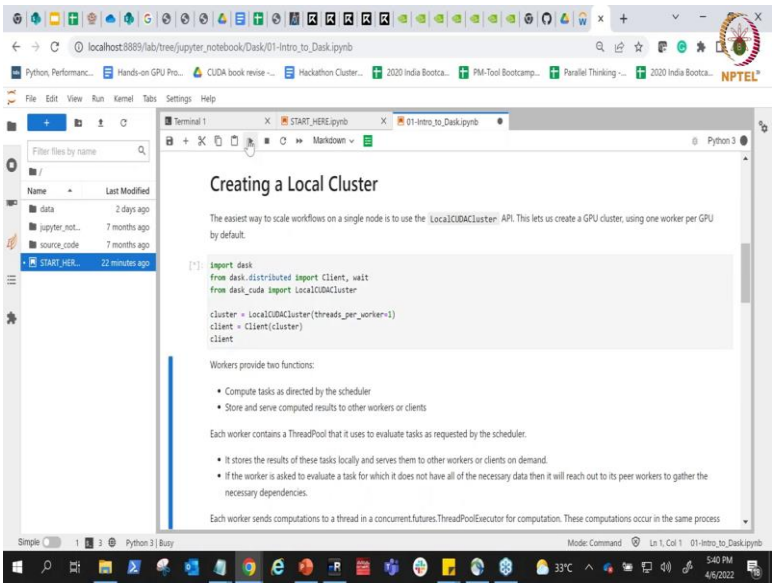
So, I said Dask is a library which is more of a parallel computing library, it provides you the scale out features, which means, you can go across multiple nodes, across multiple cores and in case of GPUs you can go across multiple GPUs as well.

(Refer Slide Time: 04:41)



So, in this particular lab, what I am going to show you very quickly is, how to create a multiple how to use multiple GPUs using Dask. So, there is a concept of cluster and everything.

(Refer Slide Time: 04:57)

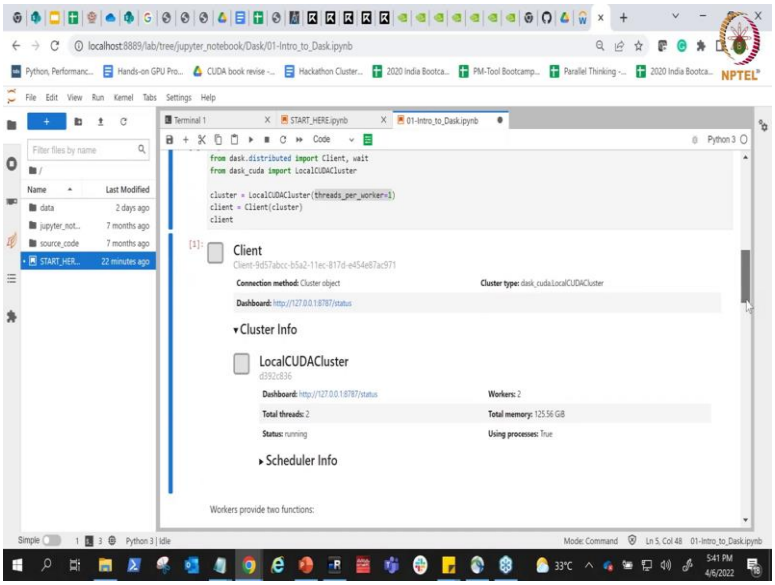


So, we are going to use in this particular scenario, I am having a machine which is a single node which means, I have 1 machine with 2 GPUs. There are different kinds of clusters that you can create in Dask. The one that I am creating here is a

LocalCUDACluster which means, I know that I am going to run only on one machine which has multiple GPUs.

I will use another kind of a cluster when I want to scale it across multiple nodes. But, here in this particular case, I am going to use a cluster strategy which is LocalCUDACluster which means, I will utilize different GPUs on one machine itself.

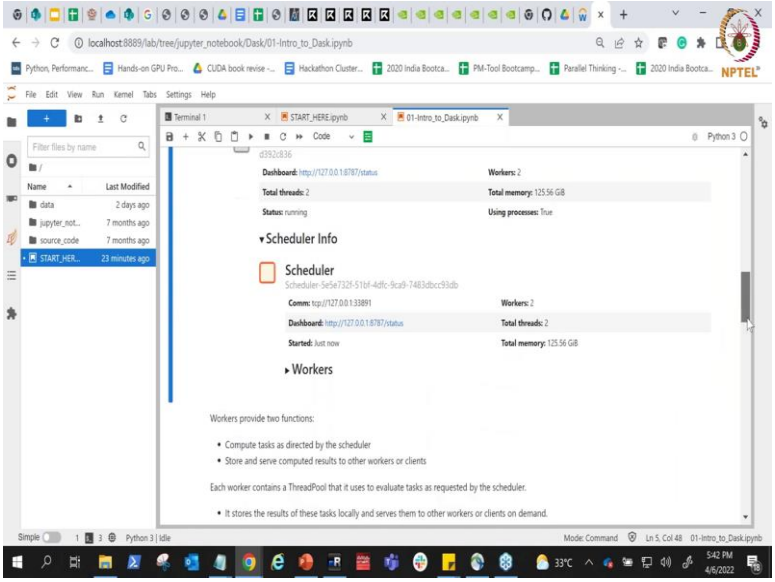
(Refer Slide Time: 05:40)



So, when I run this, it is you can see it the one that we are doing is we are trying to import dask and inside dask we are importing client and we are also including importing the LocalCUDACluster, which is an extension for CUDA or the GPUs.

And you can see here that, what it returns you is basically we are creating a LocalCUDACluster and we are seeing that every cluster will have only one worker, which is 1 thread per worker, you can have more threads also per worker if you require. But here we are explicitly stating that we are going to have 1 worker.

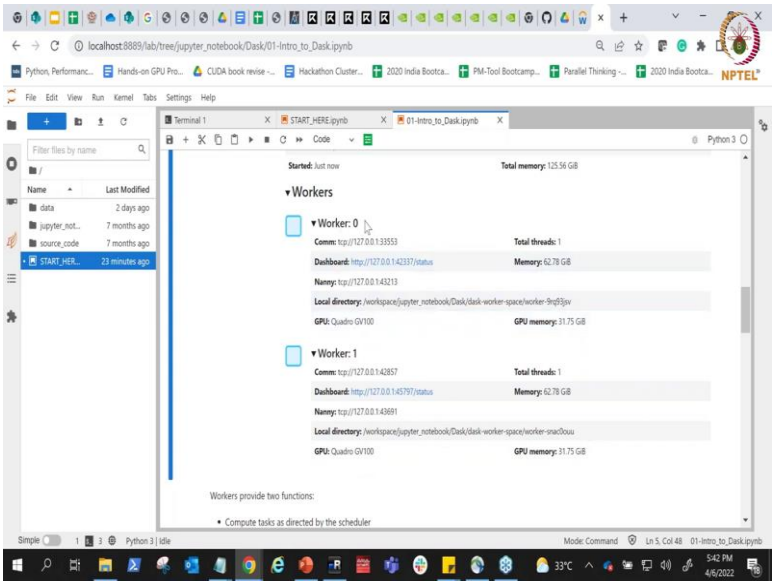
(Refer Slide Time: 06:19)



But you can keep on going further and further down like you are saying that the strategy the cluster is a LocalCUDACluster. And, I basically have 2 workers, you can see here it has said that I have created 2 workers.

And both these workers are going to have total 2 threads and the status is running which means my cluster is running at this point of time. Where does these 2 workers come from? You can go further inside and you can see here that it has it will give you much more information and you will see here the workers I have 2 workers.

(Refer Slide Time: 06:52)

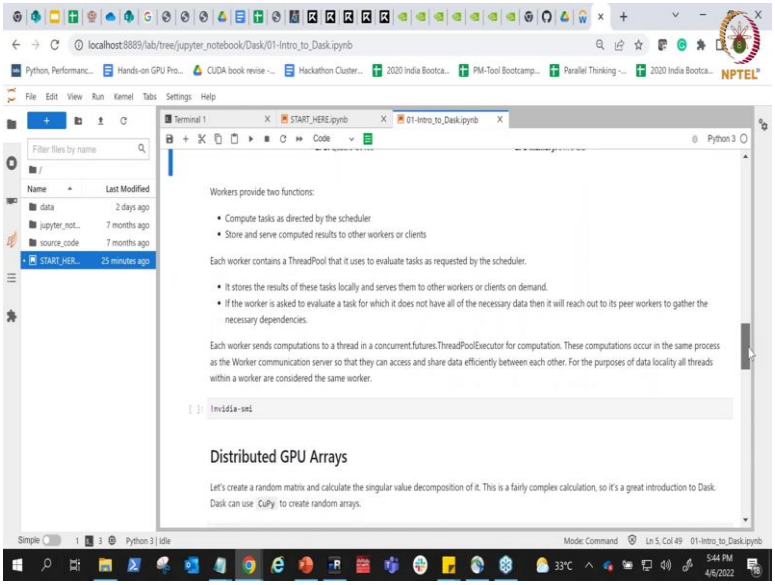


And the worker, each worker is basically accessing 2 Quadro GV 100s. So, worker 0; used 1 thread, CPU thread and it is going to utilize the 1st GV 100 which I had on my system. And, the 2nd worker is basically going to use another GV 100 and you can see here how do you know this? You can see the communication is slightly different.

This is at 3353 and this is at 42857. And, both of them GPUs are having 32 GB of memory which they have and on the CPU file both the workers can utilize 62 GB 6 to be a GV each. So, once you create the cluster it is going to automatically figure out the scenario like, how many of workers are there based on the number of GPUs you have.

And you can also define the some of these parameters can also be sent to define some of the things that you would like it to change like how many workers do you need per thread and stuff like that.

(Refer Slide Time: 08:06)

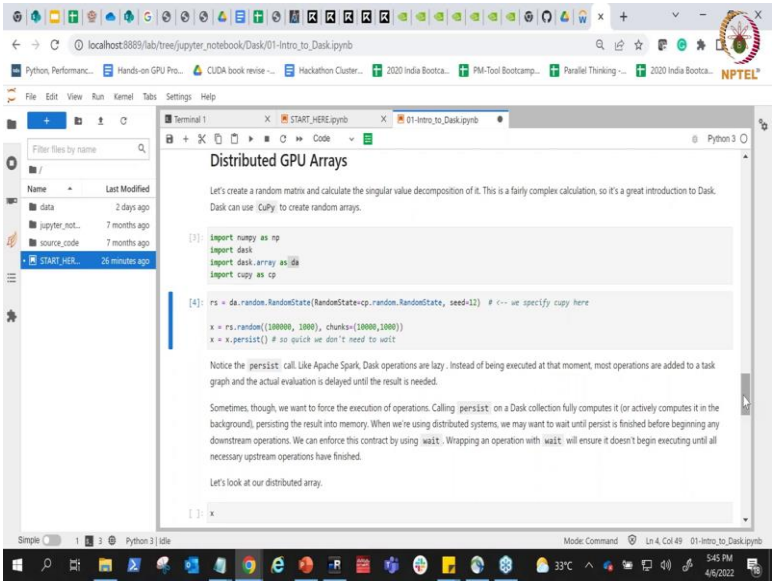


So, it will give you all of this information and the workers basically provides 2 functionality, basically it is going to perform tasks.

So, as the name itself says, these are workers who will be given some work and they will perform certain tasks. Who will give the work to them? The scheduler will give the work to them. And, basically it will serve multiple clients, right. So, the idea it is more of like a client server model and your workers are going to complete the task provided to you by the scheduler.

And the scheduler gets it to work from the client themselves, right. And, you can create multiple thread pools as I said to you can give multiple threads to a single worker if you would like to do and so on and so forth.

(Refer Slide Time: 08:49)



So, as I told you that currently we have 2 GPUs, which are assigned to 1 worker each.

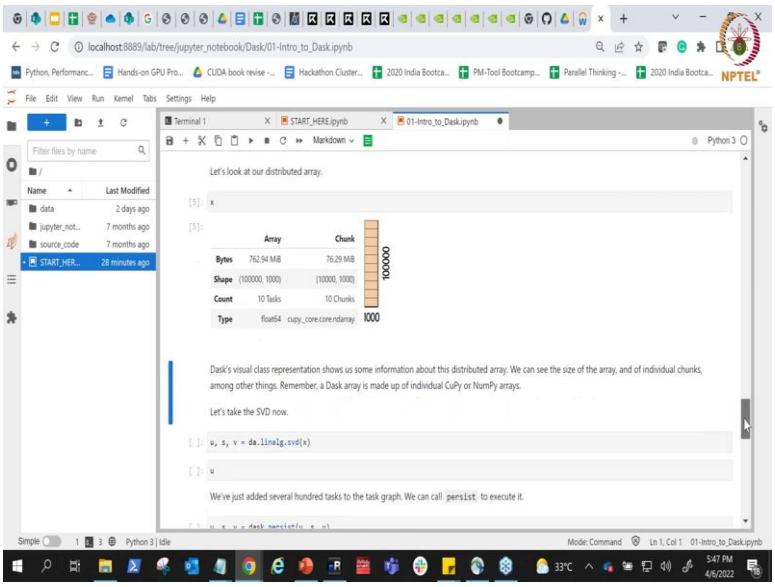
So, the first thing that we are going to do is that rather than creating a normal array, we are creating a dask array. You can see here we are going to create a dask array here and you can so instead of normal array we are creating a dask array. And we are creating multiple you can see here I am saying that create random values.

And in these dimensions so, we are saying that I want to create so many values and they would be split up into these many chunks, right. So, the so we I am saying that I need to have. So, many values which is 1,00,000 and 1,000 in the respective dimensions and then I am going to create multiple split of it, which is also referred to as chunk.

So, I am going to create multiple chunks of that larger split, larger value that I have. And, then I am saying x dot persist y persist. So, one thing about Dask that, you have to understand is that, Dask is more of asynchronous in nature or it does all of the operations lazy. So, when you ask Dask to do something, they what it does it actually adds a particular task into your graph.

So, it creates a task graph and it will do the evaluation, it delays the evaluation until the results are needed. So, until you are going to use the values which are going to call on the Dask, it will not do any execution. So, it is lazy in nature. So, if you want it not to wait you want the data or all of the things to have done previously you can just say persist. Persist will kind of guarantee that your task has kind of finished before you move on to the next thing.

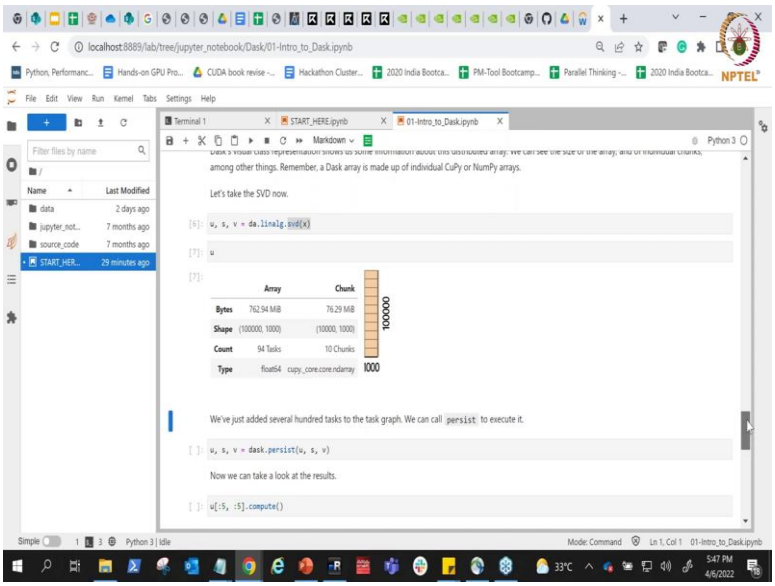
(Refer Slide Time: 10:41)



Otherwise, this asynchronous nature or lazy nature is the by default nature of Apache Spark and Dask also. So, you just have to make sure that you are either waiting for it to happen or you call persist to make sure that it is kind of indeed finished. So, it is what it creates is basically a distributed array. So, you can see here that it has created an array. The array total size is 762.94 MB, the shape is 1,00,000 by 10,000. But, you have splitted it across multiple chunks.

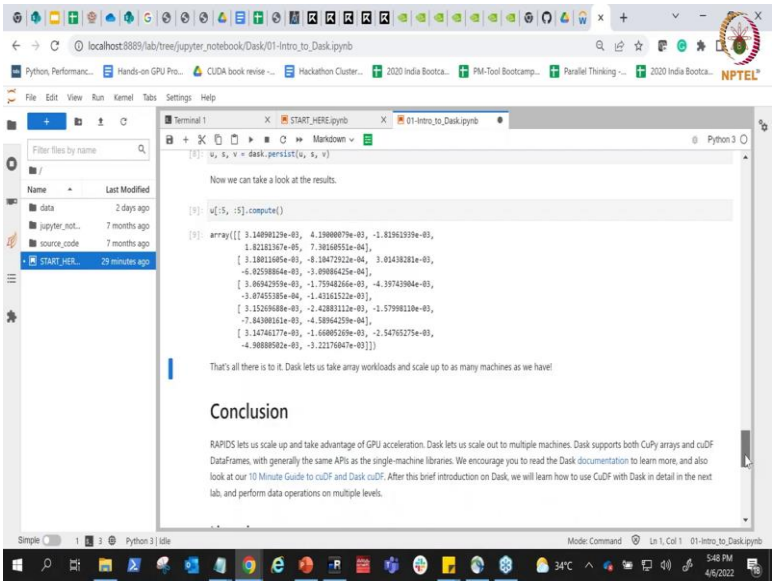
So, each chunk is of size 72 MB and it is split in this particular form where, it is of size 10,000 by 1,000. So, basically if you see what you have done is you have basically created 10 chunks, total chunks overall of the larger array. So, it is distributed across 10 chunks, 10 short and the type is of type cupy here, because and is of type cupy ndarray, N-dimensional array which is primarily mentioning that it is being created on the GPU. And after that, you can basically if you call anything on this particular array.

(Refer Slide Time: 11:52)



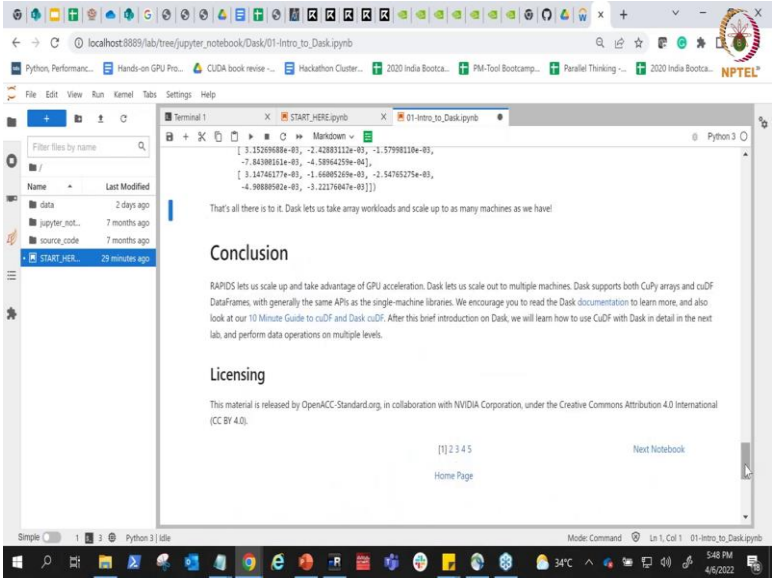
So, it shows us the information as I said of the distributed array, we can see the size of the array and of the individual chunk. And after that, if you call anything on that particular on the Dask array like here we are trying to do svd. So, we are trying to take the svd and if you do any operation here, then you can see the output is also of same type. So, the output of the svd is again you can see I am printing the characteristics of you and it is also of 10 chunks value.

(Refer Slide Time: 12:33)



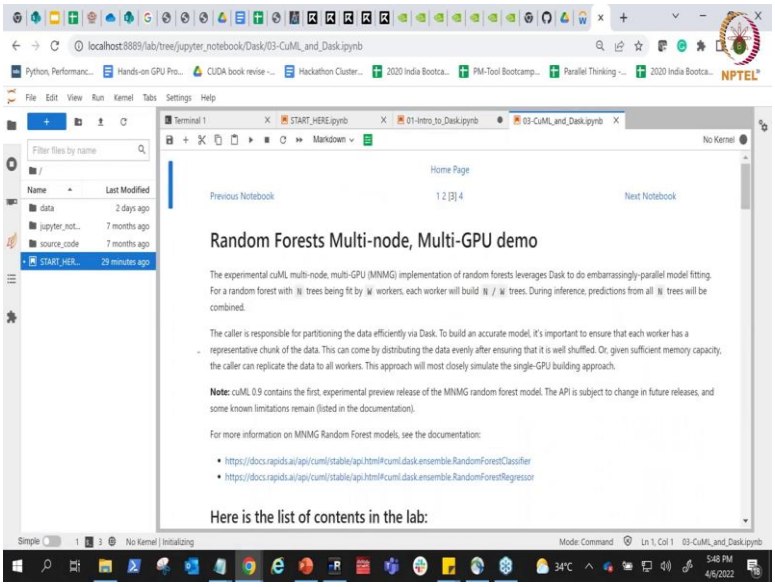
But, in order to persist it again, you have to basically call the persist API here. And, yes, that is all you have to do to make use of Dask.

(Refer Slide Time: 12:46)



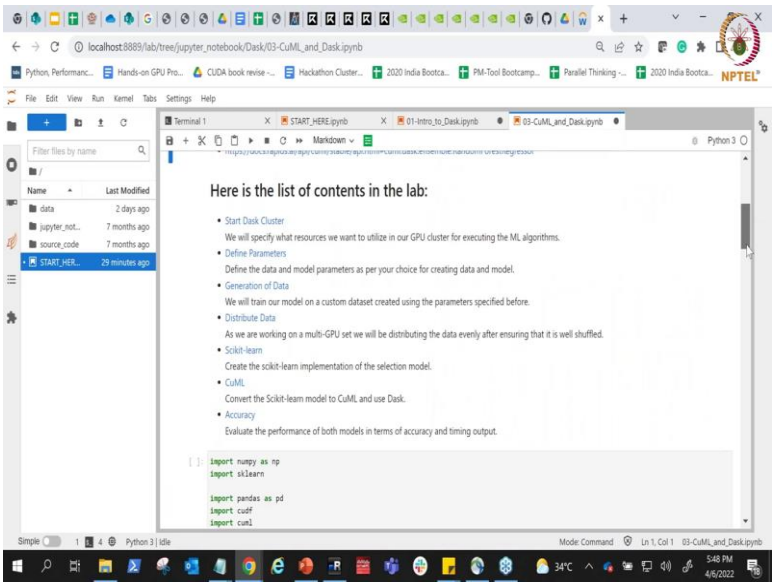
And you can do whatever activity that you would like to do with it, in terms of chunking the data across the clusters that you have created and working on it.

(Refer Slide Time: 12:56)

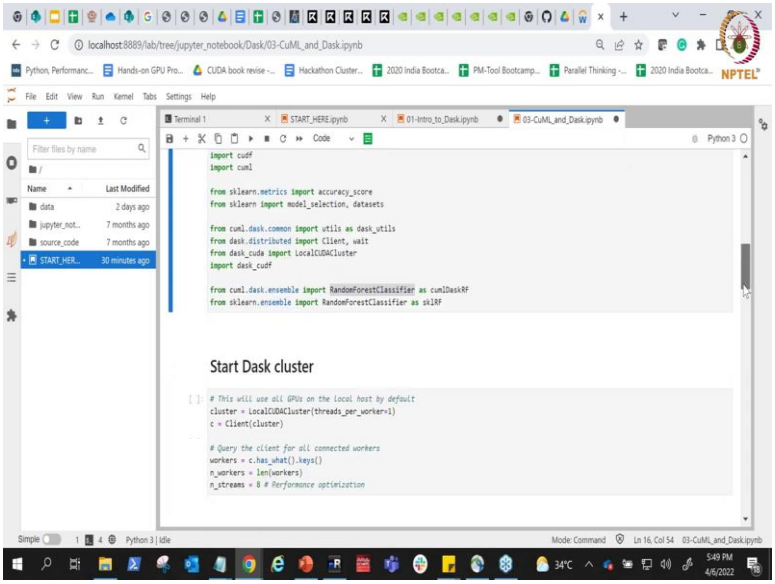


Let us take an example again on using cuML along with it. So, we are going to use random forest and we are going to run it across multiple GPUs

(Refer Slide Time: 13:09)



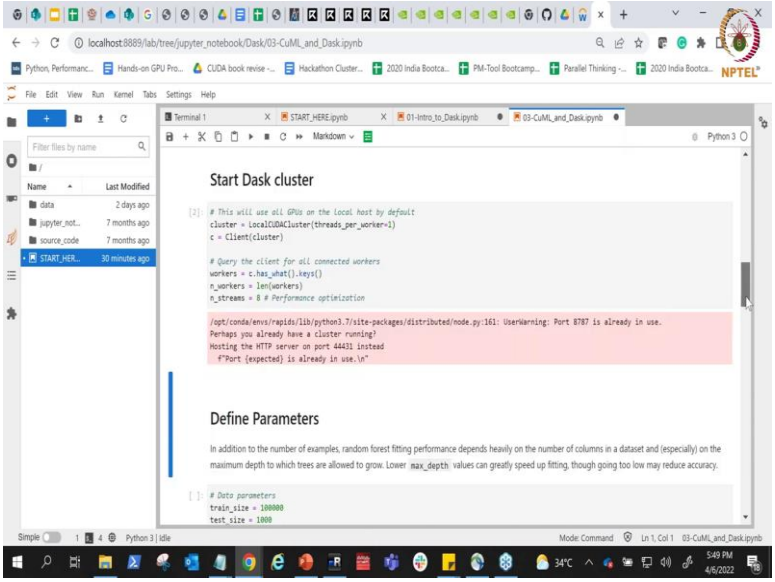
(Refer Slide Time: 13:11)



So, again like previously that we have done, we are going to use LocalCUDACluster. And, for we are going to do random forest hence we are calling or we are importing the Random Forest Classifier like the way we have done it previously also in the cuML just that this time we are importing the Dask version of it.

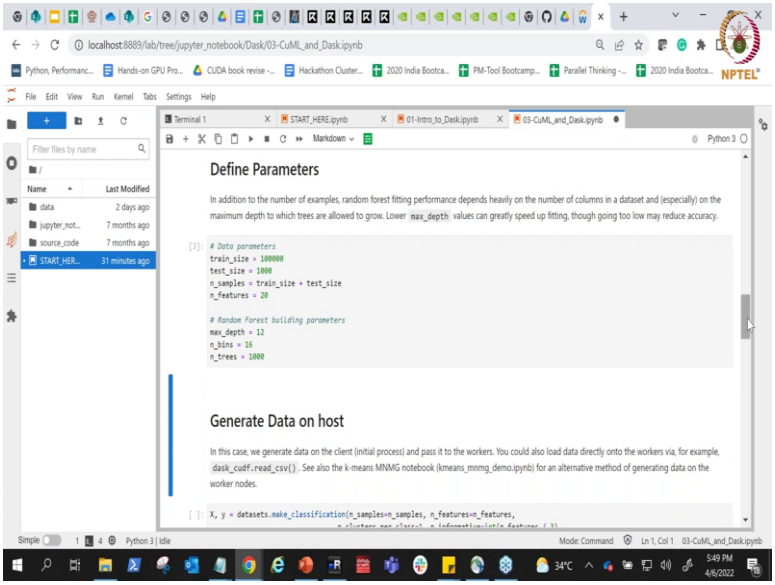
You can see here I am not doing cuml dot ensemble, I am basically saying cuml dot dask dot ensemble. So, I am explicitly importing a Dask version of supported version of Random Forest Classifier here.

(Refer Slide Time: 13:50)



And then, yes, again I am going to create the CUDACluster which is going to be the same here, there is no change which will happen it is just that, I am explicitly calling the number of workers that I need, the number of workers would be equivalent to the number of GPUs which is going to be 2 again.

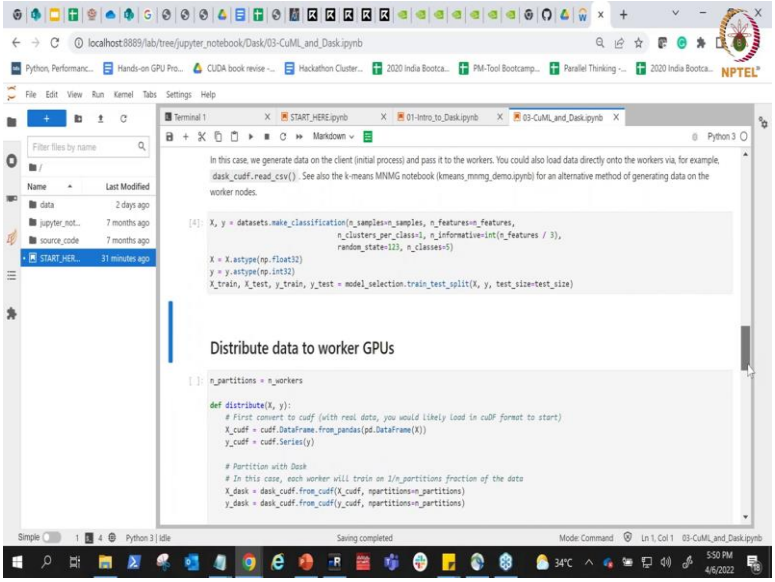
(Refer Slide Time: 14:08)



So, here what we are doing is we are just defining certain parameters for our random forest, I am not going to go into the details of the random forest itself that is not the objective of this lecture. But we are just defining certain parameters to it; including the

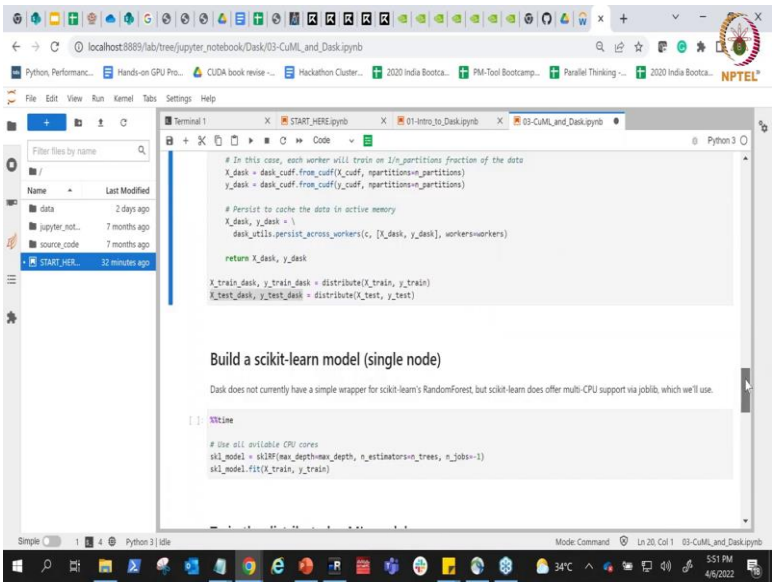
maximum depth of this particular forest that we want to evaluate it for once we have defined it we are going to generate the data on the host.

(Refer Slide Time: 14:31)



First as I said, we will use will create the data on the host itself which is on the CPU. Then we will import it inside our Dask. So, we are creating a data set, we are initializing it to certain values like random states. The number of classes we wanted for our random forest and everything we are defining all of those parameters. Once we have defined it, what we are going to do is we are going to distribute that data across the number of workers.

(Refer Slide Time: 14:57)

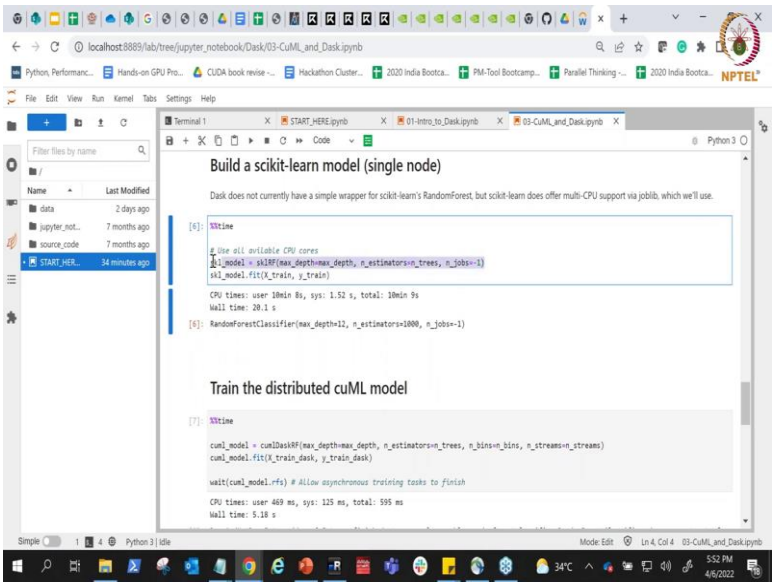


For the number of workers in our case it is 2, we are going to basically as you can see here, we are creating dask cudf from the cudf normal cudf. So, we have created our data input here and we are importing it into a dask environment.

And how many of that it is going to be split across is the number of partitions the number of partitions are equivalent to a number of workers. In our case, we have 2 GPUs. So, the number of workers are 2. So, basically, we are saying that from our cudf we are going to split into number of partitions across dask, which is 2 in our particular case, because we have 2 GPUs.

And we are calling the distribute API for both the training set as well as the task set. So, the output of distribute is basically that we have distributed our data across the Dask which is going to run across multiple workers here.

(Refer Slide Time: 16:04)

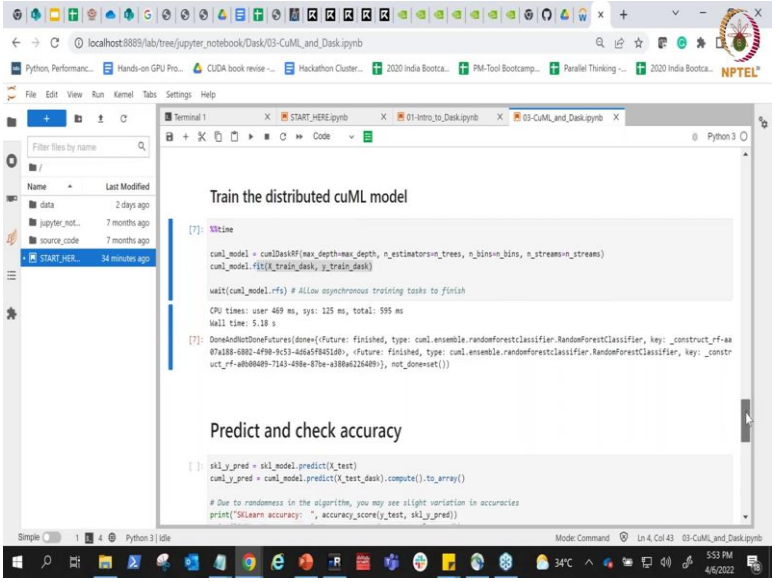


And then, yes, initially we are going to run this on scikit-learn. This scikit-learn by default is going to run on multiple CPUs ok. So, scikit-learn by default supports multiple CPU cores. So, when you run it, you will see that you can enable certain parts and you will see that it is going to run it across multiple CPUs as well.

So, let us see how much time it takes across multiple CPU cores? So, you see that it took 20 seconds in general, but the CPU time is 10 minutes. What does it mean is that, even though your wall clock time for us was 20 seconds, it actually ran across multiple cores, where the total CPU time across multiple cores was 10 minutes and 9 seconds.

So, it distributed the work across multiple CPU cores and for us it was Wall clock time was only 20 seconds overall for these many depths and all that we did. So, how do we define it? How to use all the cores in the scikit-learn? You have to just set number of jobs as -1, what -1 tells to scikit-learn is that, I want to utilize all the cores which are available to me that is how you can distribute the work even in the scikit-learn use utilizing all the CPU cores.

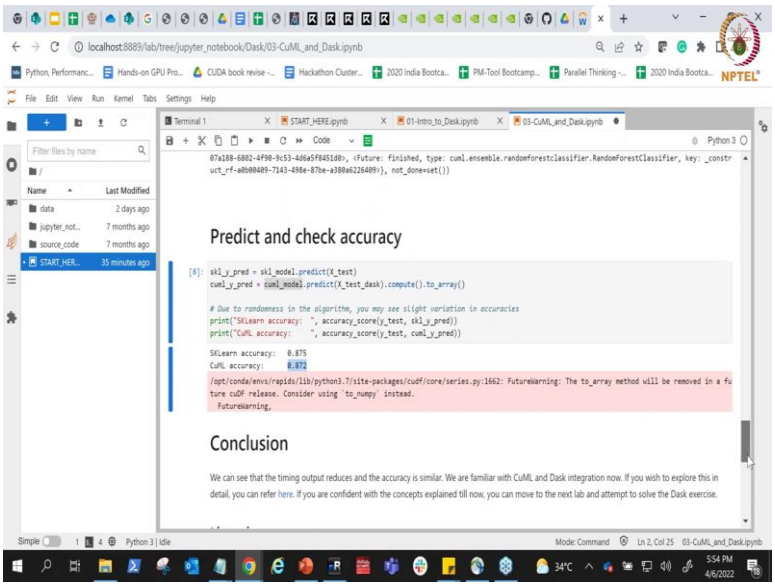
(Refer Slide Time: 17:32)



The same thing using cuML, you can see in the cuML we are doing the same thing, but using Dask version of it. You can see here we are calling cuMLDaskRF. So, we are passing it the Dask repository and that is why we are calling it cuMLDaskRF and we are passing it the Dask arrays.

And you can see here instead of taking 20 seconds, it took around 5 seconds overall, and the total time actually on the CPU was only 400 microseconds.

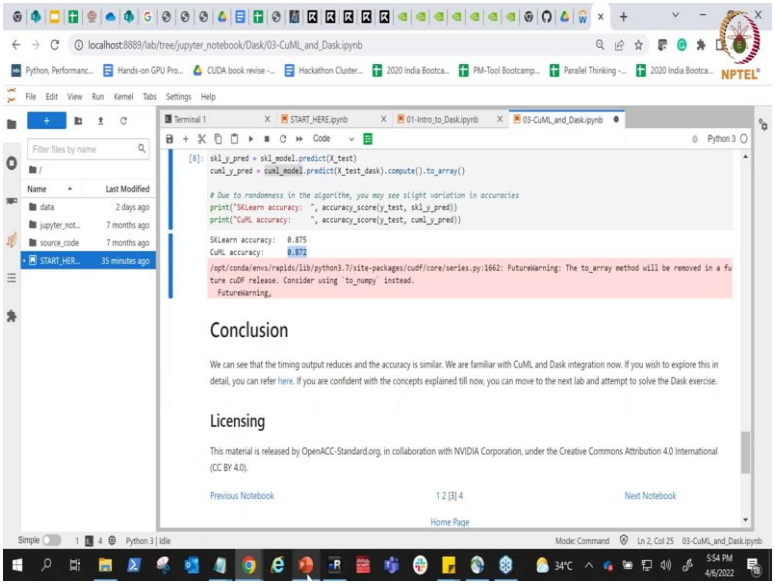
(Refer Slide Time: 18:05)



So, that is how we were able to run it across Dask across multiple GPUs without having to change a lot of things cuML used, sorry, the scikit-learn used practically all of the CPU cores by passing number of jobs as -1 and for cuML basically we use the predict option for the cuML we basically split it across the Dask API and the split the job across 2 GPUs in short.

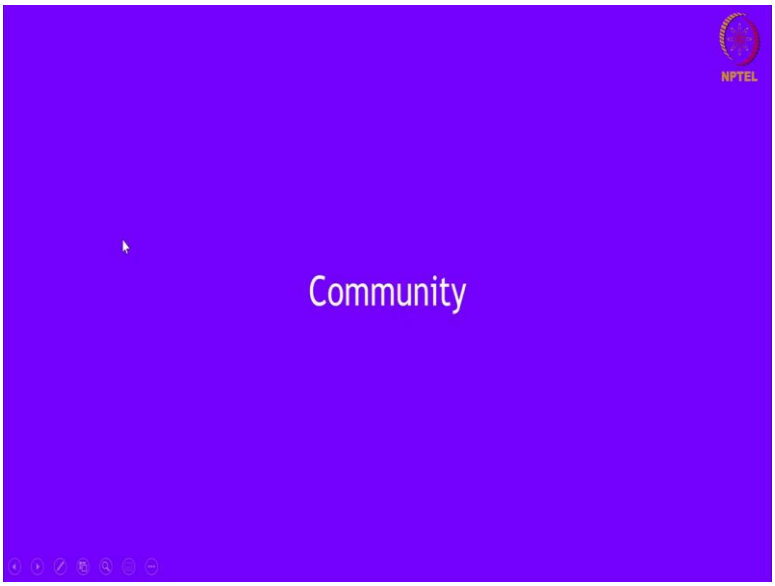
And you can hear both SKLearn and cuML are practically giving almost the same accuracy SKLearn being slightly higher 875 and cuML having a slightly lower accuracy at the third decimal place, which is negligible I would say.

(Refer Slide Time: 18:51)



So, with that, we are done with this session on the RAPIDS. We have finished three components of RAPIDS here.

(Refer Slide Time: 19:00)



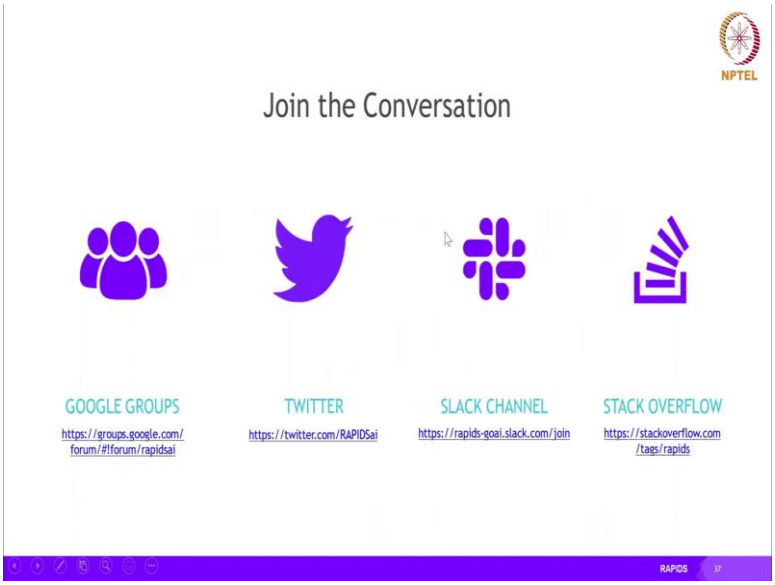
We covered the part of cuDF, we covered the part of cuML, we also showed how to distribute the work across multiple GPUs using the Dask APIs.

(Refer Slide Time: 19:13)

And as I said that, RAPIDS is open-source project and it has contributions and it has APIs to integrate various different open-source projects as well. There are various adapters out there including enterprise segments like Uber and all.

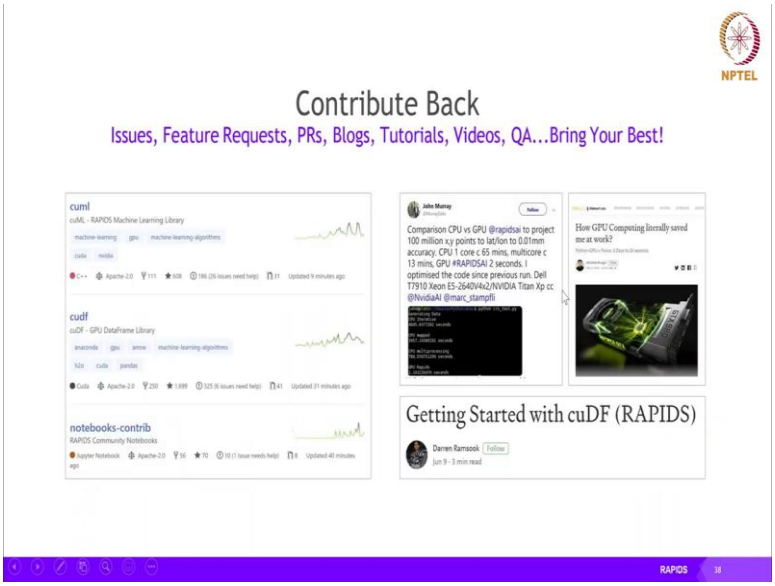
They are using it in the production environment. And there are various open-source contributors as well who are contributing to the RAPIDS project.

(Refer Slide Time: 19:43)



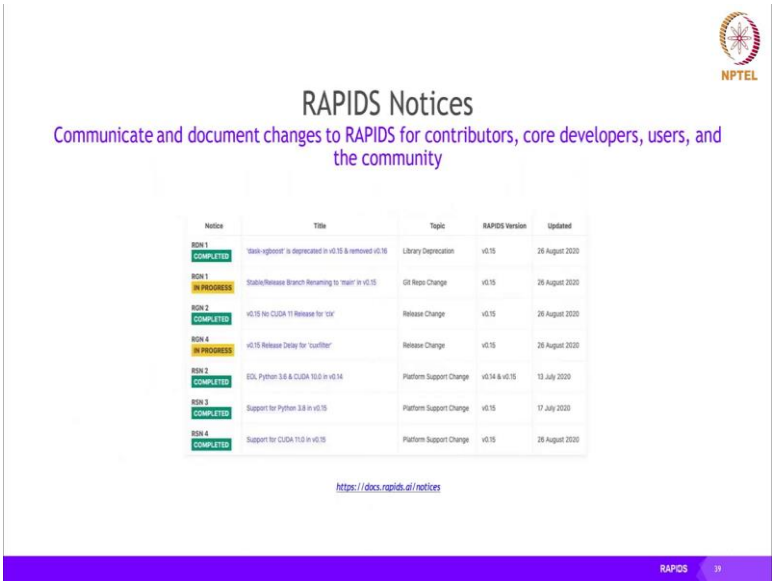
So, you can also join the Google Group, which exists for RAPIDS AI is a Twitter group, there is a Slack Channel which you can go to and ask any questions related to slack related to RAPIDS and there is a slack Stack Overflow part also for the RAPIDS.

(Refer Slide Time: 19:59)



And as I said, its open-source project, you can anytime go ahead and start contributing if you had particular thing.

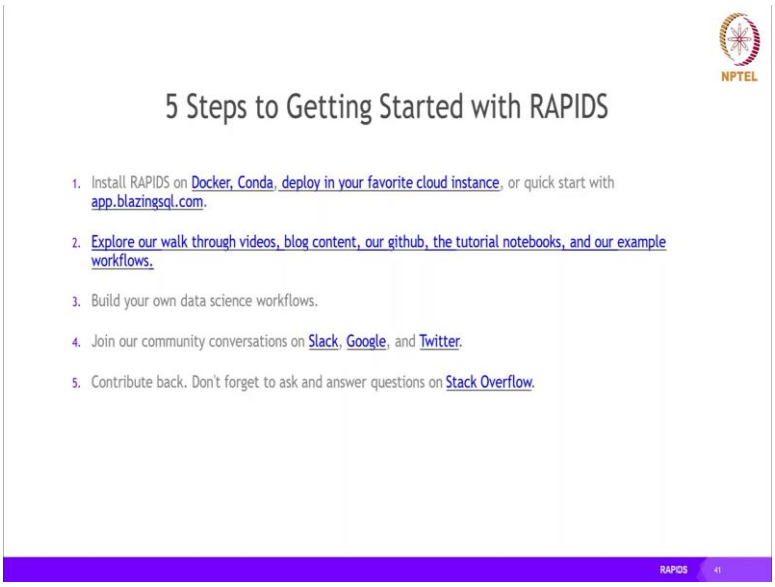
(Refer Slide Time: 20:09)



And, you can go through different videos and all also. You can be part of a forum where if there are any changes which are happening you can go to the notice section and basically see if there are any primary changes that you are interested in is happening or if there are any deprecations which are happen like you can see here.

In the dask xgboost there was a deprecation, which happened from the previous generation which existed. So, you can basically be aware of what changes are happening.

(Refer Slide Time: 20:39)

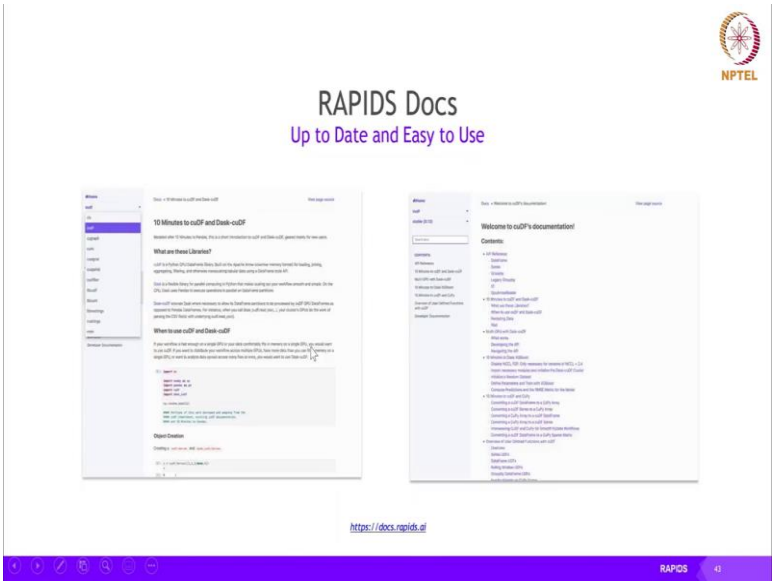


As I told you can download RAPIDS from in form of Conda environment or you can go to any of the cloud environment and run RAPIDS or you can use either a Docker or singularity or different versions of it also.

(Refer Slide Time: 20:55)

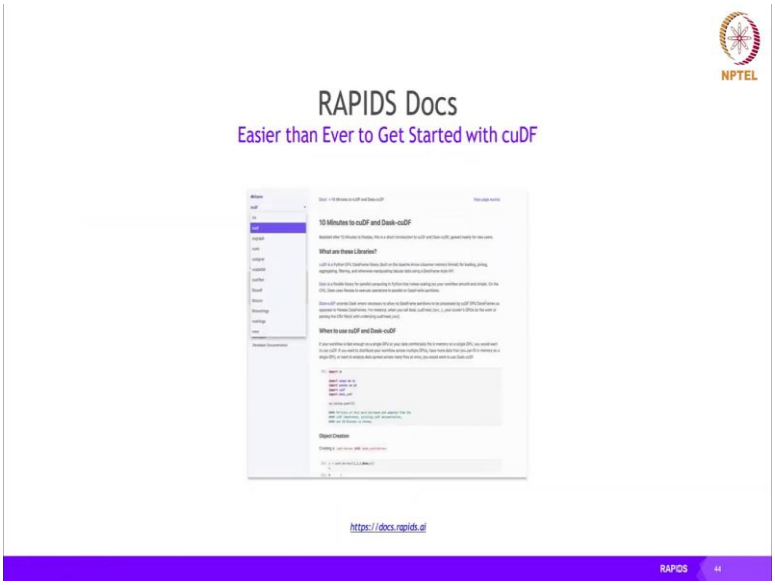
So, they are all installation guides present on the rapids dot ai platform you can go ahead and see how to install it.

(Refer Slide Time: 21:06)



In your own environment also.

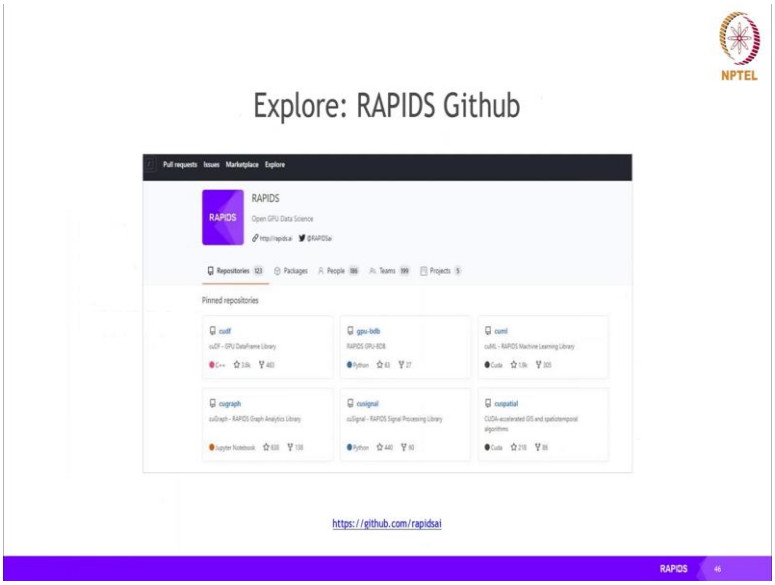
(Refer Slide Time: 21:09)



And you can keep yourself updated as well.

(Refer Slide Time: 21:10)

(Refer Slide Time: 21:10)



(Refer Slide Time: 21:10)

So, with that, I am done with this session on these three topics.