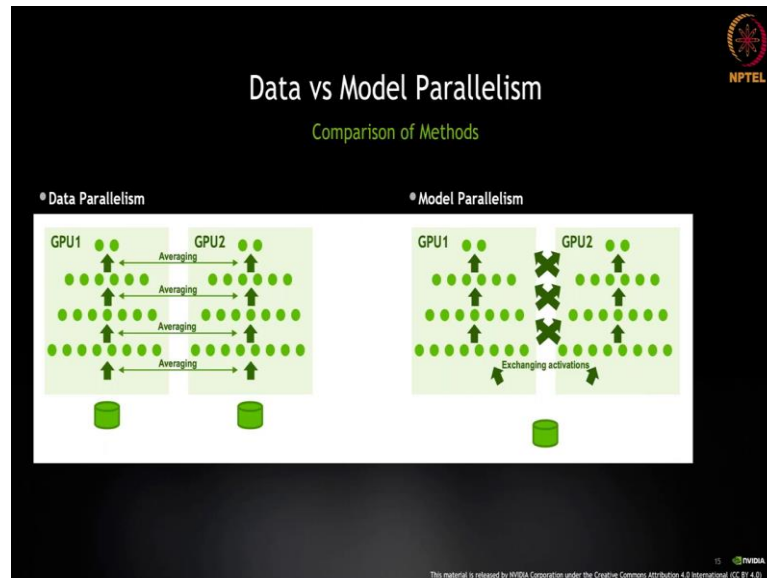


**Applied Accelerated Artificial Intelligence**  
**Prof. Bharatkumar Sharma**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Palakkad**

**Lecture - 35**  
**Fundamentals of Distributed AI Computing Session 1 Part 2**

(Refer Slide Time: 00:16)

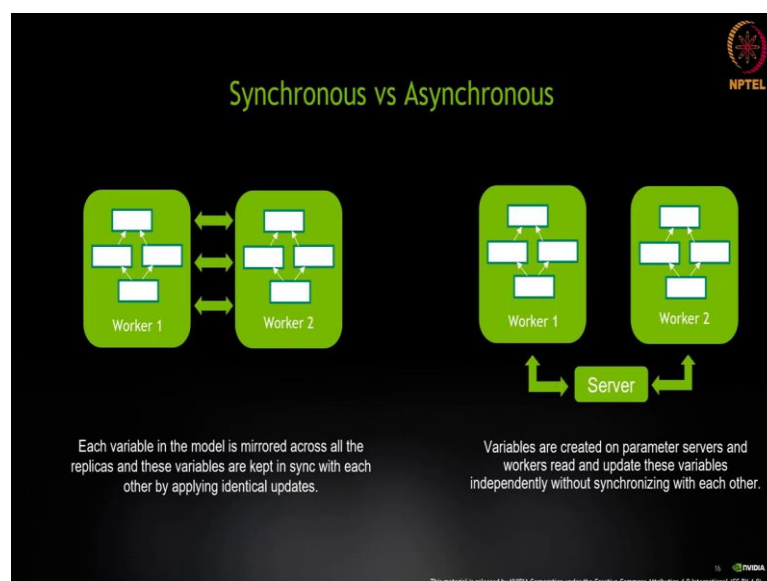


So, this is kind of the summarization of the communication part of it which is the comparison of the methods. As you can see here, here in this particular case you have on the left hand side GPU 1 and 2 and you are using data parallelism. And, as you can see you are basically passing it as a new different data, but the same set of layers exist in both the GPUs. And, at every level what you are trying to do actually is you are averaging the gradient.

So, you are basically communicating the gradients across and then finally, averaging them. On the right hand side what you see is the model itself where you will have, you are exchanging slightly different things right. So, you are exchanging the activations and you are basically the communication is of different type.

You are basically not averaging, but you are actually exchanging the activations and the communication pattern is slightly different or actually it is much-much more different. The reason to understand this will also be explained in the next couple of slides for you.

(Refer Slide Time: 01:37)



So, I was also referring to you in the terms of synchronous and asynchronous communication. So, we will be focusing primarily on the data parallel approach in the couple of lectures that we are doing. But, within that also you have two types: synchronous and asynchronous. On the left hand side what you are seeing is the synchronous approach.

And, as the name suggest the synchronous approach is an approach where all the workers are kept in sync which means every worker is in sync and they know the status and the values of the other workers. And, in the every step to make sure that they apply individual updates to the model. And, all of them at any point of time have the same copy of the model. This is very very important aspect of the synchronous communication.

While, in the asynchronous data parallel approach, the workers basically obtain the variables from a parameter server. So, you can see there is one more component which has got added which is called as a server. As the name, you might have heard the server is basically responsible for talking and giving a updates to all the different basically clients. Who are the clients here in this particular case? The clients are the workers or the GPUs themselves.

And, basically the workers obtain the variables from the parameter server and store the results back after processing. The parameter server would apply those updates and

update the variables. This allows the workers to work independently. Now, this is very-very critical. If you see on the left hand side especially in the synchronous communication approach, since every worker needs to be in sync with the other workers they are always going to go in a step by step manner.

So, even if one worker is lagging behind, it will affect the scalability of all the other workers because they have to wait till the time all the workers are at the same state. So, in the in case of load imbalance or if one worker is slower as compared to the other, you can get into scaling issues.

While, in case of asynchronous communication with parameter server, the workers actually work independently of each other. And, we are going to in the next set of slide, we are going to focus on the synchronous data parallel approach. But, asynchronous has its own way and it kind of finds it own space to make the scaling go better.

But, no matter what the we talked about so far, we talked about either using synchronous communication or using asynchronous communication using data parallelism or model parallelism and how to split that and everything. What is one thing which is common across all of them is communication. You need to communicate.

The workers needs to communicate either with each other or with the parameter server or and at what level are you going to do averaging with each and every worker, are you going to do activation, sync activation communication. So, whatever is the approach that you are going to follow, it is going to be dependent on the machine. And, how GPUs are connected to each other and how the machines are connected to each other because finally, the communication is going to define if you can scale it or not.

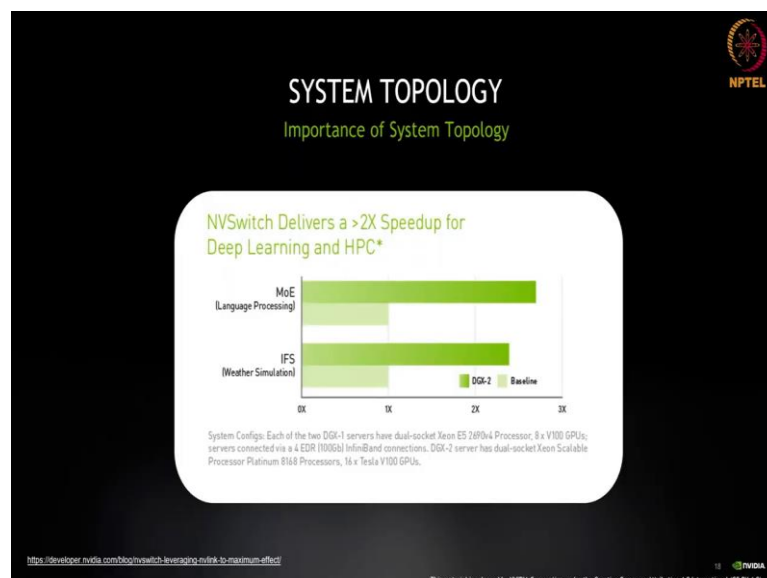
(Refer Slide Time: 05:27)



So, what matters is in this case is the system topology. In the networking in the system in this computer architecture world, the definition of it is called as system topology. So, when what I when I talk about topology within a node how are these GPUs connected to each other, across the nodes how are these two nodes connected to each other, how are the GPUs connected to the interconnect.

All of these is basically going to define the performance or the scaling of your model and that is what we are going to look at in this next couple of slides.

(Refer Slide Time: 06:10)



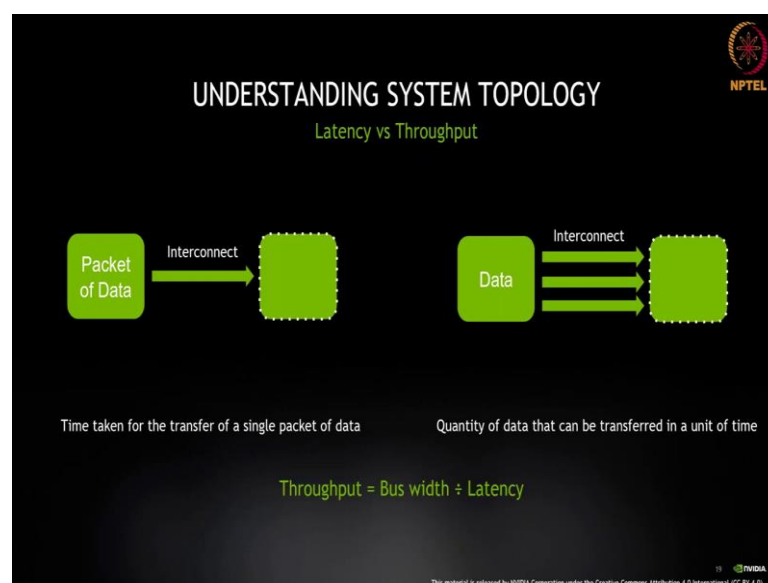
So, if I look at this particular example here, what you are seeing is an example of a system topology where you have a system which is having something called as the NVSwitch right. So, you are seeing here this is a baseline version and this is the version which is of a particular server which is referred to as NVIDIA DGX 2.

Now, what you are seeing is in the system config, in the first one the baseline consist of a server which is of type DGX 1. DGX 1 is another server, this DGX 1 basically has 8 GPUs. So, you have basically 8 GPUs and what you are doing is in the baseline version is basically that you have 2 GPU, 2 nodes which is 2 DGXs connected to each other via a interconnect like InfiniBand interconnects.

In this case, the InfiniBand type is also given right. What is I am having a node which has 16 GPUs. So, instead of going across nodes, I am within the node and within the node I am using a specialized switch which is called as NVSwitch. And, when I do not go across and within the node you can see here that the amount of performance gain that you can get over the baseline version is much much higher.

So, if your baseline version is this you can get up to 2.5X speedup by using a network topology of a machine which can do faster communication with a lower latency. And, we are going to talk about that in a bit. So, you can get a large amount of difference based on the machine which you are running it on.

(Refer Slide Time: 08:04)



So, one concept that we need to understand is the definition of throughput and latency. And, it is very important to understand it because it is not just about how much speed network I have in terms of bandwidth, but also latency matters a lot right. Because, if you would have seen in the previous set of slides, the communication is happening quite frequently.

Now, the gradients are actually maybe very small. So, you do not need a network which is having very high bandwidth, but you might require a network which is having very low latency. So, one concept that we are going to look at is this. So, latency is the time taken for the transfer of a single packet of data which means if I were to just send, if I just were to go from here to the airport alone right.

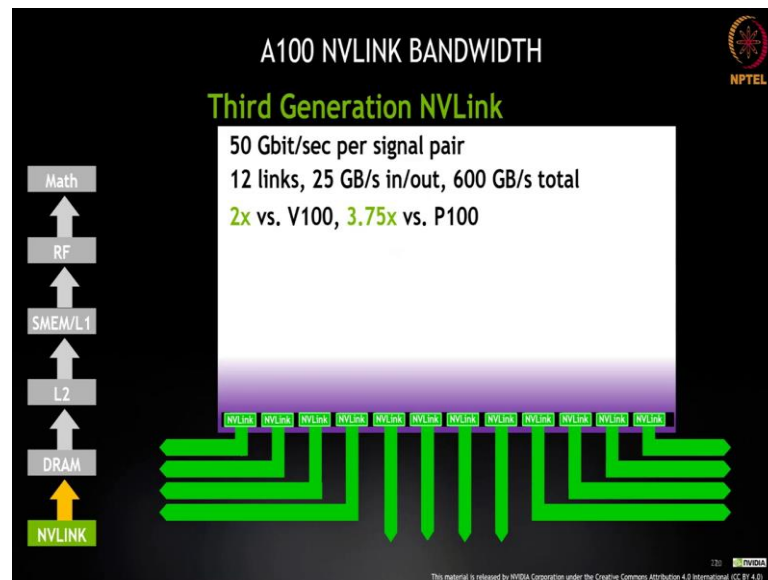
So, based on the kind of a bus or whatever method I use I can reach faster or slower. I am not confined, let us see the number of people who are going to go with me. So, it is like latency. So, whenever you are testing your networks, if you do a speed test of your network, you always get two numbers. Latency would be given in terms of how many milliseconds or microsecond it takes to do a ping.

So, it is the ping latency right. In this case, we are talking about the latency of the data being sent from one place to the other. The throughput is basically measuring the amount of data, that can be transferred in 1 unit of time. Like can I transfer 10, 10 of them together in one shot, can I transfer 1 kB of data, 1 MB of data or much more amount of data?

So, how much can I transfer in one shot? So, these are two different things and they are kind of related to each other. They can be related by equation that you see there right. The throughput is basically nothing but the bus width divided by the latency. An interconnect is the way the two systems are connected. So, basically the two systems are connected to each other via one or the other form of interconnects.

For example, two systems may be connected via LAN which is your normal Ethernet or they can be connected to each other via say a PCIe bus, if they are within the same node. 2 GPUs can be connected within the node using some other interconnect called as PCIe.

(Refer Slide Time: 10:48)

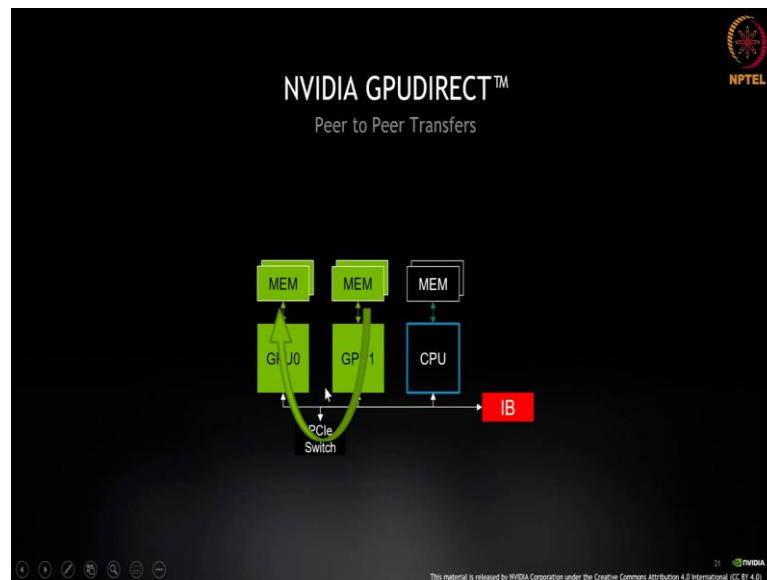


To give you an example of our latest generation architectures like A100, A100 basically has a specialized interconnect which is called as NVLink. If you look at the ampere cards which are present the later generation, they have a particular interconnect called as NVLink. And, this is a third generation NVLink and you can see here that each and every; so, you get up to 50 gigabit per second per signal pair.

So, you have different number of signals and you can get up to 50 gigabit per second per signal pair and you have 12 of them right. So, if you divide it by 2, because you can do both in and out in parallel. So, you basically have 12 links which can run at in and out speed of 25 gigabytes per second. A total amount which comes to is 600 gigabytes per second of total which is twice more than a previous generation which was V100.

So, based on the type of GPU you are going to use, the communication across and the NVLink bandwidth or sometimes you might be using like if you use the pre sensors on the Google Cloud or the other instances out there, it does not even have a NVLink. And, when you start using multiple GPUs, you may not get the scaling that you expect out of it because, it does not have this specialized network protocol which is there in A100 called as NVLink.

(Refer Slide Time: 12:19)

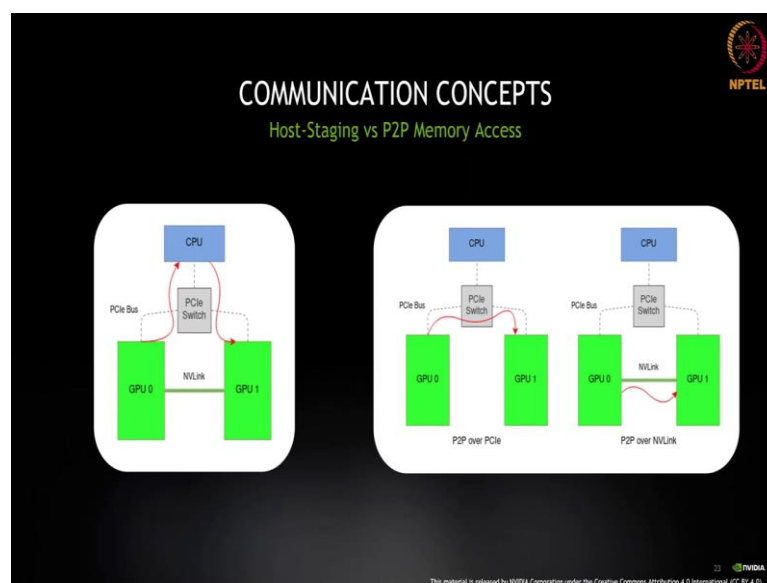


And, based on the types of system topology that you have, you may or may not have these kinds of features. Like here we are talking about a feature called a GPUDIRECT. What the GPUDIRECT basically means is that if one GPU is connected to the other GPU in a friendly peer to peer manner or which can allow GPUDIRECT; the GPUs can directly talk to each other, which means one GPU can directly transfer the data to the other GPU.

And, it does not need to depend on any other thing to do the transfer, that kind of matters. And, it is dependent completely on the kind of system that you are having GPU in.



(Refer Slide Time: 13:07)



The other method is also the communication concept which I was showing sometime to you right. So, if the communication interconnect in the previous slide that I showed you, we noticed some basic terminologies that happen in an interconnect right. We can also have multiple types of interconnects.

Like on the left hand side you are seeing a method which is called as a Host Staging. The data which is being transferred from different GPUs, the data is traversing from GPU 0; as you can see here it is traversing via the PCIe bus to the CPU first. And, then from the CPU it is staged in a buffer before being copied into GPU 1 right. So, based on the type of system topology or the method that you have, you are going to basically have these things right.

Whereas, if you see the second method, if the GPU direct was enabled and if you are using a better system topology in the peer to peer communication approach, they can directly be connected via interconnect right. Here, you can see here the interconnect is PCIe switch and GPU 0 is directly talking to the other GPU via the PCIe switch or it can use a switch the NVLink protocol that I just talked to you about, directly using much more higher bandwidth and lower latency as compared to using a PCIe switch.

Now, why am I telling this, because some of the numbers which I am showing you based on the kind of hardware that you use, based on the kind of network topology or the kind of system you have, you will get different performance numbers.

(Refer Slide Time: 14:56)

**SYSTEM TOPOLOGY**  
Inter-node communication topology of DGX-1

`$ nvidia-smi topo -m`

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	mlx5_0	mlx5_1	mlx5_2	mlx5_3	CPU Affinity	NUMA Affinity
GPU0	X	NV1	NV1	NV2	NV2	SYS	SYS	SYS	PIX	PHB	SYS	SYS	0-19,40-59	0
GPU1	NV1	X	NV2	NV1	SYS	NV2	SYS	SYS	PIX	PHB	SYS	SYS	0-19,40-59	0
GPU2	NV1	NV2	X	NV2	SYS	SYS	NV1	SYS	PHB	PIX	SYS	SYS	0-19,40-59	0
GPU3	NV2	NV1	NV2	X	SYS	SYS	NV1	PHB	PIX	SYS	SYS	SYS	0-19,40-59	0
GPU4	NV2	SYS	SYS	SYS	X	NV1	NV2	SYS	SYS	PIX	PHB	PHB	20-39,60-79	1
GPU5	SYS	NV2	SYS	SYS	NV1	X	NV2	NV1	SYS	SYS	PIX	PHB	20-39,60-79	1
GPU6	SYS	SYS	NV1	SYS	NV1	NV2	X	NV2	SYS	SYS	PHB	PIX	20-39,60-79	1
GPU7	SYS	SYS	SYS	NV1	NV2	NV1	NV2	X	SYS	SYS	PHB	PIX	20-39,60-79	1
mlx5_0	PIX	PIX	PHB	PHB	SYS	SYS	SYS	SYS	X	PHB	SYS	SYS		
mlx5_1	PHB	PHB	PIX	PIX	SYS	SYS	SYS	SYS	PHB	X	SYS	SYS		
mlx5_2	SYS	SYS	SYS	SYS	PIX	PIX	PHB	PHB	SYS	SYS	X	PHB		
mlx5_3	SYS	SYS	SYS	SYS	PHB	PHB	PIX	PIX	SYS	SYS	PHB	X		

Legend:  
X = Self  
SYS = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UP1)  
NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node  
PHB = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)  
PIX = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)  
PIX# = Connection traversing at most a single PCIe bridge  
NV# = Connection traversing a bonded set of # NVLinks

So, here you can see the system topology, now either question you might ask is how do I know what kind of a system topology I have? The good thing is if on the end, when you install the NVIDIA drivers; you can use this particular command here. Like in this case, it has `nvidia-smi topo -m`. This `nvidia-smi topo -m` is topology - m stands for matrix format.

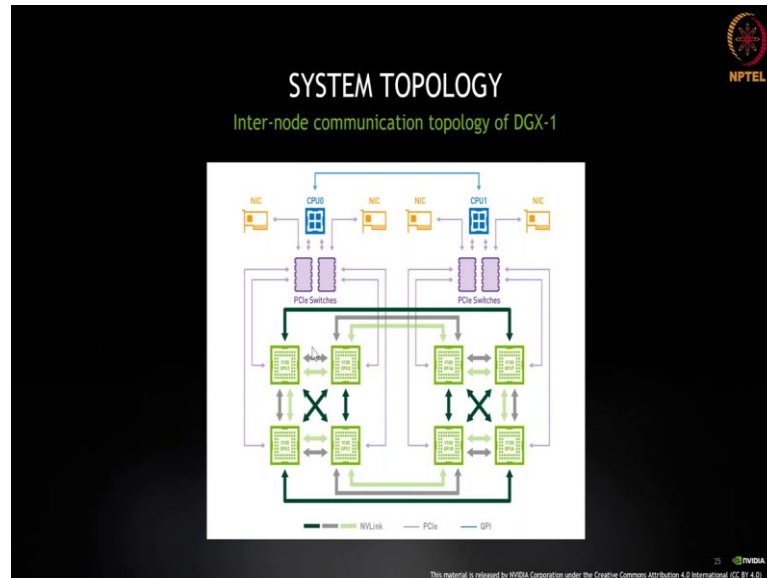
So, you can see here this is a particular output of a system which is DGX 1. What it is showing you is from GPU 0 to GPU 7. And, then it also has the InfiniBand adapters of Mellanox 4 of them. And, then if I were to go from GPU 0 to GPU 0, it is X because I am within the same GPU. But, if I were to talk from GPU 0 or transfer the data from GPU 0 to GPU 1, I can use NV1.

NV1 is what? It is nothing, but NVLink which sorry it is the NVLink 1 right. Same, a GPU 0 to GPU 2, I can use NVLink and you can see I can use NVLink from GPU 0 to GPU 4. But, when I want to talk from GPU 0 to GPU 5, it says that the communication protocol is SYS, SYS. What does SYS mean? It is connecting or traversing PCIe as well as SMP interconnect which means it is going to basically do the host staging.

So, it is going to take the it is going to take the larger path, the lower bandwidth part, the higher latency part. And, the CPU comes into the picture, resulting into really-really low performance between GPU 0 to GPU 5, 6 and 7. Now, that is how the overall topology can be found out very efficiently.

A network topologies which do not have a really good interconnect pattern, they may result into not so much of good performance numbers when you are trying to scale across. So, you can use this command `nvidia-smi topo -m` to see the network topology.

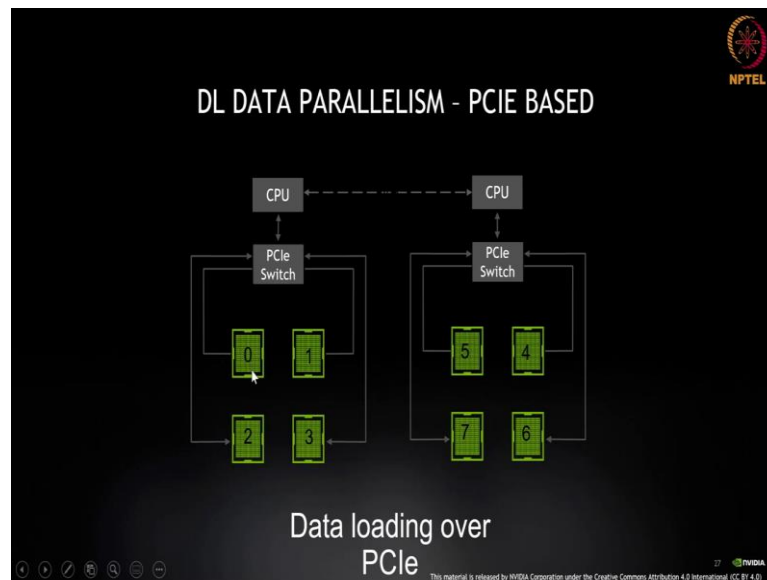
(Refer Slide Time: 17:21)



So, this is the topology that I was talking to you about which is there in DGX 1. You can see here that there are 2 CPUs, both the CPUs have PCIe switch via the PCIe switch. So, the CPUs and GPUs are connected to each other via the PCIe bus. And within the GPUs so, you have like here you can see here you have total 8 GPUs and those GPUs are connected to each other via also the NVLink. So, you can see here these ones are the NVLink protocols, while these ones are the PCIe buses right.

So, as I was showing you sometime back I said from GPU 0 to GPU 5, you can see here there is no NVLink, that is why it has to go from here to PCIe, then to CPU and then come to GPU 5. Now, you can see here that, this is what I was just trying to show you previously in the `nvidia-smi topo -m`.

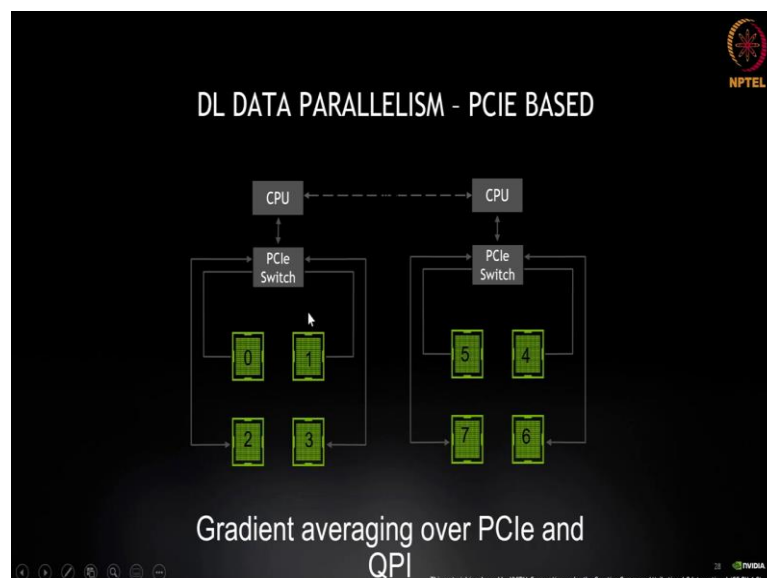
(Refer Slide Time: 18:28)



So, why does it matter for us right? Now, that is what we will see in this particular slide, where when we are trying to do this communication how does it matter. Suppose, if you had a system where you had this PCIe bus, you did not have NVLink. So, if you remember there are different kinds of communications which are happening.

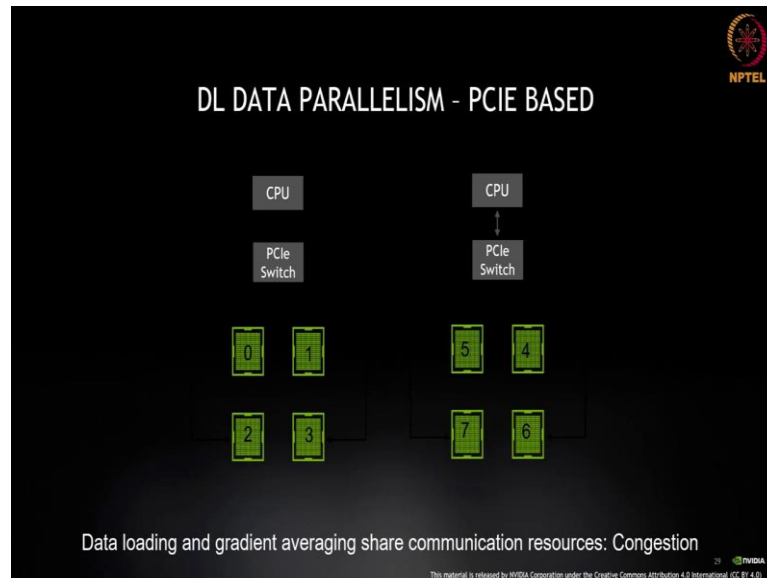
So, the first thing you are going to do is that you are going to load the data from the file system to GPU to the CPU to the GPU. So, you are basically doing the data loading over the PCIe bus. So, you load from file to CPU to PCIe to the GPU.

(Refer Slide Time: 19:08)



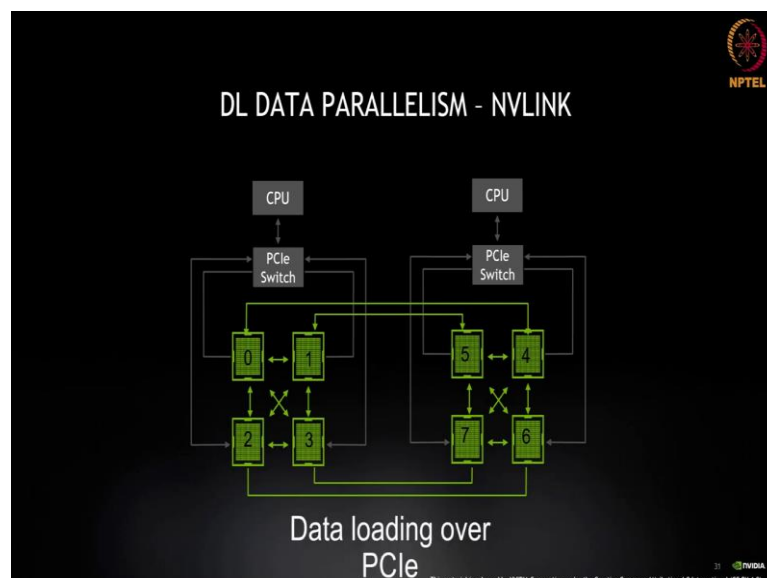
And, when you are running it in parallel, you are also doing at the same time gradient averaging and that also you will do via the PCIe. So, you are using the same PCIe for reading the data or transferring the data to the GPU. And, also you are using it for the gradient averaging part which will basically result into choking of the PCIe bus alone right.

(Refer Slide Time: 19:31)



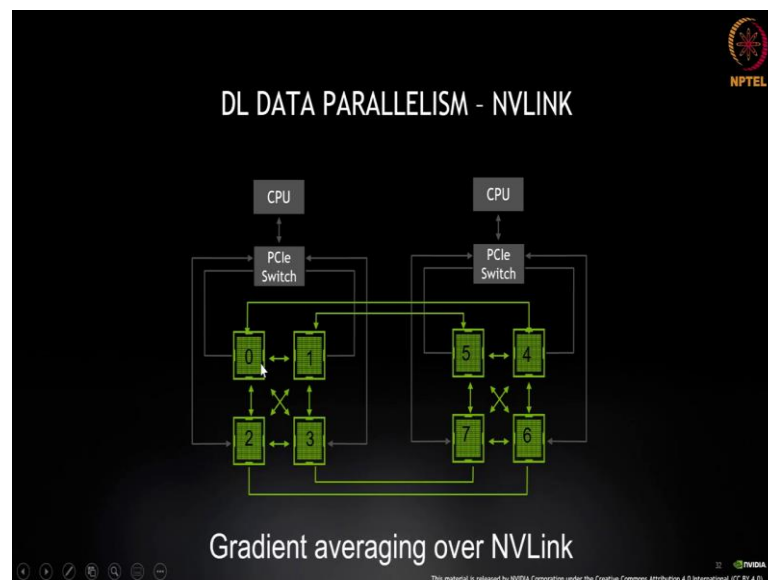
So, the data loading and gradient averaging are basically sharing the same communication resources leading into congestion right.

(Refer Slide Time: 19:41)



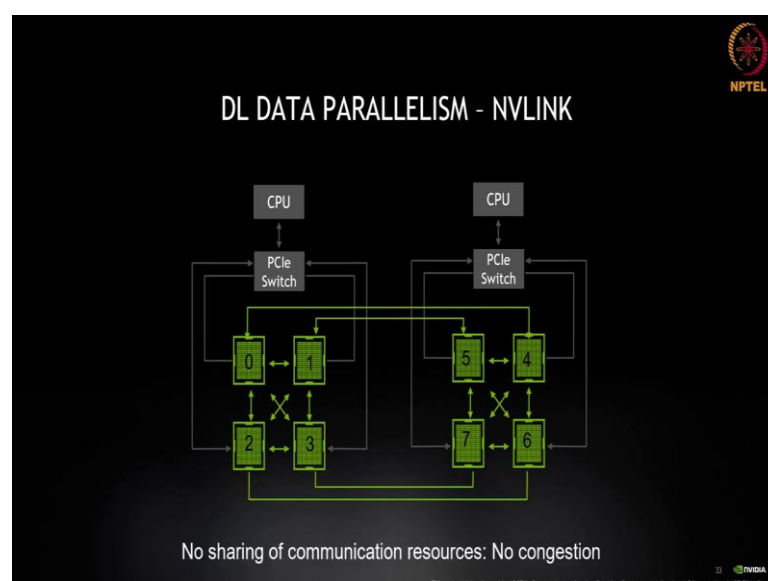
What if you had a different system and that system basically had NVLink inside it? In that particular case, you can basically do this, that the data loading actually happens while the PCIe bus from the CPU taking this particular paths. While, the gradient averaging can basically happen via the NVLink.

(Refer Slide Time: 19:58)



So, the GPUs can talk to each other via the NVLink rather than via the via the PCIe, same PCIe bus which is being used for the data transfer as well.

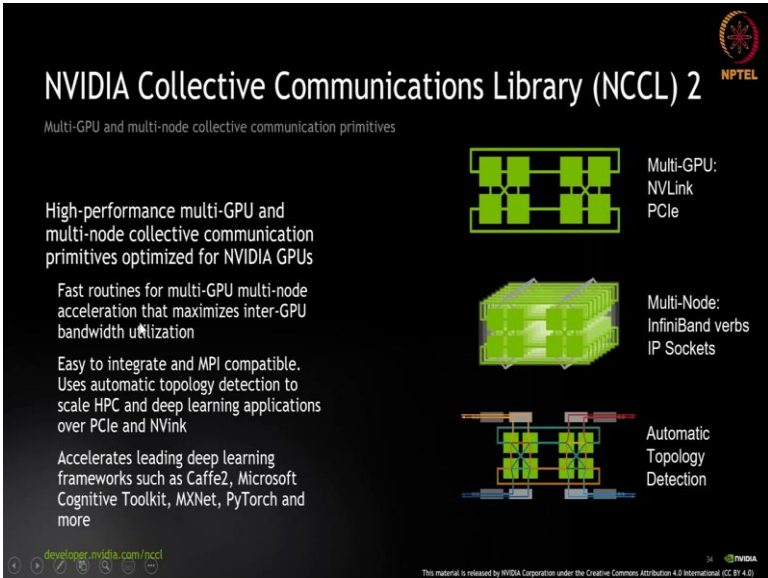
(Refer Slide Time: 20:17)



So, there is no sharing of communication happening across the resources. And, hence resulting into almost no congestion and resulting into much more better a scaling; as I would say or the performance part of it. Now, the question you might ask is should I worry about it, should I not worry about it? It is always a good idea to look at the system and then understand on whether you are going to get good scaling or not.

But, from a point of view of the frameworks like PyTorch, TensorFlow and all of them, they basically take care of all of this communication behind the scenes for you or, but you might or might not get good performance.

(Refer Slide Time: 21:09)



The slide is titled "NVIDIA Collective Communications Library (NCCL) 2" and features the NPTEL logo in the top right corner. Below the title, it states "Multi-GPU and multi-node collective communication primitives". The slide is divided into three main sections, each with a diagram and descriptive text:

- Multi-GPU:** NVLink, PCIe. The diagram shows four green squares representing GPUs arranged in a 2x2 grid, connected by lines.
- Multi-Node:** InfiniBand verbs, IP Sockets. The diagram shows three green cubes representing nodes stacked vertically, connected by lines.
- Automatic Topology Detection.** The diagram shows a complex network of green squares and lines representing a multi-node topology.

On the left side of the slide, there is a list of features:

- High-performance multi-GPU and multi-node collective communication primitives optimized for NVIDIA GPUs
- Fast routines for multi-GPU multi-node acceleration that maximizes inter-GPU bandwidth utilization
- Easy to integrate and MPI compatible. Uses automatic topology detection to scale HPC and deep learning applications over PCIe and NVLink
- Accelerates leading deep learning frameworks such as Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch and more

At the bottom left, there is a URL: [developer.nvidia.com/nccl](https://developer.nvidia.com/nccl). At the bottom right, there is a small NVIDIA logo and a Creative Commons license notice: "This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)".

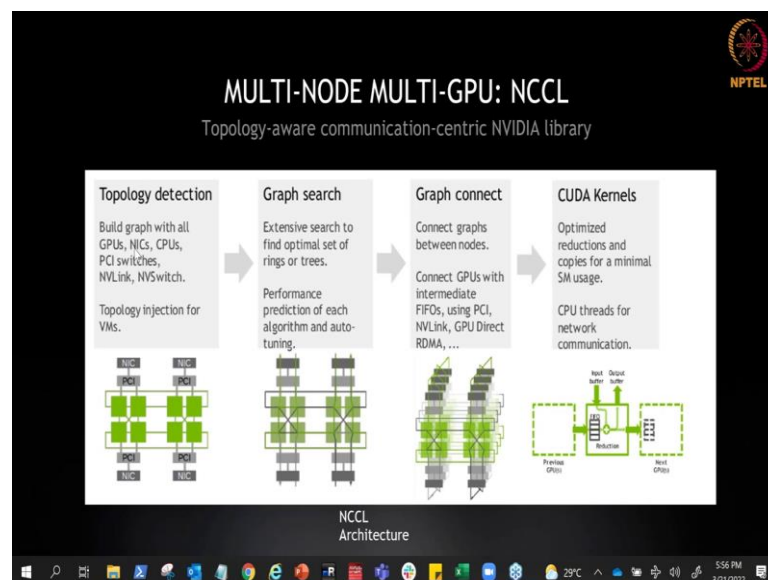
But, NVIDIA has provided libraries which can result into better performance. It tries to find a lot of things like a all of your frameworks like PyTorch, TensorFlow and all; they use our library when you activate multi-GPU computation. A library called as NVIDIA Collective Communication Library or NCCL. Basically, NCCL is a library which provides you many things. It provides you fast multi-GPU and multi-node communication.

So, it provides you fast collective communication routines for multi-GPU, multi-node acceleration. NCCL improves the application performance by maximizing inter GPU bandwidth utilization. And, also it is very easy to integrate or optimize for high bandwidth kind of networks. And, it is basically integrated into all of the major deep

learning frameworks like Caffe, Microsoft Cognitive Toolkit, MXNet and PyTorch TensorFlow and all.

So, you can see here it enables multi-GPU NVLink or PCIe based. It enables multi-node InfiniBand based topology. It also does automatic topology detection. So, let me just take you to another slide.

(Refer Slide Time: 22:22)



So, as I was saying that basically NCCL provides a lot of things. It does the lot of behind the scene things, like the first thing that it will do is to do topology detection. What it does is it looks at the overall system. It builds a graph including where all our GPUs, NIC cards, InfiniBand cards, CPUs; should I be traversing PCIe switch or does it have NVLink or does it have NVSwitch.


So, it does a topology injection for different settings. It does a graph extensive search to find the optimal set of rings or trees. So, it tries to find what is the best traversal path. It does a graph traversal and tries to find the best path, whenever you want to communicate across. And, it does the graph connection based on that right and finally, then it will it runs all of this.

The communication happens not via the CPU, but it actually runs inside the GPU as small small GPU kernels and the CPU threads are basically for network communication. So, it uses a smaller amount of GPU for basically doing these kinds of things. So, you



can see here, it does the lot of things behind the scene for you so, that you do not have to worry about it. The only thing is if you have a system which has better system topology, it will help you in doing it so.

(Refer Slide Time: 23:43)

 NPTEL

## MULTI-NODE MULTI-GPU: NCCL

API:

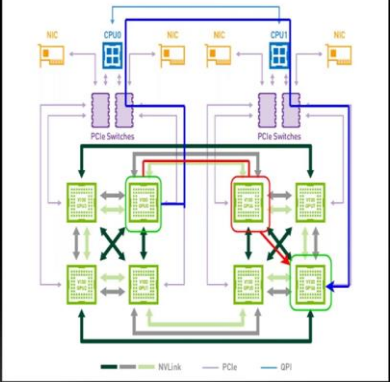
```
// Communicator creation
nnciGetUniqueld(nnciUniqueld* commid);
nnciCommInitRank(nnciComm_t* comm, int nrank, nnciUniqueld commid,
int rank);

// Communicator destruction
nnciCommDestroy(nnciComm_t comm);

// Point-to-point communication
nnciSend(void* buff, size_t count, nnciDataType_t type, int peer,
nnciComm_t comm, cudaStream_t stream);
nnciRecv(void* rbuff, size_t count, nnciDataType_t type, int peer,
nnciComm_t comm, cudaStream_t stream);

// Collective communication
nnciAllReduce(void* sbuf, void* rbuf, size_t count, nnciDataType_t type,
nnciRedOp_t op, nnciComm_t comm, cudaStream_t stream);
nnciBroadcast(void* sbuf, void* rbuf, size_t count, nnciDataType_t type,
int root, nnciComm_t comm, cudaStream_t stream);

// Aggregation/Composition
nnciGroupStart();
nnciGroupEnd();
```



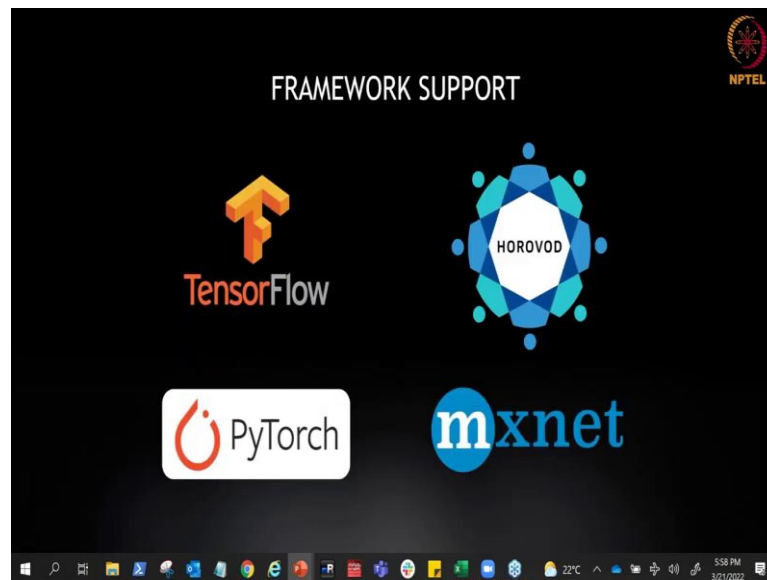
NCCL optimization: Blue path is taken by CUDA-aware MPI  
and Red path is taken by NCCL

22°C 5:57 PM 3/21/2022

So, if I were to use NCCL separately, not within this frameworks. It basically does something like this. You will write a code like first is to create a unique Id or the communicator. You are going to create communicator and then you are going to do a different communication in forms of like `nnciSend` or receive or you can do collective communications like all radio, broadcast which are particularly very well known in the MPI world.

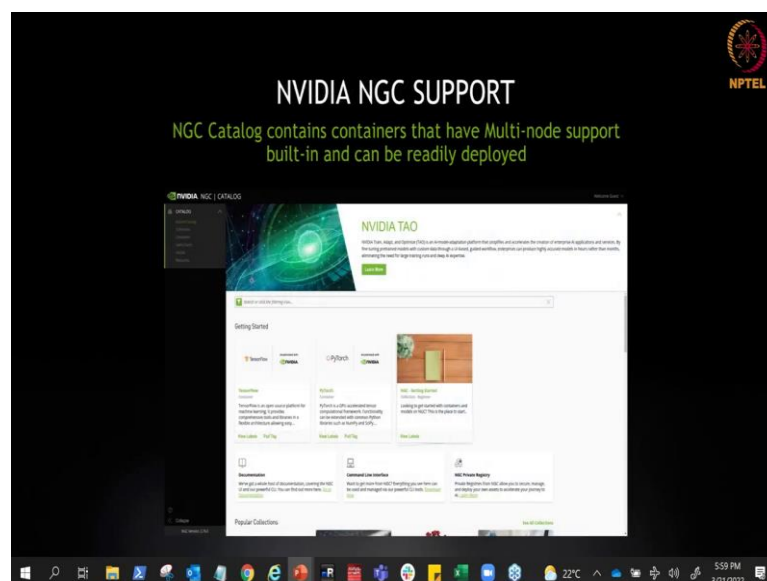
But, the good thing is basically all of these frameworks behind the scene, whenever you want to do this gradient averaging across and you want to communicate. All of this code has been written behind the scenes so, that you do not have to worry about this. But, the library NCCL is in general present and you can utilize it if needed.

(Refer Slide Time: 24:42)



And, as I said all of this frameworks TensorFlow, Horovod, mxnet, PyTorch, all of them are it is important to know that how this framework basically implement this. And, each of them have a different basically way of using multiple GPUs and they call it as strategy. Like in PyTorch, you will have torch load distributed. It has three main components. While, in Horovod is a very popular distributed framework that works with TensorFlow and PyTorch or mxnet, it is a completely different framework.

(Refer Slide Time: 25:19)



And, as we had shown you previously also, all of this is present on our NGC. So, NGC as you remember sometime back NVIDIA HTTP cloud, all of these containers supporting multi-GPU and the right version of NCCL and all are contained into one particular unit. If you do it manually, it is very very error prone. So, rather than that you can actually use the NGC and make sure that you have all the dependencies which are set.

So, with that I am done with this session today. Today, what we covered was the requirement of how to use multiple GPUs, what are the effect of using multiple GPUs when you have different kinds of systems and also how you can check the different parameters.