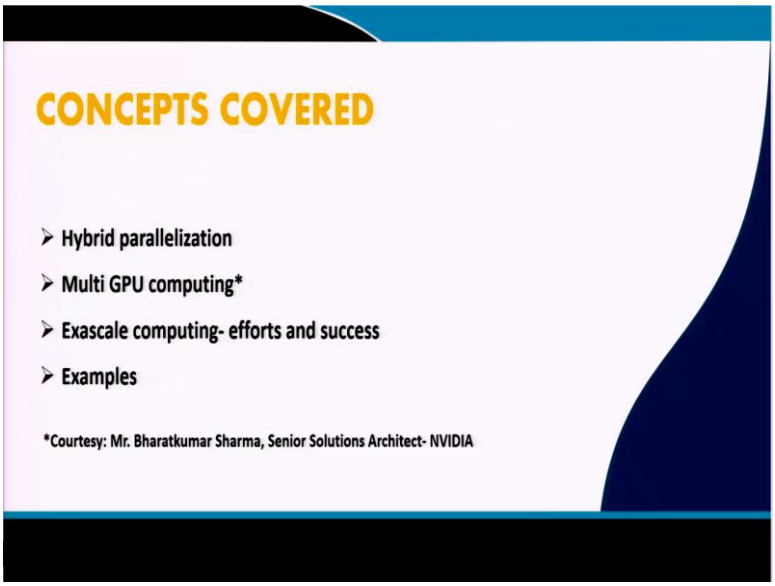
High Performance Computing for Scientists and Engineers
Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Module – 04
GPU Computing
Lecture – 40
Hybrid parallelization and exascale computing

Hello, welcome to the class of High-Performance Computing for Scientists and Engineers. We are in the last lecture of the last module. We are continuing with GPU computing and the last lecture is hybrid parallelization in which essentially, we will talk about multi-GPU computing and see how this can cater to the need of the hour in terms of exascale computing.

Exascale means; the performance of the computing infrastructure will be exaflops, exa-floating point operation; 10 to the power 18 floating point operations per second.

(Refer Slide Time: 01:10)

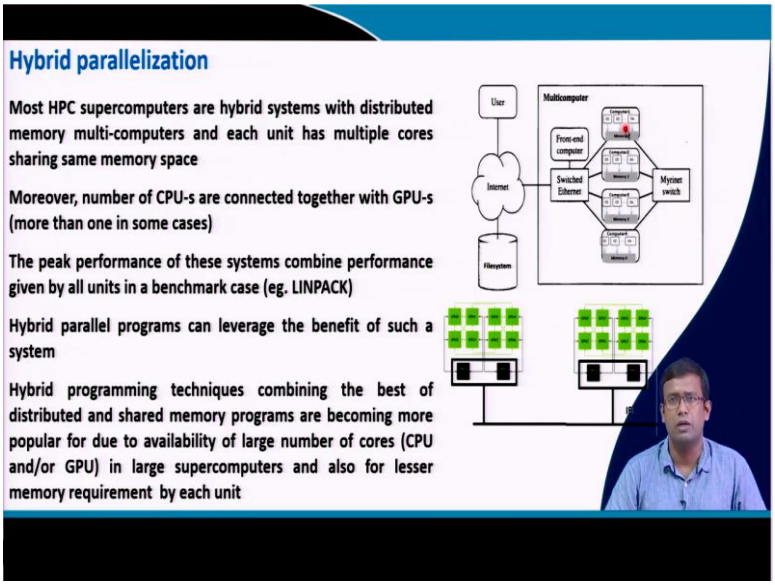


and, how that can be achieved. In this lecture we will first look into hybrid parallelization combining CPUs only that is distributed computing and shared memory computing, how they can be hybridized and we can take full advantage of a modern HPC platform using hybrid parallelization. Then we will see if there are multiple GPUs in the infrastructure then what are the computing instances that are required for that. We will also see how the performance of multi-GPU computing and I will start with discussing why this is important in context with the

modern supercomputing platforms that we are having all in all the best performance can be obtained if you think of doing multi-GPU computing and hybrid parallelization. So, we will look into that.

Then we will see what is exascale computing and what are the efforts behind exascale computing and what are the successes. We will see at least one example of exascale computing in today's date that is the only example available here. On multi-GPU computing I got resources which will be discussed in this class from Mr Bharat Kumar Sharma who is a senior solutions architect at NVIDIA.

(Refer Slide Time: 02:36)



So, if we look into a HPC supercomputer with meaningfully good performance, we will see this is basically a hybrid system. We have looked into this diagram earlier when we are discussing the architecture of parallel computers. So, what is here? That the supercomputer is essentially a combination of several computers and that is why we call it a multicomputer. Many computers have their own memory and processing units connected through a high speed ethernet cable, myrinet or InfiniBand switch and they are connected again through an ethernet to the internet and also to a front-end computer. So, you can access the front-end computer and the front-end computer can offload their jobs there.

Now, if we come to each of the computers inside the multi computer they are like a multicore computer itself. A multicore computer means if you look into your own pc, we will see that there are at least two or four or more processors available. So, each of the computers in this

architecture is itself a multicore system. So, they are sharing a common memory and many CPUs are connected with them. Many computers with their own memory and own set of CPUs are connected together. So, this is essentially a distributed and shared combined hybrid parallel architecture.

Now, if we have to leverage the best benefit of using this type of architecture, we have to think of the applications which we will be running in distributed computing more primarily, and then each job in the distributed computing application can be multi-threaded, so that they are running in all the processors inside one particular computer.

If we can write the program accordingly, and design the job schedule accordingly, then we can get the best benefit of this system. Now, if we think of the top supercomputers in today's date, we will see that there are a number of CPUs which are connected to GPUs and sometimes some have connected to more than one GPUs.

So, there are multiple CPUs inside one computer and then these CPUs are also connected to more than one GPUs in some cases. So, we can see that this is a schematic of a system where there are two CPUs in one motherboard and each CPU is connected to four GPUs.

So, this essentially gives you a one unit in the distributed computing infrastructure, but these are shared memory systems as well as they are connected with the GPUs. Then all the computers in this distributed memory infrastructure are connected through high-speed InfiniBand cable.

So, when again we think of utilizing this resource, we should write programs which can take advantage of this many GPUs present here. So, essentially, they have to distribute the program into multiple small tasks using distributed memory and each of these tasks now can be mapped into a group of tasks or threads which will be launched in the GPUs; sorry.

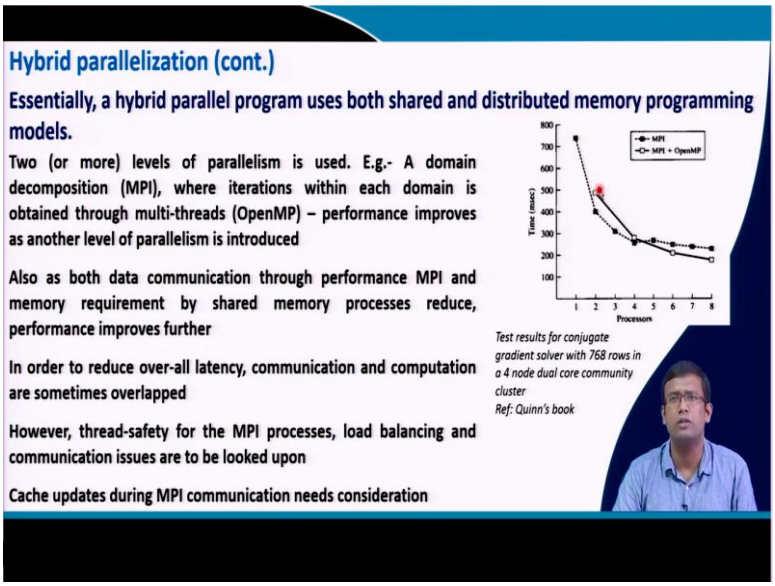
If you go to the top 500 supercomputer list you will see what is the peak performance of the system. So, this peak performance is tested when an application runs in a hybrid computing environment over all the computing codes present in the infrastructure. This application is typically some of the benchmark cases like in pack solvers linear algebra solvers and some matrix operations are done and how many floating-point operations is done per second is noted. Now, when we talk about peak performance of a supercomputer it is not the theoretical performance. it is not a combination of floating point per second speed of all the units present

there. It is the performance which is obtained while running a job which is spanning over all the processors, coprocessors or GPUs in the system. That is what we can tell that if we can write hybrid parallel programs if we can think that the program that I am going to execute will go to multiple computers and inside each computer it will use a shared memory space and launch multiple threads.

Or inside each CPU it will use GPUs then we can leverage the benefit of this large system. If we say for example, if we talk about a multi computer like this and write only a distributed memory program which will use all the CPUs present here. So, even if we come to a shared memory system within the shared memory system it is operating like a distributed memory program and all the overheads due to memory copying and data transfer will be there, but within a distributed memory system when it is a within one unit when this is basically a shared memory architecture if you can write it in a hybrid parallel model, then it can utilize that it is pointing to the same memory space and the data transfer overheads can be reduced. So, we can take the best benefit of the system.

Hybrid programming techniques combining the best of distributed and shared memory programs are becoming more and more popular due to availability of large number of cores which is in terms of CPUs as well as GPUs in large supercomputers and also if we think of shared memory system, if we use the entire memory space you need a large memory. If you think of a distributive memory system every time you have to copy data using MPI transfer data using MPI calls it is also large memory handling. But if you think about it in terms of shared plus distributed memory or hybrid memory systems then smaller memory space is being utilized as well as smaller amounts of memory being transferred across processors. So, therefore, the lesser memory requirement is also one significant advantage of hybrid parallels.

(Refer Slide Time: 09:53)



Essentially, a hybrid parallel program uses both shared and distributed memory program models. So, part of the program utilizes something like MPI for distributed memory programming and part of it uses OpenMP or if there are GPUs CUDA or something which is taking care of the shared memory program. So, it is essentially a mixed programming model that is why you call it hybrid which is using both shared memory and distributed memory programming.

Two levels of parallelization are used for example, we can think of solving something like a Laplace equation using a domain decomposition. So, you distribute into multiple domains and use MPI to transfer data across the domains. So, it is an MPI program among the sub domains. Inside each sub domain now we have to iterate over a large number of internal points, and if we can launch OpenMP threads or if it is GPU if you can launch GPU kernel that is parallelizing what is happening inside one particular task in the MPI job.

So, this is using the shared memory parallelization. So, there are therefore two levels of parallelism that can be used even if we can think of increasing one more level like MPI, OpenMP, then CUDA. So, therefore, as we are increasing the levels of parallelism we are, we can expect that the performance will improve.

Also, both data communication through a through MPI and memory requirement by the shared memory process reduces therefore, performance will also improve. But, anything under the sun as well as in parallel computing, you cannot be sure that is only increasing. If you are not doing

things correctly it might reduce. So, as you are increasing the levels of parallelism, as you are increasing more MPI calls, more synchronization etcetera there can be substantial overhead and performance can actually fall down.

We can see an example say there are eight processors and these eight processors are in 4 dual 4 node dual core systems. So, four distributed memory systems are inside each system; there are two CPUs mounted on the same memory space. So, if we think of using it as MPI we keep on increasing the number of processors, we can see that performance drastically improves the time requirement for computing drastically reduces up to 4 processors and then it becomes kind of flattened. Because now it is using a shared memory space only two processors are there; however, there is substantial data communication among the processors. So, also because the problem size is small there is some performance degradation in a larger problem performance should have been more, but data communication gives the overhead here in case of MPI.

But, if we use it in a shared memory system with larger processors the time required is small therefore, the performance is more, but at smaller number of processors we can see when we are using only two processors the MPI plus OpenMP distributed, hybrid memory parallelization gives us worse performance than simple MPI, simple distributed memory system. Because we are trying to overdo it, so one level of parallelism will be sufficient, but we are trying to pull more MPI and OpenMP calls here and bring more overhead into this system.

So, this has to be looked upon that when we are using the other levels of parallelism in a hybrid infrastructure what are we doing in terms of overhead, what is the problem size, what is the resource being used are we copying the data recopying redundant data in case the data can be mapped shared memory can be used are we still copying it. I mean we doing more in terms of copying is their cache issues etcetera sorry.

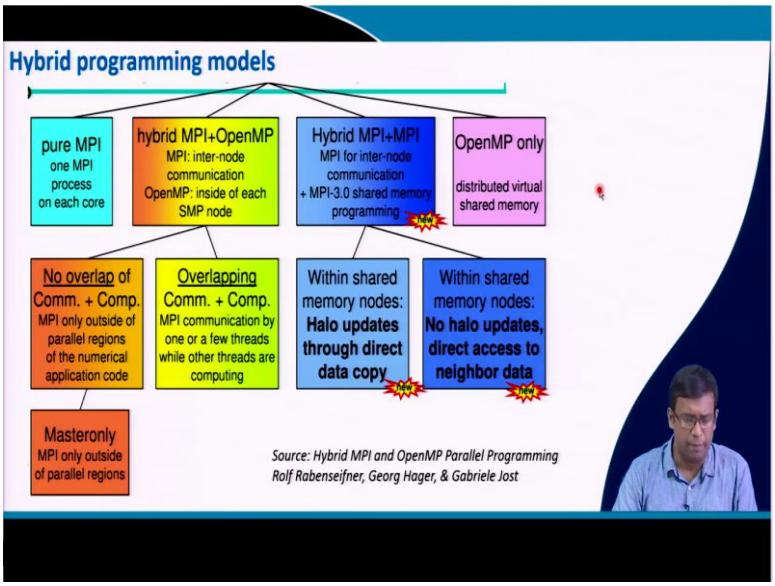
In order to reduce overall latency another thing is sometimes done overlapping of computation and communication; one CPU is responsible for communication among, say there are two different units. Both have multiple CPUs one CPUs in each side is responsible for communication and when this communication is happening the other CPUs are computing. Doing the load balancing accordingly we can overlap computation with communication and can get better speed.

However, we have to also look into thread safety. If we think of the OpenMP process multiple threads are there and if we have to transfer data across two CPUs only one thread will transfer

the data. So, which thread is transferring the data? Has there been any bottleneck or any contention in data transfer among the threads, load balancing and other communication issues because there are multiple communication issues now both shared and distributed memory issues are there that have to be looked upon?

Also, the cache updates during MPI communication needs consideration because one of the processors is taking data from the remote computer and as the data is being copied to this particular computer from a remote computer, will the cache be updated for all the threads. If we have to use cache coherent protocols that will also introduce a level of latency. So, these things have to be considered.

(Refer Slide Time: 15:50)



There are several hybrid programming models and this hybrid programming we are only talking about CPU – CPU hybrid programming. Multi GPU or CPU – GPU hybrid programming we will talk about in a while. But, right now we are talking about only CPU – CPU hybrid programming. One thing can be done by pure MPI. That is, we just discussed earlier that it is a distributed hybrid memory system distributed memory.

Inside each distributed memory there are shared memory systems, but we consider everything to be distributed memory. Even when we look into a shared memory symmetric multiprocessor where one memory unit and multiple CPUs are there, we consider MPI to be responsible for data transfer across the processors and that is also taken as a distributed memory system which is not utilizing the resource in its best possible way.

However, we can do that and many times in many of the legacy codes we have completely MPI parallelized legacy codes we use 8 core, 16 core, 24 core machines; say four 24 core machines gives us 96 core and we are using a legacy code, we use pure MPI there. The other way is MPI plus OpenMP and which is actually discussing now that you distribute the job into multiple small chunks of problems using small memory units using MPI and then each MPI job is again parallelized using OpenMP.

A couple of things that can be done there: one is overlapping the communication with computation. MPI communication will be done by only a few threads, other threads will keep on computing and by that way you can hide some of the communication overhead, subside some of the latency. But there can be cache issues, there can be synchronization issues etcetera.

So, in that case there is no overlap between communication and computation. So, all threads are doing the computation. Once computation is over outside the parallel regime only master thread which is active outside the parallel regime for an OpenMP process will call the MPI calls, communication calls ah. With MPI-3.0 it allows you to use MPI for shared memory systems also.

So, using these MPI-MPI hybrid communication is possible and one idea one is that you use a haloed zone even this is a shared memory use some halo zone and do something like data copying from the halo zone; another is that directly access the neighbour data because there is a shared memory. This MPI-3.0 is relatively new in parallel computing technology.

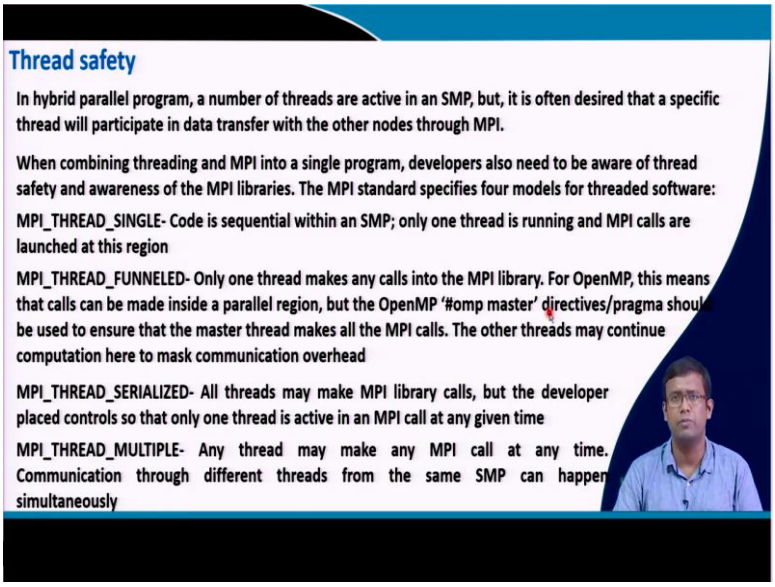
However, this can be used for shared memory programs especially you do not need to use OpenMP, you do not need to use two different APIs and these are the relevant hybrid calls, so that some of the compiler overheads while calling to different APIs can be reduced and with only with MPI you can do this hybrid memory parallelization because MPI itself now can take care of shared memory system.

Or use OpenMP only which is one end was using pure MPI only considering the entire program as a distributed memory program, though you are using a hybrid memory infrastructure. Another is use OpenMP only so, think of distributing the thought of a virtual shared memory though it is a distributed memory map it maps the distributed memories in a way so that it looks like a shared memory like NUMA infrastructure and that is another way of doing it.

But, these two are the most efficient way to utilize hybrid memory architectures. However, there can be overheads there can be issues in programming for which one might have to stick to distributed memory or virtual shared memory systems or simple MPI or simple OpenMP.

But, in between comes these two and which can give you the best result if the program is written perfectly and if the main programs algorithm supports these levels of parallelization. So, this is taken from hybrid MPI and OpenMP parallel programming tutorial from Rabenseifner and others which is present in OpenMP .org website.

(Refer Slide Time: 20:40)



Now, one of the issues is thread safety. In a hybrid parallel program, a number of threads are active in each symmetric multiprocessor. You have a large job. We have distributed this large job into small jobs each have its own small memory local memory access and this units of the decomposed tasks each of them goes to a symmetric multiprocessor.

Symmetric multiprocessor is basically a computer with a shared memory space and multiple CPUs connected with that. In a symmetric multiprocessor you would use another level of parallelization and a number of threads are active there. Now it is often desired that one particular thread will do the data transfer with the other nodes in the system in the computation using MPI. So, the MPI sets the communicator which encompasses a number of nodes. Each node is a symmetric multiprocessor; it has its own memory which is shared among different CPUs. Now, when data communication within the communicator is required then one of the CPUs in the symmetric multiprocessor has to communicate with the other nodes.

If everybody tries to communicate at the same time there can be issues, or if more than once one is transferring to the right side another is transferring to the left side, there can be many communications also and they have to be mapped correctly.

Now, it is important when the parallel execution is happening in the symmetric multiprocessor, we have to identify specific threads which we will do data transfer with the remote nodes, all the threads cannot do that. So, it uses some synchronization, some barriers or some protocols for data transfer also when you talk about OpenMP -MPI hybrid program because all the threads in the MPI program are not going to do data transfer. For that we need to be aware of thread safety that this thread is safely doing data transfer and the other threads are not taking part of it, which thread is there and how it is transferring data without being restricted by other threads.

MPI standard has four different models. First is MPI thread single. So, this is again a MPI function call which ensures thread safety. It says that part of the code is sequential. So, we are out of the parallel regime of the code, only the master thread is active only one thread is active and this thread is doing the data transfer using MPI calls. So, this data transfer is done outside the parallel region and only one thread is active for that.

MPI thread funnelled – when only one thread makes any calls into MPI library. These calls are made inside the parallel region. However, if we call this function MPI thread funnelled then using the OMP master directive one of the threads is identified and only this thread does the data transfer; the other threads may continue computation to mask the communication overhead.

If you do not put any OMP barriers before the communication or OMP barrier before and after the communication the other threads can continue their work, only the master thread will be identified and this thread can do the data transfer.

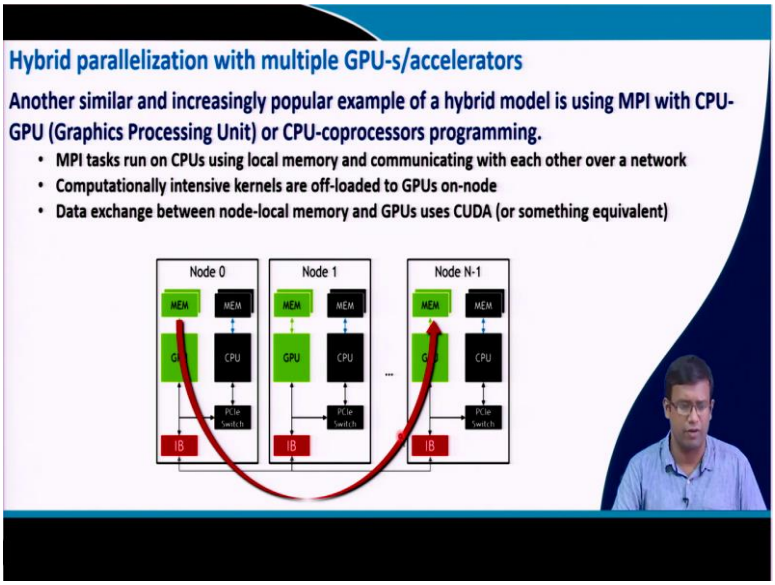
The third model the standard is MPI thread serialized that all threads now are allowed to do data transfer, however, they will do it serially. Like a critical constant all threads will execute that construct; however, one by one they will do that. So, all threads can do the MPI library call, but developers can control the fact that only one thread is active at one given time. So, the threads will do it serially.

The fourth one is MPI thread multiple any thread may make any MPI call and at the same time multiple threads can do data transfer. So, these four are the models and based on the programmer's intention, based on the requirement posed by the algorithm also based on the architecture and compiler level supports and the size of the problem one can decide which model has to be taken and how this data transfer will be done.

But now the important point is that we need to first recognize the fact that not all the threads inside the OpenMP code are doing data transfer because data transfer is through some restricted channels and also data transfer is required only for a certain amount of data in a distributed memory system.

So, if you think of a domain decomposition problem only for the boundary points the data transfer is required. So, only few of the threads will be responsible to do data transfer and programmers should be able to decide that and control that. You understand that this requires certain synchronization steps also to ensure thread safety and that might be a killer that will add to overhead in the computing and that might reduce the overall performance.

(Refer Slide Time: 26:23)



Now, another very important part is how we can do hybrid parallelization with multiple GPUs or multiple accelerators because now we see systems which have many GPUs. So, how to divide a job across these GPUs and that is also another part where we need MPI; a large job has to be broken down and part of these jobs will go to different GPUs.

So, there also we need MPI and as well as well we are offloading a job from CPU to GPU, we need something like CUDA or OpenACC or OpenCL. So, this also gives us another instance of using hybrid programming.

MPI tasks run on CPUs using local memories and communicate within each other because jobs cannot directly go to GPU or CPU has to send job to GPU and CPU has to identify that what part of the job will go to which GPU and if there are multiple CPUs all with GPUs then connection data transfer across the CPUs is done through usually through MPI. This partitioning of the job will also be done through MPI because essentially each GPU will work on its local memory; each GPU will local memory means each GPU will work on the memory on the GPU device. So, it essentially tells that each GPU is a unit in the distributed memory system. Now, in the device stream the memory they are accessing they are accessing as a shared memory. So, it is also a hybrid memory system.

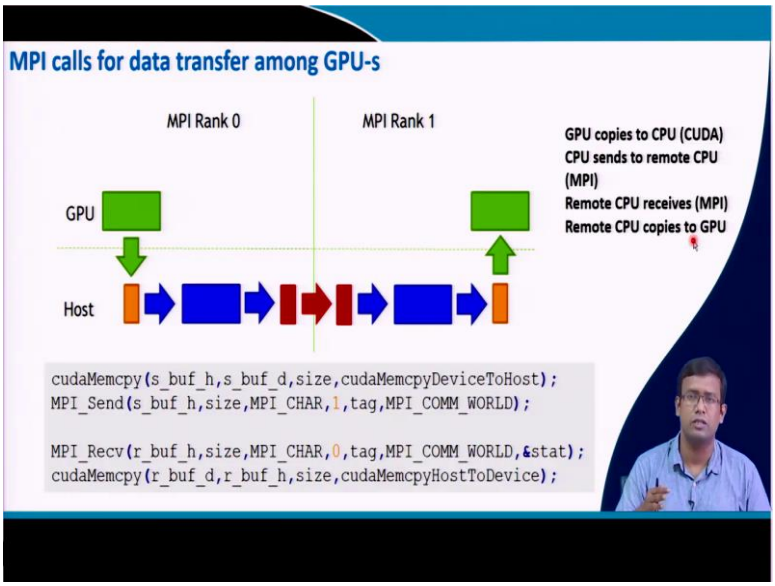
MPI task is to identify, break down the job into multiple small problems and each map each CPU to its own local memory and then set up a communicator, so that each CPU can send data from one to another. The computationally heavy part, the main high computing intensive part that will be uploaded to GPUs by the CPU nodes.

The data exchange among the GPUs will be done first is that data will be copied from GPU to the CPU through CUDA or OpenACC or OpenCL some CPU – GPU and then data can go from CPU to CPU that is the basic framework.

So, what can we see? That there are multiple nodes each node has a GPU and a CPU we consider the fact that each node is a single GPU system. So, we have a large data set which is broken down into smaller local data items and each CPU gets its smaller data and this is essentially copied to the GPUs.

Now, when there are some changes in the local data, this data from the GPU will be copied to the CPU and the CPU will send it to the other CPU which will copy to the required GPU. So, all these CPUs are also connected; all the systems are also connected through something like an InfiniBand network.

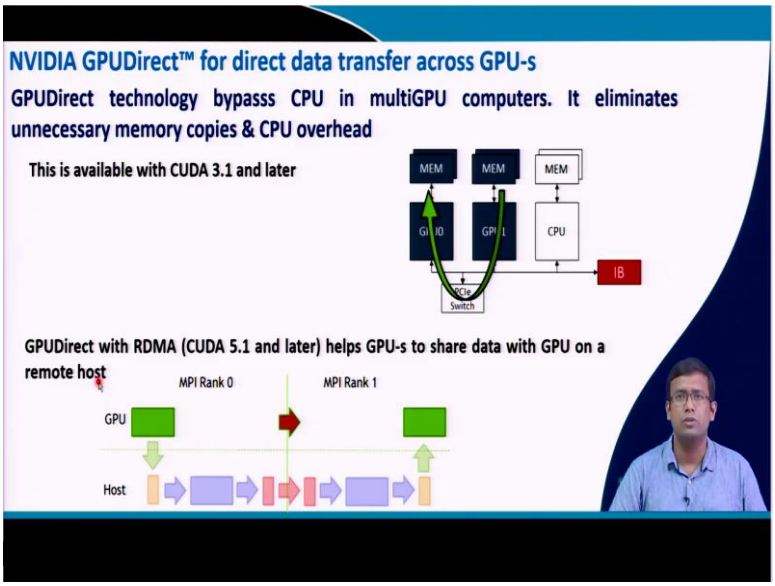
(Refer Slide Time: 29:43)



So, this is the MPI call for data transfer. There are two CPUs which have their own GPU; first is MPI rank 0, MPI rank 1. So, there will be CUDA memory copy if data comes from this GPU-GPU 0 to GPU 1. First through cudaMemcpy data will be copied from the device to the host and then this host rank 0 will send the data to rank 1. So, it will send data to rank 1 and rank 1 now will receive this data which is sent from rank 0 and the received data which is staying in r_buf_h will be copied from HostToDevice to the GPU.

So, this four-step set GPU will copy to the CPU. CPU will send to the remote CPU. Remote CPU will receive from the sending CPU and the received data in the remote CPU will be copied to the GPU associated with it. This is how data transfer happens across the GPUs. GPUs do not use MPI. In many cases GPUs do not send data directly from one to another, through the CPU it is done. Therefore, cudaMemcpy MPI Send Receive, again another cudaMemcpy is required.

(Refer Slide Time: 31:00)



Now, NVIDIA has a technology called NVIDIA GPUDirect for direct data transfer across GPUs. So, we think of a multi-GPU system where there is only one CPU, but a number of GPUs are mounted that if you buy a high-end server you will see a number of GPU slots available there. In each GPU slot there is a GPU.

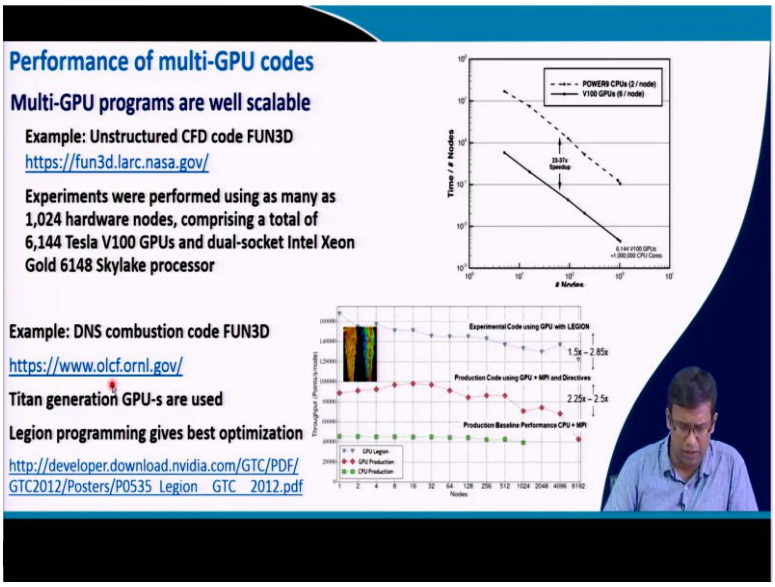
So, then you do not need to do any MPI data transfer, you can directly send data from CPU to GPU and so, NVIDIAs GPU direct technology that utilizes the connectivity inside the multi-GPU systems motherboard and copies data directly from a GPU to the other. This is available with CUDA 3.1 and the later versions. So, it is like you have a CPU and many GPUs are connected, their data is directly copied through the PCI switch from one GPU to the other GPU. You do not need to use the MPI or do not need to copy it to the CPU using cudaMemcpy and then again copy it to the CPU. So, there is some optimization in it. In case you have multiple GPUs, GPU direct this technology with remote device memory access can help the GPUs to share data with GPU on a remote host by passing the CPU channel. That can be instead of copying it to the CPU and then calling the MPI send receive, using GPU direct and RDMA protocol, data can be copied from one GPU to the other GPU.

Both the GPUs are hosted by different hosts, and different CPUs are hosting the GPU. But, the previous part like copying to the CPU and then using MPI to copy to the other CPU and then again copying to the GPU that can be avoided using GPU direct with RDMA.

Now, again in these protocols use some of the optimization some of the both at hardware and compiler level; however, they have their own overheads also. So, it is application dependence; for some applications this RDMA GPU direct parallelization gives better result, but for some of the applications especially for some of the legacy codes which are essentially MPI based code on which GPU parallelization is brought later, probably you have to go to the older method; that is copying to the CPU and then MPI send receive among the CPU, again copying to the remote GPU from the CPU, that has to be followed for some cases.

Of course, it is understandable that we are only trying to introduce these concepts, but these are one level higher concept than simple CUDA programming or MPI programming. So, if you have to explore it there is enough material available in internet resources and you can get some pointers from this discussion and its reference also and start working on that.

(Refer Slide Time: 34:17)



Now, these multi-GPU programs are well scalable. So, if we increase the number of CPUs scalable MPI jobs will say that efficiency is almost constant and almost linearly speed up is increasing. The time required will reduce almost linearly.

Now, if the MPI node is connected to a GPU this linearity still holds. So, these jobs are most of the time scalable. We can look into a CFD code FUN3D where they have experimented with over 6144 V100 GPUs and Gold 6148 Skylake processors. This is the scalability of the CPU code. Each CPU has two codes per node and this is almost kind of linear.

Then once you use GPU it also shows it actually shows more linear plot, but the speed up is 23 to 37 times which is almost in a similar order and this is fixed. So, as we increase the number of GPUs, we get 23 to 37 times speed up compared to the CPU job at all levels.

So, the CPU nodes are increased one is that without GPUs it is seen that up to thousand nodes what is the performance, when you add GPUs then performance increases and this increase in performance is nearly constant as we keep on increasing GPUs. It is not falling down.

There is another example on DNS combustion code the name is not FUN3D.You can find out the code in the Oak Ridge National Lab website and this is code for computing turbulent combustion.

The titan generation GPUs which are one which is the newer generation than the V100 voltage if you are used. Legion programming is used for best optimization; legion programming takes care of multi-GPU systems and some further optimizations are done in legion programming. You can look into legion programming on NVIDIA site.

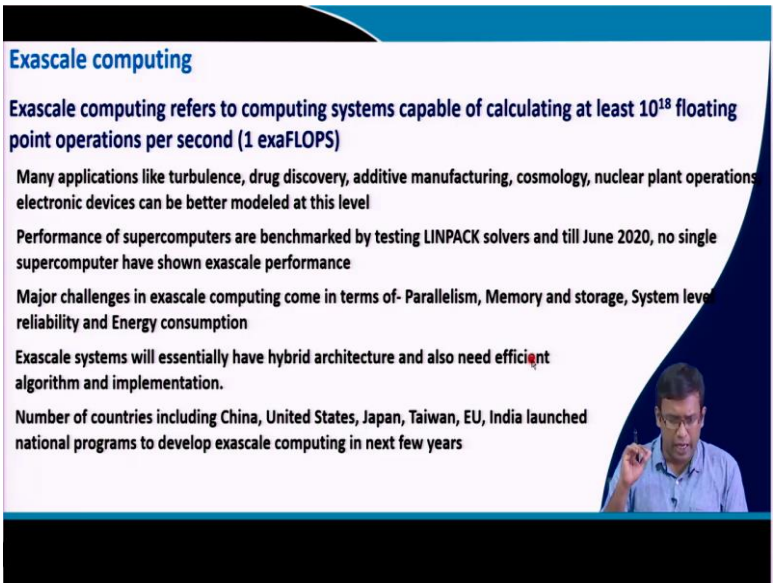
It is seen that this is the performance with CPU plus MPI code, the throughput remains constant; we can keep on increasing the number of nodes. The throughput remains constant depending on how many points are solved by each node and what is in one second as we increase the node. So, what is the speed at which the operation is done per node and this remains constant? This shows that as we are increasing nodes the total number of operations are increasing. So, the speed is increasing nearly.

Once we add the GPUs then we can see that initially the throughput improves, but at a high large number of GPUs it starts to keep falling a little more because of the overheads. But, with legion we can get near throughput and near constant throughput and also very high throughput. So, this is a more optimized program for multi-GPU parallelization, it gives better performance.

However, it is shown that the number of iterations per second per node that remains constant. So, therefore, as we are increasing the number of nodes the total number of iterations per second is increasing, the speed is increasing almost linearly. This is done up to 8000 GPU nodes and a large number of processors.

We can see that the performance improvement is the throughput improvement from 4000 in only the CPU system; in GPU it goes even close to 14000; 3 times improvement in the throughputs.

(Refer Slide Time: 38:37)



Now, we come to exascale computing; exascale computing refers to computing systems that are capable of calculating at least exaFLOPS or 10^{18} floating point per second or exaFLOPS speed is obtained by a system then we call it exaflop system. Why is it an important thing? Because; many applications like turbulence, drug discovery, additive manufacturing, cosmology or astronomy, nuclear plant operations, electronic device design etcetera can be seen that a better modelling of these operations, more physically accurate solutions can be obtained in a realistic time scale, if we can do exaFLOPS speed else it will take astronomically long time and we really cannot achieve the solution in a physically realistic time scale. So, if we need to solve these problems with the right accuracy the problem complexity or problem size is so high that an exaFLOPS computer can be essential.

Performance of the supercomputers are benchmarked by solving LINPACK solvers in them, I have discussed it earlier till last month June 2020, no single super computer I have shown exascale performance. Supercomputer in the sense of the large computing infrastructure which is dedicated to do HPC calculations connected to a together by an InfiniBand and they have CPU-GPU hybrid systems.

None of them have shown exascale performance rather they are quite far away from the exascale performance. In India today we talk about few petaflops performance of the super computers and that is the peak performance that we can think of. So, exascale means thousand petaflops; which is which will take some more time.

The major challenge in getting that performance is parallelism using the right hardware as compiler and software or algorithm to get that parallel program which can actually avoid overheads and give this speed, otherwise overheads will kill it.

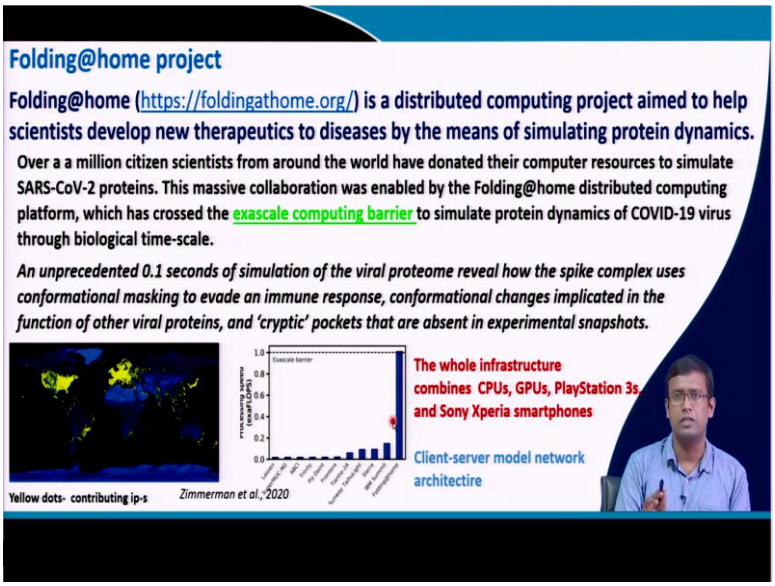
Memory and storage issue that includes both data transfer memory mapping as well as how much memory is being required well how to store that. The system level reliability when you run hundred thousand or one million of CPUs together some someone might fall down just by statistical measure you can think of that if one CPU is running for a long time what is the probability of it going to an idle stage.

So, if you run this large number of processors, then some processors may stall at some point of time, some of the networks may work wrongly. So, in such a large architecture what is the reliability in the systems level. So, that everything is in operation. Energy consumption, we have earlier talked about PUE that if we have to spend 1 unit of energy for computing 0.5 units will be spent by the AC. So, total energy consumption will be very high in that case.

So, these are the major challenges. However, keeping this in mind it is seen that essentially, we need hybrid architecture which is even for petaflops. We are using hybrid architecture and need efficient algorithms and implementation.

Number of countries China, United States, Japan, Taiwan, European Union and also India have launched national level programs where many groups, engineers as well as many companies are coming together and trying to develop an exascale computing facility in the next 1 to 5 years, each country has different plans.

(Refer Slide Time: 42:43)



Now I will come to an important point that there is a project called folding@ home. This project is run by a community of scientists working in protein chemistry looking into computing the protein dynamics for better therapeutic design or better drug discovery.

So, they want to take a virus, see what is the structure of the virus at the protein level, which drug will be effective for that and solving at the molecular or atomistic level and looking into what is happening to the proteins, how are the protein molecules in the in the virus and how they can be killed. So, this is a very gross way to put that, but this idea of doing protein dynamics simulation. Folding@home projects have been running for almost two decades on this. People in this are called citizen scientists. They can share their own computational resources on the internet, they can get connected to that project and contribute computational resources. Combining this over a million citizen scientists have come together and when I am recording these lectures, we are going to an unprecedented event of novel corona pandemic COVID-19 pandemic across the world. Here the forum of scientists many individual scientists, we call them citizen scientists everybody is sitting in his own place, his lab, his room, his office and contributing to this Folding@home project. Together the computers have just connected to the internet, but some of them are super computers also they are connected within itself through InfiniBand. But together they have joined hands and shared the resources, built a hybrid distributed computing architecture across internet connectivity and crossed the exascale computing barrier. So, this showed that together exaFLOPS calculation is obtained and some protein dynamic features of COVID-19 virus has been simulated in biological time scale.

So, this is the first time in history an exascale barrier has been crossed that is not by a conventional supercomputer, but by citizen scientists sharing their computing resources. You can see these yellow dots. These are IP addresses at different locations in the United States, Europe, some in Australia, Africa, some in India, some are in China they are contributing to the computational resource of this system.

So, this is also developing a very novel type of hybrid computing infrastructure, and, the infrastructure combines CPUs – GPUs, play stations, even smart phones and a client server model network architecture has been used where the main job is distributed into many small jobs and given to different computers.

Sometimes they are using distributed memory models and computers are communicating across themselves. Sometimes these are small shared memory programs and then the result is being uploaded to another site. So, they have calculated what will happen to the viral proteome for 0.1 seconds and shown some of the conformational changes in the viral proteome and how the COVID virus masks the immune response. Even human immunity cannot kill the COVID virus, how does it do in the protein level and get some information which is absent in experimental snapshots.

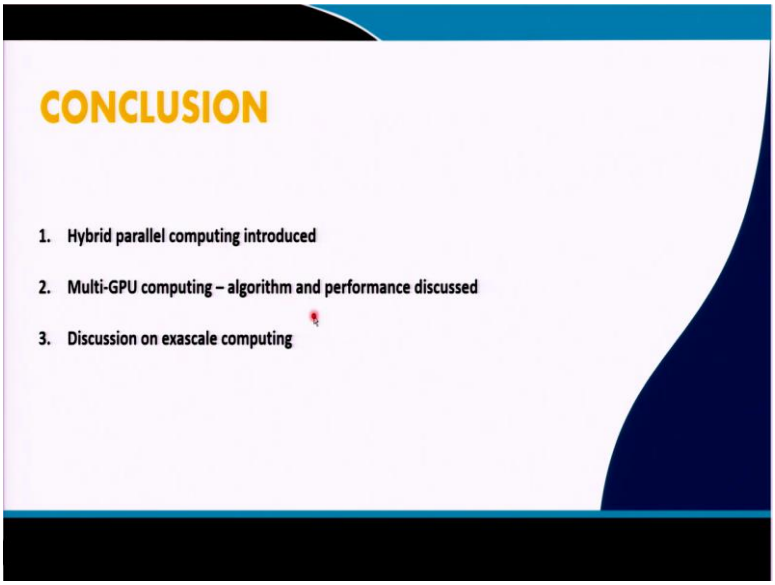
So, this is a remarkable achievement which is done through this floating at home project both in terms of COVID-19 virus research as well as in terms of computationally reaching the exascale facility. You can see that these are the leading supercomputers – IBM Summit, Sierra I have talked to you about these super computers which are much below the exascale speeds; all close to 0.2, in between 0.1 to 0.2 exaFLOPS.

This Folding@home project using the unconventional way of utilizing citizen scientist computational resources has reached an exascale barrier. So, it can be told that you might not be able to access the largest supercomputer today. However, people can think of better ideas, can come together, think of better algorithms and utilize whatever the connectivity is given, utilize them efficiently and cross the barriers in supercomputing.

(Refer Slide Time: 47:43)

Well, these are the references.

(Refer Slide Time: 47:45)



We discussed hybrid parallel computing, multi-GPU computing. We have shown some of the issues on algorithm how data transfer happens etcetera and also discuss the performance issues and we had also a discussion on exascale computing.

Thank you very much for attending this course.