

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

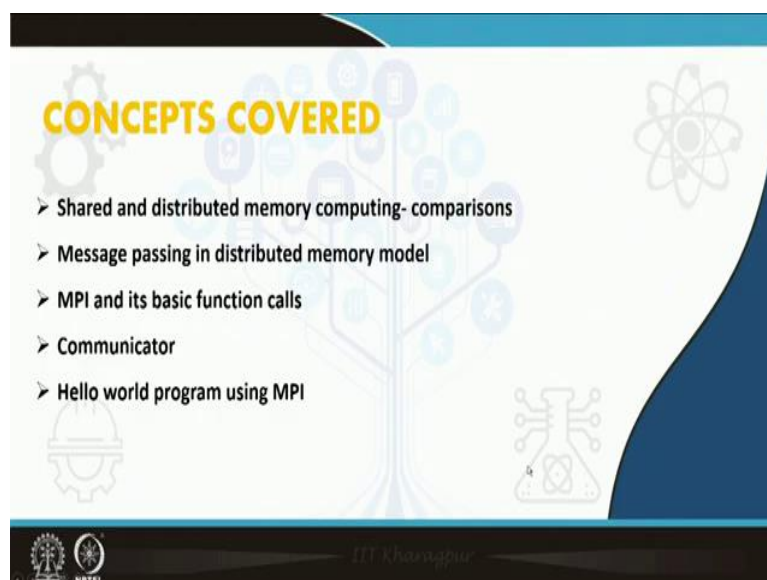
Module - 03
Message Passing Interface (MPI)
Lecture - 19
Introduction to MPI and Distributed Memory Parallel Programming.

Welcome to the class of High-Performance Computing for Scientists and Engineers and we have already discovered 2 modules of this course. The first modules were on fundamentals of parallel computing and in the second module we have discussed open MP programming, which is a programming paradigm for shared memory parallel processors.

Now we will look into distributed Memory Parallel Programming and this is the third module in which we will start discussing the message passing interface which is an API which is heavily used for distributed memory parallel computing.

This is the first lecture which (nineteenth lecture of this course) we will discuss about introduction to MPI; MPI and distributed memory parallel programming.

(Refer Slide Time: 01:15)



So, we will cover shared and distributed memory computing .We have already discussed shared and distributed memory computing, but we will again try to compare these 2 models mostly due to the fact that on shared memory programming we have already built a foundation.

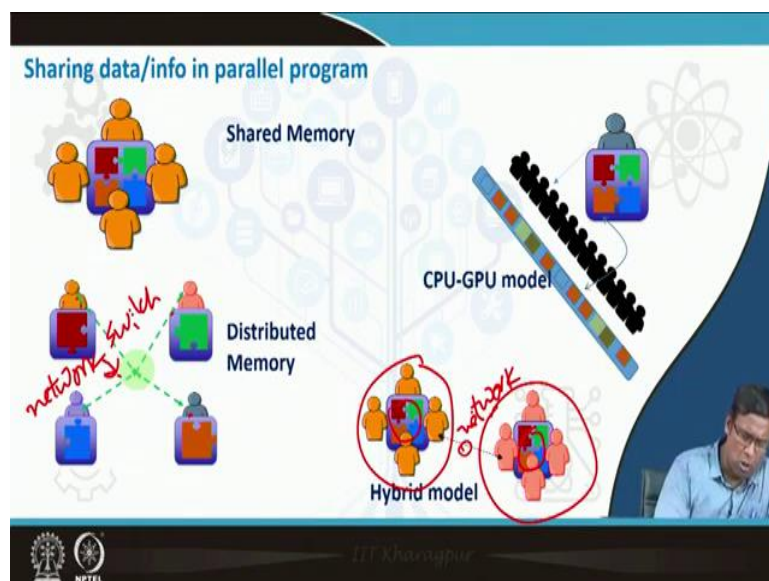
So, now we will discuss what are the differences between shared memory programming and distributed memory programming and from there we will pick up our voyage on distributed memory computing and then we will discuss message passing which is the probably most demarcating pointer of distributed memory computing model.

The API for message passing which is universally used is message passing interface or MPI. We will introduce MPI at that point of time and discuss some basic function calls. We will see what is the communicator and how do we identify the computers which are participating in an MPI job, and how do we give their identity during the program . So, we will discuss communicators in the MPI program and then we will see our first MPI program which is hello world program. If you remember for OpenMP also we started with hello world program; that means, there are multiple processors and each processor will independently write hello world while executing the same parallel programming for OpenMP , here we will do it for MPI.

This will cover the introduction to MPI and message passing based distributed memory computing and with this background in the subsequent lectures we will discuss data transfer protocols in MPI and how we will see how to write scientific computing algorithms in a parallel MPI version .

So, this lecture and the next lecture will be devoted to giving you an introduction to MPI and distributed memory computing.

(Refer Slide Time: 03:39)



When we work on a parallel program we ask that there are multiple processors and each processor we will do some part of work and these processors will work in parallel that is why we call it a parallel program. So, there will be some concurrency in the job and each of the processors will take care of one of the tasks and this task is a concurrent task.

However, because they belong to the same problem; so, there is some requirement of communication across the processors because they are trying to solve the same problem. We have seen in shared memory computing that they write to common address space and whatever each processor is doing after one parallel construct we need to consider the output of each of the processors calculations and combine them or do something with them .We have seen it many times in different OpenMP applications.

So, there is a requirement of sharing data or sharing the same data space or sending information across the processors. We have discussed it earlier also it is like there are few concurrent instructions and each of the processors is taking care of one; however, in shared memory the entire data set is visible to all the processors like 4 people sitting on a table and they have to solve some puzzle together. The concurrent parts can take care in parallel in a distributed memory, so this is a shared memory system. In a distributed memory system, there are different processors and each have their own local data. So, they are working on the local part of the puzzle given a puzzle to 4 different people they are sitting at different tables and working on the local part; however, it is a complete single puzzle. So, they need to communicate across themselves and there is a network switch through which all the computers are connected and through the network switch this computer can send some information to other computers it can receive some information from other computers. They can share data, they can get some information from some of the other computers, and this is a distributed memory model .

The third one which we will actually not discuss over this course, but which gives us a route to very large-scale supercomputing exercises is a hybrid model. There are different shared memory computers where there is single memory shared by multiple processors and these different shared memory processors with their different local data sets are connected to each other again via something like a network switch. So, all the modern hpc clusters, all the hybrid architecture, CPU systems they are like this that there are different shared memory units which are connected to each other via switch. The advantage is that you can get the shared memory computing also you can distribute different parts like large parts of the computing to different units and you can see this distributed memory computing among these parts. So, distributed

memory computing and each of the distributed memory parts is again further parallelized by the shared memory machines. So, this is called a hybrid architecture.

There is also another cutting-edge model which is a CPU GPU model that the CPU which gets the whole problem is connected to a GPU which has multiple small processing units and GPU has its own data set. So, the main data which belongs to the CPU part of the data is copied to the GPU on which these multiple processors will work.

So, these processors will work on this data because there are a number of processors each can take care of smaller words of the data and then they will give the output back to the CPU system. This is a CPU GPU architecture. GPUs cannot work independently. So, if you ask the programmer to directly work on the GPU it is very difficult especially for the scientific computing community, they give the entire job to the CPU, the CPU distributes the job into small memory pieces and small and small instructions and asks the GPU to take care of that. It is both copying memory as well as copying the main instructions or sending the concurrent instructions to the GPU.

(Refer Slide Time: 09:21)

Parallel programming models

- **Shared memory programming model**
 - Where all processors can see whole of the memory that is available
 - On a multi-core system a single shared address space
- **Distributed Programming Model**
 - Where processor can see limited memory.
 - Assume that every CPU has access to its own memory space, and can alter its own local variables.

Shared Memory: Processor 1, 2, 3, 4 can see whole memory

Distributed Memory System: CPU can see only limited memory of their own

The slide contains two diagrams. The top diagram illustrates a shared memory model where four processors (1, 2, 3, 4) are connected to a central 'Shared Memory' block. The bottom diagram illustrates a distributed memory system where three separate CPU-Memory units are connected to a 'Network Cable'.

To be more specific there are basically 2 types of programming models in parallel programming ,which are shared memory programming model and distributed programming model and what we discussed about hybrid programming model is a combination of both shared and distributed memory. GPU also works in a shared memory model. We can see that the data space or address space which is copied to the GPU memory is visible to all the GPU codes.

So, in the shared memory programming model all the processors can see the whole of the memory that is available or the entire memory is a single shared address space and multiple processors are connected to that. So, each processor can see any part of the memory and all of our multi core systems (like if we buy a dual core laptop or quad core laptop) is basically a shared memory system.

In shared memory we can see that there is a common address space in which multiple processors are connected. So, whatever they are working on they are working on the common address space variable.

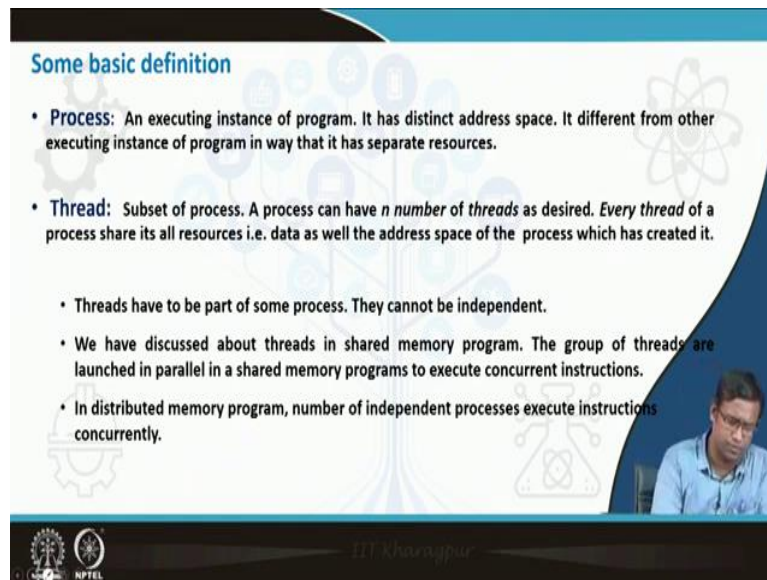
The distributed memory programming has a difference that the processors can see limited part of the memory .The entire memory space is not visible to the processors; each processor can see a limited part of the memory and it one CPU can only access that memory and alter it the local variables (it cannot see entire memory or cannot do anything any). So, whatever tasks are given to the processor that must work on the local data or the local ram of that particular processor it cannot go to a remote memory and work there.

If it needs to get anything from the remote memory it has to be done through the network cable through the data transfer or message passing. So, multiple CPUs with their own local memory are connected via network cable ,there is a distributed memory programming involved therefore, we can very easily understand that there will be differences in the programming approach also.

So, here we look into a same memory and we have seen that in OpenMP it is a very similar for loop which is operating over a data set and we just write OpenMP construct ,and that part of the iterations are given to processor as some of the threads and different threads can work in parallel if there is no dependency among them.

A distributed memory programming will be of course different, we have to identify this space of data and give it to the processor and this processor only sees this data. So, the whole problem has a global reference, but when a processor looks into the problem its viewpoint is very local it cannot look into the global problem. It can only look into its own local memory; none of the processor can look into the whole memory here.

(Refer Slide Time: 12:52)



Some basic definition

- **Process:** An executing instance of program. It has distinct address space. It is different from other executing instance of program in a way that it has separate resources.
- **Thread:** Subset of process. A process can have n number of threads as desired. Every thread of a process share its all resources i.e. data as well the address space of the process which has created it.
 - Threads have to be part of some process. They cannot be independent.
 - We have discussed about threads in shared memory program. The group of threads are launched in parallel in a shared memory programs to execute concurrent instructions.
 - In distributed memory program, number of independent processes execute instructions concurrently.

NPTEL

Process is an executing instance of a program. It has a distinct address space. It is different from other executing instances of the program in a way that it has separate resources. So, one particular instance which is being executed by a program which works on its own distinct address space is known as process.

We can also tell that in a distributed memory system each processor is executing a process; we usually do not mention processor here rather we take it as a process because there can be some hybrid architecture when one processor is executing multiple processes or some way there can be coprocessors there. So, the word processor is a little confusing here as even multiple processors can point to the same address space. But process is a very distinct term used in parallel computing, a process means an executing instance of the program which works on its own distinct address space, this executing instance does not share its address space with any other executing instance. There can be multiple processes running for the same program, but each of the processes has its own memory space.

Thread is a different concept we have looked into; threads that multiple instances are working on multiple instructions are being operated on; however, they point to the same address space. So, the entire group of threads which is pointing to that address space are part of the process. So, we can call thread is the subset of the process.

A process has its own address space; it is an executing instance of the program which has its own address space. A thread is something like an executing instance of the program which shares its address space with some of some other threads.

So, when on the same address multiple instructions are going, these are threads when different instructions are going only on the particular distinct address space. If there are multiple instructions there are multiple address spaces this instruction instance is processed.

So, thread is what we have looked into. Shared memory program threads have to be part of a process. There is a what whatever the instructions or instances going on are being executed over a particular address space that executing instance is a process. Inside a process there can be multiple threads, many sub instructions which are executed concurrently on the same address space. So, threads are subsets of process ;processes which has it is own address space and one executing instance the program going through part of the program which is working through the same address space is a process .Inside the process if there are multiple instructions, but they are using the same memory space they are threads. So, threads are part of the process, threads cannot be launched independently.

We have discussed threads in shared memory programs. So, in parallel we launch a group of threads, they execute concurrent instructions using the same shared memory.

But again, in distributed computing we will not talk about threads because we are not sharing the same memory space, we will talk about processes which have their own distinct memory space. In a distributed memory program a number of independent processes execute instructions concurrently. In the shared memory program a number of threads being executed connect to the same global address space. In distributed memory programs a number of independent processes are being executed and each of the processes by definition is connected to its own distinct address space.

(Refer Slide Time: 17:45)

Pro-s and Con-s of shared memory programming

Pros

- Ease of programming
- Lower latency
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Cons

- Limited Scalability between memory and CPUs
- High coherence overhead
- Adding more CPUs can geometrically increases traffic on the shared memory-CPU path
- Programmer is responsible for the synchronization constructs that ensure "correct" access of global memory.
- It becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

✓ OpenMP makes it relatively easy to work on this environment.

Dr. Khanna

The advantages of shared memory program is ease of programming .We have seen that in OpenMP programming it is actually very easy, you only have to know about basics of OpenMP program you have to identify that region where concurrently instructions can be executed and write some OpenMP construct . You do not need to do anything about the memory because they are writing to the same memory space (yes you need to take care of the memory synchronization).So, if you see that there is some possibility of cache conflicts or contention you have to write a proper OpenMP construct there like critical or atomic or reduction. However, you really do not need to think about that which thread will take care of which part of the iteration .Even if you take care of that OpenMP has very simple clauses for that or which part of the instruction it will be better to execute in which thread ; programmer does not really need to look for these things this is something which OpenMP takes care of that.

The load balancing ,OpenMP it itself can take care of that, at best you have certain control over it.

Lower latency because there is no requirement of sending data from one processor's memory to another processor's memory through a network cable because it is the same data space which all the processors are using.

Data sharing is fast because it is through something like a pci bus, processors are connected to the memory and with the very first channel they are writing the memory .It is also uniform because this is the same memory where all processors are sitting there. It is like a same table

which is being shared by multiple people, so, if I am eating something here everybody is dining on the same table. If I am eating something here this guy can directly see that this part is being eaten out .

The disadvantages are that there is limited scalability between memory and CPU. So, you really cannot bring many people to sit here, the space is constant, the bandwidth constant is there ,you really cannot increase the number of CPUs or the memory size.

One of the big disadvantages is cache coherence protocol, false sharing etcetera which will add certain overhead. If you increase more CPUs the traffic on the shared memory CPU path increases that makes the program slower. So, you cannot go up to thousands of CPUs in a shared memory system or you just cannot take your laptop which is a quad core system and think that I will put another CPU on that there are certain restrictions on that.

For any of the synchronization constructs the programmer is responsible for, it is not that the programming model or the algorithm itself takes care of the synchronization. You have to identify the part, it does not itself take care of the synchronization, you have to identify the location where synchronization is required. You as a programmer have to put a synchronization clause or a synchronization construct there, so that access to global memory is correct and without latency for different processors.

So, this is a programmer's job who needs to identify that this should be a critical operation in this part of the program. In terms of increasing the resources if we think of getting a large number of processors or large memory it is difficult if from the design point of view, from the manufacturing point of view ,from the heat dissipation point of view we really cannot keep on increasing processors there.

There is a space constant, heat dissipation constant, electricity constant and many other constant. So, you really cannot go to a very large computing system here , with a very high end parallel symmetric multiprocessor you are restricted by few hundreds of processors

In NUMA you can probably have a greater number of processors, but still some of these restrictions are there. So, you really cannot increase your resources by just adding processors or increasing the memory here.

If you have a very large job you cannot use a shared memory machine because you will not be able to find out the right hardware which will provide you the large number of processors which are required to solve this job. On the other hand, the biggest advantage is that OpenMP is a very simple library to learn and with 3 and half hour's discussion on OpenMP and I think any of you can now jump and take your code and make it OpenMP parallelized. OpenMP makes it is easy to work with the shared memory environment well.

(Refer Slide Time: 23:56)

The slide is titled "Distributed memory programming". It features a central diagram of a network switch connected to several computer icons, each representing a processor with its own memory. The text on the slide highlights two key advantages:

- Scalable seats (scalable resource) – Adding more CPU-s to the network does not pose challenge to hardware, software or space
- Less contention from private memory spaces- No two processor try to access same memory element, no cache conflict.

The slide also includes the NPTEL logo in the bottom left corner and a small inset image of a speaker in the bottom right corner.

In the distributed memory programming, there are multiple computers each having their own ram and local memory space in the ram and they are connected via network switch.

The big advantage is that this is scalable. You can add more CPUs and each CPU will come up with its own ram. So, there is no restriction by the CPU memory bandwidth or CPU memory channel width. You can increase as many as CPU and each CPU will have its own rack. So, only you need a network switch which has many ports and you can connect it via network switch. It is not restricted by the hardware software or either by space or heat transfer issues because these are basically connecting many computers and they may not be physically located at the same space because they are connected by network switch and you can even use the internet to connect them.

You can actually bring more computers each has their own ram and put it there that gives us the best advantage of using distributed memory programming that we can have a very large system we can have very large scale high resource with high large number of processors in

which we can solve large problems .We can solve problems which will require many compute hours because we can use high quite higher number of processors compared to the shared memory system ;this is the biggest advantage here.

Less contention from private memory spaces because none of the processors are trying to access the same memory element, there is no cache conflict, there is no requirement of using cache coherence protocol.

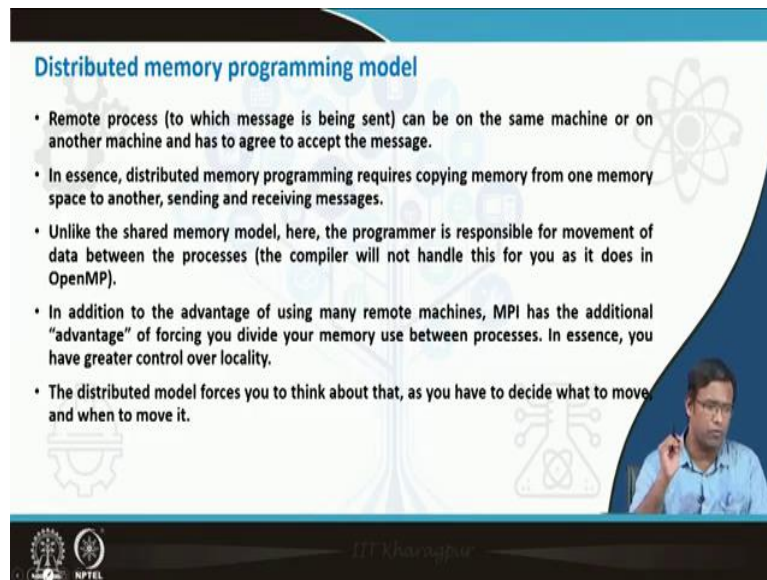
We will discuss this later that today's high-end computers (I am not talking about super computers just computing units with CPU ram cache triode) rely heavily on the cache-based architectures as well as the software or compilers they rely heavily on how CPUs will access cache and how memory will flow from cache to CPU and how memory will be prefetched from ram to cache.

Now, if you are bound by cache coherence then many times you cannot take this advantage, though the computer can perform very fast, but it gets significantly high overhead due to cache coherence and false sharing. If you are putting something like a critical construct to avoid false sharing ,many times you are not utilizing the best parallel performance of the system, but if you have a distributed memory system there is no same address space which will be shared by multiple computers. So, this cache conflict issues are not there as well as the bandwidth issues while multiple computers trying to write to the same memory location are also not there. So, this is another significant advantage here.

Also, another advantage we will see that as we have a large problem and we break it down to smaller memory problems, the small memory units are cheap (if you have to buy many 4 gig rams, these 4 gigs you will see that can save money because in terms of motherboard and other requirement this is less expensive).

So, from expenditure wise fact also if you can break the problem into small memory units in what you do in a distributed memory system then that is advantageous.

(Refer Slide Time: 28:10)



Distributed memory programming model

- Remote process (to which message is being sent) can be on the same machine or on another machine and has to agree to accept the message.
- In essence, distributed memory programming requires copying memory from one memory space to another, sending and receiving messages.
- Unlike the shared memory model, here, the programmer is responsible for movement of data between the processes (the compiler will not handle this for you as it does in OpenMP).
- In addition to the advantage of using many remote machines, MPI has the additional "advantage" of forcing you divide your memory use between processes. In essence, you have greater control over locality.
- The distributed model forces you to think about that, as you have to decide what to move and when to move it.

NPTEL

In the distributed memory programming model what is important is that they are not writing to the same address space; however, all the processors are working to solve the same problem therefore, there is some data which is shared across the processors which is common across the processors. How this data sharing will be done as they are not sharing those data spaces. There are 2 people who are not sharing their bank account (rarely people share bank accounts unless they are close relatives), but I need to send money to you. So, what will I do ? A bank transfer , an online transfer or I will give you cash etc.

So, as they are not sharing the same address space ,if they have to share some information, if they have to give some information to others, the processor will send this information to a remote processor. So, the remote process can be on the same machine (in the same machine there are multiple processors I can make separate address space for each of the processors) and run it as a distributed memory program or it can be a different remote machine. While writing a distributed memory program we have to be sure that these processors which are sending the data and who are receiving the data they both have to be on the same page that the processor which is sending the data should give the sending instruction, and the receiving processor should be able to accept the message. This is a very important step in the distributed memory programming model. They should be connected, the processors should be connected to each other, the processors that are being executed should be allowed to send data from one to another.

So, in essence, because one processor will send data, others will receive it. It is basically copying data from one's memory and it will go to another processor's memory. It is not like money that if I give it to you, I do not have that money, it is information. So, there is a difference between money and information. I know something I tell you I do not forget it. So, if one processor gives the information it still retains the information therefore, it is not giving it is not cut paste it is copy paste, it is copying the data from one processor and putting it into the receiving processor.

Unlike shared memory models, the programmer is responsible for movement of data between the processes. In shared memory we have seen that we didn't bother about sharing data between the processes. We didn't even bother about writing local data and shared data; if we write anything and define anything in shared data, it is shared. So, there is how each of the processes or how each of the threads will know about certain chains made by other threads in OpenMP it is a shared memory variable. So, if one thread changes, the other knows it. The programmer does not even worry about that programmer cannot even do something on that unless he only can define it as private. In distributed memory programming, a programmer has to ensure that data moves from one processor and is copied to another processor.

The movement of data at which instant from which processor to each processor, which data will be moved, it is the programmer's responsibility to specify it. In addition to the advantage of using many remote machines has another advantage that it forces to divide the memory between processes. So, instead of using a large piece of memory the main problem is it is broken down into small memory units and each process is attached to one of the memory units.

The help is that each process is now working on a localized part of data. So, the problem is very much local on which each process is working, what is the advantage is that each one is working on a small data set, so, the cache miss type of issues when almost entire matrix can be copied into the cache miss type of things or there the registers, they can use the issue of data localization. Data locality is very good for computational performance.

Now as each process now works on a small memory unit the processors are forced to work on the localized data. Moreover in distributed memory programming model which is good as well as a little difficult also, the programmer has to think on a distributed memory paradigm he shouldn't think about the global problem while writing the program. What matters to him is

the local part of the program which is being local part of the data which is being accessed and operated by each of the processor.

So, a programmer looks from a local perspective on the data set and he also has to decide which part of the data has to be moved and when to move that. So, programmer gets more responsibility and the programmers job is also little more difficult in distributed memory programming model because instead of the global view which generally comes to us when we think of a problem he now needs to look from a local perspective based on each processor and then he also needs to think about which data has to be moved and when to move.

(Refer Slide Time: 34:52)

The slide is titled "Essentials of distributed memory programming" and lists several key concepts in a list format. The concepts are: "How many people doing the work. (Degree of Parallelism)", "What is needed to begin the work. (Initialization)", "Who does what. (Work distribution)", "Access to work part. (Data/I/O access)", "Whether they need info from each other to finish their own job. (Communication)", "When are they all done. (Synchronization)", and "What needs to be done to collate the result." Below the list, the text "Message Passing Interface (MPI)" is displayed in red. The slide also features a small video inset of a man in a blue shirt on the right side. At the bottom, there are logos for NPTEL and IIT Bombay.

We can look into the essentials of the distributed memory programming here and after that we will go look into message passing which is the stem of distributed memory programming.

How many people are doing the work (degree of parallelism) and these are the aspects from the programmer's point of view one need to think about when writing a distributed memory program how many people are how many processors are working on that what is the degree of parallelism.

What is needed to begin the work, we need to identify which processors are working , initialize an environment in between them, so, that they can communicate in between each other and there are some other important issues also which are to be initialized.

Who does what(work distribution) ,OpenMP itself is doing load distribution? It is allocating a number of threads to a number of processors, but here who does load distribution? Programmer has to take care of load distribution and ask one particular processor to work on that part of the data.

Access to work part(Data/IO access) or how will the data access ,how will the io access have to be thought from a distributed memory perspective.

Whether the processors need any info from each other to finish their own job, what are the requirements of getting information or data from the other processor to finish it is their own job, what is the requirement of communication, when the communication will be done and which data will be communicated.

When they are all done(synchronization) there like in shared memory as multiple processors are working for the same problem there are requirements of synchronization also. So, when synchronization will be done.

What is required to collate the result. So, there are certain pointers which we need to take care of when developing a distributed memory programming, and fortunately we have an API which has been developed by number of people working in this area for very long time almost for 3 decades it is available to us called message passing interface, which can take care of all these parts .We need to know about the library functions on this API on message passing interface use the wrapper to call the library functions use a wrapper while compiling the program using MPI and these parts will be taken care of.

In the subsequent lecture we will see about message passing, what is message passing and what are the features of message passing and distributed memory programming. We will also try to compare a distributed memory program with a shared memory program and then we will look at the message passing interface in detail.