High Performance Computing for Scientists and Engineers

Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Module - 02
OpenMP Programming
Lecture -12
Introduction to OpenMP

We are discussing the topics in the course High Performance Computing for Scientists and Engineers. And today we will start the 2nd module of the course which is OpenMP Programming and this lecture and the subsequent lectures, we will discuss on Introduction to OpenMP. Before this, we had 2 weeks of lectures in our 1st module, which was fundamentals of parallel computing.

In the 1st module we have discussed the architectures of parallel computing systems, shared memory and distributed memory parallel computing systems. We also discussed appropriate algorithms for parallelizing a job for both of these systems and the performance metrics of the parallel systems.

So, this has given us a background to start working on actual parallel programming. There are two broadly different systems of parallel computing. One is a distributed memory system; another is a shared memory system.

In the distributed memory systems, there are number of computers which have their own local memory space and a large job is broken down into multiple smaller jobs; each will have their own local memory space and each of the computers will update the data and do the processing in that local memory space, and they will interact between within each other using message passing algorithm.

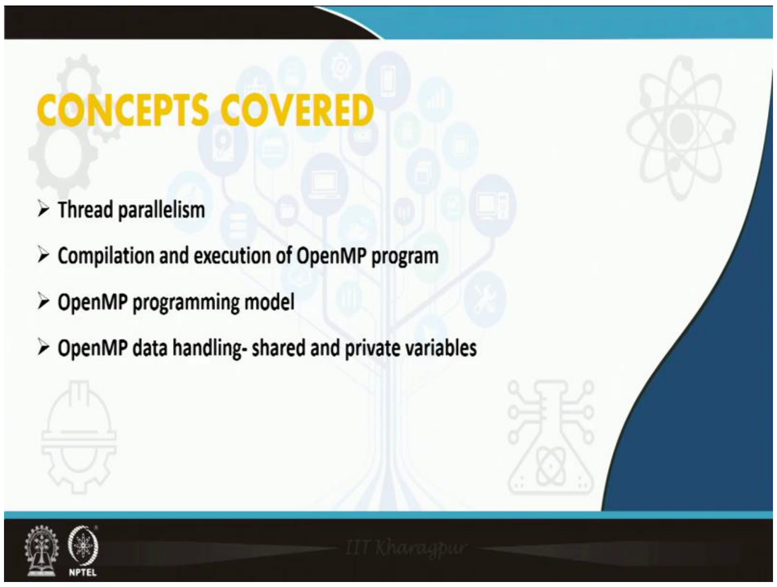
In the shared memory it is much simpler than distributed memory computing both program wise as well as architecture wise. We will also discuss it in a while, but there is a common global address space in which multiple processors are connected and they try to update data, at different locations at the same time instant or they work parallel to update data of these different locations of the same memory space.

Well, the third one we will discuss about GPUs or Graphics Processing Units at the end of our course. But right now, we will first focus on shared memory parallel computing and writing programs on that using an API called OpenMP. This is actually much simpler than CUDA based GPU programming or MPI based distributed memory programming. Also, most of us are probably having a computer or a PC or a workstation which has multiple processors.

So, when we go to buy computers, we go for dual core ,quad core processors. Therefore, a computer which we already have has multiple processing units or multiple CPUs connected with a common memory space. So, we can learn OpenMP very easily, and we can get our hands dirty in parallel programming, using OpenMP because the computers which we are having in our home or in our office which is readily available to us without doing any further hardware augmentation like using a GPU card or using a network cable, we can use it for doing OpenMP programming.

So, well so let us start with the 1st lecture of module 2 and this is the 12th lecture in our course which is Introduction to OpenMP.

(Refer Slide Time: 04:23)



CONCEPTS COVERED

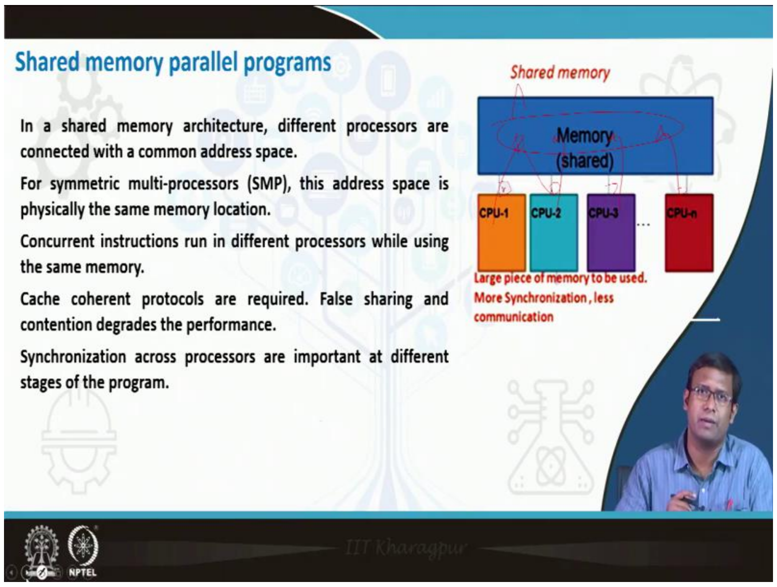
- Thread parallelism
- Compilation and execution of OpenMP program
- OpenMP programming model
- OpenMP data handling- shared and private variables

IIT Kharagpur
NPTEL

In this lecture and in subsequent lectures, we will discuss thread parallelism, compilation and execution of OpenMP program, OpenMP programming model, and OpenMP data handling using shared and private variables and different nuances of that.

When we talk about OpenMP programming we are trying to write a parallel program.

(Refer Slide Time: 05:04)



Shared memory parallel programs

In a shared memory architecture, different processors are connected with a common address space.

For symmetric multi-processors (SMP), this address space is physically the same memory location.

Concurrent instructions run in different processors while using the same memory.

Cache coherent protocols are required. False sharing and contention degrades the performance.

Synchronization across processors are important at different stages of the program.

Shared memory
Memory (shared)
CPU-1
CPU-2
CPU-3
CPU-n
Large piece of memory to be used.
More Synchronization, less communication
IIT Kharagpur
NPTEL

What we are trying to do here is that we go to a shared memory system where there is a common memory which is attached with multiple CPUs and this can be done in different ways. Each of the CPU will look into different memory locations of the memory, but it is the same memory space and each CPU is free to look into any of the memory locations.

But through the program we will see that each of the CPU will look into different locations of the memory and try to do their arithmetic and logical operations on these locations. Therefore, if there are multiple operations; they can be done actually in parallel instead of running a do loop over them. Precisely that is what we are trying to do using OpenMP.

As this course is about high-performance computing for scientists and engineers, most of the domain people who work in some areas of science or engineering or even some people from economics or business. We can consider this group also very much into using high performance computing for their own domain problems. These people many times have some legacy codes or many times they have some algorithms which they can very easily convert into a program and run their problem.

Now in case they use some technique like OpenMP, it helps them in the sense that they can use multiple processors of the computers they are accessing, day in and out and use full utility of that computer where there are multiple processors of all the processors.

Also, when we use some of the commercial software or some of the community developed software like Open FOAM , NAMD, Romax ,these software also have some optimization within them so, that if you build them correctly for a multi CPU system, they have their own OpenMP constructs in between them which you probably cannot see them directly. But it is inside the source code of the software which launches such you have been multiple CPUs. So, you get an actual parallel computation using that software.

So, as a user also, domain experts may sometimes need to know what is happening within the source code and or if they have to change something in the source code. In many cases these are open source codes, so if you find that there is some incredible way of getting better performance in the code or there is some new physics which you can explore using software source code is available to you, you will jump into the source code and try to update it.

Then it also becomes important for the person to know OpenMP because for many of this operation inside the program OpenMP constructs are called. For people from computer science backgrounds who are supposed to develop the source codes, these source codes are finally developed through computer science people. It is extremely essential if they are working in the domain of scientific computing that they have to work with parallel codes and specially OpenMP type of parallel codes in most of the cases.

So, it is very important that we learn OpenMP properly and this is also the first step in this course to learn how to write parallel programs or this will be the first step for many of us while developing their own parallel program. Once you know some constructs of OpenMP, you will feel the joy of asking multiple processors of a computer to work in parallel.

When you run a simple program or when you write a simple code and compile it and execute it, you see that only one processor is active on that. But when you learn OpenMP, you can see yourself asking multiple CPUs of the computer to run the job and crunch numbers for you and give output to you.

So, this is also important and as I said writing OpenMP programs, executing OpenMP programs does not require very high-end computing systems like a data center or something with a GPU card or a at least HPC cluster with multiple servers connected via MIDI NET, so it does not require that.

It is your simple desktop or laptop computers with multiple processors which you can use to do OpenMP programming. There can be another use when you are trying to do something, say post processing of some data you got through your scientific computing program, you can also use OpenMP to do it in a faster way. So, we are using multiple computers looking into different files and giving you data, but they are looking from the same global address space.

We will see that typically single instruction multiple data program and, in some cases, multiple instruction multiple data program ,you can do using OpenMP. So, let us go into the business.

We can see that if there are multiple CPUs connected with the common memory space, we can write an OpenMP type of program which is for shared memory parallel programs in that. What is good here is that there is a common data space, so all the computers are basically accessing say if you have a variable a if you have an array a, all the computers are accessing different locations of the array.

You really did not do not need to make different copies of a map into some separate local memory and then you have to track which part of the memory should go and then combine them, like a distributed memory system.

So, there is less communication across the CPUs because everybody knows that they are connected to a common memory space and they are updating the data assigned by them. Who has assigned it? The programmer has assigned that data which CPU will do and that is precisely what will do through OpenMP.

Another thing is important here because it is the same variable a, it is the same array a whose different elements are being accessed by different CPUs; we need a lot of synchronization in between them.

So, OpenMP or I will say shared memory programming requires more synchronization, but less communication and also programming wise it is probably much simpler than distributed memory programs. So, in a shared memory architecture, different processors are connected with a common address space.

For a symmetric multiprocessor, there is a physically single or connected chunk of memory and all the CPUs are connected with that chunk of memory. We also call it uniform memory access or UMA. This architecture we have discussed in our first few classes in our first week's class.

So, it can be a symmetric multiprocessor where they are using physically the same memory location or it also can be multi-processor arrangements where non uniform memory access is given; that means, different computers are connected with different pieces of memory. However, these memories are somewhere mapped to a global address space.

Now, when you connect the same memory with different processors, it might be a symmetric multiprocessor that is physically the same memory that is connected with different processors. It can be a non uniform memory access multiprocessor where different computers with independent memories are connected via an interconnect switch and all these independent memories map to a global address space.

In this case, concurrent instructions can run in different processors while using the same memory. As I have shown here that there is a same variable a and different parts of this variable, different elements of this variable are being updated by different processors; at the same instance they can go ,so, therefore, there are concurrent instructions which can be processed by different processors in parallel.

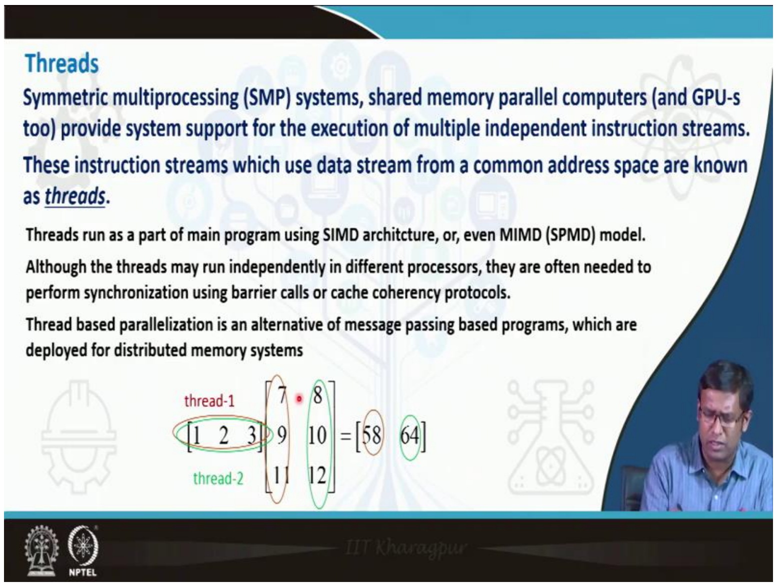
Cache coherent protocols are required. This is very important; we have discussed cache coherency in detail earlier that if some part of a is updated by CPU 1 and CPU 1 has some cache and this variable also is called by CPU 2. So, CPU if the update which has been made on this variable by CPU 1 must be reflected in the cache of CPU 2.

We have discussed cache coherency earlier ; because it is a shared common global address space whenever one processor updates something there, this must be reflected in all other processors' cache.

And that there are protocols for doing cache coherency which can lead to false sharing which can lead to some of some sequential operation across the processors etcetera and it can degrade the performance. So, because cache coherency has to be established, we have to be careful while dealing with the data in the shared memory system.

Whatever I am discussing here is not specific for OpenMP, but is general for all shared memory programming paradigms. Synchronizations across processors are important at different stages of the program because it is the same memory which all the processors are trying to update and it must be synchronized, otherwise some part of the memory may not be updated and the processor starts working on something else and there can be something wrong because the variable is or the array is not completely updated.

(Refer Slide Time: 16:32)



Threads

Symmetric multiprocessing (SMP) systems, shared memory parallel computers (and GPU-s too) provide system support for the execution of multiple independent instruction streams. These instruction streams which use data stream from a common address space are known as *threads*.

Threads run as a part of main program using SIMD architcture, or, even MIMD (SPMD) model.

Although the threads may run independently in different processors, they are often needed to perform synchronization using barrier calls or cache coherency protocols.

Thread based parallelization is an alternative of message passing based programs, which are deployed for distributed memory systems

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}7 & 8\\ 9 & 10\\ 11 & 12\end{bmatrix} = \begin{bmatrix}58 & 64\end{bmatrix}$$

In symmetric multiprocessing systems, shared memory parallel computers which are non uniform NUMA, symmetric multiprocessing is uniform memory access UMA and shared memory parallel computers are non uniform memory access or NUMA systems (and GPUs also),they provide system support for execution of multiple independent instruction streams. So, they are connected to the same memory space either physically or virtually, and the memories are mapped to a common address space.

Now, there also is a requirement of the system level support by the operating system, support by the other hardware, support by the scheduler of the processors which is inbuilt within the computer in the motherboard.

This system level support is important so, that all these computers can in parallel execute independent instruction streams or concurrent operation of the processors on the same data space is possible that is done through system level support and symmetric multiprocessors and shared memory parallel computers as well as GPU cards, they provide this system support.

So, it is not only just the hardware, but using the software, the operating system, the scheduler inside the motherboard everything needs to give support for execution of parallel programs. Now each processor is executing instruction streams on its own and there are multiple processors and each of these executing independent instruction streams.

These instruction streams which use data streams from common address space are known as threads. So, the instruction streams are being operated by the processors, but they are taking data from a common address space, these instruction streams are known as threads and threads are the units of parallelism here.

So, when we will write a parallel program in a shared memory system, we will mostly use this thread parallelism paradigm and we will launch threads for concurrent operations at any instant.

Threads run as part of the main program using single instruction multiple data architecture. So, essentially the same instruction operates over different parts of the same data set, say SIMD ,that is how the threads will work. But also, multiple instruction multiple data; that means, instruction streams are only concurrent and independent, but they are different instruction streams depending on the data they are working on; that is, it is also possible to use the MIMD model in thread parallelism.

Although the threads may run independently in different processors, they are often needed to perform synchronizations . I mean if say if there is one variable all the processors are trying to update, they cannot do it ,there is some synchronization required for them and cache coherency is also required ,if one is updating it and another is trying to read it requires cache coherency.

So, because all the threads though they are running independently, they are pointing to the same memory space. It is important to have synchronization in between them, it is also important to have cache coherency. So, we can think of something like this: there is a trained compartment where there are 8 berths and there are 8 passengers who will go to each of their independent berths.

So, this process can be done very well in parallel with whoever is boarding the train. You know that it is not that one has to wait for the other to take his own berth. However, some synchronization is needed otherwise there will be contention, there will be racing between them; somebody will probably by obstruct others to go into his own place. It is the same area

;it’s the same memory location whose different elements are being accessed, it requires some synchronization.

Also, it is possible that some memory location will be accessed by multiple threads and therefore, some sort of sequentiality will be required there. So, these are different types of synchronizations and also cache coherency protocols are required. We have discussed cache coherency in much detail. We will also discuss some of the aspects later.

Thread based parallelization is an alternative to message passing based programs. In message passing what happens is that it is typically a distributed memory system programming that each processor is connected with its own local memory space.

So, when one processor tries to update something it updates only in its local memory space and at particular instances when the programmer thinks that these updates are to be informed to some other processor, it passes a message or passes some data through network cable to that other processor. The programmer has to control it when the data passing will be done, who will pass data to whom.

But in thread parallelism programmer really cannot control; he has some control, but the entire think he cannot control it because it goes on the hardware and operating system of the system that is sharing the same memory and different processors are writing to and reading from the same memory.

So, it is less on the program and more on the system that how this memory will be accessed by different processors. Well we can see a small example here row matrix is multiplied by another matrix and we are getting product as a row matrix. So, how is this multiplication coming that this row is being multiplied with this row.

So, each element two elements or a row of this element is multiplied by the first row of this column; the first, second column of this row is multiplied by the second row of this column; the second, third column of the row is being multiplied by the third row of the column. And this is summed up and 1 *7 + 2 * 9 +3*11= 58. This comes as 58.

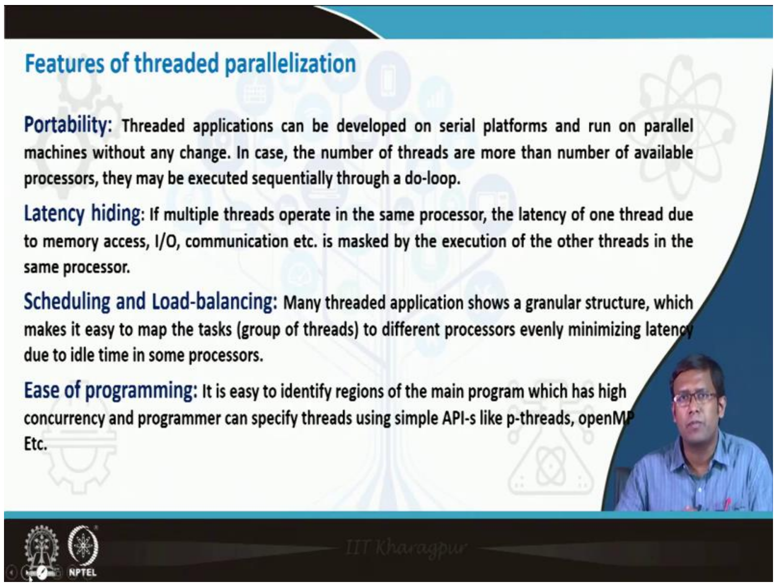
The second one is this is multiplied with this is multiplied with 8, this is multiplied with 2 and this is multiplied with 3 and the product comes here. So, I will show there are two columns who can be independently multiplied with the row and their result will come here.

$$\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\end{matrix} \quad \begin{matrix}7 & 8\\ 9 & 10\\ 11 & 12\end{matrix} = \begin{matrix}58 & 64\end{matrix}$$

So, we can launch two threads. First thread will multiply this row with the first column and this will be the output and a second thread will do for the second row, second column. So, these two elements can be obtained in parallel if we run two threads in two different processors.

So, this is an example of a thread parallelism in that these two are obtained basically operating over two columns of one of the input data. And these operations can be done independently.

(Refer Slide Time: 24:55)



Features of threaded parallelization

Portability: Threaded applications can be developed on serial platforms and run on parallel machines without any change. In case, the number of threads are more than number of available processors, they may be executed sequentially through a do-loop.

Latency hiding: If multiple threads operate in the same processor, the latency of one thread due to memory access, I/O, communication etc. is masked by the execution of the other threads in the same processor.

Scheduling and Load-balancing: Many threaded application shows a granular structure, which makes it easy to map the tasks (group of threads) to different processors evenly minimizing latency due to idle time in some processors.

Ease of programming: It is easy to identify regions of the main program which has high concurrency and programmer can specify threads using simple API-s like p-threads, openMP Etc.

So, we will see some of the features of the threaded parallelism. One is portability ;threaded applications can be developed in serial platforms and run on parallel machines without any change. In case the number of threads is more than the number of available processors, they may be executed sequentially through a do loop. So, we write an OpenMP program with the threads as discussed.

Now, when we will try to run it in a single processor architecture, we have a computer which has only one processor, but we launched ten threads. So, these ten threads are ten independent instruction sets. They actually will be operated simply as a do loop; 1 to 10 each of the threads will be operated.

So, if we have a computer system in which the number of processors is less than the number of threads, what is the number of threads; that we can control through several ways? We can within the program also run that part, there will be this number of threads.

Say earlier we saw two threads for the matrix ,if there is a matrix with 8 columns, there will be 8 threads, so now we try to run it on a single processor computer.

These 8 threads will be sequentially run using a do loop and the advantage is that when writing a threaded program, you really do not need to bother about how many processors are available there. You can write the threaded program the way you want to parallelize that particular part of the program.

Parallelize in the sense that the way you want to get the number of concurrent instructions for that particular instance of the program. Once you run it into a CPU which has a smaller number of processors, a group of threads goes to each CPU and these groups of threads run as a do loop in the CPU, but all the CPUs execute these each of their groups in parallel.

So, there is some parallelism in between, if there are 4 CPUs and you launch 8 threads each CPU will run 2 threads. But one instant if 4 of the CPUs are launched have launched 4 threads, so you are getting four-time speed. In some sense its speed will not be exactly 4 x times because of overheads, but you are getting that here.

So, it is a portable system of parallelization that you can take to any computer system and run it. If these processors are less than the number of threads specified there, there will be some sequentiality then it will run.

The next is latency hiding. In case multiple threads are operational along the same processor, then each thread has introduced some latency. There is some part due to say memory access IO, communicating, establishing, cache, coherency etcetera; so, each thread has introduced some latency. But while this thread is taking time for this latency, say this thread is trying to write something to the file system, there is some latency there.

At the same instances a next thread in the group of the threads given to the CPU processor will be up launched. So, the latency of one thread will be hidden by operation of the next thread.

So, in some sense this latency is if we have threads much more than the number of processors, some part of the latency will be hidden because when one thread will go to the latency state or take its overhead time, the other thread in the same group will be launched at the time on that processor.

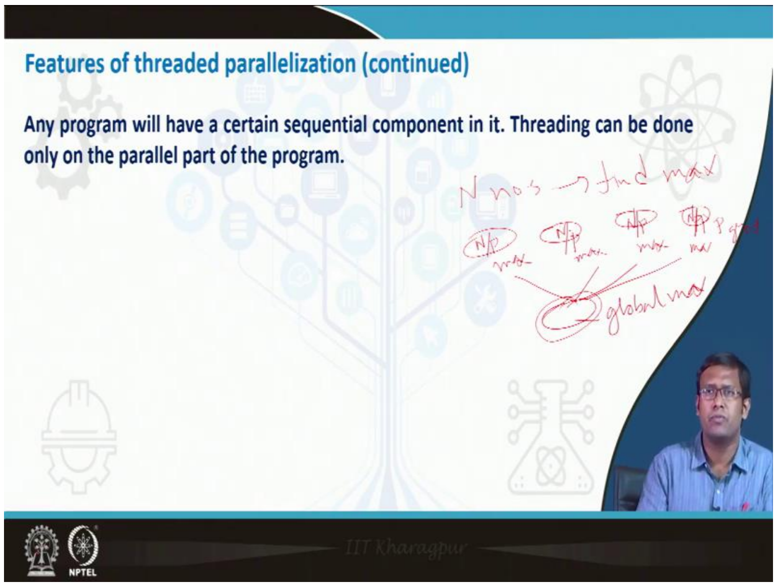
Scheduling and load balancing because we have developed the granularity, we have found a number of concurrent instruction streams; a finite number of concurrent instruction streams which should be running at that instance of the program. It is very easy to distribute it among a finite number of computers.

So, load balancing becomes easy because its finite number of threads to be distributed across finite number of processors and scheduling also becomes easy. We will see out some of the schedulers which OpenMP provides later.

Ease of programming, it is easy to identify the region of which the main program has high concurrency. It is something like a do loop where each of the operations are independent in the do loop; the next operation does not have to wait for the previous operations results. So, they are independent instruction streams instead of do loop you , you will launch threads there. Therefore, programmers can identify the region where it can be parallelized, there are high concurrency and specify threads using simple APIs like p threads, OpenMP etcetera .

Well so, I have given you some backgrounds on shared memory parallel programming and we came to the idea of thread parallelization.

(Refer Slide Time: 30:29)



Features of threaded parallelization (continued)

Any program will have a certain sequential component in it. Threading can be done only on the parallel part of the program.

We will see little more details on these and then we will conclude this session. If we look at the other, there is another advantage of thread parallelism. If you look into a certain program, there are some sequential components in it. Say we can consider this case say, I have a set of n numbers and find; you have to find out the maximum of n numbers.

So, I make some groups say P groups each have N /P numbers; P groups and ask N/P threads to find out the max for each of this and then all these local maximums are collected somewhere and I find out the global max.

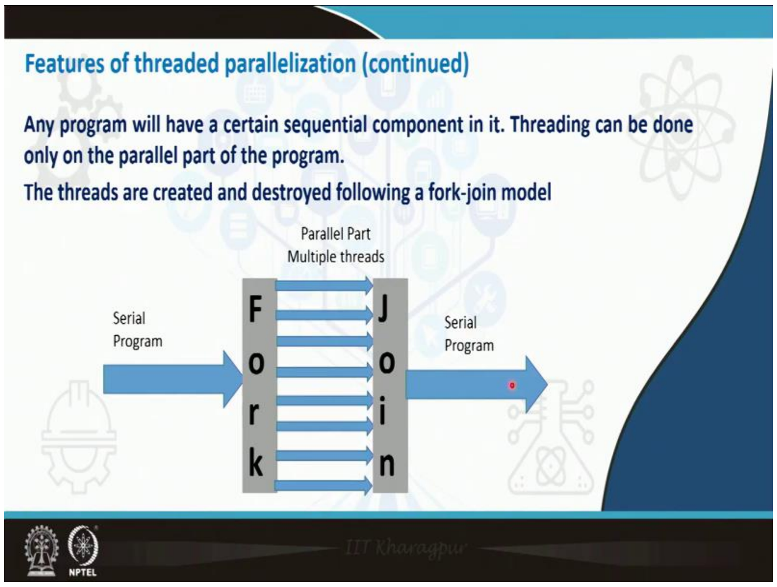
So, instead of doing this over N numbers, each thread is operating over N/ P numbers and if each thread is going to an independent processor, it is operating fast using parallelism.

Now this can be done in parallel in four processors, but this part is a sequential part because output from all the processors will come and only one processor will learn it. So, we cannot launch four threads here; if we have launched P threads, here we cannot launch P threads at this location; this part is inherently sequential.

And we have seen that any program has a certain sequential component in it. We remember Amdahl’s law when he found out what is the highest possible efficiency of a parallel program. We have seen that there is a sequential part which will be a limiting part on the efficiency of the algorithm.

So, we have to be able to control the number of threads. There will be a large number of threads. When we will go to the sequential part, most of the threads will be destroyed only; one thread will be active and will take care of the sequential part.

(Refer Slide Time: 32:46)



Features of threaded parallelization (continued)

Any program will have a certain sequential component in it. Threading can be done only on the parallel part of the program.

The threads are created and destroyed following a fork-join model

Parallel Part
Multiple threads
Serial Program
Fork
Join
Serial Program
IIT Kharagpur
NPTEL

Again, when another part of a parallelism will come, these threads will be active. This is done through creation and destruction of the threads using a fork join model. We can see that if we have a parallel program which the main program is running in the x direction.

In the serial part only one thread is active ;when the parallel part comes the thread, parallelism forks it into multiple threads and different threads take care of different parts or different groups of the main program. So, different instruction streams are launched.

When we have to go to the serial part again, all these threads join and one thread remains active and we go to the serial part. So, this is another very important feature of thread parallelism and we should keep in mind that it's not that all threads are active throughout the parallel program. At different instance, we can launch number of threads which will be active and in the sequential region only one thread is active

So, I will conclude the introduction to Open MPs first lecture here and we will in the next few lectures, we will look at the programming model of an OpenMP and how to compile and run OpenMP jobs.

And very importantly, we will also look at how different types of data handling is done in OpenMP ;how to access shared memory variables in OpenMP and in case some of the variables are very locally updated by the threads. What should we do for that?