High Performance Computing for Scientists and Engineers

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

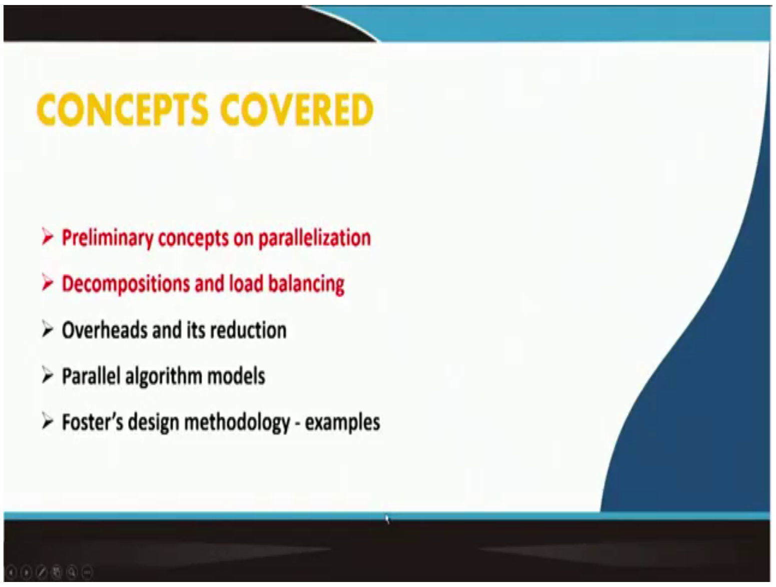
Module - 01
Fundamental of Parallel Computing
Lecture – 09
Parallel Algorithms (continued)

Welcome. We are going through our course on High Performance Computing for Scientists and Engineers. This is the 1st module, which we are discussing now fundamentals of Parallel Computing. The idea of this module is really to introduce some very essential concepts on parallel computing, which in some case are not quite well taught to the students and researchers who are coming from the background of non-computer science.

So, we will, we are trying to introduce you with some of these concepts which are essential in parallel computing. After this module will start working on high performance programming using OpenMP, MPI and CUDA.While doing so, these concepts will be vital.

So, right now we are discussing on parallel algorithms and this is the third lecture on parallel algorithms. We are basically continuing from what we have learnt earlier on parallel computing in last two lectures.

(Refer Slide Time: 01:36)



CONCEPTS COVERED

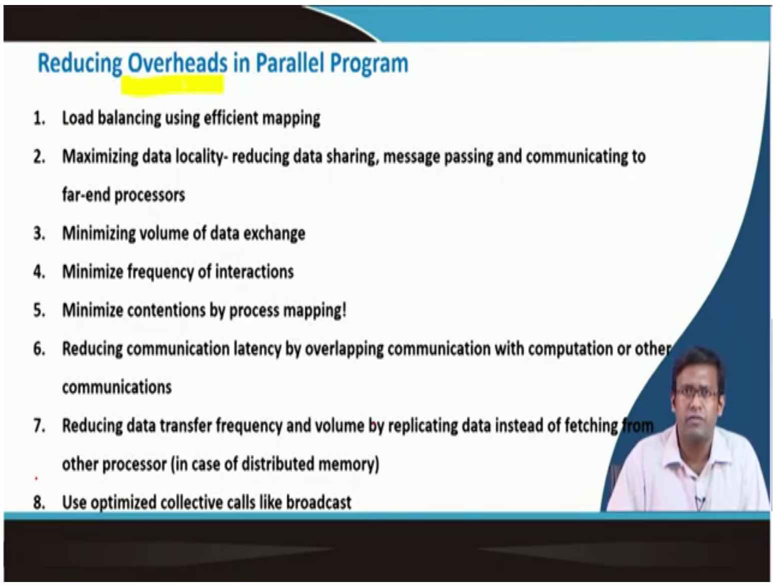
- Preliminary concepts on parallelization
- Decompositions and load balancing
- Overheads and its reduction
- Parallel algorithm models
- Foster's design methodology - examples

So, we have covered some preliminary concepts on parallel creation, including task dependency, decomposing a job into multiple tasks, and how these tasks interact. Depending on the interaction and dependency on the tasks how to determine a critical path and get a parallel algorithm out of that. We have also discussed about concepts like concurrency and granularity that how many tasks our job can be broken down into and how many of these tasks can work concurrently, so that we can ask a parallel computing architecture to take care of different tasks at the same time while they are trying to solve the same job.

And then this these decompositions have certain techniques. We have discussed about the decomposition techniques in which we primarily focused on recursive decomposition and data decomposition. These two decomposition techniques are important in terms of scientific computing. Especially, focused a lot on data decomposition; that means, if you have a large pool of data on which similar type of multiple tasks will be done. A part of the data will be taken by one task, another part of the data will be taken by one task. There can be in some cases overlapping in the data also in which these tasks are working, however, the tasks are independent. So, how to decompose the job into different tasks considering the input output or intermediate data that we have discussed earlier.

Then we are discussing about overheads in parallel programming in a specially reduction of overheads in parallel programming. So, quickly to ensure a continuity in the discussion will quickly revise the present topic which is overheads and its reduction in a parallel program. So, let us go to the next slide.

(Refer Slide Time: 03:38)



Reducing Overheads in Parallel Program

1. Load balancing using efficient mapping
2. Maximizing data locality- reducing data sharing, message passing and communicating to far-end processors
3. Minimizing volume of data exchange
4. Minimize frequency of interactions
5. Minimize contentions by process mapping!
6. Reducing communication latency by overlapping communication with computation or other communications
7. Reducing data transfer frequency and volume by replicating data instead of fetching from other processor (in case of distributed memory)
8. Use optimized collective calls like broadcast

We have discussed what are the overheads in a parallel program. The overheads are basically the add-ons over the main task or the seek main job or the sequential program which is required to ensure that this sequential program is not running in a single computer, rather this has been paralyzed and running in multiple computers.

And, we can understand that we have a single processor job and we are trying to divide it into multiple processor jobs which will run in parallel. The main aspect while doing so will be that the jobs will be distributed among different computers.

And, this process that getting the job dividing it into multiple tasks and asking different computers to take care of different tasks this itself will take some of the computational effort. Once this is done every time, we cannot ensure that all the processors are having equal number of tasks.

So, some processors will have a smaller number of tasks and they will be finished early and some processors will take more time because they have a greater number of tasks. So, there will be in some cases uneven distribution or improper load balancing of tasks among different computers because we have a large chunk of the job, we have divided into multiple small chunks and asked all the computers to work in parallel.

While doing so if somebody gets more, somebody gets less, the person who got more will take more time. So, others will sit idle in that part of time. This idleness will add to some computational overhead because we are utilizing these computers; however, we are not getting any work out of them. So, this is the load balancing process itself; that means, dividing the main job into multiple tasks and assigning it to different computers. This will take some time. This will add some overhead also. The improper load balancing which is inevitable in many cases in case of complex problems, will also add to some of the overhead.

And, there is another part of substantial overhead which is communication overhead because these computers need to communicate in between each other, or if we are using a shared memory machine the computers need to write the data at one shared memory. And there can be cache a coherency issues, race conditions, false sharing.

On the other hand, if you are using distributed computing system then one computer needs to pass some information to another computer which will go through a network cable and will have its own communication time. So, the time required for communication in between computer, be it distributed, computing be it shared computing will also act to overhead. So, these are the main two sources of overhead while executing a parallel program.

One of the important aspects when designing a parallel algorithm will be that how to how we can reduce the overhead. So, the important points on reducing the overhead is first that we need to ensure that there is a proper load balancing across the computers. The tasks are grouped or broken down, I mean the job is broken down into groups, so which has equal number of tasks. And this task goes to goes to computers, so that each of the computers gets almost same share of that task that will reduce the latency or idle time up for some of the computers and there will be some overhead reduction.

Maximizing data locality. So, data locality means if one particular computer is using some data and whatever data it requires that must reside in the same zone. In case, it is using a distributed memory it should avoid as much as possible getting data from other computers memory or it should reduce the data sharing.

In case, it is using a shared memory machine the data used by one particular processor must be on same contiguous piece of memory, so that it can avoid false sharing or cache a coherency type of issues, also reduce the message passing. It should communicate to the nearer processors in case of a distributed memory while communicating with a far-end processor will have its own overhead due to the hop hopping time and other communication overhead. This, we have discussed these things earlier while discussing about architectures.

Minimizing the volume of a data exchange, that is important specially in case of distributed memory computers. As much amount of data one computer is sending to other the data transfer time is more if the volume of data is more. So, minimizing the data volume.

Minimizing the frequency of interaction. So, again in terms of in cases of data transfer every data transfer call or every instance of data transferred requires a start-up time, that this processor needs to transfer data to another processor, both of them should first communicate and set the protocols and then the data will go. So, there is always a start of overhead in one data transfer call.

So, as many data transfer calls will be there the start-up time will be cumulatively increased. So, if we can reduce the frequency of interaction; that means, if you have to send a large piece of data instead of breaking it into smaller pieces of data and sending it. At many goals if I can send take the whole data and try to send it at one go that will probably reduce the overhead.

Minimize the contention by process mapping. That means, we should map the tasks rightly in the processors, the process should be mapped correctly so that contention between different process on writing or accessing same part of the data or same part of the file can be avoided.

Reducing communication latency by overlapping communication with computation or other communication. That is understandable, that communication takes substantial amount of time. If at the same instance when one computer is trying to communicate some data to the shared memory space or to another computer, if some amount of data which is not being affected by this communication can be considered by the computer and computation on this data go on. So, that means, that communication time will be overlapped by some of the computation time even that will also reduce overhead.

However, there is a caveat here. That, if you are trying to communicate some data, however, at the same instance there is a non-blocking communication. So, the computer keeps on computing. If the computation takes some data which was supposed to be communicated by other computer, but it has not been done, so it will work on the older data because many of the time when we are using scientific computing algorithms, we are using iterative methods.

So, the data especially the solution data is frequently updated through the iterations. In case, it has not been exchanged properly or the communication has not been completed, but we are trying to use the data by overlapping computation with communication, we might end up in using garbage data or long data. So, there has to be a caveat or there has to be a proper synchronization on that.

This is also another important parameter that reducing the data transfer frequency and volume by replicating data instead of fetching from other processor. Sometime we say we have done a data decomposition a part of data is assigned to one particular computer, but it needs some data which is originally assigned to another processor. However, the communication of data may be costly.

So, instead of communicating that data if this computer has some information about the nature of the data which it is supposed to get from other computer, but it itself does the computation, instead of asking the other computer to compute it and send it to this particular computer. So, instead of fetching the data from other processor, if it replicates the data in it itself that can reduce the overhead in terms of communication overhead.

However, we can understand that this thing will add replicating the data transfer reducing the data transfer frequency and volume by replicating data, replicating the data itself will take some amount of computational effort. So, we are saving in terms of communication, but the computer will do some extra computation. But usually and we have seen it earlier also communication is costlier than computation, so that might help in reducing the overhead.

And lastly, using optimized collective calls like broadcast. So, when some data is sent from one processor and it will go to all the processors instead of using something like send receive protocols if you use collective calls like it will broadcast to other computers, This collective calls has some advantage usually over the computer hardware; that means, when the parallel computing hardware and the compilers are developed they are already optimized for this collective calls.

So, processor 0 has some data. It tries to send it to processor 1, then to processor 2, then the processor 3, so on. Instead of doing that if it uses this collective call that means, it is at a go broadcasted to other processors that sometimes still use overhead because these calls are optimized by the hardware and the compiler.

So, these are these are some issues, some pointers on reducing overheads. In parallel program you have discussed this earlier also, but I thought like it is important to again discuss it because this is something on which efficiency of the parallel program depends.

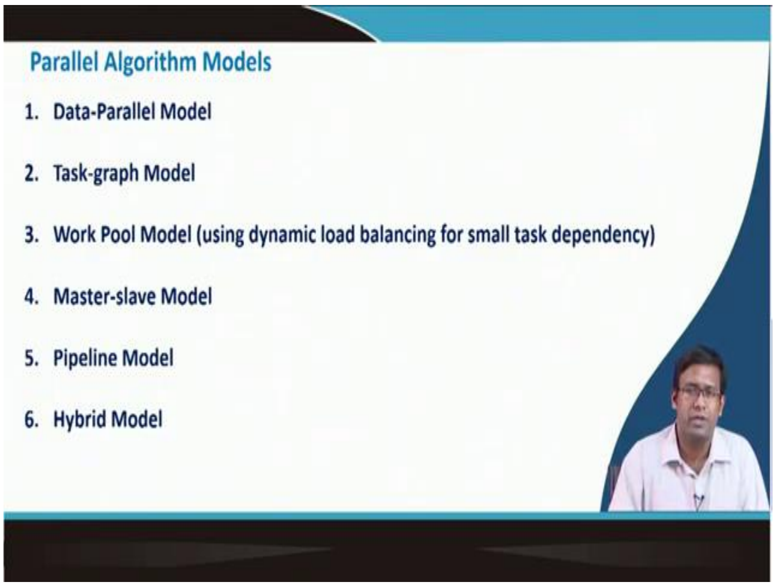
So, in the next discussion after this class we will discuss about performance matrix of parallel processors. And then, we will see that this particular term overhead, this particular term overhead is a very vital parameter while quantifying efficiency of a parallel processor.

If we have a parallel program and we are keeping on increasing the number of processors will see that at certain point the efficiency is not as good as expected. It might not be at all beneficial to increase number of processors because this overhead which includes many of these things like communication across the processors, proper load balancing etcetera. These overheads also increase and make the overall parallel program inefficient when we are increasing the number of processors.

Therefore, understanding what is overhead and understanding the pointers by the parameters by which we can reduce overhead of a parallel program is extremely important. For any practical application we try to paralyze any algorithm and get the benefit of a large parallel computing infrastructure. We will see the main focus becomes that how we can reduce certain overheads and make it efficient. So, that as we are increasing number of processors and we are trying to solve large problems we are getting right performance.

So, overhead is extremely important. It is very easy to make a job parallel. But it is also extremely difficult to reduce the overhead and get the optimized optimal performance. And this is something which you in which we will focus in our subsequent discussions also specially when we learn about parallel programming.

(Refer Slide Time: 16:43)


Parallel Algorithm Models

1. Data-Parallel Model
2. Task-graph Model
3. Work Pool Model (using dynamic load balancing for small task dependency)
4. Master-slave Model
5. Pipeline Model
6. Hybrid Model

So, we will be discussed about models of parallel algorithm, means that given a sequential program how parallel algorithms can be developed out of it. So, first thing we can see that what is data parallel model. This means, you consider the entire data set on which the serial algorithm or the original algorithm is supposed to work, and that must be a for loop or do loop which takes care of the data from the entire data space address. From the first point on the data space to the last element on the data space, you paralyze it. That means, you take part of the data and give it to different computers.

So, we have done data decomposition. Just previous lecture we have discussed about data decomposition. So, it is basically using the data decomposition technique and ask different processes to work on different parts of the data in parallel. Only thing you have to see that there is no data dependency across processes, some processes data is not readily being required by another process therefore, and in that case it might not at all work in parallel or if even if it is working in parallel that there can be contentious and false sharing and cache a coherency issues which will add to overhead. So, this has to be looked into that this though when we are talking about a data parallel model sufficient granularity is existing between different tasks.

The next one is a task graph model. We have seen when you are solving the identification of their schools and their cell fees structure problem in the last lecture, that you get a task graph generator task dependency graph and see that how many tasks can be operated concurrently at one particular instant. And based on that you give some tasks to some of the computers.

So, there are task dependency; that means, some of tasks are dependent on previous tasks which is not usually seen in data parallel model because it is a same data in which different tasks can independently work. But now there are some task dependency and therefore, looking into the task dependency model you need to find out the critical path and get up a parallelized model like that. So, this is probably little more involved experience. It requires little more involved effort to get a parallel algorithm than a data parallel model.

Now, next one is a work pool model. So, in case you have a small amount of task dependency across different tasks. So, you get what is the total amount of work and group different tasks together and use a dynamic load balancing to get part of task done by one processor, another part of task will be done by another processor like that. Make pools of tasks and assign it to different processors and this can be done using dynamic load balancing, in case task dependency is smaller or almost no task dependency.

In Master slave model, one processor is called the master. The master knows that it has many tasks to do like it has to ran a for loop and it asks different slaves or different processors under its own supervision to take care of some of task given to it. So, there is a master task which is forked or broken down into many slave tasks and different processes takes care of different task. And this type of master slave model will see in some cases of OpenMP programming.

Pipeline model, this is interesting. So, if we think for auto industry, the cars are produced using a pipeline model. That means, there is one part where probably the gearboxes are assembled. There is another part where probably these gearboxes are fitted into the engine. There is another part where the motor is fitted into the engine. There is another part where the batteries are putting to the engine box. So, what is happening in the assembly line at one point, some of that one particular task is been done and many cars car engines are coming and one by one every or car engines are being processed.

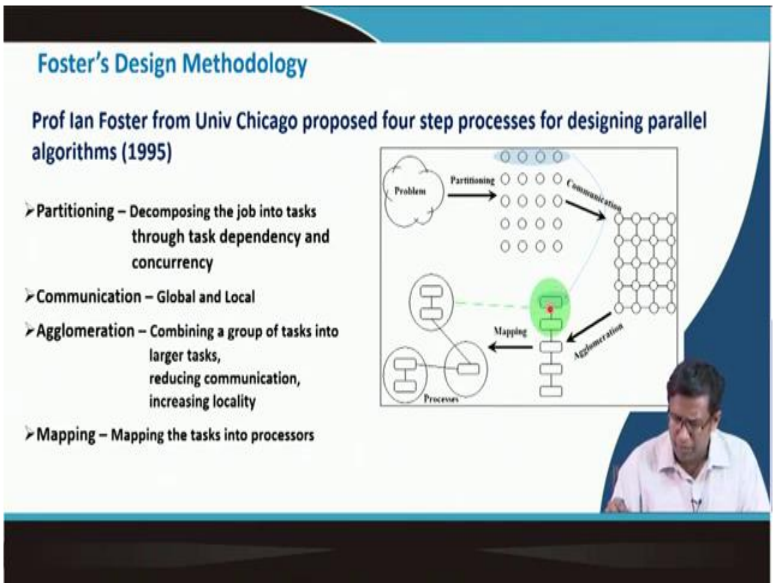
So, when one the gearbox assembly unit is working on the gearbox then at the same time the previous cars, battery assembly unit is working on the battery. So, there are a number of cars going through the same pipeline, at one place some job is being done and another place some job is being done.

So, different processes are being executed on different cars, but one process will go to, but one particular car will go through one particular all these processes. So, it is like a batch job where a different instance, different part of this batch job is being worked by worked on by different processors. And this can be very very efficiently utilized pipeline model in some of the computers, some of the problems which I say it works in a batch job.

And any combination of them can give you a hybrid model. Data parallel model combined with a master-slave model which again will see in terms of in cases of traded parallelism using OpenMP or CUDA or pipeline model combination with task graph model. This, all these things can be combined as and a hybrid models can be evolved.

So, when whenever there is a possibility of developing a hybrid model then there are there is possibility of being innovative in terms of designing the algorithms. So, you can combine some of these ideas and make your own model.

(Refer Slide Time: 23:01)



Foster's Design Methodology

Prof Ian Foster from Univ Chicago proposed four step processes for designing parallel algorithms (1995)

- **Partitioning** – Decomposing the job into tasks through task dependency and concurrency
- **Communication** – Global and Local
- **Agglomeration** – Combining a group of tasks into larger tasks, reducing communication, increasing locality
- **Mapping** – Mapping the tasks into processors

Now, in order to formalize parallel algorithms Professor Ian Foster from University Chicago, in 95 proposed 4 steps, in which given a problem you can get a parallel algorithm of this problem. Trying to address parallelization of a problem, some way or not you have to go through these 4 steps. So, it is important that we know for Foster's design methodology and when designing our own parallel algorithm will see that how does it compare with the formally formal method methodology prescribed by Professor Foster.

So, it is said that first part is partitioning, you have a job, it has to be decomposed into several tasks through task dependency and concurrency and to find out how many tasks can work concurrently at one part of time. And therefore, the main job will be decomposed into multiple concurrent tasks, group of tasks rather or multiple concurrent tasks. And then we need to see that what is the communication across the tasks that; that means, like if we see two tasks how do they communicate in between another or if you see one particular task how does it communicate with all the tasks in a given in the job or a group of tasks given in the job.

So, what is a global communication? What is communication across all the tasks or what is the local communication? What is communication across two specific tasks? That has to be looked into because I told earlier communication acts to the main part of the overheads, we need to see how the communication is happening. And we have to take care of communication overheads while designing the parallel algorithm.

Now, based on that, so we have seen that the job can be broken down into many concurrent tasks and these tasks require communication in a global sense a task need to communicate with all tasks in some point of time, also this task required communication in between them.

Now, looking into the communication, communication means one task need to send data to another task, one task needs to get information from another task. If these tasks reside in different processors which have in case of distributed memory programming and Foster's methodology is mostly applicable for very mostly important for distributed memory programming.

In case of distributed memory jobs if two tasks need to communicate in between them if they are given two same process ok, there will be done serially, but they do not need to send data through a network cable and go through all the communication overhead. It will be on the same memory; they are accessing and one by one the data can be accessed and task can communicate in between them.

If the tasks are residing in different processors, if they had given two different processes then they have to communicate across themselves. So, there will be communication overhead. So, depending on the communication, we know or the partitioning is already done depending on the communication overhead how many tasks will be given to how many processors that will be found out. We assumed that we have a limited number of processors, but the problem is large it can be broken down into number of great numbers of concurrent tasks.

So, the next step will be, agglomeration that is combining a group of tasks into larger tasks. So, several few tasks will be taken and they are combined into one a larger task. In between these tasks all the local communications they do not need to go through the network cable switch.

So, therefore, the communication overhead will be reduced here. The overall communication in terms of network switch communication that is reduced here because locality of data has been increased within a processor. But for global communication or for some of the communications which one tasks need to send to another task which is residing on another different processor with a different memory in a distributed memory system, that needs communication.

However, that communication can be reduced when we agglomerate the task looking that great number of local tasks. All these tasks are communicating locally. So, a great number of local communications have been taken down into one particular processor and therefore, they do not need to communicate by a network switch. So, this is called agglomeration.

And once this is done then the processors have to be tasked or mapped into the processors. This also has an issue with communication. As group of tasks goes to one processor another group of tasks goes to another processor. And, now in between these tasks there will be some communication probably, but if these processors are very far from each other there will be many hops in the network, through which the message has to go it will add to the communication overhead.

Therefore, there is also requirement of locality, data locality when mapping the task that the tasks which are communicating across between them if they are not residing into same processor this must go to neighbouring processor. Well so, if we see there is a large problem which is broken down into multiple concurrent tasks, and, now we can see how the task are communicating in between another.

Looking into the communication pattern we can agglomerate some of the tasks. So, when we agglomerate this task, they do not need to communicate across themselves, they are in the same processor, so therefore, they can talk easily. And now once this agglomeration is done that the locality is ensured then depending on the number of processors, we map them into different processes.

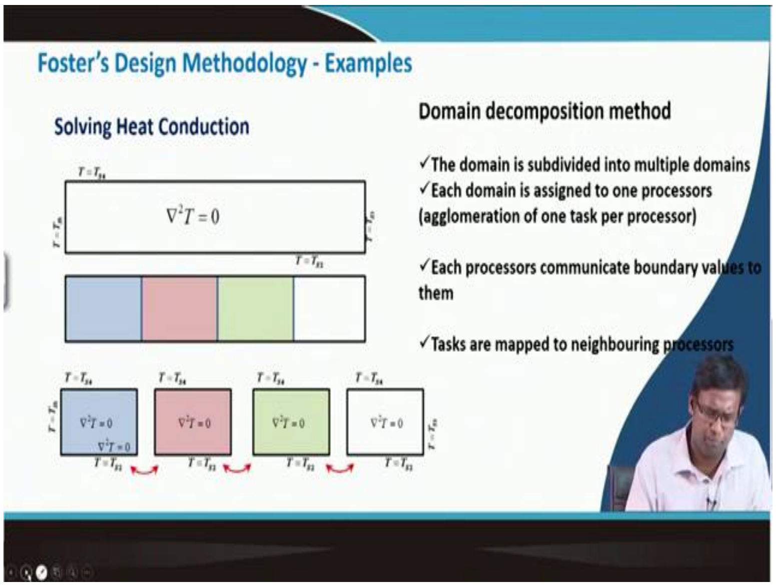
And now we can see that maybe one processor needs to communicate with one another processor, the other processor needs to communicate with them and the overall distance between the processors. Here it is shown 3 processes, but if you have a greater number of processors, we can map them accordingly.

This will also take care of load balancing. One of the significant differences between agglomeration and mapping is that agglomeration is grouping the task looking into communication over it. Mapping is mapping the group of the tasks looking into localization, some part of communication overhead, also when this group will be done this will take care of load balancing, that they have nearly same number of tasks. Also, they will take care of the fact that some of the processors which are interacting with another processors task and not very far from each other.

So, here we only take care of the communication across the tasks how data locality can be increased communication locality can be increased, here how many processors are increased and how load balancing will be done. This is an extremely important task that through looking into communication the partitioning the decomposed tasks are grouped and he had agglomeration into a group of tasks which is considered as a new task set for the for the actual problem. And the load balancing is done based on the agglomerated task not based on the granular task or concurrent set of tasks, but based on the agglomerated tasks.

From agglomeration, the group of tasks will go to different processors through a mapping which will also take care of load balancing.

(Refer Slide Time: 31:13)



Foster's Design Methodology - Examples

Solving Heat Conduction

Domain decomposition method

✓The domain is subdivided into multiple domains
✓Each domain is assigned to one processors (agglomeration of one task per processor)

✓Each processors communicate boundary values to them

✓Tasks are mapped to neighbouring processors

So, we see quickly couple of examples. First is a heat conduction problem which is solving Laplacian of temperature over a domain and we use something called a domain decomposition method in which the entire domain is broken down into 4 sub-domains. Here we are seeing it 4 it can be more and each in the problem inside each sub-domain is solved concurrently.

So, independently we solve problem inside each sub-domain, but there is a requirement of specifying boundary condition at the domain boundaries as well as establishing the continuity of the solution. Therefore, the solution at this domain needs to be mapped with the solution at this domain as well as there is a data exchange requirement from this domain to this domain.

So, while solving it and this equation is solved using a numerical method in which it is converted into a matrix equation. While solving this matrix equation we need to take data from one processor to another processor and this has to be done dynamically at different stages of this solution. So, this is known as domain decomposition method, we will discuss it in very great detail when you will discuss about MPI programming, but quickly see it in terms of Foster's methodology.

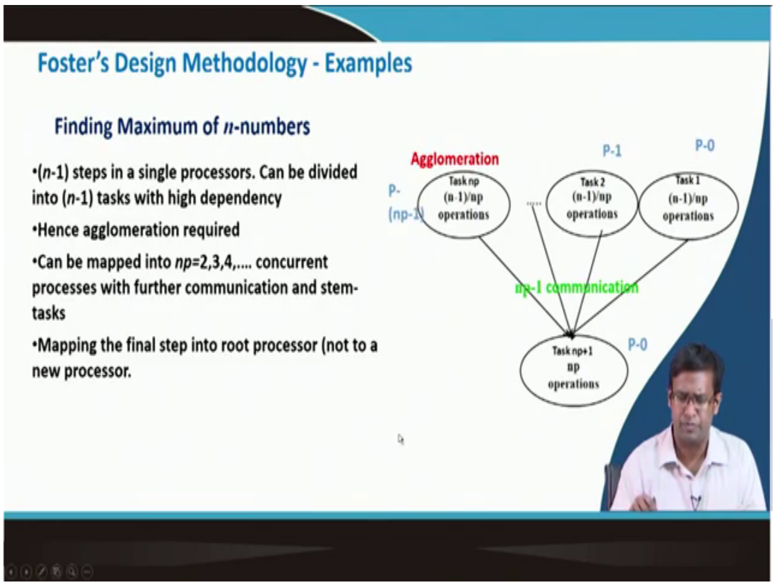
The domain is subdivided into multiple domains. Each domain is assigned to one processor. So, problem inside each domain is considered as one task. So, one task when we consider our domains problem it is one task which is agglomerated as again as one task. One finally, one agglomerated task comes out of our domain. Inside a domain we have if we think of hybrid parallelization, we can see that we can form multiple tasks inside the domain, but let us not look into that at this point. And then each processor communicates boundary values to them.

So, each processor needs to communicate with the neighbouring processors only and therefore, the tasks must go to the neighbouring processors. The task must be mapped to the neighbouring processors only. So, this is following Foster’s design methodology. This probably the simplest example of Foster’s design methodology that you have 4 tasks you consider agglomerate them.

Agglomeration means that tasks becomes an agglomerated task itself and you give it to neighbouring processors, so that there is some locality in data communication. And if we have to if this is given to processor 0 and this is given to processor 1, they need to communicate by a network switch which goes across the 4 processors.

In case of a in case of large computing infrastructure there can be multiple hubs there, but if they are given two neighbouring processors, it is much easier to communicate across neighbouring processors and also for the programmer’s perspective it is it becomes easier.

(Refer Slide Time: 34:11)



Foster’s Design Methodology - Examples

Finding Maximum of *n*-numbers

- (n-1) steps in a single processors. Can be divided into (n-1) tasks with high dependency
- Hence agglomeration required
- Can be mapped into *np*=2,3,4,.... concurrent processes with further communication and stem-tasks
- Mapping the final step into root processor (not to a new processor.

Agglomeration
P-(np-1)
Task np (n-1)/np operations
.....
P-1
Task 2 (n-1)/np operations
P-0
Task 1 (n-1)/np operations
np-1 communication
P-0
Task np+1 np operations

The next one is finding maximum of n-numbers. So, n is again a large number, we have we have a data set of large number of integers and we have to find out which is maximum of that. And it requires if we have in a single processor. So, one number has to be taken and compared with the next number, so on. So, it can it will take n- 1 steps.

And this can be divided into n-1 tasks. Now, these tasks have very high dependence each task is dependent on the previous task. So, agglomeration is required, so that we can reduce task dependency and group it into multiple groups of tasks. So, it can be mapped into np=2, 3, 4 concurrent processes with further communication and same tasks. So, what will like this that n-1/ np number of operations will be considered as one agglomerated task, n-1 / np as another task. Similarly, task 1 to task np, np tasks will come if np processes are there and each one is having n - 1 / np number of processors.

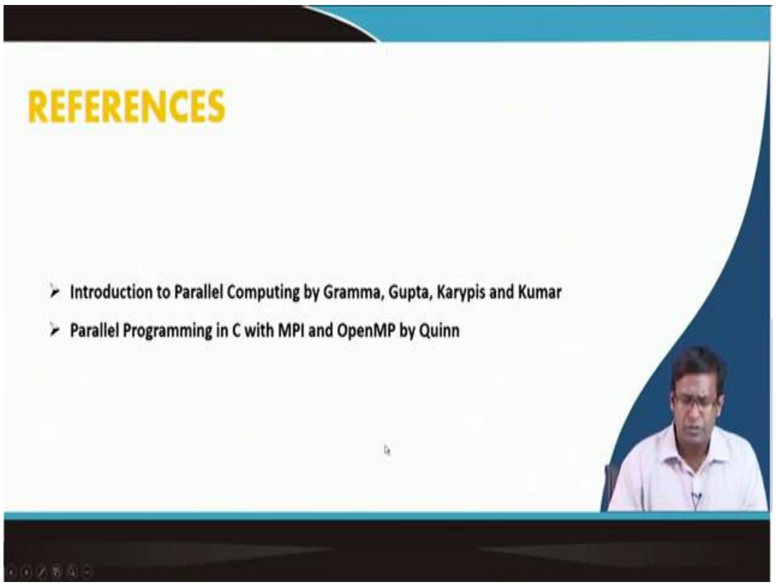
Now, here what we are doing we are also taking care of the fact that inside each agglomerated task, there are multiple tasks which are interdependent. So, the task dependency is also inside the local agglomeration. And then, once it finds out the maximum, so we have a large data set we have broken it down into np number of data set, each data set has n- 1/np operations.

And in each of the operations we find out what is a local maximum and all these local maxima are given to the stem task which has total there are total np processor. So, total np maximums are given and it will do np again, np-1 job to find out what is the final maximum.

So, when working on n-1/np operations they are being operated in parallel, but this is an essentially sequential part. So, this has to be the agglomeration is n minus 1 by np operations and they are assigned to each of the processes. And finally, all the results come in to one particular processor. This is an agglomeration step. And this is the communication step. So, this communication now is a strictly globe communication. The processors do not need to communicate across themselves.

And then mapping the final step it should go to a route processor, not to a new processor because if we take the take one of the one make one processor as master or route and ask all the processor to send data to that particular processor, the new task will be mapped to one of the old processor and there is at least one communication which we can reduce.

(Refer Slide Time: 37:09)

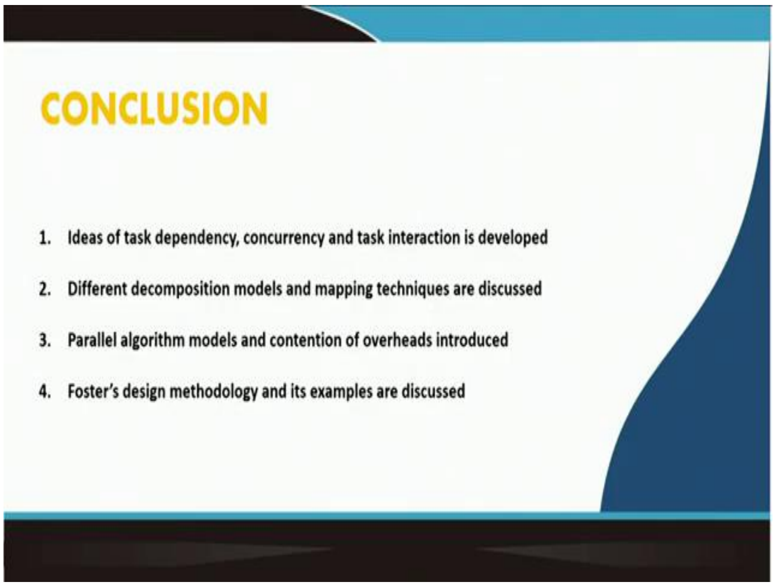


REFERENCES

- Introduction to Parallel Computing by Gramma, Gupta, Karypis and Kumar
- Parallel Programming in C with MPI and OpenMP by Quinn

So, we come to end of parallel algorithms and that is how you can take some of the algorithms yourself and look into its parallelization using Foster's design methodology. And, we will also discuss about it later in several applications when we will develop using a parallel programming or like MPI or OpenMP.

(Refer Slide Time: 37:31)



CONCLUSION

1. Ideas of task dependency, concurrency and task interaction is developed
2. Different decomposition models and mapping techniques are discussed
3. Parallel algorithm models and contention of overheads introduced
4. Foster's design methodology and its examples are discussed

So, here through the parallel algorithms series of lectures we had 3 lectures on that, first we learnt about, in the first lecture we learnt about ideas of task dependency, concurrency and task interaction. Then we discussed about different task different decomposition models and mapping techniques. Then we discussed about parallel algorithm models and how the overheads can be contained.

And later we discussed about Foster’s design methodology, and looked into two examples, two very simple examples on Foster design methodology, but you can take some of the complex problems which you have to paralyze and we will see that it is also following Foster’s design methodology.

Thank you.