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

Module - 01
Fundamentals of Parallel Computing
Lecture - 08
Parallel Algorithms (continued)

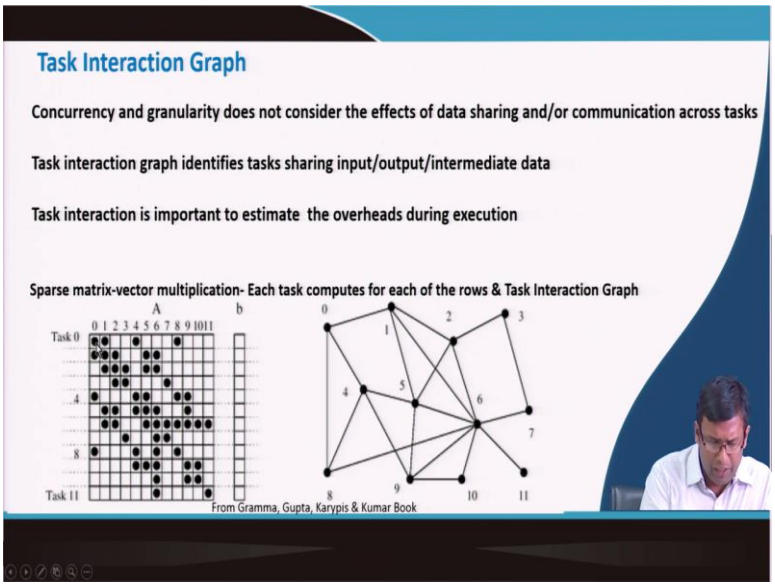
Welcome we are continuing our discussions on Fundamentals of Parallel Computing in the Parallel Algorithms lecture. That is what we are continuing today in the course of High-Performance Computing for Scientists and Engineers.

And we have been discussing the preliminaries that we need to know before looking into the methodologies to design a parallel algorithm. And we have still we discussed some of the important aspects of this preliminary concepts using including granularity, concurrency, critical path length, task graphs etcetera.

(Refer Slide Time: 00:58)

We will look into few other aspects there and then start discussing on the subsequent concepts which are decomposition and load balancing, overheads and it is containment and parallel algorithm models and fosters design methodology.

(Refer Slide Time: 01:17)



So, we have discussed about task dependency and drawing task dependency graphs and observed that we need to find out how many task can run concurrently in a parallel system. And while also noting down the fact that a particular job can be broken down into many tasks and task dependency graph can be drawn and this process that does generating a task dependency graph from one for one particular job can be done in several ways.

And we to we need to find out one particular task dependency graph which will give us better parallel performance. For that, identifying the concurrency ,specially the average concurrency of that as dependency graphs and which one has the best average concurrency ,that will give us a better parallel best better parallel algorithm that decision has to be taken.

The degree of concurrency means how many processors can run in parallel. So, the task dependency graphs which gives on an average more number of processors or processors can be active to solve the problem gives us a better parallel algorithm. Now the dependency of the tasks means one tasks output is required for the other tasks as input mostly.

There is another important thing called task interaction graphs. The, task dependency graph involving concurrency and granularity does not consider one particular aspect ,that is there is also requirement of data sharing and communication across tasks. So, we need

to do another thing which is a task interaction graph which identifies that which of the tasks have some sharing of data.

Sharing of data is important in couple of sense, if we are in a shared memory machine it is a same address space in which both the tasks are pointing to.They are has to be cache coherency and some protocols for cache coherency can introduce some of latency in that process, there can be false sharing which will also introduce latency there.

And for a distributed memory system if there are sharing data some data has to be shared from one processor to other processor, some data has to be physically passed through the interconnectors from one processor to another processor which will also put certain overhead in the calculations.

Therefore, this task interaction graphs graph is important to identify or estimate how much overhead is going to happen for one particular task dependency graph if we look into the decomposition and see the task interaction graph and how much data is shared across different processors if we try to find out that will give us some estimate on the overhead.

And we have seen a sparse matrix vector multiplication example which we have given in last discussion that each way is multiplying with the vector and generating one product in the right hand side vector. And we can see that say for the task 0 which is for the first row ,it is operating on the data belonging to first 0th column, 1st column, 4th column and 8th column.

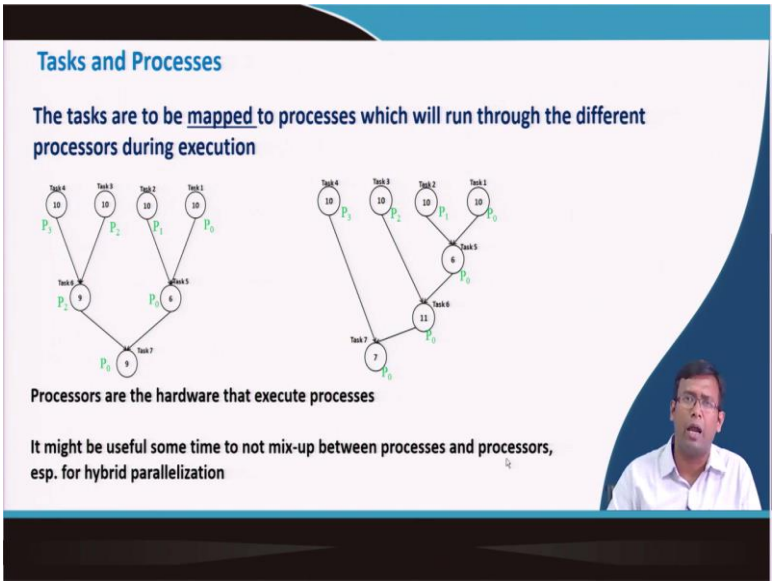
Therefore, when multiplying this it is multiplying with the first zeroth element of b vector, first element of b vector, fourth element of b vector and eighth element of b vector. Now the zeroth element of b vector is also multiplied with task once that is a column 1 row 1, task 4 row 4 and task 8 row 8.

So, the same data each of the 4 tasks are accessing therefore, task 0 has interaction with task 1, 4 and 8. So, and that is how we can get our task interaction graph and take a decision that which tasks are interacting with each other in terms of data.

In case of a shared memory system they are trying to access the same data there can be contention, there can be cache coherency protocol, there can be false sharing and

overhead.In terms of distributed memory system a same copy of these data in some sense here it will not happen probably, but in some cases same similar copy of some of the data has to be transferred from one processor to other processor. And therefore, it is also important to look into task interaction graph to identify the overheads in parallel program.

(Refer Slide Time: 06:03)



And now once we got all the tasks and drew the task dependency graph we need to map the tasks to the processors and see that which processors will run during the execution and which task will go to which of the processor. So, for example, here we can see that the previous class example that the finding out a school which is based on particular medium of instruction, which has a fee annual fee less than certain amount and belongs to one particular board from a list often boards and this was the task dependency graph for that.

And now once we have identified the tasks in previous class example now mapping means each of the task has to be mapped to one particular processor .And some of the tasks will be mapped some of the processors will be still active once the first set is done because there will be further tasks.

Initially there are 4 tasks now there are 2 tasks so, 2 processors will be in dormant state and 2 will be active. And finally, there is only 1 task so, 3 processors are not doing anything and only one is active therefore, the other processors are going into a latent

stage here. So, even we are using 4 processors we are not getting speed in terms of speed up of the factor of 4, because many processors are going into the latent stage.

And also it is important in the second part we can see similarly that the processors are working like that and another thing these we will not call them as processors ,we will call them processes that this process is working and taking care of the task, why I will come in a moment.

And the maximum amount of data that should retain to one process that we will be another idea of mapping, that you will avoid data transfer across the processes. So, that one process which is active from start to end, it should have maximum amount of data and that is also another important concept here.

And we call what is running in concurrently and in each of the processing the task that running in instance we call it to be process and processors are the hardwares which is executing process. So, in many cases processors and processors are equivalent, only process is the programming thing that each there are multiple processes which are taking care of multiple parts of concurrent parts of the program and processors are the hardware which are executing the processes.

They can be same in many cases, but it is also useful not to mix up between processor and processes because there can be hybrid parallelization where one process takes care of certain task and there are coprocessors in which this task is divided. So, processor and process may make wrong confusion if we mix them up for more technically we should use the term process which is going which the tasks will be mapped. So, one of the important part is mapping that each of the task will be mapped to a process.

(Refer Slide Time: 09:24)

Now we need to see how the task can be decomposed into many tasks, we have earlier discussed about decomposition that given a large job it has to be decomposed into many tasks.

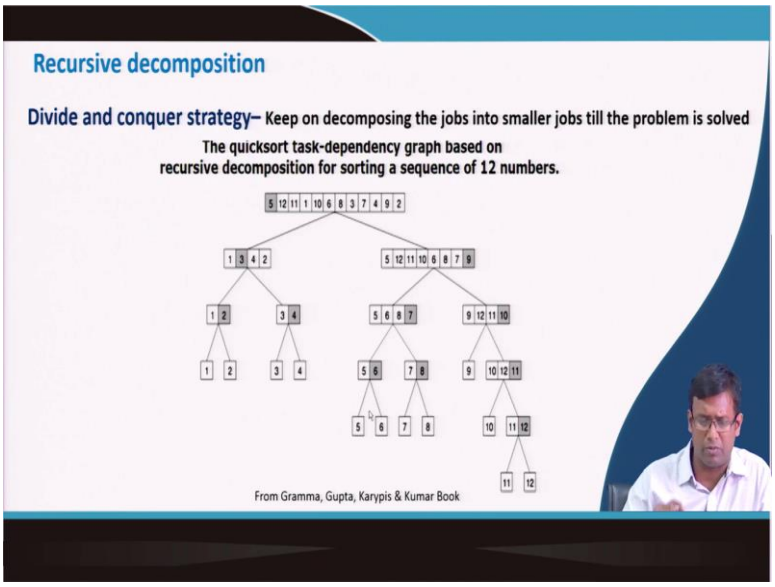
And we have seen some of the methods of the decompositions in our matrix vector multiplication or the such problem to given two examples of decompositions, but what are their ways .There are five different standard methods of decomposition one is called a recursive decomposition, the next one is data decomposition the third one is known as exploratory decomposition.

So, I will not spend much time because it is not a very high use in scientific computing, but in a sense you need to find out solution of one problem and you are not sure about the right way of solving it. So, instead of solving it sequentially you have to make few trial and error and each trial case you give to different processors make different tasks for each trial case and this decomposition that is a it is a trial error based solution in each trial is going being a task is a exploratory decomposition.

There can be speculative decomposition that for example, for chess playing after one we in our childhood it was to be a story that a supercomputer has beaten Garry Kasparov. So, how did it work? So, when Kasparov is giving one move the supercomputer is calculating that what are the branches of this move what are the next possible moves and what will be the solution for it.

So, speculatively you calculate the next moves and find out the next solutions and once he gives the next move it is already computed by that and there is mix-up of that. So, let us spend much time on exploratory and speculative decomposition, but let us go to recursive decomposition.

(Refer Slide Time: 11:34)



Recursive decomposition is also called divide and conquer strategy, a big problem is there it is broken down into small problems and these small problems have further broken down into small problems and finally, it is a very small problem which goes to each of the computer and it can solve it quickly.

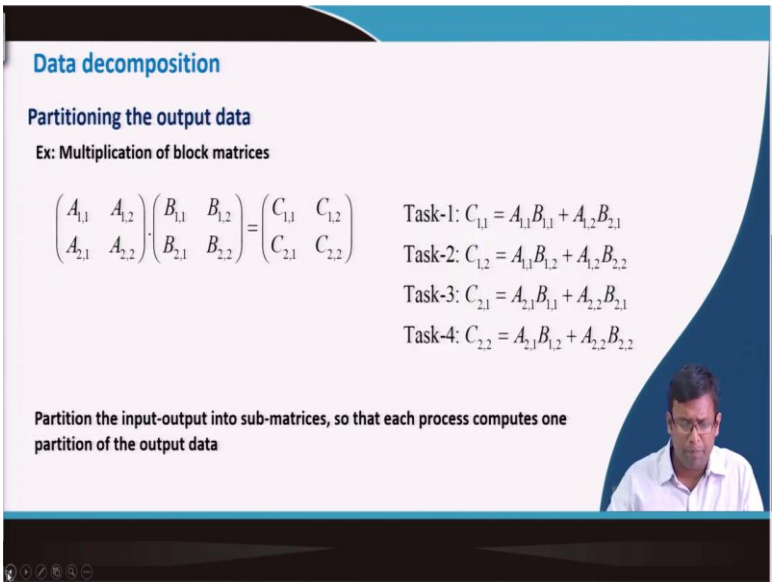
For example, you have to do a sorting of a set of 12 numbers. So, first identify the numbers which are less than 5 and identify the numbers which are greater than 4 and make two. So, you get 1, 2, 3, 4 here which is less than 3 identify the numbers which are greater than 2 and get two of the parts and then find out the 1 and 2 and 3 and 4 there.

So, as we branch this decomposition finally, we end up in a process where say these 2 processes can run concurrently, these 4 processes can run concurrently, again these 4 processes can run concurrently, these 2 processes can run concurrently .We break them into many concurrent processes and do a recursion on that.

As we break down the task a job into smaller tasks ,we get more concurrent processes and we get a simpler problem in each of the processors which can be solved first. And

through this finally, we have conquered the space. After a simple algorithm is the broken down this less than this less than this half like thing is executed we get many concurrent jobs and finally, each processor is coming out with the solutions. So, this is a recursive decomposition.

(Refer Slide Time: 13:17)

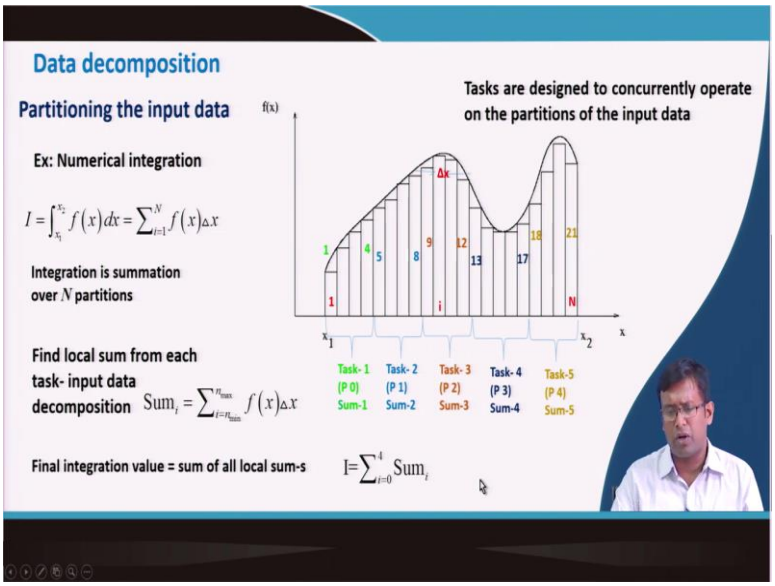


But for scientific computing we are more interested in data decomposition. And that is we divide the job by the data ,by what is going to happen in the data. For example, while doing a matrix vector multiplication, we can partition the output data and say 2 matrix multiplication, each of the matrix is considered as a block matrix and the output is 4 matrices of these 4 block matrices and each process is calculating one of the block matrices.

So, the actual task is decomposed into many subtasks by partitioning the output data. That first task compute C_{11}, second task compute C_{12}, third task compute C_{21}, fourth task compute C_{22}, this is partitioning based on the output data, each process computes one partition of the output data that is done here.

However, that there is one particular issue that is say both C_{11} and C_{12} is operating on same data A_{11}. So, there is huge amount of task interaction which is possible.

(Refer Slide Time: 14:36)

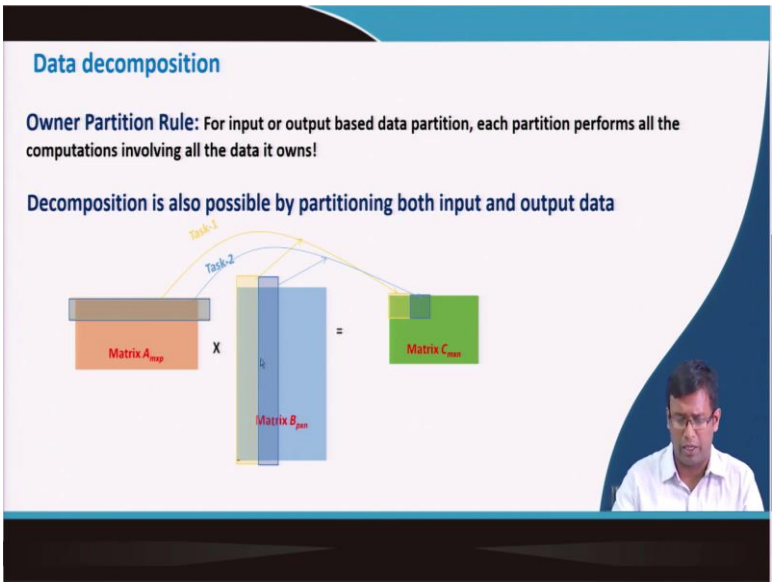


Now there can be also input data-based partitioning, say I have to do a numerical integration. So, find out f (x)dx within a range. This is the f(x) function in x axis, I distribute x_1 to x_2 , the range in to N small partitions, each partition is of with delta x. So,

$$\int_{x1}^{x2} f(x)dx = \sum_{i=1}^{N} f(x)\Delta x.$$

And then I divide the summation into multiple tasks, task 1 which goes to processor 0 takes care of the first summation And finally, task 2, task 3, task 4, task 5 ,many tasks are there and each task is taking care of the small amount of data given to it. So, we are dividing the input data , generating the task set, depending on decomposition of the input data ,which is what is the sum.And the final value input data, is what are the input points and the final value is, sum of the all of all points.The tasks are designed to concurrently operate on the partitions of input data.

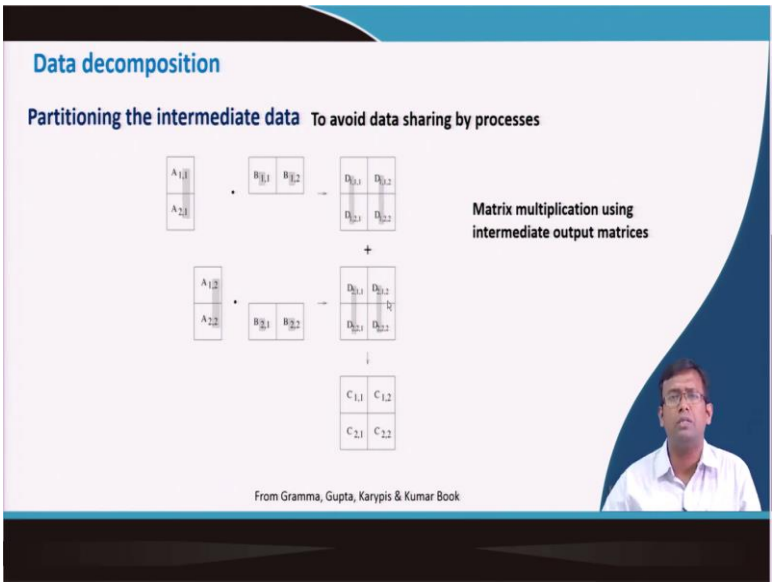
(Refer Slide Time: 16:00)



Both input and output data based decomposition follows owner partition rule that the amount of data given to one particular process it will do all the computation for that amount of data.If it is input data it will do all the computations involving that input, if it is output data it will do all the computations to get that output and it is called the owner partition rule.

Decomposition is also possible by partitioning both input and output data, say we take up a matrix vector multiplication and first block of rows with first block of column gives one part one block of the product matrix.First block of row with second block of columns given gives second block of the product matrix. We are partitioning the based on the output as well as making partitions based on the making partitions on the input also. So, it is partitioning both input and output data.

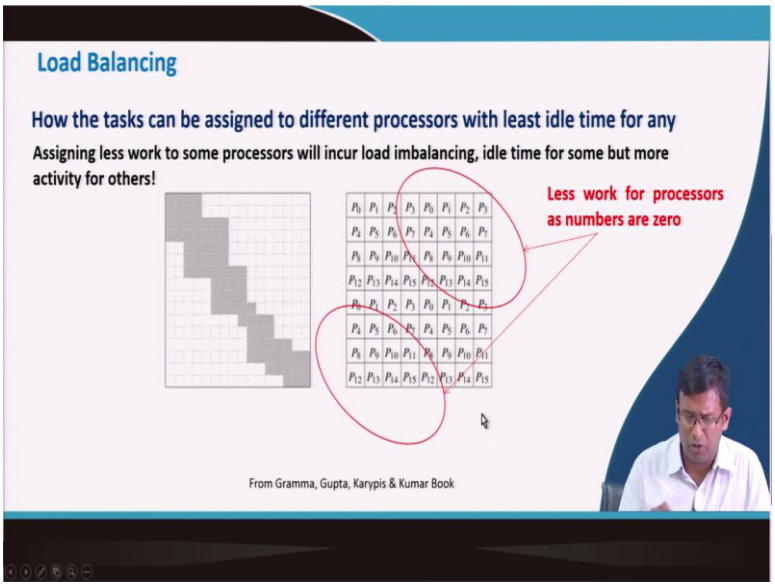
(Refer Slide Time: 16:57)



There can be partitioning on the intermediate data because in both the previous cases they sharing lot of data which will give good or how huge overhead. Instead of that we do take intermediate matrices so, instead of writing A into B is equal to C we take first rows of first block of rows of A columns of A and first block of rows of B and get a intermediate matrix D_{11}.

Similarly we get a intermediate matrix D_2 by second block of columns of A and second block of rows of B and then sum them up. So, this is creating an intermediate data and partitioning the main matrix into these two intermediate data set, which reduces the data sharing across the processes.

(Refer Slide Time: 17:52)



Now, mapping is important so, the tasks are mapped to different processes, say we think of A and where it the important part becomes load balancing.We have earlier seen that all the processes are not active, initially for the search problem four processes at active, then two will be active, then one will be active. So, some processes are less active some are more, which one is less active is sitting idle and therefore, we are not getting right speed.

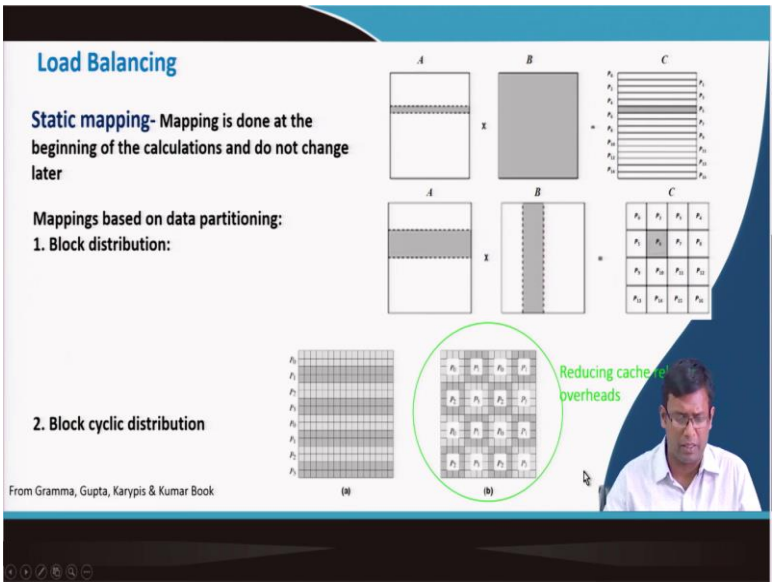
So, in case if we try to map the job from tasks to processes and some of the tasks are less heavy than some other then the less heavy task processes will move finish their job faster and will sit idle. And therefore, we need to do a load balancing, so that each one is getting almost similar amount of load. If we have a sparse matrix vector multiplication and these are the non-zero elements of the matrix and these are the 0 elements of the matrix.

And now we do a decomposition in which different block is given to different processor, we can see that the blocks processes which takes care of these blocks this sparse matrix vector. So, these elements are 0 they are practically working with 0s therefore, they need not have to do anything, 0 multiplied by 0 is anything 0 we know the number we know the result only you will not ask the processors to work there. So, this will be loading balancing and less work for the processors as a numbers are 0 .While others are working they are still sitting idle, instead we could have given some amount of work to this

processors in computing the non-zero part of the matrix this would have been a better load balancing.

So, the actual work will be in reality not in practice not by assignment wise, but in practice given to the all the processors and each processor can finish their job faster compared to the case that some processors have done already, but others are waiting that others are still working that would not have happened.

(Refer Slide Time: 20:19)



So, load balancing is also important when writing a parallel program and it is usually programmers' duty to tell which tasks will go to which process. So, he has to do a mapping, he has to map the tasks into the process so, that load balancing is taken care of and this mapping .There are two classes of mapping, one is static mapping where mapping is done beginning of the calculations and it is fixed it does not change throughout the computing. So, initially this some of the tasks are identified and this task will go to this process, this task will go to this process this is I fixed.

And say we do a data partitioning. So, we can do block distribution. So, we do a matrix vector multiplication and do it based on the output data. So, these are the different tasks which calculates different parts of this data and each of these tasks goes to one particular process and that task takes care of one block of the output data. And each task goes to one particular processor that is called a block distribution, block by block we distribute the tasks.

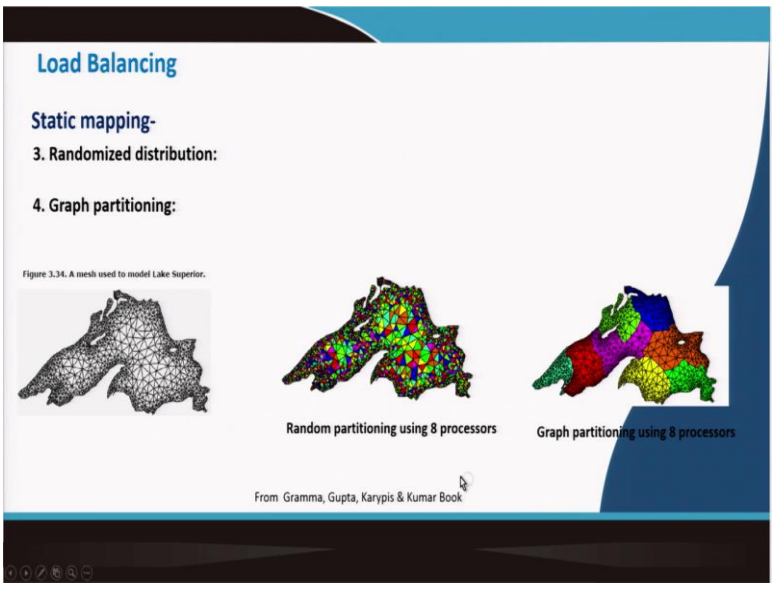
There can be different type of block distribution also based on both output input data what we are that decomposition here also you can do block distribution. So, sequentially 1 to 16 or sequential here 1 ,2, 3, 4 up to 15 jobs are given to different processes.

Now there can be block cyclic distribution, instead of giving the jobs 1 to 16 you give first 1 ,2 ,3, 4 and then again you repeat a cycle of that 0,1, 2, 3, again 0, 1, 2, 3 and you do 0 0 1 0 1 2 3 2 3 0 1 0 1 2 3 2 3.

So, you make a cycling of block distribution, what is the advantage? That, there will be some of the cache coherency related issues where some read write protocols are there, some processors are not working or some message transfer issues, some processors are not working, others are working to take care of the communication or share data sharing issues.

And then you can do a read write red black algorithm that this processor is working this one is working, but these two blocks are not working because they may have to same process instead these two blocks are working or say these two blocks are working. So, that they do not have cache overhead related issues. So, block cyclic distribution is good in some senses to reduce cache overhead.

(Refer Slide Time: 23:16)



And there can be randomized distribution for cache over also, then there can be graph partitioning like all these are not matrix vector type of products when you know right

number of rows and right number of columns and get very uniform number of blocks and distribute the job.

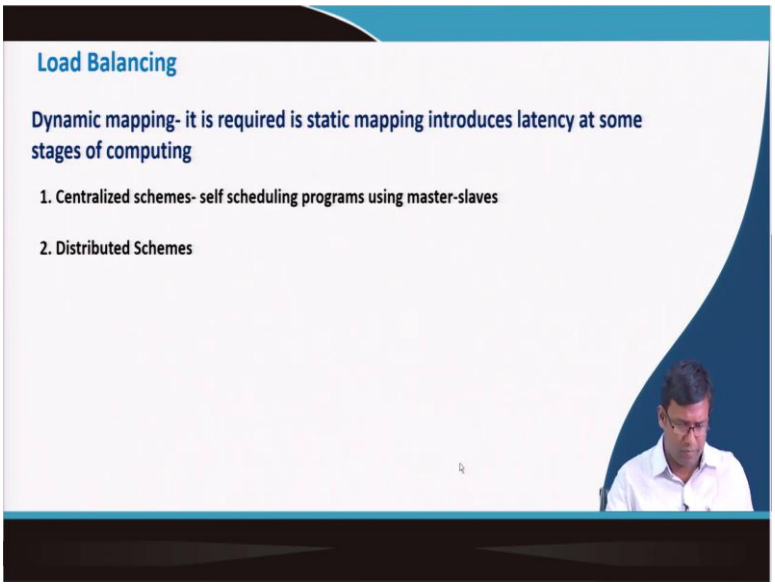
There can be cases for example, finite element this simulation an lake superior which is again taken for from Rama Guptas book that the number of tasks and each small triangle represent one tasks say here which is computation on that particular element in the lake superior domain. So, each task has different size and the task where very non uniformly unstructured distributed over the geometry.

One idea is taking some of that ,we know that this will be distributed among 8 processes. So, generating random numbers and allocating within 1 to 8 and finding out which will be the location so, it is a random way of distributing the job.

The problem is that there can be different part of the domain given to different processes and there can be huge data sharing requirement, because the neighbouring in job goes to different process instead of localizing the data we area randomly distributing the processes so, the data is less localized and more sharing is required.

The other one is a graph partitioning in which more localized job distribution is possible and the graph partitioning is better and this these two are randomized and graph partitioning are the techniques of task distribution especially for Laplace equation type of solvers when you have a very irregular domain and you have unstructured distribution of mesh points. If you have a structured distribution of mesh points like matrix vector multiplication or working in a Cartesian geometry or polar geometry the block or the cyclic methods are better.

(Refer Slide Time: 25:33)



And there can be dynamic mapping also, the computing is completely changing during the scenario say we are solving about molecular dynamics simulation and the number of molecules interacting with each other their distribution etcetera are changing solving some chemistry equation ,the composition is changing.

In that particular case instead of static mapping it might be of importance that dynamically the mapping is done a different stage of calculation as the arrangement has been changed that your problem configuration problem size everything has been changed it might be important to do a dynamic mapping.

There can be cases where the geometry itself is changing, the topology is change and therefore, need to do dynamic mapping. This can be done by centralized schemes that there are self scheduling programs using master slaves.

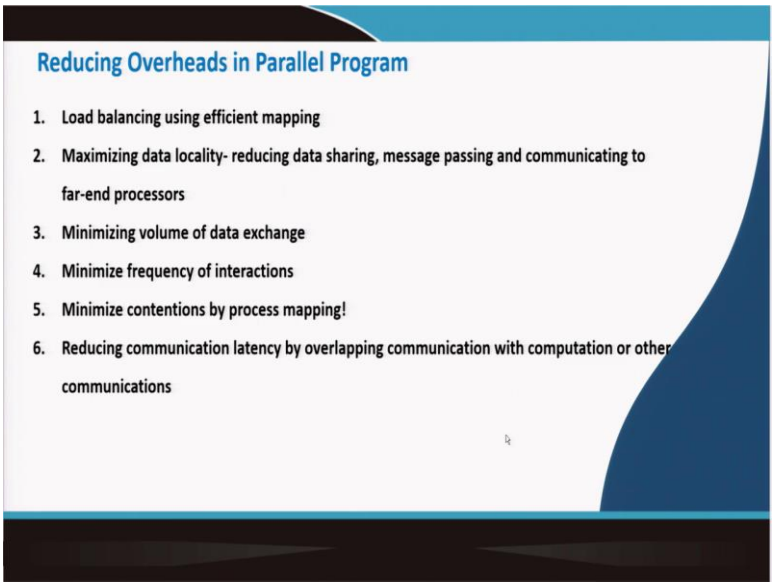
So, one computer is asked or one processes is asked to be a master and it will see that what is the total amount job at and each different instance of calculation, it will do a new task partitioning and assign different new decomposition and assign different tasks to different processors. There can be distributed schemes where there is multiple zone ,it is like a recursive type of distribution scheme is given to it, multiple processors are again breaking down and giving their jobs to other processors.

(Refer Slide Time: 27:16)

We understand that when computers will work in parallel all of them are not working in together some much going to be idle and as well as there will be communication requirement across the processors.

So, the next important part will be understanding the overheads in parallel program and the sources of overhead and performance degradation at load imbalance that some of the processors are getting more work, some are getting less work and communication overhead .Communication overhead means there is a need of data transfer across the processes.

(Refer Slide Time: 27:55)



Now, in order to reduce the overhead one important step is doing an efficient mapping where load is balanced. These are also very important steps for a parallel programmer that he knows what are the overheads and how to reduce it. Maximizing the data locality compared to randomized and graph partition decomposition, in randomize ,the processors are one particular processes has tasks spread across entire domain of the given task set given job set, but in graph partition case it was much more localized.

So, if we can increase the data locality the data sharing our message passing will be reduced and it will this there will be also not in much need to share data with the processes which is at far end of the node, it will only share data with the neighbouring processors. So, the data sharing's time will be less and over it will be less minimizing the volume of data share.

So, one is that data local, it will minimize the requirement of data sharing, but again when data has to be shared some way the volume of the data exchange has to be minimized and the frequency of the interactions, frequency of data sharing that has to be also reduced. Once data is shared it will be shared as a packet because every time a data sharing will take some start up time. So, as many times data sharing is happening my more start-up time is included and more overhead is imposed.

Minimizing the contentions by process mapping ,that is, like a block cyclic mapping not all the processors are trying to read from the same location of the data and contention is

reducing . And reducing communication latency by overlapping communication with computations or other communication when computation is costly.

So, when one processor is computing it has some overlap ,when one processor is communicating ,that time data has gone from this processor tries to reach another processor at that time also this processor goes from for some other work. So, there is some non blocking miss during communication that it does not goes into halt because it has sent a data to somebody else. While it has sent the data to somebody else some other task has been picked up by the process and it is working and that also reduces the overhead .

So, we have seen some of the issues regarding the parallel algorithm and the points on which a programmer has to take care, that involves doing a right decomposition, then mapping it accordingly and then looking into the overheads. So, that the communication time can be minimized and even the communication time cannot be minimized by a communication, computation or communication-communication overlap some of the communication time can be utilized by the processor itself.

Now in the next class we will look into that using these concepts how a good parallel algorithm model can be developed and what are the available parallel algorithm models and we will take some examples and see how these how these models can be implemented to parallize that particular example problems.

Thanks.