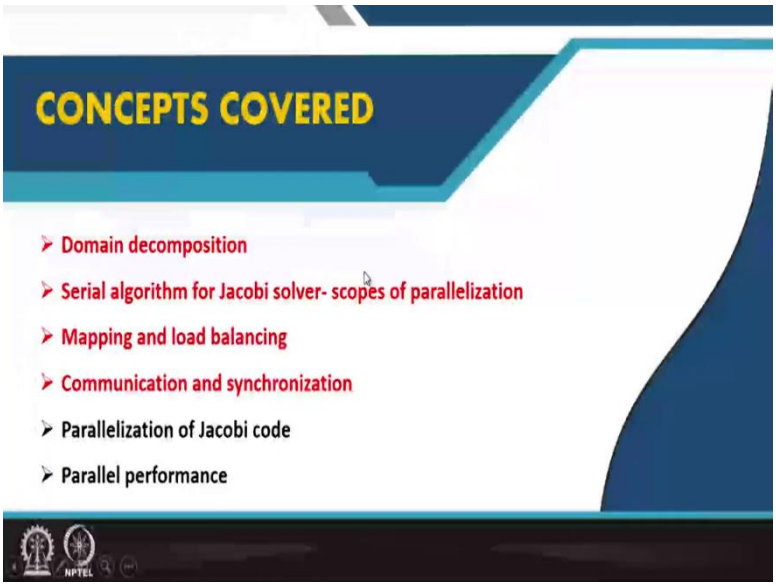
High Performance Computing for Scientists and Engineers
Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Module – 03
Distributed Computing using MPI
Lecture - 27
Domain Decomposition based parallelization of matrix solvers (continued)

Welcome to the class of High-Performance Computing for Scientists and Engineers. We are discussing Domain Decomposition based parallelization of Jacobi solvers. This is the 3rd module of the course where we are talking about Distributed computing using Message Passing Interface or MPI and continuing from previous lectures.

(Refer Slide Time: 00:45)

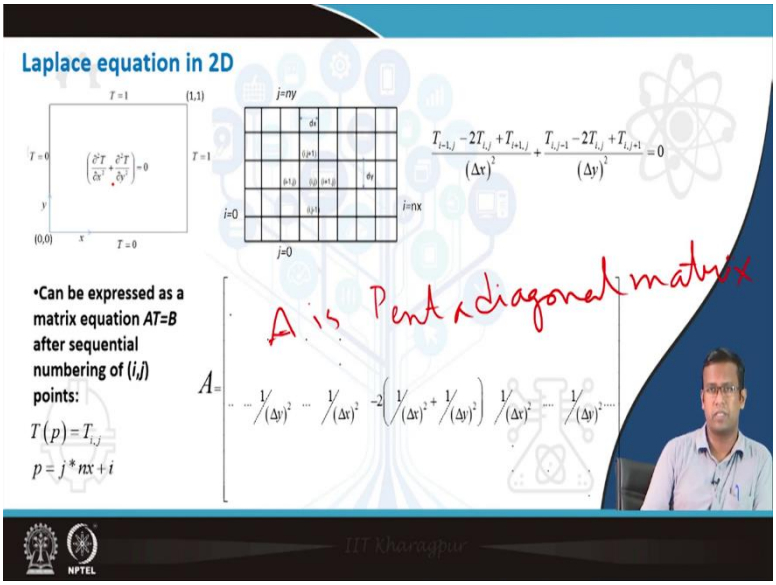


In the previous lecture, I have discussed about the basic idea of domain decomposition, then we looked into the serial algorithm or single processor algorithm for Jacobi solver, identified the scopes of parallelization looked into mapping and load balancing for doing domain decomposition in a Cartesian geometry Laplacian solver and also looked into communication and synchronization issues.

Based on the previous discussion now we will look into a Jacobi solver for the Cartesian geometry Laplace equation. Identify the steps in which we can write the parallel constructs and

develop a template for a parallel solver. Then subsequent class will take this parallel template and try to develop the parallel Jacobi solver and look into its performance.

(Refer Slide Time: 01:33)



So, we are considering a Laplace equation in 2D which is a Cartesian domain. We have a Cartesian domain (0, 0) to (1, 1). So, it's a unit square inside which you have to solve $\frac{d^2T}{dx^2} = 0$. The boundary conditions are x = 0 y = 0 lines are at T= 0 and the other boundary is at T = 1. It is very easy to find out the analytical solution for this equation which we will be using separation of variables, you will get a Fourier sine series and find out the solutions. We are discussing the Jacobi code for matrix equations here, but if you are interested you can also find out the analytical solution and compare the solution that you are getting using this Jacobi solver and see the accuracy.

So, the geometry is discretized into a number of points in a number of lines in x and y direction. So, we get a set of points there i starts from i = 0 to i = nx total nx divisions in x. Similarly, in y there are total ny divisions. So, total number of points in this geometry will be (nx + 1) *(ny + 1).

However, the 0 and nx ,0 and ny lines are the boundary lines. So, only the internal points which are nx - 1 into ny - 1s are important to solve. But if we can include the boundary conditions, we will get a matrix which is of the order of total number of points including the boundary

condition is nxny. So, you will get a matrix of the order of nxny X nxn y. Now, for each internal point using a finite difference method we get the discretized equation.

This equation Laplacian of T = 0 can be substituted by this difference equation. A differential equation is replaced by difference equation in numerical method and the error is of the order of $(\Delta x)^2 (\Delta y)^2$.In case we get large number of points within this geometry, $\Delta x \Delta y$ is small then the error will vanish and will get more very accurate solution.

Again, getting a large number of points means the matrix size will be very high. So, that is where doing parallel computing equations is beneficial. Now, this equation is given in terms of i j index. We cannot directly consider it as a matrix equation.

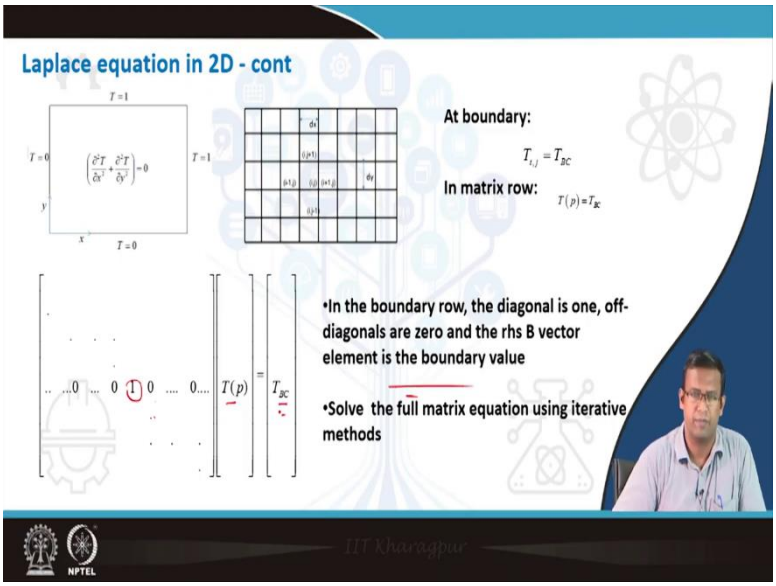
In order to get it in the form of a matrix equation we have to convert this T into a 1 d vector and that can be done by numbering the T s. So, T_1, T_2, T_3, T_4, T_5, T_{nx}, T_{nx+1}, T_{nx+2} so on. If we can number the T 's we will get the single T vector over which we can rewrite the equations and get a matrix equation. This can be done instead if T_{ij} is called as T_p, p is the pointer for any location of $T_{i\,j}$ and p can be found out as j * nx + i.

So, for j =0 ,p = i. This is 0, 1, 2, 3, 4, 5, 6, nx j = 1 p = nx + i ,nx ,nx + 1 , nx + 2 so on. Sorry, this should be nx + 1 right. Yeah this should be nx + 1. So, after doing this numbering we substitute we rewrite the equation and we get a matrix equation.

This matrix has a diagonal value $-2(\frac{1}{(\Delta x)2} + \frac{1}{(\Delta y)2})$ which is the i,jth terms coefficient in this equation and most of the off diagonals are 0. $\frac{1}{(\Delta x)2}$ for i + 1 and i – 1. $\frac{1}{(\Delta y)2}$ for i + nx and i – nx.

Similarly, all the terms appear in this equation except the boundary terms and we get a matrix equation out of this. These are sparse matrices. Because there are only 5 terms near the diagonal which are non-zero, we write it to be a penta diagonal matrix. A is a diagonal matrix. If we are solving the 1D equation, we get a tridiagonal matrix, 2D penta diagonal. If you are solving 3D, we get a septa diagonal matrix 7 non-zero number. So, 2 will come for the z direction.

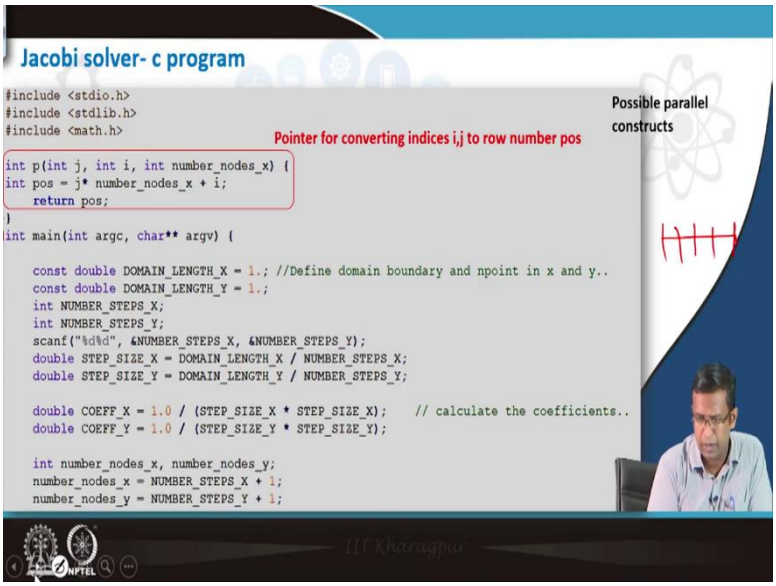
(Refer Slide Time: 06:47)



At the boundary, we get $T_{ij} = T_{BC}$ with the essential or Dirichlet boundary conditions. So, boundary condition is directly specified. If we take the boundary points in the matrix, we get $T_p = T_{BC}$. So, the equation for this point is $T_p = T_{BC}$ and in the matrix that can be substituted as all the up diagonal 0 only the diagonal 1 and again all the up diagonal 0 into T_p and the RHS matrix (b matrix) gets the boundary condition value T_{BC}.

So, that is how we can substitute all the boundary conditions also and form a matrix using (nx + 1)*(ny + 1)rows. In the boundary row, the diagonal is one ,off diagonals are zero and the rhs B vector element in is the boundary value. We will build the matrix for our Jacobi Laplace solver code using this boundary substitution. Now, we get a full matrix equation. You do not have to separately consider boundary conditions that are already given in a matrix and the b vector ,get the full matrix and solve it using iterative methods.

(Refer Slide Time: 08:09)



We have discussed the Jacobi solver in the previous session. So, first is that we use a pointer for converting because we have looked into the geometry and formulated finite difference points based on i j number of points in i and the number of points of j. But we have to convert in order to get a matrix equation from this ij index into a 1 D vector.

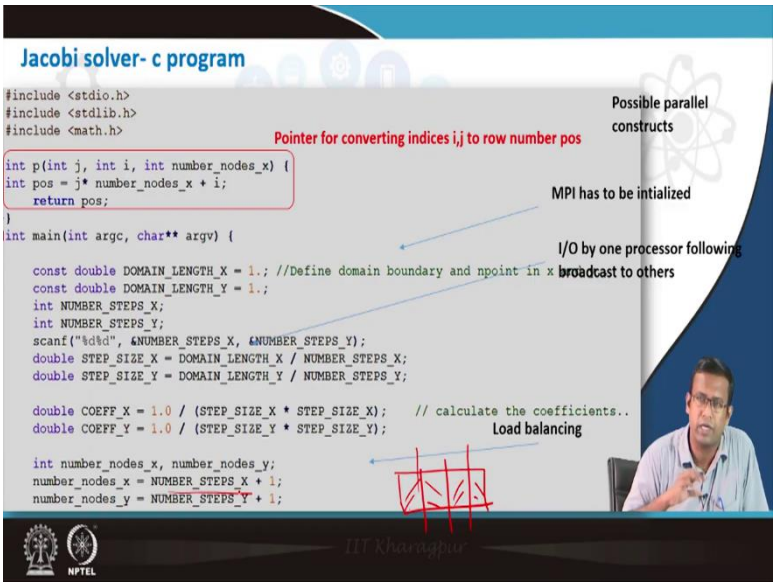
So, use a function p which will take the j index, the i index and number of points in the x direction and return the position pos for that particular point. If we call p it will return the location of that particular point in the vector .p is an integer valued function. Now we come to the main subroutine.

The domain length in x and y is 1 .The number of steps in x and number of steps in y are read from the user. Step size($\Delta x, \Delta y$) is domain length by the number of steps. The coefficients for x ,y is $\frac{1}{(\Delta x)2}$, $\frac{1}{(\Delta y)2}$and the central term will be $-2(\frac{1}{(\Delta x)2} + \frac{1}{(\Delta y)2})$.

Now, the number of nodes including all the boundary nodes also will be the number of steps + 1. So, if we have one particular length, we divide it into 2 steps the number of nodes is 2 + 1= 3. Similarly, if we divide into 4 steps number of nodes is 4 + 1 =5.

So, the number of nodes is the number of steps + 1 in both directions. This is the pointer for converting i j indices to row number. What are the possible parallel constructs here? Where should we try to write parallel functions MPI subroutines or something else or some other programming clauses to make it a parallel program.

(Refer Slide Time: 10:50)



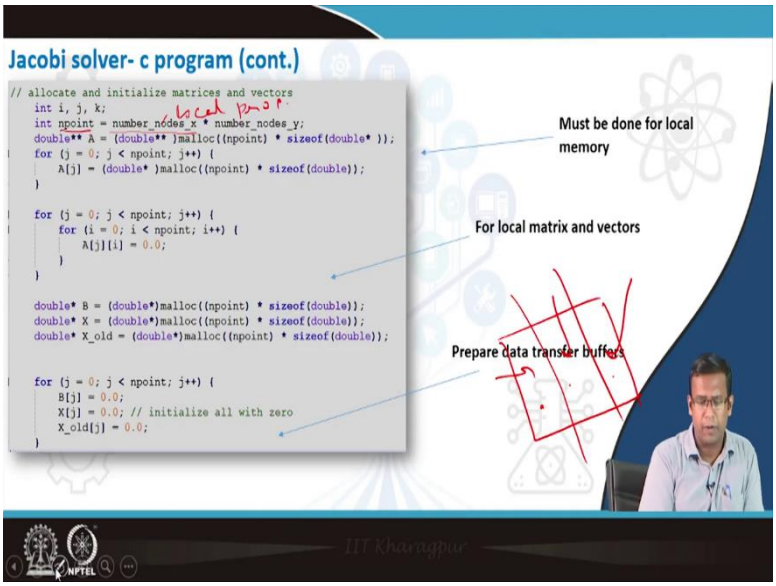
Well, first is that we need to initialize MPI if we think of a parallel program. The Message Passing Interface will be used for distributed memory data transfer based parallel programs. So, MPI init, MPI com size, MPI com rank these calls have to be given there, Now, it is reading something from the user number of steps in x and number of steps from y.

If all the processors try to read from the user that will be a time consuming and conflict arising step because it is the same user who is giving the input and multiple processors are trying to read it.I have one piece of information or one letter I have to give it to someone and if 10 people come and try to get the letter there will be conflict.

So, what we do, we will ask only one processor to read these things and then once it has read it will broadcast it to the other processors. Now, number of nodes in x and y; we will take the geometry and do domain decomposition in the x direction as we have shown in the previous lectures. So, these will be my domains in x direction.

Number of steps in x will be small compared to the global number of steps; we will divide the total number of steps by the number of processors and do load balancing. So, each processor will get a small number of steps compared to the global number of steps in x . So, the number of steps in x will be the domain decompose number. Here we are not doing anything in y direction. So, all the steps in y will remain the same.

(Refer Slide Time: 12:43)



After that we have to allocate and initialize matrices and vectors. So, you allocate the matrix A of the size npoint is the total number of points. A is of the same number of rows, so it is a 2-dimensional matrix. Then b, x and x_old the guess variable all these are allocated using npoint and everything is initialized as 0 at the beginning.

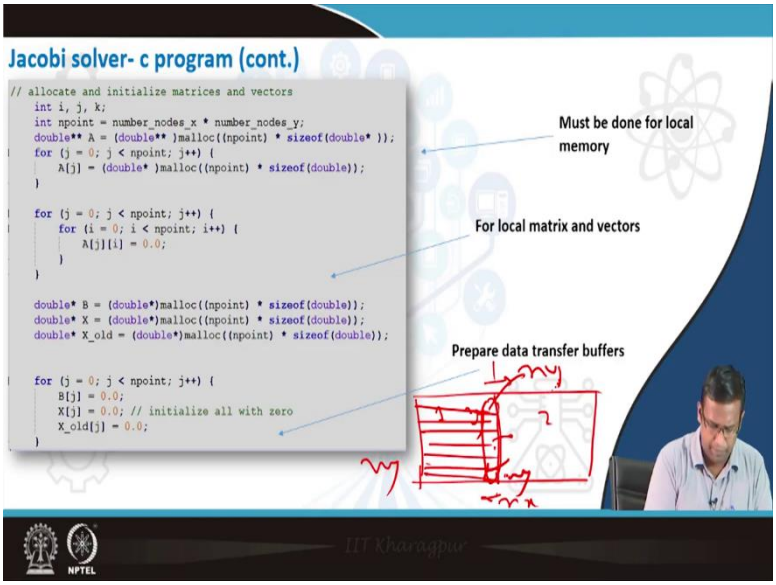
Now, what is to be done to parallelize it? This allocation must be done for the local matrix and vector. Ax = b here we are solving for the global matrix. If we parallelize it, we will get a subdomain. So, now, we are solving Ax = b for the global matrix in this particular code.

If I parallelize it, I will solve A x = b for all the sub matrices coming out from the sub domains. So, the allocation has to be done for the local matrices and local vectors as well as the initialization also. How can that be done? Local processor number of nodes in x is small; it is given for the local processor. The end point will be different for each of the processors there will be a different endpoint each of the sub domains and therefore, this allocation and the initialization will be for the local points. This must be done for the local memory.

Another important part will be you have to prepare the data transfer buffer. Remember in the last discussion we are talking about 2 steps. One that data has to be sent that has to be put into one contiguous array and that will be sent because when you are sending data, we will only send the address and the size of the buffer.

So, the entire send data has to be put in a buffer which will be sent. When we are receiving data, we are receiving it as a buffer and now that has to be mapped in the right location. So, these buffers, what is the send buffer, what is the receive buffer, these buffers are to be prepared at this stage because once we have initialized everything ,allocated all the variables we know the size of the buffers.

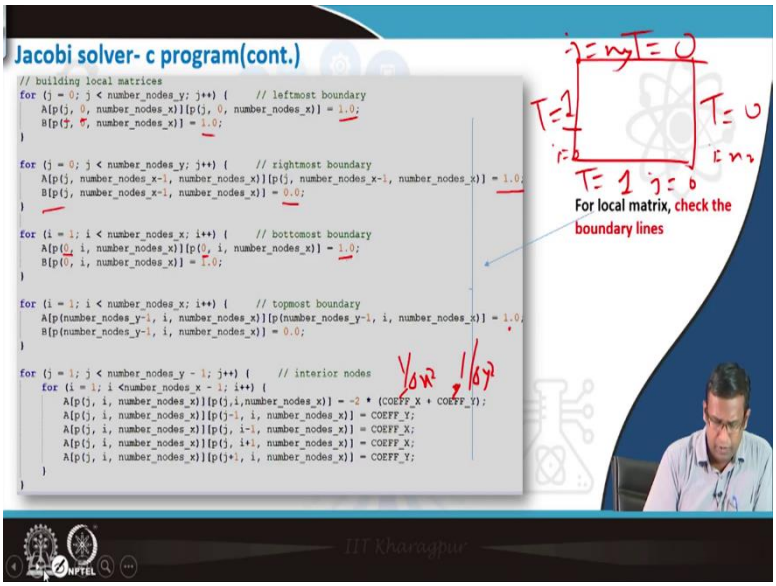
(Refer Slide Time: 15:54)



So, what is the size of the buffer? Say I have total nx points here and I have total ny points here and this is my decomposed domain 1 and 2. There will be data transfer from 1 to 2 and 2 to 1. Which data will be transferred from 1 to 2? This data will go from 1 to 2. What is this data? This is nothing but the y line which belongs to the last x point of the first domain line means how many data points are there? There are total ny data points because there are ny points here. So, there are total ny points that means ny amount of data has to be transferred from processor 1 to processor 2.

Similarly, processor 2 will also give ny data from 2 to 1. So, these data transfer buffers have to be allocated and they have to be prepared. You have to define a data transfer buffer and allocate based on the number of points in the y direction.

(Refer Slide Time: 16:45)



Now these are the boundary conditions .We are even now looking in the Serial Jacobi code. The boundary conditions are to be given. How a boundary condition is given ,we have seen in the first few slides of this discussion that at the boundary row the diagonal is 1 in the A matrix, and the off diagonals are 0 and the right-hand side vector contains the boundary value. So, the leftmost boundary, if we remember the boundary conditions right, is T = 1, T = 1, T = 0, T = 0.

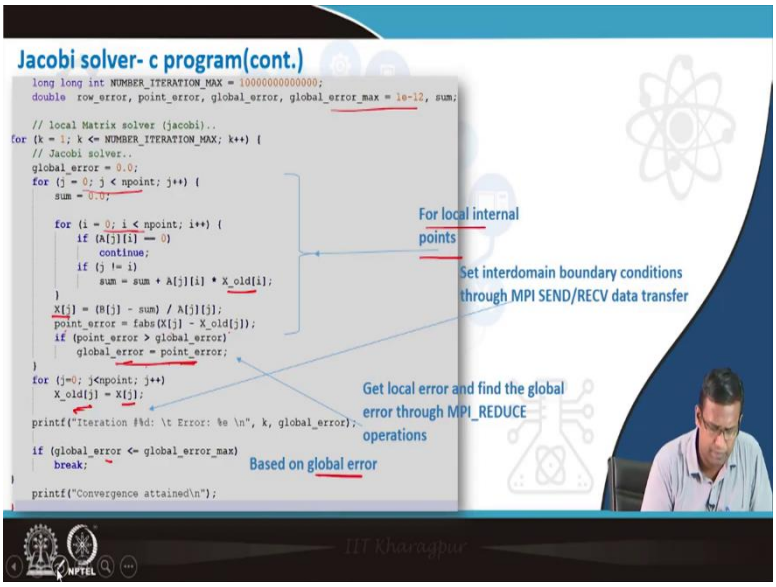
We have to use the p function to get the vector location from the index location. So, we will take the i value 0 for any j. So, this is i = 0, this is i = nx or number of steps in x, this is j = 0, this is j = ny. So, I will send for all the j values i = 0 and the total number of x which is called by the p function is given. The diagonal is 1 and off diagonals are already initialized with 0 when we initialized a matrix and the right-hand side vector also gets the boundary value which is 1.

Similarly, for the rightmost boundary the diagonal is 1 ,off diagonals are already initialized to be 0 and the right-hand side gets 0. For the bottom most boundary, again diagonal is one ,bottom most boundary is identified by j = 0; the diagonal is 1 and the right-hand side is 1. Topmost boundary diagonal is 1, the right-hand side is 0.

So, the boundary conditions are substituted in the A matrix and in the b vector and then we build the main A matrix. We have calculated coeff x is $\frac{1}{(\Delta x)2}$ coeff y is $\frac{1}{(\Delta y)2}$. We have

calculated these values and accordingly we give the diagonal values $-2(\frac{1}{(\Delta x)2}+\frac{1}{(\Delta y)2})$.. The others are $\frac{1}{(\Delta x)2}$, $\frac{1}{(\Delta y)2}$accordingly. So, we have completed defining the interior points of the matrix and the boundary points by diagonal 1 off diagonal 0 and substituted the boundary conditions in the rhs b vector.

(Refer Slide Time: 19:19)



Now, the matrix has been built. This is the Jacobi iteration solver. We say that we can go up to a very large number of iterations we do not need to go because we will check with the tolerance value. Tolerance value is that if the maximum error is less than 10 to the power - 12 it will say that it is the tolerance value and convergence has been achieved. So, do for this large number of iterations, set the global error initially 0, find out this sum A[i][j]* x _old[i] (x_old [i] has already been set to 0).

So, we will find out the sum of A[i][j]* x _old[i] subtract this from the right-hand side vector divided by A[j][j],this is plain Jacobi steps. Find out the error for this point. What is the error? This x is my updated x and this is my old x. The absolute difference between updated and old is the local error and then if the local error is greater than global error, we substitute global error as the point error for the local point and find out what is the local error and then substitute. My new guess will override the old case.

So, x is the new updated value that will be the new guess and the iterations will continue if global error is less than the tolerance value then it will break out and write convergence is

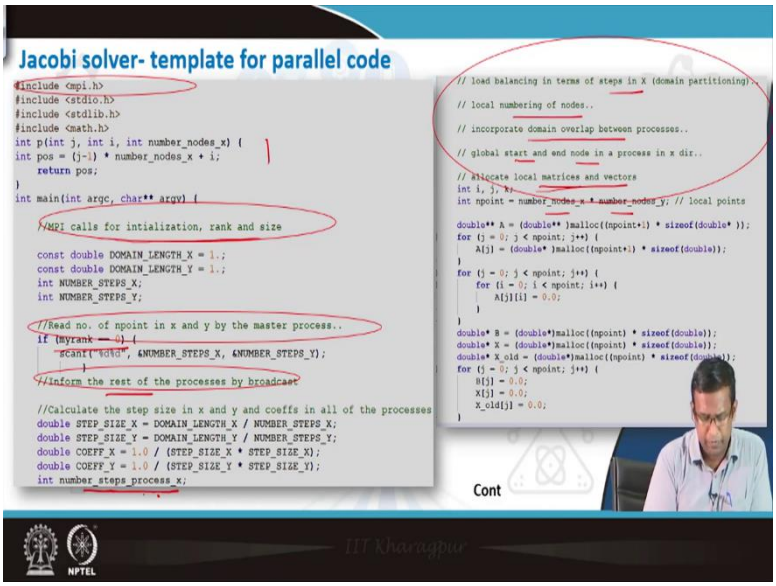
achieved else it will continue with the iterations. Well, what are the scopes of parallelization here? This entire loop looping over the rows and within each row looping over all the column elements of the row this will be done only for the local internal points if we are parallelizing. From point error we are calculating global error. If we have parallelized the program then each processor will give us its own local error. So, these local errors have to be taken and then we have to use MPI reduce to find out which is the maximum of the local errors and that will be a global error.

During the iteration loop we have to also ensure that inter domain boundary conditions are rightly satisfied. So, send receive operations have to be given. Also, when specifying the boundary conditions here we only have specified the physical boundary conditions.

If we do for domain decomposition methods specifying the boundary conditions, we have to do the diagonal 1 ,off diagonal 0 for a coefficient matrix for the inter domain boundary nodes also. In the next class we will look into the actual parallel Jacobi code and we will see that these things are done there. So, global error will be the maximum of all the local errors and based on that global error you have to make a decision whether iterations will be continued or not.

So, what are we doing here? We are not converging Jacobi solver for each of the sub domains independently we. After one Jacob iteration we are interchanging the values and checking whether global convergence has been achieved. So, we are solving for global convergence. We are not solving for independent local convergence, then that is a smarter way because it reduces the number of iterations.

(Refer Slide Time: 23:10)



Now, we look into a template parallel code. This is not the parallel code, but this is a template of the parallel code we have inserted as comments the instructions where parallel constructs can be called and that is the template code.

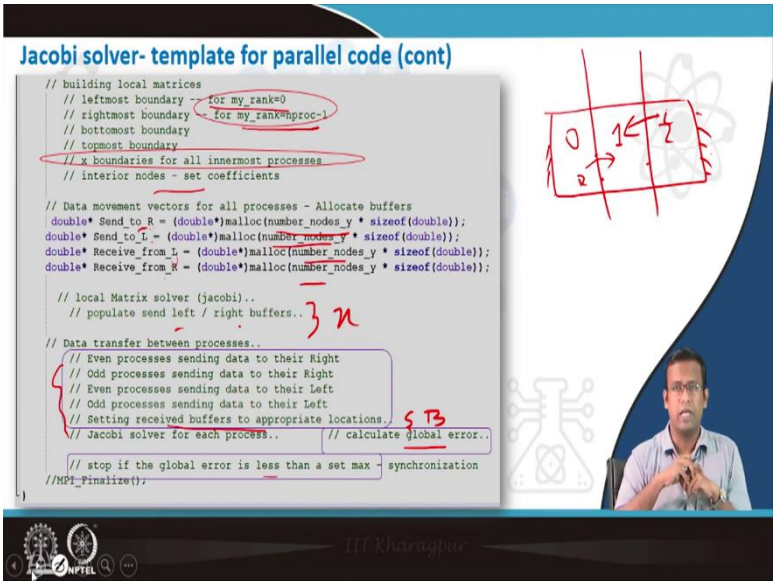
Similarly, the position vector calculation will be there after the program starts .MPI calls will be there for initialization rank and size. Then when you are reading the number of steps in x and y that has to be done for the myrank 0 or for the zeroth processor and then there will be a broadcast it will give it to the global processors. There will be step size calculation for all the processors calculating delta x delta y etcetera.

Then a new variable is included which is num_steps_process_x. In x direction, what is the number of steps in one particular processor? This is the output of the load balancing step. We will do a load balancing in terms of steps in x and can get that what is the number of steps in x in each processor. Based on that you do a local renumbering of the load, incorporate the domain overlap between the processes, identify the global start and end because you need to collect the results finally, and get a continuous solution. Allocate the local matrices based on the domain overlapped.

So, local matrices will be allocated based on the number of steps in the x and steps in the y which is the same for all the processors. Then this allocation and initialization will be for the local matrix and vectors find out that these are the number of points in the local domain and

initialize the bx and x_old. So, these are the important parallel steps here. This locally numbering is an important parallel step here load balancing, domain overlap, etcetera.

(Refer Slide Time: 25:26)



Now, start building the local matrix for the leftmost boundary that belongs to only 1 processor. So, I have divided it into 3 processors. This is 0, rank 0, rank 1, rank 2. The leftmost physical boundary when you will give the boundary condition that belongs to rank 0. Rank 1's leftmost boundary is not a physical boundary and is an inter domain boundary.

Similarly, the rightmost boundary belongs to rank 2, rank 1 rightmost boundary or rank 0's rightmost boundary is not the physical boundary. So, for rank 0 give the left most boundary condition, for rank nproc - 1 (because rank starts from 0 to nproc – 1) give the rightmost boundary condition. Bottom most and topmost boundary conditions are the same.

Then for all the innermost domains what are the x boundary conditions. These x boundaries are to be identified and boundary conditions are to be incorporated in the A matrix; that means, off diagonal 0, diagonal 1. Now, for all the interior nodes you need to set the coefficients for one particular domain.

Then you have to prepare the data transfer buffers and you can see what is the size of the data transfer buffer that is the number of nodes in y. The data that will be sent to the right processor is R? The data that will be sent to the left processor is L this is the left. So, the data that will go in the right and the data that will come in the left.

Similarly, the data that will be received from the left processor ,this data will go to the right processor; that means, the right processor will receive the same data as the left processor. These buffers are allocated then the local matrix solver is called. These buffers are populated at the local matrix solver.

What is in this buffer? This buffer will go and set in the boundary condition. What will go in the buffer? The updated values of x. So, these buffers are populated as the updated values of x and then there is an odd even ordering of the data transfer, so that there is no conflict in the data transfer and then these buffers will be received to the appropriate location.

What is in the same buffer the data that will go and work as the boundary condition for the receiving processor. Where do boundary conditions sit in for any processor? In the b matrix. So, the sent data from x vector and it will go and sit in the b vector; x because it is the updated solution.

This will be the boundary condition for the neighboring sub domain. So, an updated solution will be put into the same data, this will be received by the neighboring sub domain and neighboring sub domain will be put in the b vector to ensure continuity and then MPI finalize will be there.

These are the important steps here and you have to calculate the global error and the program will stop or the iterations will stop if the global error is less than a maximum set value. You need to do a synchronization here also and go for the MPI finalize. Now, based on this template in the next lecture we will develop a working parallel code for domain decomposition of Jacobi solver and we will also look into its parallel efficiencies and the and will also see if the results are right.