

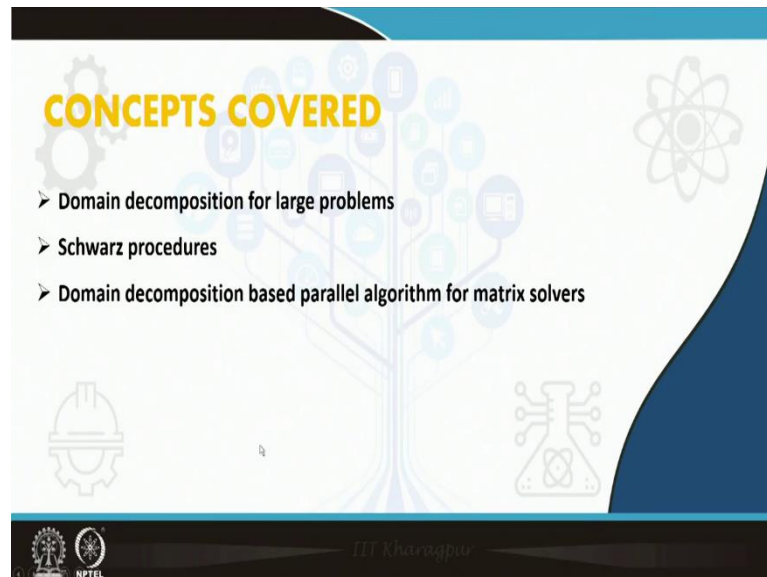
High Performance Computing for Scientists and Engineers
Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Module - 03
Message Passing Interface (MPI)
Lecture – 25
Domain Decomposition Technique

Hello everybody, welcome to the class of High-Performance Computing for Scientists and Engineers. In today's class, we will discuss about Domain Decomposition Technique. We are in our module 3, Message Passing Interface. If you remember in the previous discussion, we are discussing about matrix solvers and we have established the fact that many scientific computing problems involve solution of matrix equations.

So, today we will see a method by which we can parallelize the matrix solvers or we can take a scientific computing problem which involves matrix solvers and parallelize it.

(Refer Slide Time: 01:04)

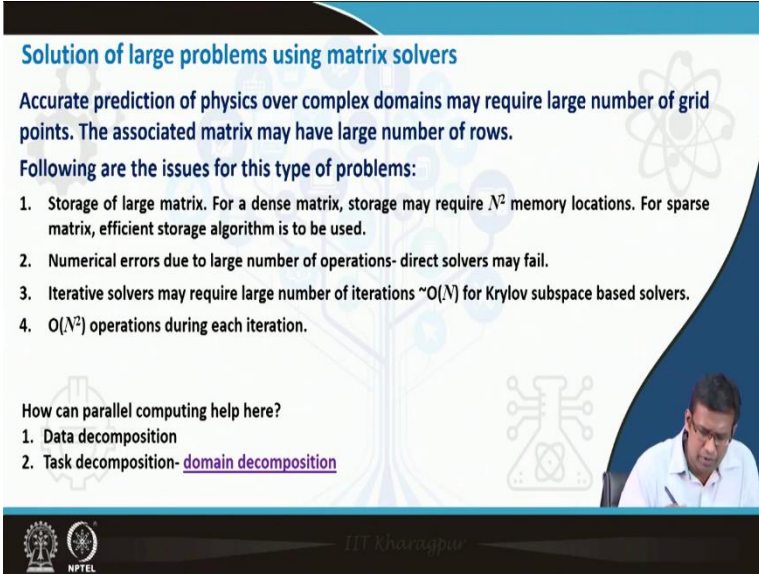


Domain decomposition is heavily used in scientific computing problems involved. In computational mechanics, computational fluid dynamics, computational physics, computational astrophysics we see people doing domain decomposition. Domain decomposition is suitable mostly for distributed memory systems, that is why MPI; when

you learn MPI program, it is and specially from the perspective of learning scientific computing, it is essential that you know the domain decomposition technique.

So, today we will start discussing about domain decomposition technique for the large problems. We will see mathematical procedures which is known as Schwarz procedure, and then we will see how this can be used to get design parallel algorithm for matrix solvers.

(Refer Slide Time: 02:03)



Solution of large problems using matrix solvers

Accurate prediction of physics over complex domains may require large number of grid points. The associated matrix may have large number of rows.

Following are the issues for this type of problems:

1. Storage of large matrix. For a dense matrix, storage may require N^2 memory locations. For sparse matrix, efficient storage algorithm is to be used.
2. Numerical errors due to large number of operations- direct solvers may fail.
3. Iterative solvers may require large number of iterations $\sim O(N)$ for Krylov subspace based solvers.
4. $O(N^2)$ operations during each iteration.

How can parallel computing help here?

1. Data decomposition
2. Task decomposition- domain decomposition

The slide features a blue header, a white body with a blue tree-like graphic, and a video inset of a man speaking in the bottom right corner. Logos for IIT Kharyappa and NPTEL are at the bottom.

We have discussed in last class, physics-based problem where we get a governing equation in form of a differential equation or rate equation can be posed as a matrix solution problem. We have to solve a large matrix equation, that is the numerical equivalent of solving a differential equation for physics-based problem.

So, accurate representation of physics over complex domains may require large number of grid points; because in order to get accurate solution, the approximation terms must be good and the errors or the omitted terms from the Taylor series expansion must be very small. So, we need to ensure that the distance between the points are small which can be again established by using a large number of points.

If we use large number of points, the accuracy will be good; but we will get matrices with very large number of rows or large size. So, we will get a large matrix problem if we try to accurately solve a physics-based problem and the following will be the issues for these

problems that, storage of large matrix; if it is a dense matrix, if there are n rows, for storing we have to store N square memory locations. If we have a million by million matrix, we have to solve 10 to the power 16 square, 10 to the power 12 elements which is large; we will find it difficult to fit in a ram, and specially while accessing the matrix, the cache miss will be very high. So, we need to find out some efficient storage.

Specially, in many cases we get sparse matrices. If you remember sparse matrices, there are only few nonzero numbers in a row and most numbers are 0 . So, you do not need to store the 0 s. You only store the non-zeros; but you need to use a mapping, so that when you are storing the non-zero numbers in the main matrix, they sit in the right place. You need an efficient storage algorithm and again the cache issues have to be looked upon when you are storing that.

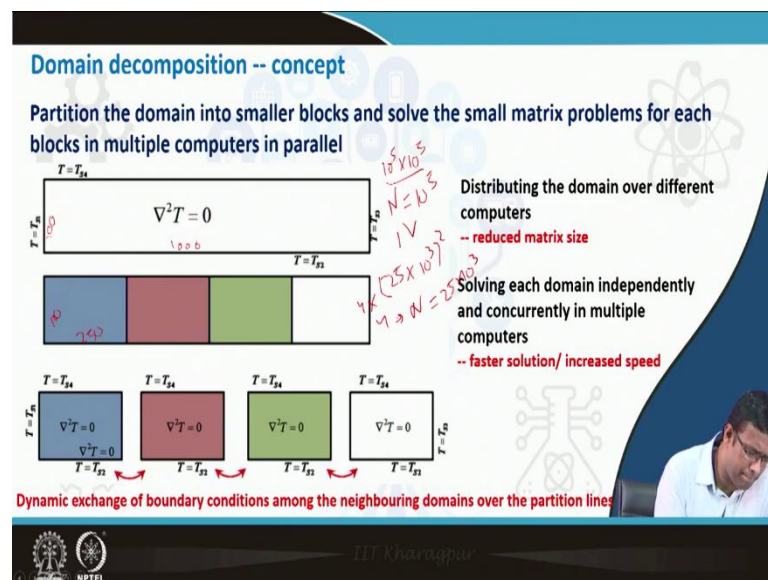
So, one will be the storage issue; because for a large matrix, the number of elements is very large. The next issue will be; if the number of rows is high and the number of elements in each row will be similarly high, therefore the numerical errors will be high. Numerical errors are mainly round off error; you get an irrational number or get something like a recurring decimal, get some number which does not have finite decimal space, after certain decimal numbers, you truncate the number, you round it off and that also gives you some error. If you have large number of problems, these errors pile up and you get high amount of error, and specially the direct solve matrix solvers fail in most of the cases for large matrices due to this round off error, and sometime we get numerical instability issues also due to round off errors.

Iterative solvers may require large number of iterations, in case of Krylov subspace-based solvers like conjugate gradient; the number of iterations of the order of N , N is the number of rows. In case of basic iterative methods like Gauss Seidel, Jacobi or even SOR; the number of iterations is much high. So, if you are not using direct solvers to avoid round off error; using iterative methods the number of iterations is high. Also, in each iteration almost N square operations are required; because each iteration requires a matrix vector product. So, total number of operations are very high in case of large matrix solvers. So, we have to do parallel computing. Well, how can we do parallel computing here? One way of doing parallel computing is data decomposition, which we have done for the openMP type of problem, and we will see some examples of data decomposition for MPI based parallelization also in some of the later class.

Data decomposition means, you have a large matrix vector multiplication matrix which has to be multiplied with a vector. Ask each of the processor to take care of few rows of the matrix and finally, combine that. The other one is a task decomposition. What we will do in domain decomposition? We will take a large domain over which many points are there, which gives us a large matrix; we will decompose the domain into small sub domains and ask different processors to look into different sub domains.

But we have to see that how we can do that, we have a large matrix problem; but each small sub domain will give me a small matrix problem, how this can be related?

(Refer Slide Time: 07:07)



Well, partitioning the domain into smaller blocks and solve small matrix problem for each block in multiple computers in parallel, that is the essence of domain decomposition.

You are solving Laplacian of T or conduction equation over a large Cartesian domain; you decompose it into four small sub domains; get small matrix problems in each sub domain and solve them. While solving them you need to ensure continuity of the solutions; so, you get the boundary conditions by dynamically exchanging the values across the domains.

So, value of this domain will act as its boundary condition; dynamically exchange the boundary conditions among the neighbouring domains over the partition lines. This is in a nutshell what is domain decomposition method.

But one question will come that, how can we be sure that this gives us right solution? In actual problem we are changing it to different sub problems, including the boundary value exchange, introducing new iterations; because we will of course will not get it directly. So, we have to use iterations to ensure continuity across the boundaries; how can we do that? We will, look into that.

Well, let us see the advantages; first is that once we distribute the domain over different computers, we reduce matrix size. Say, we can understand that as the number of points are small, the matrix in each subdomain will be smaller. So, instead of storing a large matrix, we are storing many small matrices; but overall matrix size will also reduce.

Let us see that we have say 1000 points here and 100 points here. So, the matrix size will be 10^5 into 10^5 , $N = 10^5$. But now we get 250 here and 100 here; so, we get 250 into 100 is 25 into 10^3 .

The total number of elements in each matrix will be 25000^2 into 4. This is much smaller than this number. So, there will be 4 matrices of $N = 25$ into 10^3 . The overall matrix size is also small, that is one big advantage.

The next advantage is of course parallelization that, each domain can be independently and concurrently solved in multiple computers; only iterative methods can be used for that, because you cannot directly solve this while exchanging the boundary conditions, you need iterations to ensure continuity.

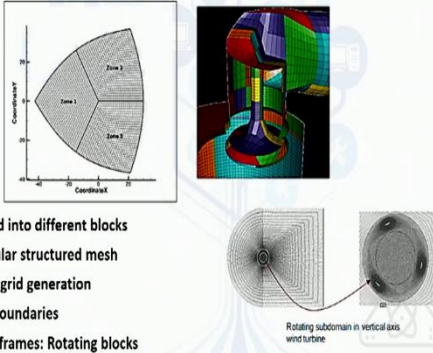
So, when you are using multiple computers; only after one iteration you are exchanging data across computers, but at any point of time four processors are active and therefore this is parallelized. The solution will come fast and you will you can get increased speed. So, you can get better speed in terms of crunching numbers, in terms of solving equations; because four parts of the main matrix is being solved in four different processors.

Also, the data in each processor is smaller than the large matrix and the total data is also smaller, than if we have to store the main large matrix. So, this in terms of the matrix size, it is also helpful.

(Refer Slide Time: 11:06)

Grid blocks for domain decomposition

For a regular Cartesian domain, decomposing it along x, y-z planes is relatively simple
However, it might be important to obtain domain decomposition for complex geometries



- Geometry is divided into different blocks
- Each block has regular structured mesh
- Complex geometry grid generation
- Mapping at block boundaries
- Multiple reference frames: Rotating blocks

Rotating subdomain in vertical axis wind turbine

NPTEL

We have seen the previous example where you have a Cartesian domain and it is very simple exercise to make partitions. But in complex problems the domains are not Cartesian; they are curvilinear, the geometry looks much complex.

So, doing getting domain decomposition might be little difficult, there are certain way out for that. In case of a complex geometry or of an industrial problem; say turbine, blades rotating inside a casing if you have to do domain decomposition, one idea is that the divide the geometry into different blocks and then inside each block you generate a mesh.

So, each block will be a domain and, in this way, a complex geometry can be well handled and the block boundaries are to be mapped correctly. So, which block is neighbour of which one and which block will supply data to exchange data with which neighbouring block that has to be ensured. In case you have multiple reference frame, like one block is rotating another is fixed; one part of the geometry is rotating, another is fixed etcetera, you can use multiple reference frame and you can put different reference frame with different blocks and model that geometry.

So, domain decomposition can be extended for complex geometry problems also; but even for completely unstructured problem, there are graph-based partitioning by which one can get domain decomposition.

(Refer Slide Time: 12:39)

Classification of domain decomposition

- **Non-overlapping domain decomposition**
 - The sub-domains intersect only on their interface
- **Overlapping domain decomposition methods**
 - The sub-domains overlap by more than the interface

May be used to ensure higher level continuity of solutions

The slide features a blue header with the title 'Classification of domain decomposition'. Below the title, there are two bullet points. The first bullet point is 'Non-overlapping domain decomposition' with a sub-bullet 'The sub-domains intersect only on their interface'. Below this is a diagram showing two adjacent rectangular domains, each divided into four vertical sub-domains. The second bullet point is 'Overlapping domain decomposition methods' with a sub-bullet 'The sub-domains overlap by more than the interface'. Below this is a diagram showing two overlapping rectangular domains, each divided into four vertical sub-domains. A red outline highlights the overlapping region. To the right of the diagrams, there is a text box that says 'May be used to ensure higher level continuity of solutions'. In the bottom right corner, there is a small video inset of a man speaking. The bottom of the slide has a black bar with the IIT Kharagpur logo and the text 'IIT Kharagpur'.

There are standard techniques for that. Now there are two types of domain decomposition; one is non overlapping domain decomposition that is one domain ends, another domain starts from there, the domains just touch each other. The other is overlapping domain decomposition, the domains overlap certain points; one domain is ending here, the neighbouring domain is starting before that. Using this we can ensure higher level of continuity, C^2 , C^3 level of continuity; because a greater number of points are same in between each domain and they have same values. Each point should have a unique value, whichever domain it belongs to. So, therefore, overlapping and non overlapping domain decompositions are there.

(Refer Slide Time: 13:35)

Domain decomposition for a matrix problem

Distribute the domain into several subdomains

Form the matrix equation for each subdomain using the inter-boundary domain values

Propose an algorithm to solve each domain independently

Transfer inter-domain solutions to obtain continuity across the boundaries!

Converge to a final solution satisfying equations over subdomains including inter-domain boundaries

$\Omega = \bigcup_{i=1}^3 \Omega_i$

$\Omega_1, \Omega_2, \Omega_3$

Γ_{12}, Γ_{13}

NPTEL

Dr. Khanna

We will discuss about overlapping domain decomposition later; because it helps us to ensure continuity of the solution.

We come to a matrix problem. So, you have a L shaped geometry over which solving some differential equation, which is laid as a matrix equation and we are solving it here. The entire domain is given by Ω and $\Omega_1, \Omega_2, \Omega_3$ are the small sub domains obtained through the domain decomposition, and Γ_{13} and Γ_{12} are the inter domain boundaries.

So, the main domain is distributed into several sub domains. Now, this can be overlapping, can be non-overlapping; here it is shown as non-overlapping, we will see later an example of overlapping domains.

Form the matrix equation in each sub domain using the inter domain boundary values. Consider Γ_{12} for Ω_1 as the boundary condition, which is coming from the solution within the sub domain Ω_2 , and while forming the matrix equation for this sub domain, use this boundary conditions.

So, what will be the next step? That we will solve this matrix equation, here you solve it for, here you solve it for here; then re exchange the boundary conditions and iterate till you get convergence. So, propose an algorithm to solve each domain independently; this is a matrix solver algorithm, you can use something like Jacobi, conjugate gradient biconjugate gradient, depending on your requirements and certain conditions of the matrix.

Solve each domain independently. Transfer the inter domain condition solutions to obtain continuity across boundaries; this is an important step and this will ensure that you are basically solving a large problem. This is a data transfer process and therefore, you will require message passing interface calls, send and receive calls for this. We will later see that we have spent times on MPI based communication, they will find their application in this particular step.

Now do this iteratively; because in one go you cannot ensure that the solutions have continuity across the domain boundaries, so the final solutions satisfying equations over all sub domains including the inter domain boundaries; the final solution satisfy everything.

(Refer Slide Time: 16:18)

Matrix solution in the decomposed domain

Solve: $\nabla^2 u = f$ in $\Omega = \bigcup_{i=1}^3 \Omega_i$

with boundary conditions: $U = U_{BC}$ on outer domain boundary Γ

The resulting matrix equation: $Au = b$

Let x Σx_i be the solution at the domain-internal points and y be the solution in the inter domain boundaries Γ_{ij} . Then $Au = b$ can be written as:

$$Au = b \Rightarrow \begin{bmatrix} B_1 & & \\ & B_2 & \\ & & B_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} + \begin{bmatrix} E_1 \\ F_2 \\ F_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$u = \begin{bmatrix} x \\ y \end{bmatrix}$

NPTEL

Say, we are solving $\nabla^2 u = f$ in the domain. The boundary conditions are $U = U_{BC}$ given at the outer domain, some boundary condition is specified, the outer domain boundary Γ .

Now, if you write this as a single matrix problem with the boundary conditions, you will get the matrix equation $Au = b$. We have to be sure that what we are doing domain decomposition is right; otherwise it will not converge, it will give us wrong result or it might not give us end result. So, let us see mathematically what is happening here.

Now, x is the solution, in each of the sub domain the solution is, x_1, x_2, x_3 , they are the solutions in the sub domain. And in the inter domain boundary, there are also points on

the inter domain boundary, few points lie on the boundary, the solutions are y . So, $\sum x_i$ be the solution at the domain internal points and y is the solution in the inter domain boundary. Then this equation $A u = b$ can be written as

$$\begin{bmatrix} B_1 & & E_1 \\ & B_2 & E_2 \\ & & B_3 & E_3 \\ F_1 & F_2 & F_3 & C \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ g \end{pmatrix}; u = \begin{pmatrix} x \\ y \end{pmatrix}. \quad A u = b \text{ will give us block matrices for}$$

the three domains, and few matrices for the inter domain boundaries.

So, $A u = b$ will can be written as these three block matrices plus the block matrix for the inter domain boundary and the matrix which connects the intra domain point, the within domain point with the domain internal point with the inter domain boundary E_1, E_2 and E_3, f_1, f_2, f_3 . So, this is basically representation of $A u = b$ in the domain decompose method and A matrix is nothing, but these elements.

(Refer Slide Time: 18:29)

Matrix solution in the decomposed domain (cont)

$$Au = b \Rightarrow \begin{bmatrix} B_1 & & E_1 \\ & B_2 & E_2 \\ & & B_3 & E_3 \\ F_1 & F_2 & F_3 & C \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ g \end{pmatrix}$$

or,

$$Bx + Ey = f \quad \text{and} \quad Fx + Cy = g$$

$$x = B^{-1}(f - Ey)$$

So, $FB^{-1}(f - Ey) + Cy = g$

$$(C - FB^{-1}E)y = g - FB^{-1}f$$

S (Schur Component)

$$\Rightarrow Sy = g - FB^{-1}f$$

$$\Rightarrow y = S^{-1}(g - FB^{-1}f)$$

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \Rightarrow \begin{pmatrix} B & E \\ F & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \text{ or } Bx + Ey = f \text{ and } Fx + Cy = g \Rightarrow x = B^{-1}(f - Ey)$$

$$\text{So, } FB^{-1}(f - Ey) + Cy = g \Rightarrow (C - FB^{-1}E)y = g - FB^{-1}f$$

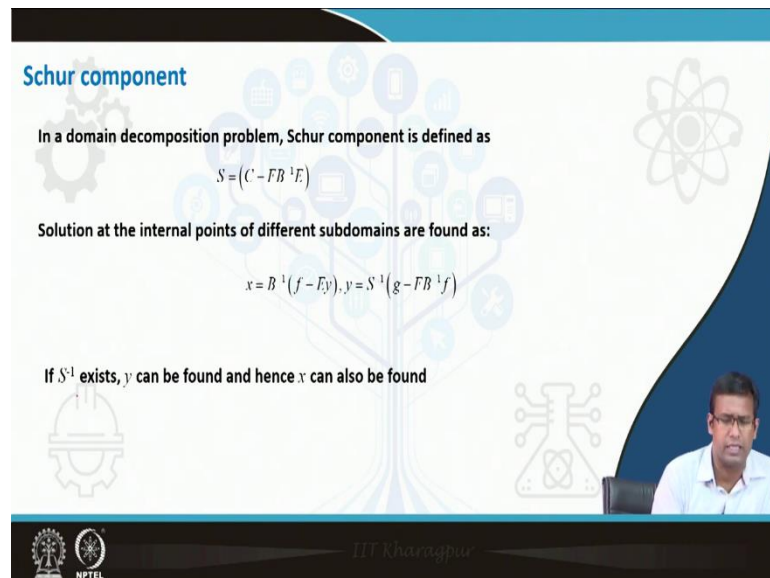
$(C - FB^{-1}E) = S$ is known as Schur component

$$\Rightarrow Sy = g - FB^{-1}f$$

$$\Rightarrow y = S^{-1}(g - FB^{-1}f)$$

This is a point which is not parallelizable, rather which is a synchronization part which involves multiple processor output from multiple sub domains.

(Refer Slide Time: 21:00)



Schur component

In a domain decomposition problem, Schur component is defined as

$$S = (C - FB^{-1}E)$$

Solution at the internal points of different subdomains are found as:

$$x = B^{-1}(f - Ey), y = S^{-1}(g - FB^{-1}f)$$

If S^{-1} exists, y can be found and hence x can also be found

NPTEL IIT Kharagpur

So, in a domain decomposition problem, this is the Schur component; solution of the internal points are found that

$x = B^{-1}(f - Ey), y = S^{-1}(g - FB^{-1}f)$. If S inverse exists, y can be found and hence x can also be found.

(Refer Slide Time: 21:20)

Domain decomposition parallelization

To solve: $x = B^{-1}(f - Ey), y = S^{-1}(g - FB^{-1}f)$

B is a block diagonal matrix,

$Au = b$

$$\Rightarrow \begin{bmatrix} B_1 & & & E_1 \\ & B_2 & & E_2 \\ & & B_3 & E_3 \\ F_1 & F_2 & F_3 & C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ y \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ g \end{bmatrix}$$

Hence, inverse of B can be found in a decoupled sense as disjoint processes.

Hence the set of equations given can be solved, provided y is made available to the particular process.

So, one focus in doing a domain decomposition is that, we should do domain decomposition in such a way that this Schur component is invertible and the standard techniques which will show as Schwarz technique ensures that.

So, the domain decomposition problem of solving $Au = b$ will be finding solving these two equations x and y , this is B . Hence, the inverse of B can be found in decoupled sense as disjoint process; because B is combination of the block diagonal matrices. Hence, the set of equations can be solved, provided y is made available to the particular process.

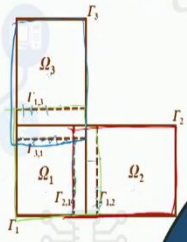
So, if y is known a priori say, we guess the values of y and, each of these vectors can be solved and we can find out the x 's. Now based on the x 's, we have to again guess the next iterated value of y and then we can update the x 's that is the method.

(Refer Slide Time: 22:26)

Schwarz multiplicative procedure for overlapping domains

Solve Dirichlet problem on one domain in each iteration and consider boundary conditions based on the most recent solution of the other domains. This is established through boundary overlaps.

Algorithm



The diagram shows three overlapping rectangular domains: Ω_1 (bottom-left), Ω_2 (bottom-right), and Ω_3 (top-left). The boundaries are labeled Γ_1 , Γ_2 , and Γ_3 . The overlapping regions are labeled Γ_{13} , Γ_{12} , and Γ_{23} . A person is visible in the bottom right corner of the slide.

NPTEL

Dr. Khazanchi

This is formalized as Schwarz multiplicative procedure and this will do for overlapping domain. Overlapping domain means, this is my main domain which is divided into these three sub domains; $\Omega_1, \Omega_2, \Omega_3$. Omega 1, this sub domain actually ends here, this is Ω_1 . Similarly, the sub domain Ω_3 ends here. This is my omega 1; this has an overlap here; this has an overlap here.

Ω_1 has overlapped with 3 and 2; this is my omega 2, I use another pen here, this is my Ω_2 , this is the overlap with Ω_1 , and this is my Ω_3 which is overlap with Ω_1 also and the boundaries are given similarly. So, when we will solve Ω_1 , we will solve up to these points.

Now, the boundary condition for Ω_1 which lies in domain 2 will come from the solution of the sub matrix problem in domain 2. Now once we solve this similarly once Γ_{13} the boundary condition for Ω_1 which is at Ω_3 will come from its solution.

Now, look what we will do? Once we solve for Ω_1 , we will update along these lines and this line; this will be boundary condition for Ω_3 and this particular line will be boundary condition for Ω_2 and we will solve it.

So, again while solving this we get the updated values at this line and this line. So, this will be boundary condition for Ω_1 and an entire thing can be done parallelly; that means,

every time you use the last updated value like a Jacobi iteration at the boundaries and solve all the domains and update the boundary values. Boundary value for one sub domain is an internal point for another sub domain. So, from the solution of the neighbouring sub domain update the boundary values of the next of the neighbouring sub domain or constant sub domain. Well, so let us solve Dirichlet problem; because boundary is known it is a Dirichlet problem; already the boundary condition is given.

On one domain in each iteration and consider boundary conditions based on the most recent solution of the other domain. This is established through boundary overlap.

(Refer Slide Time: 25:10)

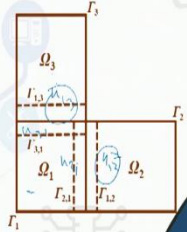
Schwarz multiplicative procedure for overlapping domains

Solve Dirichlet problem on one domain in each iteration and consider boundary conditions based on the most recent solution of the other domains. This is established through boundary overlaps.

Algorithm

1. Chose an initial guess u to the solutions
2. Iterate until convergence
3. For $i = 1, \dots, s$ *parallel*
4. Solve $B_i x_i = f_i - E_i y$ in Ω_i with $u = u_{ij}$ in Γ_{ij}
5. Update u_{ij} values in Γ_{ij}
6. Till convergence in all $\Omega_1, \Omega_2, \dots, \Omega_s$

The algorithm sweeps through the s subdomains and solves the original equation in each domain based on the boundary conditions that are updated from the most recent values of u .



NPTEL

The algorithm is: chose an initial guess u to the solution which includes the inter domain boundary points as well as the internal points.

Iterate until convergence; keep on doing that till you see that there are certain changes and once these changes are not there, it has converged. The previous iteration and last iterations solution have very small difference, we have discussed about convergence in our last lecture.

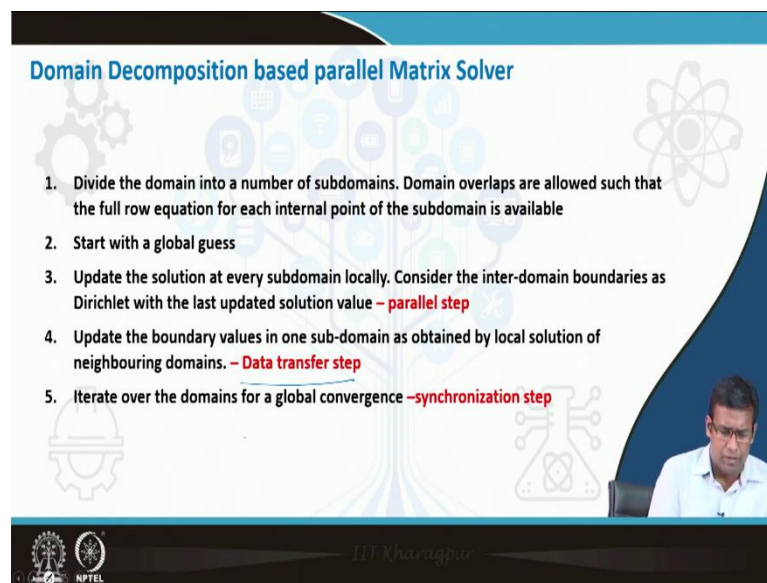
For $i = 1, \dots, s$ there are s sub domains. So, solve $B_i x_i = f_i - E_i y$; the local problem inside each of the sub domains, with $u = u_{ij}$ in Γ_{ij} . So, say if you are solving for block one, you know that u_{13}, u_{12} are the boundary conditions here.

If you are solving for Ω_3 , you know that u_{31} is the boundary condition here; if you are working solving for Ω_2 , you know that u_{21} is the boundary condition here. So, using this boundary condition, solve for each of these domains and this is a parallel step; you can do this in parallel, in different processors.

Update the values: once you have solved this you update these values; once you have solved for Ω_3 you update this, once you have solved for Ω_2 , update these values. So, once you are solving within the particular subdomain, you update for the boundary conditions, after the solution update the boundary conditions of the neighbouring sub domains. Till convergence you solve this in all the sub domains. This algorithm sweeps through s sub domains and solve original equation in each domain based on boundary conditions that are updated from the most recent values of u.

It is shown that this is a convergent method, it works exactly like a block Jacobi or block Gauss Seidel algorithm; here, the implementation shows a block Jacobi algorithm and within the finite number of iterations, it converges to the right solution.

(Refer Slide Time: 27:28)



Domain Decomposition based parallel Matrix Solver

1. Divide the domain into a number of subdomains. Domain overlaps are allowed such that the full row equation for each internal point of the subdomain is available
2. Start with a global guess
3. Update the solution at every subdomain locally. Consider the inter-domain boundaries as Dirichlet with the last updated solution value – **parallel step**
4. Update the boundary values in one sub-domain as obtained by local solution of neighbouring domains. – **Data transfer step**
5. Iterate over the domains for a global convergence – **synchronization step**

The slide features a background with faint icons of gears, a network, and an atom. A small video inset in the bottom right corner shows a man in a white shirt. The bottom of the slide has logos for IIT Kharagpur and NPTEL.

So, based on this we can write down that, these are the essential steps for domain decomposition-based parallelization of a matrix solver.

1. Divide the domain into a number of sub domains. Domain overlaps are allowed such that the full row equation for each internal points of the sub domain are available.

So, how the overlaps are given? Overlaps are given in a sense that, if you are solving within the last points in that particular sub domain, there should be right boundary conditions, so that each of the row equations of that sub domain is solvable. The points in the neighbouring sub domain will be considered, so that you can complete the equations, you can get all the points for writing out writing down the equations of all internal points in the sub domain. We will discuss about it later from perspective of Jacobi iteration of Laplace equation we can see that. We take points in another sub domain, so that the equation of each row of the internal points of one sub domain gets sufficient number of boundary points, the boundary conditions are sufficiently defined. In that way you allow the domain overlaps.

2.Start with a global guess. This is very important; if we start with local guesses, we can find sometime it might not converge even. Start with the global guess, so that initially there is a continuity of the solution. The continuity at the domain boundaries should be there from the initial step and then only it will converge; convergence is not unconditional; it has to start from a global guess that is the main condition of the convergence. Otherwise, you will get Schur component which is not invertible; there can be certain cases like this.

3.Update the solution at each subdomain locally. Consider inter-domain boundaries as Dirichlet with the last updated solution value. So, each sub domains the internal points are updated locally; that means each subdomain is solving the matrix equation locally. There is a small block matrix associated to each sub domain and each sub domain is solving it locally. Now, this can be parallelized. If there are hundred sub domains, there are hundred processors which can look into each of the sub domain as allotted to it and solve the governing equations there. Because this solution is based on the number of internal points, the solution will take at least 10 order of n iterations and n square operation. So, this solution is costly. Now, this is the part which will be executed in multiple processors in parallel and this is a disjoint part; each of this can be done concurrently, one processor does not need to know about what the other processor. Once this update is done; then do the data transfer step.

4. Update the boundary values in one sub domain as obtained by the local solution of the neighbouring domains. As the neighbouring domains are updated, the boundary points which of which is belonging to this particular sub domain, get it updated after the one

iteration in the neighbouring domain or all the domains you do the boundary exchange. This is the data transfer step associated here.

5. Iterate over the domains to get a global convergence: This will require a synchronization; that after the iterations see whether your global convergence has been achieved else you still continue iterations.

(Refer Slide Time: 31:34)



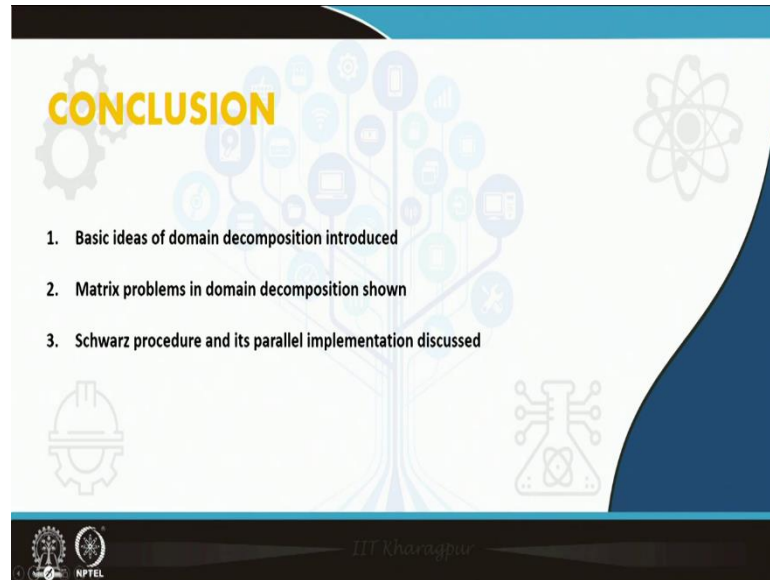
We will take a Jacobi solver for Laplacian of $T = 0$; we look into the sequential program, simple Jacobi solver. We will identify the steps that where we can distribute in multiple sub domains, that is the load balancing step each subdomain will go to a processor, and then we will see that how we the parallelization can be done that each subdomain we will get a small problem and solve it independently.

Then we will see about the data transfer that after independent iterations in each sub domains, there will be exchange of the boundary values across the sub domains and then synchronization. We will take a Jacobi solver, we will identify the locations in which parallel constructs can be brought into the Jacobi solvers, and I will show you the parallel version of the Jacobi solvers developed through this exercise. Then we will see what are the performance of this course, especially the speed up and efficiency of the programs.

So, these are the references William Gropp's MPI book, Gramm, Gupta, Karypis Kumars MPI book, Quinn's MPI and OpenMP C programming book and Yusuf Saad's book which

I discussed in last class, Iterative methods for sparse linear systems specially to get the mathematical part Schur procedure and Schur component; you can go through Yusuf Saad's book, part of this book is available online where you will get the relevant material.

(Refer Slide Time: 33:09)



This class we looked into basic ideas of domain decomposition, we look into how this can be applied over a matrix problem and what are its mathematical foundations, what is Schwarz procedure and what are the steps in parallel implementation.

So, in the next class we will take a sequential Jacobi solver and look into its domain decomposition-based parallelization. We will do this exercise through these lectures only.