18.337 Parallel Scientific Computing	March 1995
Lecture 12	
Topic: Domain Decomposition	*1995 Scribe:*

Domain Decomposition Notes

(These notes need substantial work)

Domain decomposition is a term used by at least two different communities. Litterally, the words indicatie the partitioning of a region. As we will see later, an important computational geometry problem is find good ways to partition a region. In scientific computing, domain decomposition refers to the technique of solving partial differential equations using subroutines that solve problems on subdomains. The notion, already useful on serial computers, takes on a greater importance when one considers a parallel machine with, say, a handful of very powerful processors. In this context, domain decomposition is a parallel divide-and-conquer approach to solving the PDE.

To guide the reader, we quickly summarize the main issues that arise in the domain decomposition literature.

Issues in Domain Decomposition :

1. **To overlap regions or to not overlap regions** Figures 1 and 2 indicate regions on whcih one might wish to solve a PDE. In Figure 1, we have a non-overlapped situation. Methods to solve such problems are known as *substructuring*. These methods tend to focus primarily on solving the problem on the interface between the two regions. In Figure 2, we show a region with overlap. Methods for solving the problem with overlap are known as *Schwarz* methods. Methods for solving problems when there is overlap are often viewed as block Jacobi or block Gauss-Seidel procedures.

2. **Continuous vs Discrete** The discrete domain decomposition algorithms have fairly easy to understand continous analogs. When regions overlap, there is the method due to Schwarz in 1870 who suggested an alternating procedure for solving a PDE on a large domain by solving smaller problems on subdomains and iterating. In figure 2, one alternately solves the problem on the square, then the circle, using any initial guess, and then using the solution of previous sub-problem as boundary data for the next.
This method does not work when regions do not overlap since an arbitrary choice of data on the interface would not be updated, and the solution would not be differentiable along the interface. Indeed one can define an affine linear map T from data defined at the interface to the jump in the derivative at the interface. This map is the Steklov-Poincaré operator. The solution to $Tu = 0$ on the interface yields a decoupling of the problem. Discrete versions require the solution of a linear equation on the interface with decouples the problem.

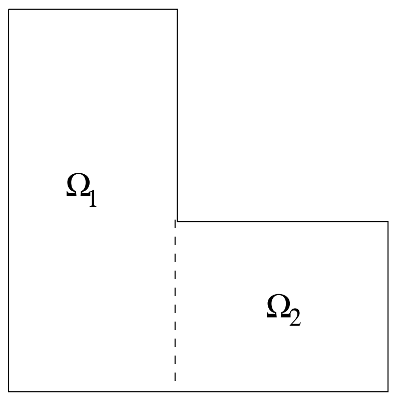


Figure 1: Domain divided into two subdomains without overlap

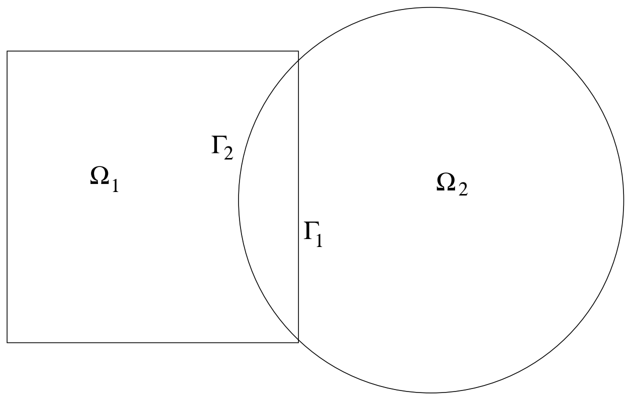


Figure 2: Example domain of circle and square with overlap

3. **Discretization: Finite Differences or Finite Elements** Finite differences is actually a special case of finite elements, and all the ideas in domain decomposition work in the most general context of finite elements. In finite differences, one typically imagines a square mesh. The prototypical example is the five point stencil for the Laplacian in two dimensions. An analog computer to solve this problem would consist of a grid of one ohm resistors. In finite elements, the protypical example is a triangulation of the region, and the appropriate formulation of the pde on these elements.

4. Matching vs Non-matching grids. When solving problems as in our square-circle example of Figure 2, it is necessary to discretize the interior of the regions with either a finite difference style grid or a finite element style mesh. The square may be nicely discretized by covering it with Cartesian graph-paper, while the circle may be more conveniently discretized by covering it with Polar graph-paper. Under such a situation, the grids do not match, and it becomes necessary to transfer points interior to Ω_2 to the boundary of Ω_1 and vice versa.

5. Additive vs Multiplicative Approaches - The alternating procedure that we described does not seem to be parallel at the highest level because if processor 1 contains all of Ω_1 and processor 2 contains all of Ω_2 then each processor must wait for the solution of the other processor before it can execute. Such approaches are known as multiplicative approaches because of the form of the operator applied to the error. Approaches that allow for the solution of subproblems simultaneously are known as additive methods. The difference is akin to the difference between Jacobi and Gauss-Seidel.

6. Preconditioning Approaches It seems wasteful to solve subproblems extremely accurately during the early stages of the algorithm when the boundary data is likeley to be fairly accurate. Therefore it makes sense to run a few steps of an iterative solver as a preconditioner for the solution to the entire problem.

7. Multilevel Approaches: These modern approaches are designed to greatly speed convergence by solving the problem on different sized grids with the goal of communicating information between subdomains more efficiently.

Jacobi vs Gauss-Seidel :

To better understand the issue raised in the comparison between additive and multiplicative approaches, let us review the basic classical methods of solving the problem within the subdomains:

1. Jacobi - At step n, the neighboring values used are from step $n-1$

2. Gauss-Seidel - Values at step n are used if available, otherwise the values are used from step $n-1$

3. Red Black Ordering - If the grid is a checkerboard, solve all red points in parallel using black values at $n-1$, then solve all black points in parallel using red values at step n

Analogous *block* methods may be used on a domain that is decomposed into a number of multiple regions. Each region is thought of as an element used to solve the larger problem. This is known as block Jacobi, or block Gauss-Seidel.

1. Gauss-Seidel - Solve each region in series using the boundary values at n if available.
2. Jacobi - Solve each region on a separate processor in parallel and use boundary values at $n-1$. (Additive scheme)
3. Coloring scheme - Color the regions so that like colors do not touch and solve all regions with the same color in parallel. (Multiplicative scheme)

was "owned" by specialist in theoretical finite elements. There was also a number of engineering computations that captured the spirit of domain decomposition. More recently, because of the advent of parallel computers, the subject has taken upon a new computational importance, forcing researchers to take into account many practical concerns not considered in the theoretical literature.

We begin by considering the problem of solving $\nabla^2 u = g$ on the domain in Figure 1 consisting of two rectangles. We assume that the boundary values of u are specified along the boundary of the "L." The solution of Poisson's equation on each individual rectangle is well understood, indeed there are many choices of (black box) library software available. Most likely one would use a multigrid, FFT, or a preconditioned conjugate gradient approach on each rectangle. The classical iterative schemes (Jacobi, Gauss-Seidel, SOR) would probably converge much slower. One might consider a solver that allows for the boundary conditions to be either Dirichlet (solution known on the boundary) or Neumann (normal derivative known on the boundary).

If we have subroutines for the solution of Poisson's equation on subdomains, how might we "glue" together these routines to solve the equation on the entire domain? This is what domain decomposition is all about. One simple approach is to alternate between Ω_1 and Ω_2, solving the problem on Ω_1 using the given boundary conditions on the solid border, and using the normal derivative computed from Ω_2 on the previous iteration as Neumann conditions on the dashed line.

Figure 2 illustrates our second example which is even simpler than our previous example. Here we have the union of the interior of a square and a circle that overlap. Again we would like to solve Poisson's equation $\nabla^2 u = g$, with Dirichlet boundary conditions given on three sides of the square and the external portion of the circle. On a region with overlap, the basic idea is very simple:

Basic Idea - Schwarz alternating procedure (1870)

1. Start with any guess for u_2 on Γ_1.
2. Solve $\nabla^2 u = g$ on Ω_1 by taking $u = u_2$ on Γ_1. (i.e. solve in the square using boundary data from the interior of the circle)

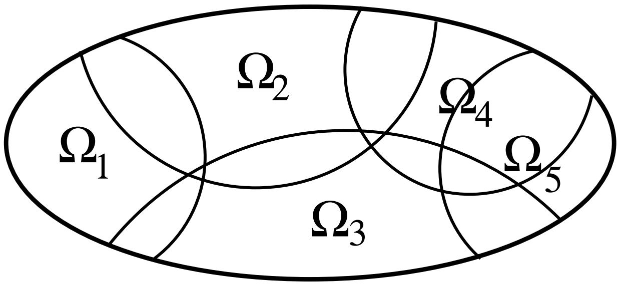


Figure 3: Multiple Regions

3. Solve $\nabla^2 u = g$ on Ω_2 by taking $u = u_1$ on Γ_2 (i.e. solve in the circle using boundary data from the interior of the square)

4. Goto 2 and repeat until convergence is reached

Domain Decomposition as a Preconditioner

The modern approach is to use a step of block Jacobi or block Gauss-Seidel as a preconditioner for use in a Krylov space based subsolver. There is not too much point in solving the subproblems exactly on the smaller domains (since the boundary data is wrong) just an approximate solution suffices → **domain decomposition preconditioning**

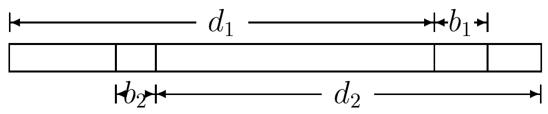
Krylov Methods - Methods to solve linear systems : **A**u=g . Examples have names such as the Conjuate Gradient Method, GMRES (Generalized Minimum Residual), BCG (Bi Conjugate Gradient), QMR (Quasi Minimum Residual), CGS (Conjugate Gradient Squared). For this lecture, one can think of these methods in terms of a black-box. What is needed is a subroutine that given u computes Au. This is a matrix-vector multiply in the abstract sense, but of course it is not a dense matrix-vector product in the sense one practices in undergraduate linear algebra. The other needed ingredient is a subroutine to *approximately* solve the system. This is known as a preconditioner. To be useful this subroutine must roughly solve the problem quickly.

0.1 Algebraic formulation for overlapping domains

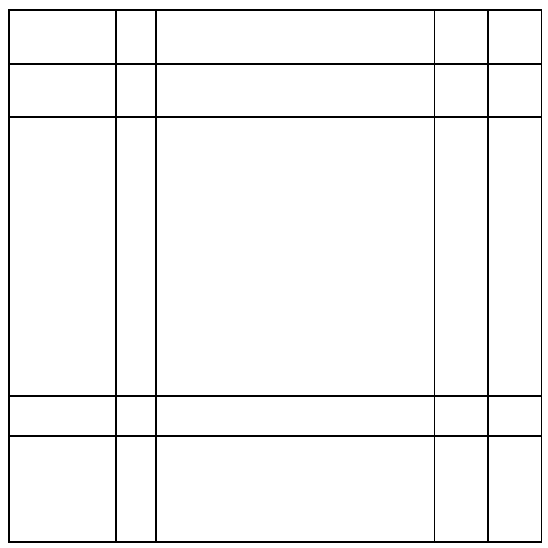
Solve : $\mathbf{A}u = f$ on a domain with overlap, matching grids , and with $u = 0$ on the boundary.

Unfortunately, we have a notational difficulty at this point. It is this difficulty that is probably the single most important reason that domain decomposition techniques are not used as extensively as they can be. Even in the two domain case, the difficulty is related to the fact that we have domains 1 and 2 that overlap each other and have internal and external boundaries. By setting the boundary to 0 we can eliminate any worry of external boundaries. I believe there is only one reasonable way to keep the notation manageable. We will use subscripts to denote subsets of indices. d_1 and d_2 will represent those nodes in domain 1 and domain 2 respectively. b_1 and b_2 will represent those notes in the boundary of 1 and 2 respectively that are not external to the entire domain.

Therefore u_{d_1} denotes the subvector of u consisting of those elements interior to domain 1, while A_{u_1,b_1} is the rectangular subarray of A that map the interior of domain 1 to the internal boundary of domain 1. If we were to write u^T as a row vector, the components might break up as follows (the overlap region is unusually large for emphasis:)



Correspondingly, the matrix A (which of course would never be written down) has the form



The reader should find A_{b_1,b_1} etc., on this picture. To further simplify notation, we write 1 and 2 for d_1 and d_2,1_b and 2_b for b_1 and b_2, and also use only a single index for a diagonal block of a matrix (i.e. $A_1 = A_{11}$).

Now that we have leisurely explained our notation, we may return to the algebra. Numerical analysts like to turn problems that may seem new into ones that they are already familiar with. By carefully writing down the equations for the procedure that we have described so far, it is possible to relate the classical domain decomposition method to an iteration known as *Richardson iteration*. Richardson iteration solves $Au = f$ by computing $u^{k+1} = u^k + M(f - Au^k)$, where M is a "good" approximation to A^{-1}. (Notice that if $M = A^{-1}$, the iteration converges in one step.)

The iteration that we described before may be written algebraically as

$$A_1 u_1^{k+1/2} + A_{1,1_b} u_{1_b}^k = f_1$$

$$A_2 u_2^{k+1} + A_{2,2_b} u_{2_b}^{k+1/2} = f_2$$

Notice that values of $u^{k+1/2}$ updated by the first equation, specifically the values on the boundary of the second region, are used in the second equation.

With a few algebraic manipulations, we have

$$u_1^{k+1/2} = u_1^{k-1} + A_1^{-1}(f - Au^{k-1})_1$$

$$u_2^{k+1} = u_2^{k+1/2} + A_2^{-1}(f - Au^{k+1/2})_2$$

This was already obviously a Gauss-Seidel like procedure, but those of you familiar with the algebraic form of Gauss-Seidel might be relieved to see the form here.

A roughly equivalent block Jacobi method has the form

$$u_1^{k+1/2} = u_1^{k-1} + A_1^{-1}(f - Au^{k-1})_1$$

$$u_2^{k} = u_2^{k+1/2} + A_2^{-1}(f - Au^{k})_2$$

It is possible to eliminate $u^{k+1/2}$ and obtain

$$u^{k+1} = u^k + (A_1^{-1} + A_2^{-1})(f - Au^k),$$

where the operators are understood to apply to the appropriate part of the vectors. It is here that we see that the procedure we described is a Richardson iteration with operator $M = A_1^{-1} + A_2^{-1}$.

In a modern approach to the solution of the entire problem, a step or two of block Jacobi would be used as a preconditioner in a Krylov based scheme. It is important at this point not to lose track what operations may take place at each level. To solve the subdomain problems, one might use multigrid, FFT, or preconditioned conjugate gradient, but one may choose to do this approximately during the early iterations. The solution of the subdomain problems itself may serve as a preconditioner to the solution of the global problem which may be solved using some Krylov based scheme.

0.2 Algebraic Formulation for Non-Overlapping Domains: Schur complement approaches

Figure 4 shows an example domain of a problem for a network of resistors or a discretized region in which we wish to solve the Poisson equation, $\bigtriangledown^2 v = g$

In matrix notation, $Av = g$, where

$$A = \begin{pmatrix} A_1 & 0 & A_{1I} \\ 0 & A_2 & A_{2I} \\ A_{I1} & A_{I2} & A_I \end{pmatrix}$$

One of the direct methods to solve the above equation is to use LU or LDU factorization. We will do an analogous procedure with blocks. We can rewrite A as,

$$A = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A_{I1}A_1^{-1} & A_{I2}A_2^{-1} & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & S \end{pmatrix} \begin{pmatrix} A_1 & 0 & A_{1I} \\ 0 & A_2 & A_{2I} \\ 0 & 0 & I \end{pmatrix}$$

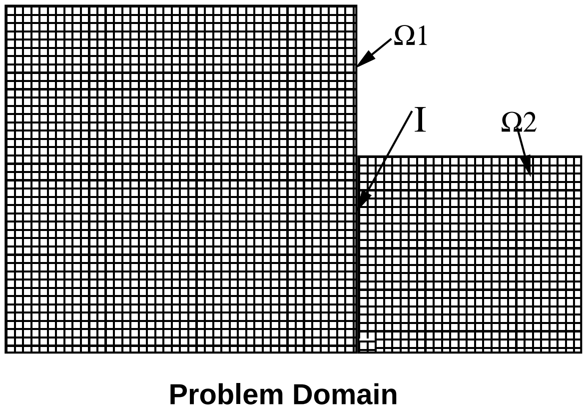


Figure 4: Problem Domain

where,

$$S = A_I - A_{I1}A_1^{-1}A_{1I} - A_{I2}A_2^{-1}A_{2I}$$

We really want A^{-1}

$$A^{-1} = \begin{pmatrix} A_1^{-1} & 0 & -A_1^{-1}A_{1I} \\ 0 & A_2^{-1} & -A_2^{-1}A_{2I} \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -A_{I1}A_1^{-1} & -A_{I2}A_2^{-1} & I \end{pmatrix} \quad (1)$$

Inverting S turns out to be the hardest part.

$$A^{-1} \begin{pmatrix} V_{\Omega 1} \\ V_{\Omega 2} \\ V_{Interface} \end{pmatrix} \begin{array}{l} \rightarrow Voltages \quad in \quad region \quad \Omega 1 \\ \rightarrow Voltages \quad in \quad region \quad \Omega 2 \\ \rightarrow Voltages \quad at \quad interface \end{array}$$

Let us examine Equation 1 in detail.

In the third matrix,
A_1^{-1} - Poisson solve in $\Omega 1$
A_{I1} - is putting the solution onto the interface
A_2^{-1} - Poisson solve in $\Omega 2$
A_{I2} - is putting the solution onto the interface

In the second matrix,
Nothing happening in domain 1 and 2
Complicated stuff at the interface.

In the first matrix we have,
A_1^{-1} - Poisson solve in $\Omega 1$
A_2^{-1} - Poisson solve in $\Omega 2$
$A_1^{-1}A_{1I}$ and $A_2^{-1}A_{1I}$- Transferring solution to interfaces

In the above example we had a simple 2D region with neat squares but in reality we might have to solve on complicated 3D regions which have to be divided into tetrahedra with 2D regions at the interfaces. The above concepts still hold.

Getting to S^{-1},

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ c/a & 1 \end{pmatrix} \begin{pmatrix} a & b \\ 0 & d - bc/a \end{pmatrix}$$

where, $d - bc/a$ is the Schur complement of d.

In Block form

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ CA^{-1} & 1 \end{pmatrix} \begin{pmatrix} A & B \\ 0 & D - CA^{-1}B \end{pmatrix}$$

We have

$$S = A_I - A_{I1}A_1^{-1}A_{1I} - A_{I2}A_2^{-1}A_{2I}$$

Arbitrarily break A_I as

$$A_I = A_I^1 + A_I^2$$

Think of A as

$$\begin{pmatrix} A_1 & 0 & A_{1I} \\ 0 & 0 & 0 \\ A_{I1} & 0 & A_I^1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & A_2 & A_{2I} \\ 0 & A_{I2} & A_I^2 \end{pmatrix}$$

Schur Complements are

$$S^1 = A_I^1 - A_{I1}A_1^{-1}A_{1I}$$

$$S^2 = A_I^2 - A_{I2}A_2^{-1}A_{2I}$$

and

$$S = S^1 + S^2$$

$A_1^{-1} \rightarrow$ Poisson solve on $\Omega 1$
$A_2^{-1} \rightarrow$ Poisson solve on $\Omega 2$
A_{I1} $\Omega 1 \rightarrow I$
A_{21} $\Omega 2 \rightarrow I$
A_{1I} $I \rightarrow \Omega 1$
A_{2I} $I \rightarrow \Omega 2$

Sv - Multiplying by the Schur Complement involves 2 Poisson solves and some cheap transferring.

$S^{-1}v$ should be solved using Krylov methods. People have recommended the use of S_1^{-1} or S_2^{-1} or $(S_1^{-1} + S_2^{-1})$ as a preconditioner

0.3 Hierarchical/Multilevel Techniques

Up until now, we saw that subdividing a problem did not directly yield the final answer, rather it simplified or allowed us to change our approach in tackling the resulting subproblems with existing methods. It still required that individual subregions be composited at each level of refinement to establish valid conditions at the interface of shared boundaries.

Multilevel approaches solve the problem using a coarse grid over each sub-region, gradually accommodating higher resolution grids as results on shared boundaries become available. Ideally for a well balanced multi-level method, no more work is performed at each level of the hierarchy than is appropriate for the accuracy at hand.

In general a hierarchical or multi-level method is built from an understanding of the difference between the damping of low frequency and high components of the error. Roughly speaking one can kill of low frequency components of the error on the course grid, and higher frequency errors on the fine grid.

Perhaps this is akin to the Fast Multipole Method where p poles that are "well-separated" from a given point could be considered as clusters, and those nearby are evaluated more precisely on a finer grid.