1991

Collaborating PDE Solvers

H. S. McFaddin

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
91-068

COLLABORATING PDE SOLVERS

H. S. McFaddin
John R. Rice

CSD-TR-91-068
October 1991

Collaborating PDE Solvers

H.S. McFaddin*
John R. Rice†
Computer Science Department
Purdue University
West Lafayette, IN 47907
Technical Report CSD-TR-91-068
CAPO Report CER-91-33

October 21, 1991

Abstract

Collaborating PDE solvers refers to a methodology for solving sets of partial differential equations (PDEs) by iteratively solving one PDE on one domain at a time. The set of PDEs are related through *interface conditions* which, in their simplest form, are shared boundary conditions. For example, two rectangles with a common edge might have the requirement that the solutions across the edge be continuous and have a continuous first derivative. Schwartz splitting is a classic method of this nature and in recent years other instances have been found which are effective. In this paper we explore the possibilities that (a) the methodology might be effective for a wide range of PDE problems, (b) it might be one of the more effective methods for the enormously complex PDE problems (e.g., complete simulation of a vehicle) that will be attempted in this decade.

The paper consists of three parts: A brief discussion of potential applications, a discussion of the interface relaxation problem, and description of the RELAX system for experimenting with this methodology.

1 INTRODUCTION

In this paper we describe a methodology for solving sets of partial differential equations (PDEs) by iteratively solving one PDE on one subdomain at a time. We assume for this discussion that the PDEs are two-dimensional, second order, linear elliptic problems although the methodology is not restricted to this class. The problem can be stated mathematically as follows

*Work supported in part by IMSL, Inc. and the National Science Foundation grant 86-19817.

†Work supported in part by the Air Force Office of Scientific Research, grant 88-0243; the Strategic Defense Initiative through ARO grant DAAL03-90-0107; and the National Science Foundation grant 86-19817.

$$L_i \mathbf{u}_i = f_i \qquad \text{for } (x,y) \in D_i \qquad i = 1,2,\ldots,m$$
$$M_i \mathbf{u}_i = g_i \qquad \text{for } (x,y) \in \partial D_i$$

where L_i is a linear, second order elliptic PDE operator, $\mathbf{u}_i$, f_i and g_i functions, D_i is a subdomain, M_i is a linear first order operator (boundary conditions), and ∂D_i is the boundary of D_i. Throughout this paper we use bold face type for the solutions of the PDEs or their discretizations. Wherever two of the subdomains join, the boundary conditions become interface conditions involving the solutions on both sides of the boundary or interface.

The methodology of collaborating PDEs solvers is to have a PDE solver assigned to each subdomain and then successively solve the PDEs while adjusting the function values along the interfaces so as to better satisfy the interface conditions. Thus we have an iterative method whose two basic steps are to solve a PDE and to relax interface conditions. This methodology is related to domain decomposition where the L_i are all the same and the interface conditions are smoothness of the solution. This relationship is discussed in Section 2 where the approaches of "discretize first" (common in domain decomposition) and "decompose first" (used here) are compared. The application of this methodology to more complex PDE problems is first discussed in this section and then in Section 3 it is related to some "grand challenges" of computational science.

Section 4 addresses the principal component of this methodology, how to relax interface conditions. It is clear that this process is not yet well understood but there is ample theoretical and experimental evidence that robust, widely applicable interface relaxation formulas may exist. We present a brief survey of types of interface relaxations including a new one which we find to be quite robust. No proofs of its effectiveness are known to us. Finally, in Section 5, we describe the computer system RELAX which can be used to explore this methodology experimentally. It is also a prototype of a PDE solving system of a type that shows promise for future high level application systems.

2 DOMAIN DECOMPOSITION

Consider for the moment the task of solving one PDE on one domain, both of which are complex. *Domain decomposition* is the process of subdividing the domain into many parts and dividing the solution process into two parts: (1) computations on the subdomains, (2) computations on the whole problem. The principal motivation here is to apply parallel computers although there are other good reasons to use domain decomposition. See [2], [5], [6] for more information.

Solving PDEs usually requires *discretization* where space (and time) is replaced by point sets or collections of elements (grids, meshes, etc.) and then the PDE is approximated locally in terms of variables defined on these points or elements. One may either discretize the PDE and then decompose the discrete domain or first decompose the domain and then discretize on the subdomains. The order chosen has a strong influence on the nature of the computation methods and each has its strengths and weaknesses, listed below:

Decompose, then discretize:

- Allows the use of independent PDE solving methods and software on each subdomain. Advantageous when nature or difficulty of the PDE varies greatly over the entire domain.
- May required special, explicit formulas to relate discrete variables between subdomains.
- Can be used to reduce geometric complexity by making each subdomain have a "simple" shape.

Discretize, then decompose:

- Reduces problem to discretize variables immediately.
- Allows direct application of methods for solving linear (or nonlinear) systems for equations.
- Easy to "lose" information related to problem geometry during the problem solving process.

One may view the "decompose, then discretize" approach of allowing (in fact, forcing) one to analyze problem solving methods at the level of simultaneous PDEs instead of simultaneous algebraic equations (usually linear). The approach of collaborating PDE solvers is to use iterations based on *interface relaxation* techniques as discussed in Section 4.

To further develop the comparison of these two approaches, consider a single linear PDE problem that is to be decomposed into three parts. If the PDE is discretized first, one obtains the matrix problem illustrated in Figure 1. In this common approach, one factors the three matrices on upper left either exactly or approximately, depending on the method. One then uses this factorization to reduce the problem to the *interface equations* on the lower right. The resulting equations are often called the *capacitance matrix* or *Schur complement.* If an approximate factorization is made of the diagonal blocks, then an iteration is used. Either a direct method or iterative method can be used on the interface equations if the diagonal blocks are factored exactly.

This process is illustrated more concretely in Figure 2 where a PDE is discretized into a 37 by 37 grid on a rectangular domain using ordinary finite differences. This problem with 1369 unknowns is then decomposed into 16 parts to produce the matrix shown in Figure 2(a). Factoring the diagonal blocks produces the pattern of unknowns shown in Figure 2(b) and eliminating all the subdomain variables produces the interface matrix shown in Figure 2(c). This example uses a particular nested dissection method of ordering the equations and unknowns, other orderings produce different patterns of non-zeros [10].

In the "decompose first" approach one must pass information between subdomains about the discretized variables. Some believe that this is a messy, even unmanageable, complication inherent in this approach. This should not be so and, in fact, a PDE solver will have all the messy analysis already done when it provides the capability to plot the computed

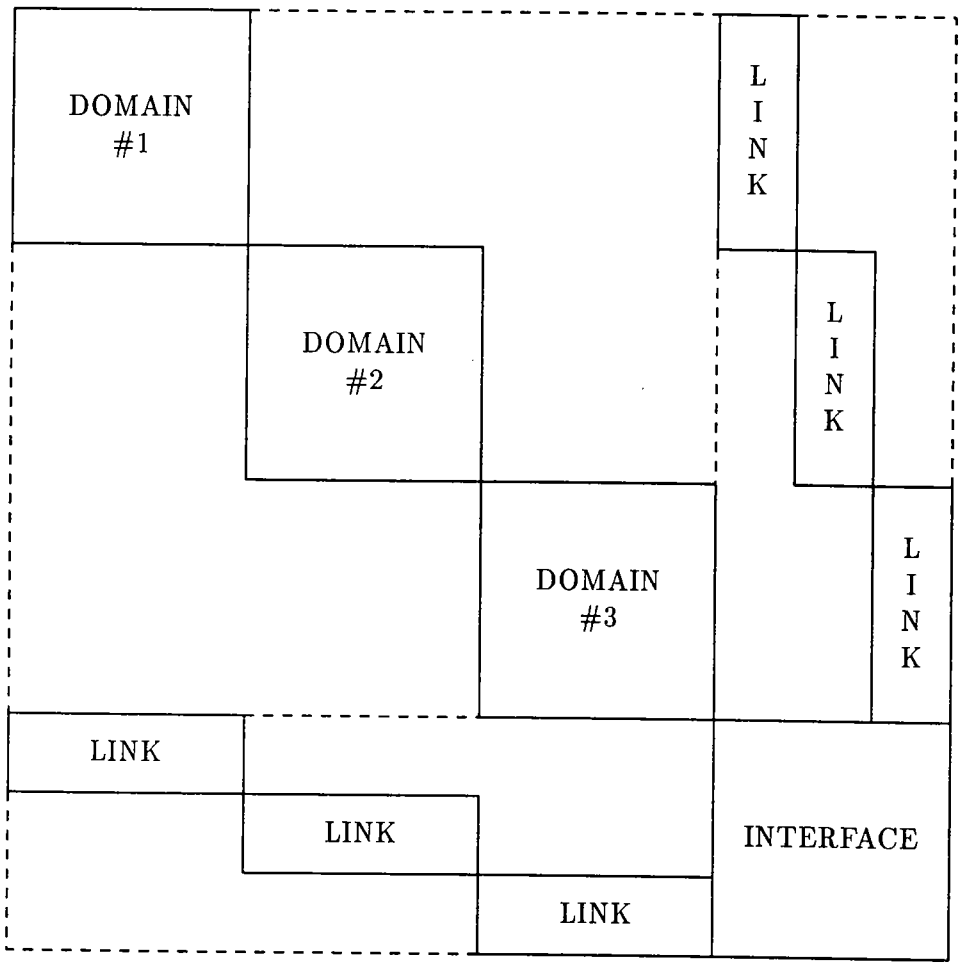


Figure 1: The matrix obtained by discretizing a PDE problem and then decomposing the problem into three parts.

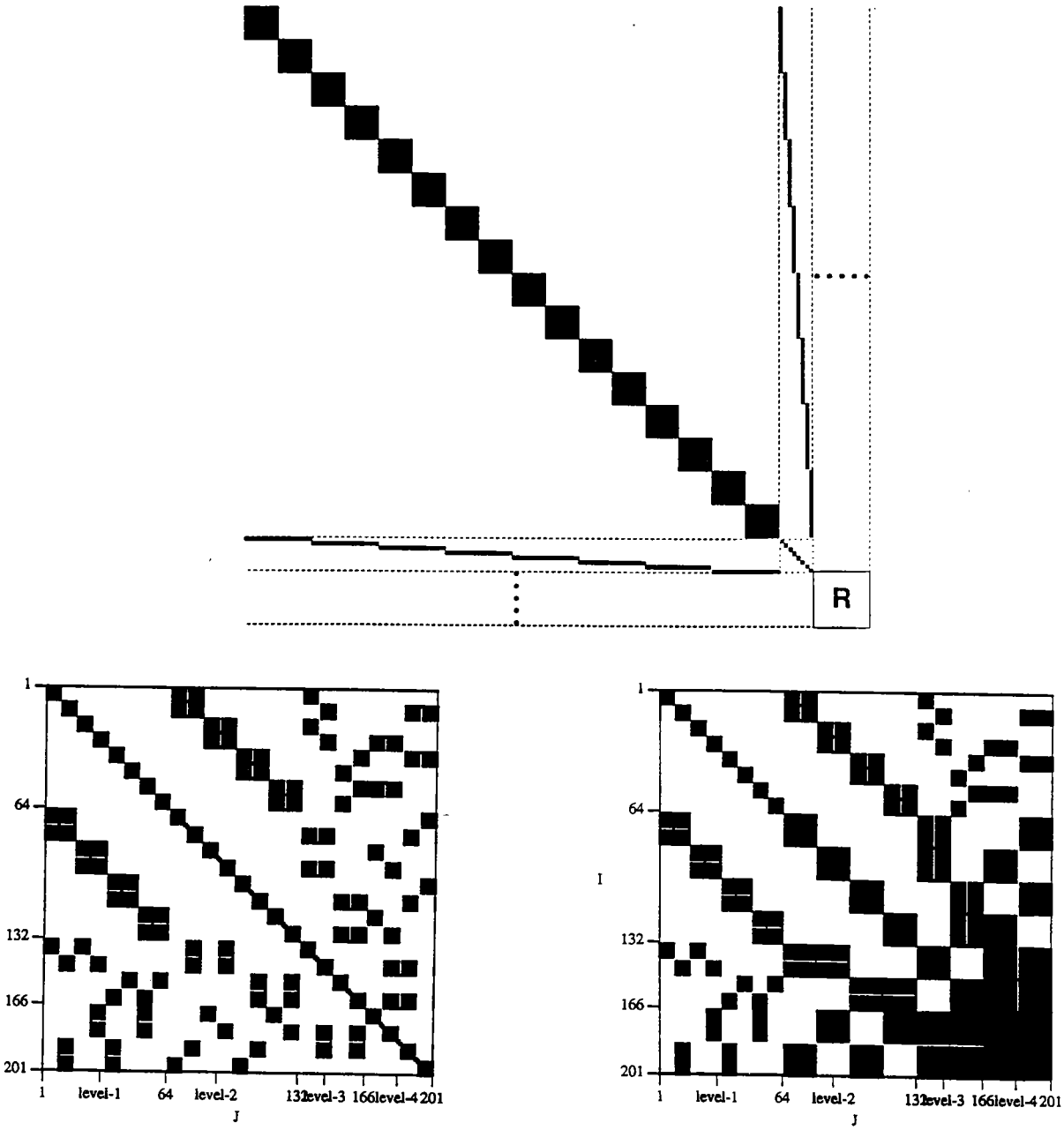


Figure 2: Patterns of non-zero obtained by discretizing with ordinary finite differences on a rectangular domain. The problem is then decomposed into 16 pieces. (a) The original pattern of non-zeros. (b) The pattern after factoring the diagonal blocks. (c) The pattern of the interface equations after eliminating the rest of the variables.

PDE solution and its derivatives. Every well developed PDE solving module will have this capability.

Figure 3 shows a part of the boundary between two subdomains where unrelated discretizations have been used. The unknowns in the top subdomain are represented by big dots and those in the bottom subdomain by small solid rectangles. The discrete unknowns need not be point values, but it is easier to visualize the situation if they are. There are interface conditions between the subdomains that may involve arbitrary points p_k in the interface, in particular, the p_k need not be the dots on the rectangles. These conditions are typical of the form.

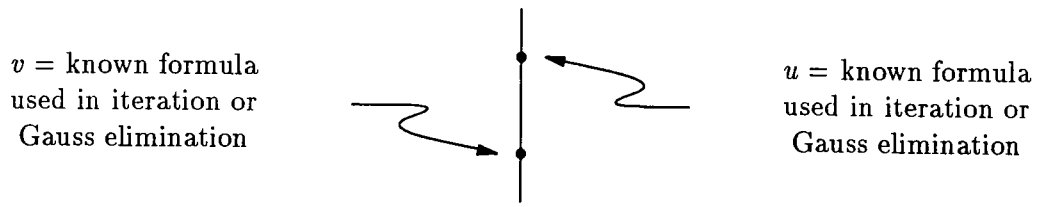


Figure 3: A piece of the interface between two subdomains after unrelated discretizations have been made of the PDEs defined on them.

$$\mathbf{u}(p_k) + \alpha \mathbf{u}_N(p_k) + \mathbf{v}(p_k) + \beta \mathbf{v}_N(p_k) = g \tag{1}$$

where $\mathbf{u}$ and $\mathbf{v}$ are solutions on the top and bottom subdomains, normal derivatives are indicated by the subscript N and α, β, g are functions. This equation is to hold for all points p_k selected in the discretization of the interface conditions between the two subdomains.

The PDE solver must not only be able to compute an approximate solution to the PDE but it must also be able to evaluate this solution and its derivatives at any point in the subdomains. The evaluation procedures must, therefore, contain expressions for $\mathbf{u}(s,t)$, $\mathbf{u}(s,t)$ [or perhaps $\mathbf{u}(s,t)$ and $\mathbf{u}(s,t)$] for any point (s,t) in the subdomain. More specifically, there are basis functions (or equivalent) $b_i(s,t)$ and $c_j(s,t)$ so that

$$\begin{aligned} \mathbf{u}(s,t) &= \sum \ell_i b_i(s,t) \\ \mathbf{u}_N(s,t) &= \sum m_i b_i(s,t) \\ \mathbf{v}(s,t) &= \sum q_j c_j(s,t) \\ \mathbf{v}_N(s,t) &= \sum r_j c_j(s,t) \end{aligned} \tag{2}$$

where the coefficients ℓ_i, m_i, q_j and r_j. are constants that define the approximate solution. Usually local basis functions are used so that the above sums contain only a few terms. These formulas (2) are present, either explicitly or implicitly, in the output routines for the PDE solver. Therefore they can be transformed into explicit formulas to be used in (1).

Further, they can be used to apply iteration or direct (e.g., Gauss elimination) methods to the interface equations (1) in solving for the discrete unknowns in both subdomains. The formulas (1), (2) are not passed across the interface between the two subdomains, rather actions to be taken or information requests are passed and these formulas are used to carry out the actions or provide the information.

Now consider the same example as in Figure 1 where one decomposes first. We have three discretized PDE operators (matrices) A_i and associated PDE problems with solutions $\mathbf{x}_i$

$$A_i \mathbf{x}_i = f_i \qquad i = 1, 2, 3. \tag{3}$$

On the interface boundaries between subdomains i and j we have interface conditions and associated unknowns $\mathbf{b}_{ij}$, $i \neq j$. Let B_{ij} denote these conditions (see equation (1) above), so we have

$$B_{ij}\mathbf{b}_{ij} + B_{ji}\mathbf{b}_{ji} = g_{ij} \qquad i, j = 1, 2, 3; \quad i \neq j \tag{4}$$

The operators B_{ij} can, for example, represent continuity of solutions or physical conditions such as continuity of flow. As discussed above, we also have interpolation operators that relate solutions $\mathbf{x}_i$ on domain i with interface unknowns $\mathbf{b}_{ij}$. Before discretization these interpolation operators are simply the identity, that is $\mathbf{x}_i \equiv \mathbf{b}_{ij}$ on the interfaces of domain i. However, after discretization they become interpolation operators (see equations (2) above), so we formulate the problem that way from the start. We then have

$$\mathbf{b}_{ij} = I_{ij}\mathbf{x}_i \qquad i, j = 1, 2, 3; \quad i \neq j \tag{5}$$

The functions f_i and g_{ij} are from the original PDE problem or the interface conditions. Equation (3), (4) and (5) involve 9 unknown functions and they can be represented in the form of a matrix of operators as seen in Figure 4.

3 APPLICATIONS OF COLLABORATING PDE SOLVERS

The collaborating PDE solvers approach may prove useful as a technique to apply parallel computers to a single PDE on a single domain or to simplify the geometry for a single PDE on a domain with complex shape. Its principal attraction, however, is for problems involving multiple PDEs. A simple example is shown in Figure 5. We see that the number of unknown variables (and hence PDEs) can vary from domain to domain. The interface conditions can be quite simple (e.g., the temperature and heat flux are continuous across the iron-brass interface) or quite complex (e.g., there is radiation from the water surface as well as conduction from the water to the air).

$$
\begin{pmatrix}
A_1 & & & & & & & & & \\
& A_2 & & & & & & & & \\
& & A_3 & & & & & & & \\
& & & B_{12} & B_{21} & & & & & \\
& & & & & B_{13} & B_{31} & & & \\
& & & & & & & B_{23} & B_{32} & \\
I_{12} & & & I & & & & & & \\
I_{13} & & & & I & & & & & \\
& I_{21} & & & & I & & & & \\
& I_{23} & & & & & I & & & \\
& & I_{31} & & & & & I & & \\
& & I_{32} & & & & & & I &
\end{pmatrix}
\begin{pmatrix}
\mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{b}_{12} \\ \mathbf{b}_{13} \\ \mathbf{b}_{21} \\ \mathbf{b}_{23} \\ \mathbf{b}_{31} \\ \mathbf{b}_{32}
\end{pmatrix}
=
\begin{pmatrix}
f_1 \\ f_2 \\ f_3 \\ g_{12} \\ g_{13} \\ g_{23} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix}
$$

Figure 4: The system of operator equations obtained by decomposing a PDE problem into three pieces.

We discuss two applications which are well beyond the capabilities of current supercomputers but which are illustrative examples of computations expected to be made in the coming decade. First is the accurate simulation of a complete vehicle. One wants to have engineering accuracy in the stresses on the engine block and the springs, the gas flows over the hood and through the exhaust system, the heat flows in the radiator and the passenger compartment, the mechanical motion of the pistons and the wheels on the road, the combustion in the cylinders, the radiation of sunlight through the windows — and everything else of interest. To remind the reader of the complexity of this problem, Figure 6 shows a cross section of part of an engine. It is estimated for this problem:

- Approximately 100 million discrete variables are required to simulate the state of a vehicle at any one moment to within engineering accuracy.
- Approximately 10,000 subdomains are required to decompose the vehicle into pieces that have simple shapes and which are reasonably homogeneous in physical behavior.
- The "answer" to a typical engineering design question (e.g., what happens it...) requires about 20 gigabytes of memory to store and about 10 teraFLOPS to compute.

While the computation for this simulation seems large, note that an accurate 3 day weather forecast requires about 3 billion discrete variables, 100 mega-giga FLOPS, and 10 gigabytes to store the answer. The weather forecasting problem does not, however, have nearly as much diversity in the physical phenomena considered nor as much geometric complexity.

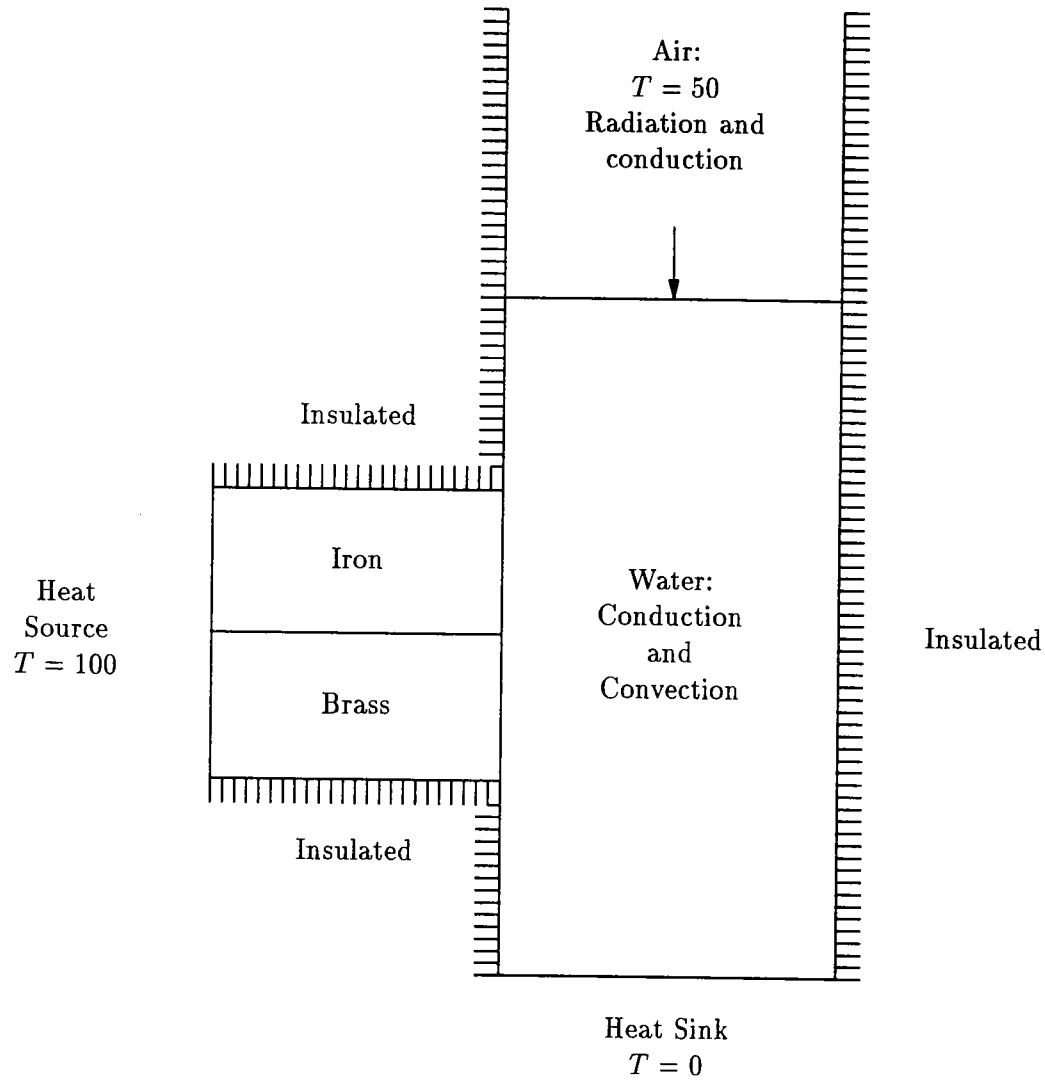


Figure 5: A simple heat and fluid flow problem involving multiple PDEs in a domain with simple geometry.

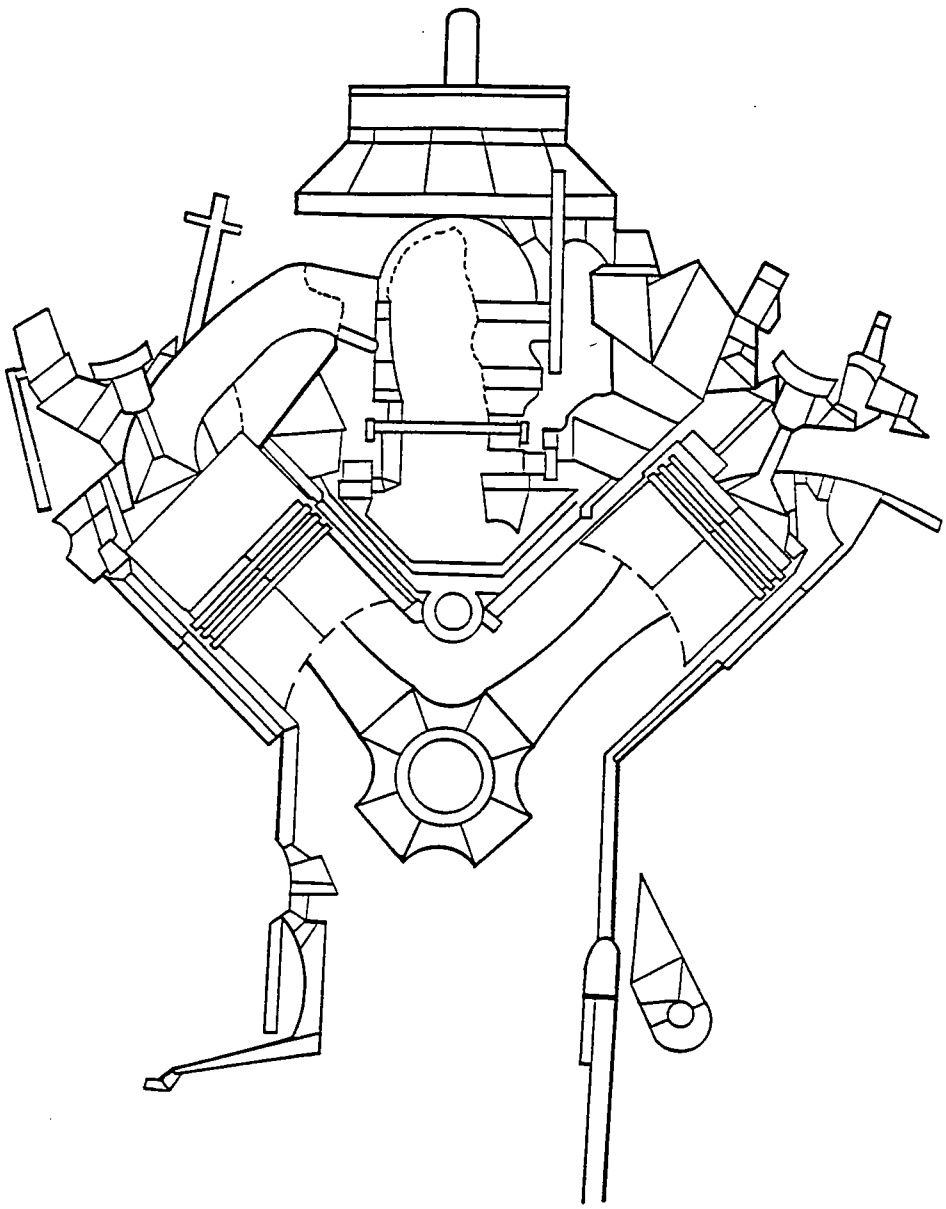


Figure 6: Cross section view of an automobile engine. The objects viewed have been decomposed into subdomains so that each has a simple geometric shape.

The second application for the future is the simulation of a tank battle. Assume that six tanks are involved and their motion, internal operation, and the terrain are modeled only roughly. Perhaps less than 100,000 discrete variables are used to represent these items. Assume further that an accurate simulation is required of the weapons and their effects. Thus, once a cannon, rocket, laser or phaser is fired, the weapon's path and effect on the target tank's armor is to be computed accurately. In this computation one sees that the simulation can progress very quickly (assuming a powerful supercomputer) until a weapon is fired. Then a large scale computation "opens up" at some unpredictable place in the problem domain. The time scale of the computation changes from seconds to milliseconds and then to microseconds. We estimate that the "answer" to a typical problem of this type is about 8 terabytes (the equivalent of a 100 hour color movie) computed with about 2 mega-giga FLOPS.

4 INTERFACE RELAXATION

Consider the situation illustrated in the diagram below where one has two second order, elliptic PDE problems with interfaces between them.

$$L_1\mathbf{u} = f_1 \quad \Big| \quad L_2\mathbf{v} = f_2 \tag{6}$$

$$Interface$$

The solutions $\mathbf{u}$ and $\mathbf{v}$ on the left and right are to satisfy some interface conditions, typically involving solution values and derivatives such as

$$\mathbf{u}(x,y) = \mathbf{v}(x,y) \tag{7a}$$

$$\mathbf{u}(x,y) + \alpha\mathbf{u}_x(x_{10}) = \mathbf{v}(x,y) + \beta\mathbf{v}_x(x,y) + g(x,y) \tag{7b}$$

for all (x,y) on the interface. An iteration involving an interface relaxation is of the following form:

1. Have $\mathbf{u}$ and $\mathbf{v}$ which solve the PDEs (6) but do not satisfy the interface conditions (7).
2. Apply *interface relaxation* to obtain new values for $\mathbf{u}(x,y)$ and $\mathbf{v}(x,y)$ on the interface that better satisfy (7).
3. Resolve the two PDE problems with these new boundary values.
4. Iterate steps 2 and 3 until convergence.

The goal then is to find interface relaxation formulas that provide fast convergence. The classical method of this type is the *Schwartz alternation methods* [7], [13], [14] formulated as follows (see Figure 7). Let domain D_1 and D_2 be $[0,B]\times[0,C]$ and $[A,1]\times[0,C]$, respectively and assume the same PDE operator on each domain. The Schwartz alternation method then iterates as follows:

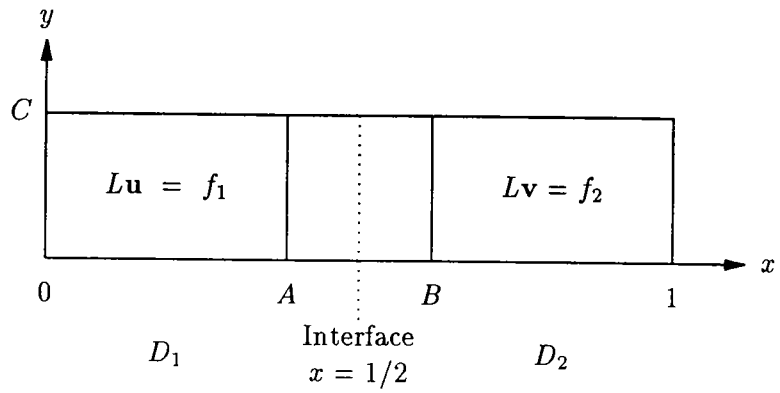


Figure 7: Two domains which overlap the interface. The left PDE problem is solved on the domain $x \in [0,B]$, the right one on $x \in [A,1]$.

1. Guess at $\mathbf{u}(x,y)$ on the line $x = B$.
2. Solve for $\mathbf{u}$ on domain D_1.
3. Set $\mathbf{v}(x,y) = \mathbf{u}(x,y)$ on the line $x = A$.
4. Solve for $\mathbf{v}$ on the domain D_2.
5. Iterate steps 2 to 4 until convergence.

The interface relaxation is defined implicitly here as values along the interface are not manipulated directly. It has been shown [3], [15] that Schwartz alternation does define an interface relaxation without involving overlapping domains.

A more recent and simpler interface relaxation is the alternating Dirichlet-Neumann method introduced by Quarteroni [4]. This method applied to the problem in Figure 7 is as follows:

1. Guess at $\mathbf{d}(0.5,y) = \mathbf{u}(0.5,y) = \mathbf{v}(0.5,y)$ along the interface.
2. Solve the PDEs on each part with Dirichlet boundary conditions $\mathbf{d}(0.5,y)$.

3. Set $\mathbf{n}(0.5, y) = (\mathbf{u}_x(0.5, y) + \mathbf{v}_x(0.5, y))/2$ on the interface.

4. Solve the PDEs on each part with Neumann boundary conditions $\mathbf{n}(0.5, y)$.

5. Set $\mathbf{d}(0.5, y) = (\mathbf{u}(0.5, y) + \mathbf{v}(0.5, y))/2$ on the interface.

6. Iterate steps 2 to 5 until convergence.

The interface conditions that are being satisfied here are

$$\mathbf{u}(0.5, y) = \mathbf{v}(0.5, y) \tag{8a}$$

$$\mathbf{u}_x(0.5, y) = \mathbf{v}_x(0.5, y) \tag{8b}$$

For simple PDE problems (e.g., Poisson problems) satisfying the conditions (8) implies that $\mathbf{u}$ and $\mathbf{v}$ join with all derivatives continuous and satisfy the PDE on the entire domain.

The convergence of these two alternation methods has been proved for some general classes of PDE problems and has also been established in practice, see [1], [11], [12] for recent work and references to earlier work. On the other hand, examples show that these two methods do not converge for all PDE problems (6), indeed, they both require that $L_1 = L_2$. This naturally suggests to search for interface relaxation methods with more general convergence properties. We list a set below, but it is fair to say that the convergence properties are poorly understood once one gets away from the simpler PDE problems. There are many complex examples where convergence can be observed for certain interface relaxations and there are simple cases where some interface relaxation fail to converge for no apparent "reason". We have experimentally observed convergence with several relaxation formulas for a variety of elliptic problems, e.g.,

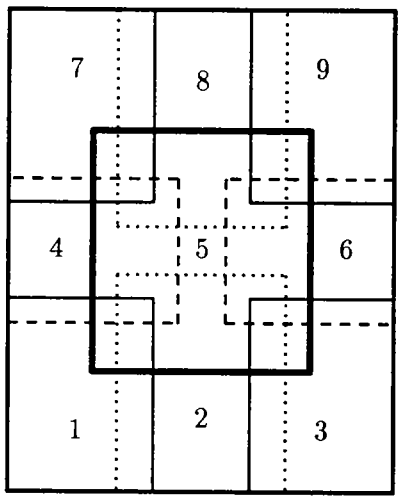
$$\begin{gathered}\mathbf{u}_{xx} + (1+y^2)\mathbf{u}_{yy} - \mathbf{u}_x - (1+y^2)\mathbf{u}_y = f \\ \mathbf{u}_{xx} + \mathbf{u}_{yy} - (100 + 2\cos(2\pi x) + \sin(2\pi y))\mathbf{u} = f \\ x^2\mathbf{u}_{xx} + \mathbf{u}_{yy} + 2x\mathbf{u}_x + (\cot y)^3\mathbf{u}_y = -100x^2.\end{gathered}$$

Quarteroni [11] reports convergence on problems with the Navier-Stokes equation and a parabolic-hyperbolic pair of PDEs. We have observed convergence on a "skyline" domain (a set of rectangular blocks placed side by side) with up to 40 blocks and yet some methods fail with an L-shaped domain or three identical tall, narrow rectangular domains placed side by side.

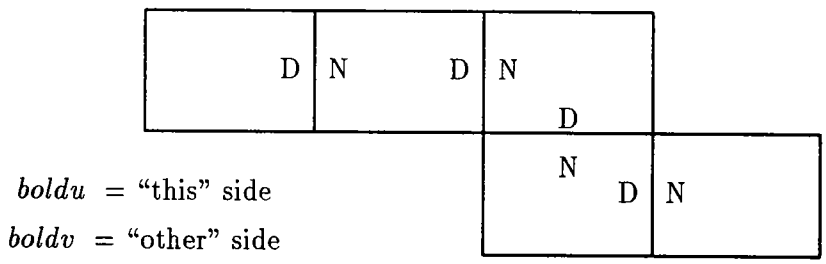
Our list of interface relaxations follows with brief descriptions and references to more details.

Schwarz Alternation. There is a large literature about this method, see [2], [5], [6], [7]. Its principal drawbacks are (a) it only applies to decomposing the domain of a single PDE operator, (b) the management of overlapping domains can become quite complex, see Figure 8(a).

Alternating Dirichlet-Neumann. One can alternate the Dirichlet and Neumann boundary conditions in various patterns. In Figure 8(b) one sees a domain decomposed into 5 subdomains with 4 interfaces and a pattern of D's and N's indicating that Dirichlet



(a)



(b)

Figure 8: (a) The overlapping of domains that occurs with 2-dimensional Schwartz alternation. (b) A simple example of a pattern for the alternating Dirichlet-Neumann method for five subdomains.

and Neumann conditions are alternated along the subdomains. One uses the relaxations (here k is the iteration index):

At each D use:

$$\mathbf{u}^{k+1} = \theta \mathbf{u}^k + (1-\theta)\mathbf{v}^k \tag{9a}$$

At each N use:

$$\mathbf{u}^{k+1} = -\mathbf{v}_N^k \tag{9b}$$

Here $\mathbf{u}$ refers to the solution in the subdomain being considered and $\mathbf{v}$ refers to solution on the other side of the interface. In the discussion above we alternated the D and N during the iteration instead of along the set of interfaces. Thus in Figure 8(b) we could use all D's on the even iteration and all N's on the odd iterations. A few "random" experiments revealed no substantial differences between various patters of D's and N's.

The range of geometric shapes for which this method converges is unknown. The simple presence of "corners at an interface" (such as occurs in the skyline domain mentioned above) prevents convergence even for just two domains. Adding some "something" from the previous iteration, as in (9a) or

$$\mathbf{u}_N^{k+1} = -\alpha \mathbf{v}_N^k + (1-\alpha)\mathbf{u}_N^k$$

helps the convergence. The presence of *cross points* (where two interfaces cross) substantially decreases the observed rates of convergence.

General Linear Relaxers. Various heuristics lead to interface relaxation of the form

$$\mathbf{u}^{k+1} = \frac{\mathbf{u}^k + \mathbf{v}^k}{2} + a(\mathbf{u}^k - \mathbf{v}^k) + b(\mathbf{u}_N^k + \mathbf{v}_N^k) + c(\mathbf{u}_{NN}^k - \mathbf{v}_{NN}^k) \tag{10}$$

where, as above, k is the iteration index, $\mathbf{u}$ is the solution on the domain under consideration, $\mathbf{v}$ is the solution across the interface, and the subscript N indicates a derivative normal to the interface. Note that since normals point in opposite directions along an interface, the third term is actually a difference in values. The coefficients a, b and c depends on the units and the geometry of the subdomains. One heuristic that leads to (10) is to use least squares to try to satisfy simultaneously

$$\mathbf{u} = \mathbf{v}, \quad \mathbf{u}_N = -\mathbf{v}_N$$

along the interface. Experimentation with (10) suggests that choosing $a = c = 0$ does not degrade the convergence rate significantly.

Smooth Along the Interface. The above interface relaxations have a "smoothing" nature (hence the name relaxation) and that suggests other smoothing might be effective. The most natural method is where $Lu = f$ is known to satisfy on the interface, which we assume is the line $x = const.$ for simplicity of notation. Once $\mathbf{u}$ and $\mathbf{v}$ are determined away from the interface, one discretizes near the interface, replacing x derivatives with differences.

This leaves an ordinary differential equation in y along the interface. Solve this equation for values along the interface and use them for the next iteration. Glowinski has shown that this approach converges for a class of PDE problems even if one uses the Laplacian instead of the PDE operator of the original problem. We have experimentally observed that simple data smoothing often improves or induces convergence. That is, whatever interface values one has, smooth them by a least squares cubic spline or local polynomial fits.

Newton's Method. Recall the shooting method from ordinary differential equations where one tries to obtain a value for an unknown derivative at one end of the interval in order to match a known value at the other end. The problem is posed as solving a nonlinear equation and Newton's method (or any other nonlinear equation solving method) is applied. In the present instance we have PDEs and interface conditions to be satisfied. We have several functions which satisfy the PDEs and approximately satisfy the interface conditions. The application of Newton's method to improve the interface values may be formualted as follows. We assume the interface conditions $\mathbf{u} = \mathbf{v}$ and $\mathbf{u}_N = \mathbf{v}_N$ for simplicity, see (6).

1. Assume one has $\mathbf{u}^k$ and $\mathbf{v}^k$ known along the interface.

2. Compute $\mathbf{u}_N^k$ and $\mathbf{v}_N^k$ by solving the PDEs for $\mathbf{u}^k$ and $\mathbf{v}^k$ inside the domains.

3. Let $\delta\mathbf{u}$ and $\delta\mathbf{v}$ be corrections to $\mathbf{u}^k$ and $\mathbf{v}^k$, they should satisfy

$$\begin{aligned} (\mathbf{u}^k + \delta\mathbf{u}) - (\mathbf{v}^k + \delta\mathbf{v}) &= 0, \\ \mathbf{u}_N^k(\mathbf{u}^k + \delta\mathbf{u}) - \mathbf{v}_N^k(\mathbf{v}^k + \delta\mathbf{v}) &= 0. \end{aligned}$$

   Note that parentheses here indicate functional arguments, not multiplication.

4. Linearize these equations (apply a discrete Newton's method) using approximations of the form

$$\frac{\partial \mathbf{u}_N^k(\mathbf{u}^k)}{\partial \mathbf{u}^k} \sim \frac{\mathbf{u}_N^k - \mathbf{u}_N^{k-1}}{\mathbf{u}^k - \mathbf{u}^{k-1}},$$

5. Then solve for $\delta\mathbf{u}$ and $\delta\mathbf{v}$ which are added to $\mathbf{u}^k$ and $\mathbf{v}^k$ to give the next iterates.

A few expriments with this method suggest (a) it has the usual property of Newton's method of working well when close to answer and not when far away, (b) combining it with smoothing along the interface helps.

Continuation Methods. In the real world everything is time depedent and perhaps "nature" does interface relaxation only for small perturbations of "known" situations. This suggests imbedding the elliptic PDE in a time depednent PDE and starting the continuation from known situations. This method seems intuitively attractive, but we have not seen any theoretical or experimental studies of it.

Finally, we note that the rate of convergence of interface relaxation should **not** depend on the method used to solve the PDE on the subdomains. There are a number of

instances where this lack of dependence can be observed in practice or proved to be so. In particular, one does not see any dependence on mesh or element sizes for finite difference of finite element methods. Most studies where the discretization is done first shows a strong dependence of the convergence rate on mesh size. This might be an advantage of the "decompose first" approach or, perhaps more likely, it might mean that there are better iterative methods for the "discretize first" approach which are yet to be discovered.

5 RELAX: A SYSTEM FOR COMPUTING WITH COLLABORATING PDE SOLVERS

RELAX is an experimental system for using collaborating PDE solvers. From a traditional numerical analysis point of view RELAX is a system for experimenting with iteration methods in the spirit of Southwell's work in the 1930's. Now the basic iteration step is to solve a single linear PDE instead of a single linear algebraic equation. The relaxation step is to improve functions defined on the interfaces instead of improving the values of discrete variables. As Southwell did, the iteration can be human directed and one can experiment with different relaxation techniques.

From the computational science point of view RELAX is a prototype of a new problem solving methodology for complex PDE problems. The geometry is specified by an interactive building block approach. The physics is specified on each building block in a natural mathematical way. Simple parameters of numerical methods are specified on each block and for the overall computation. The solution is then displayed directly or passed on to another process.

From the computer science point of view RELAX is an experimental system to support high level user interfaces and an object-oriented framework for sets of problem solving modules. The problem solving modules are encapsulated into software objects and interact only through the RELAX framework. These objects have their own numerical methods, their own editors to interact with users, their own display capabilities, etc. Some objects may, of course, be clones of a single master object. The RELAX system allows many master objects including those created by the user on the spot by combining and/or specializing existing objects.

The RELAX system is described at length in [8], [9], we present here an example oriented view of what one can do with it. Its capabilities are of six types:

Geometry: One can create collections of building block shapes (subdomains) to define a complex geometric object (domain). The basic shapes have parameters (e.g., width, rotations) to help shape the composite object.

PDEs: One can define a different partial differential equation and associated boundary conditions on each subdomain.

Solvers: The building blocks also have associated PDE solving methods (in principle, one could have several such methods) whose parameters (e.g., mesh size) can be specified.

Interfaces: As the subdomains are assembled, interfaces are created and explicitly identified as objects in the RELAX system.

Relaxers: Interface relaxation formulas are assigned to each interface. These may be written by the user or selected from a menu of predefined formulas.

Schedules: An ordering of applying the PDE solvers on the subdomains and the relaxers on the interfaces constitutes a schedule. This ordering may be a simple algorithm (e.g., round-robin) selected from a meu or interactively specified step by step.

RELAX is an experimental prototype so some of these capabilities are limited in various ways. For example, all interfaces are straight lines.

The use of RELAX is illustrated by the problem in Figure 9. This elliptic PDE problem involves seven subdomains and five operators. One may interpret the problem as one of a temperature distribution. These are explicit heat sources on two subdomains (where the right side is -1), there are solution and location dependent heat sources or sinks on two subdomains (with the $x\mathbf{u}_y$ and $y\mathbf{u}_x$ terms), and there are two radiating boundary conditions which also may be sources or sinks. The other subdomains have simple heat flow and the other boundaries are held at zero temperature.

Figure 10 shows a snapshot in building the domain for the problem. Shapes have been picked up from the bottom of the window and some assembled. The interfaces are identified by the large dots. At this point only geometric information is specified, the PDEs and interfaces still have simple defaults for their equations. Figure 11 shows the domain fully constructed and the initial PDE solution is displayed via contour plots. These solutions are computed independently by each subdomain object with all interface functions set to zero and each contour is displayed by the subdomain object.

The iteration proceeds with the interface relaxation formula

$$\mathbf{u}^{k+1} + b\mathbf{u}_N^{k+1} = \mathbf{v}^k - b\mathbf{v}_N^k \tag{11}$$

Thus mixed boundary conditions are imposed on the $\mathbf{u}$ domain from corresponding values on its neighboring domains (note that $\mathbf{u}_N \sim -\mathbf{v}_N$ because the normals point in opposite directions). We have found (11) to be the most robust of those we have used. This relaxation formula is related closest to the alternating Dirichlet-Neumann and general linear relaxers but not quite a special case of either. No convergence proof has been given for it. The parameter b depends on the scale and units. A simple round robin ordering is used and the contour plots of the solutions for several iterates are shown in Figures 12($k = 1$), 13($k = 3$), 14($k = 10$) and 15($k = 13$). Contour plots are rather sensitive to slopes so the smoothness seen in Figure 15 indicates very good accuracy in satisfy $\mathbf{u} = \mathbf{v}$, $\mathbf{u}_N = -\mathbf{v}_N$ across the interfaces.

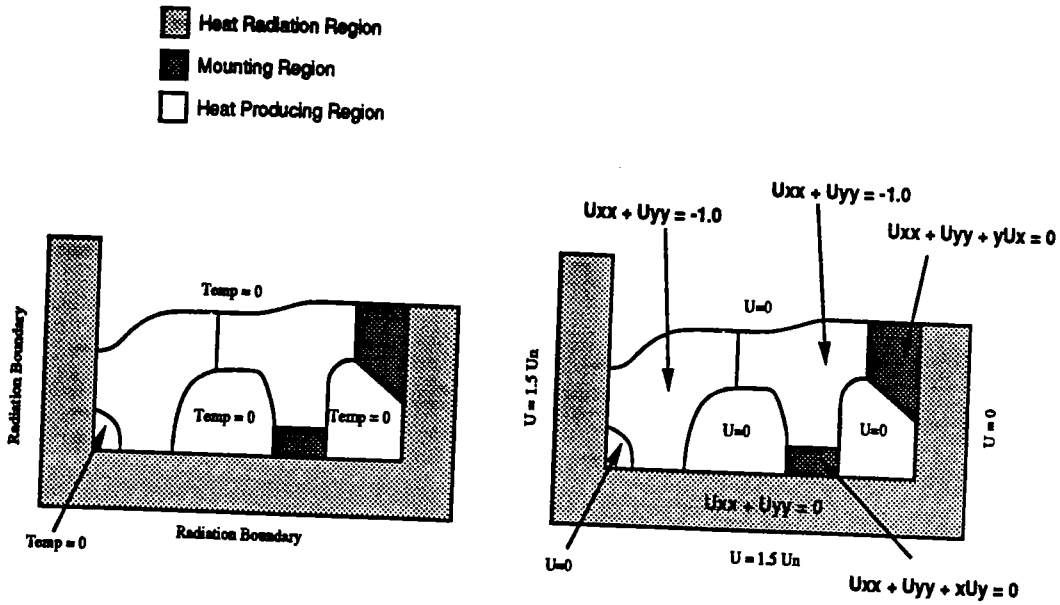


Figure 9: A temperature distribution problem involving seven subdomains and several PDEs.

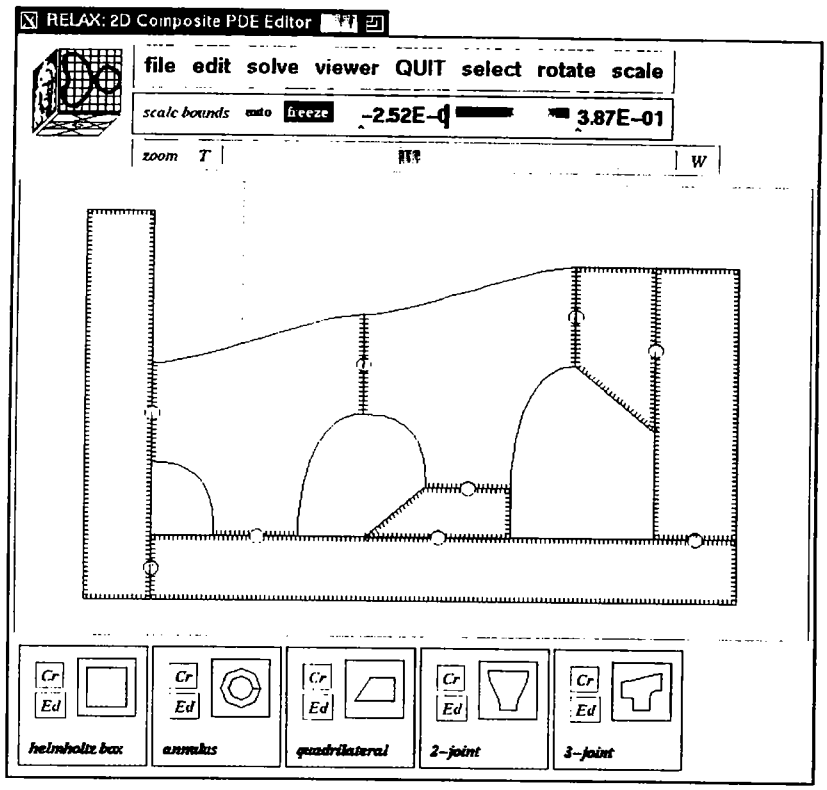


Figure 10: Creating a complex domain in RELAX by manipulating master objects (at bottom of window). The dots identify interfaces.

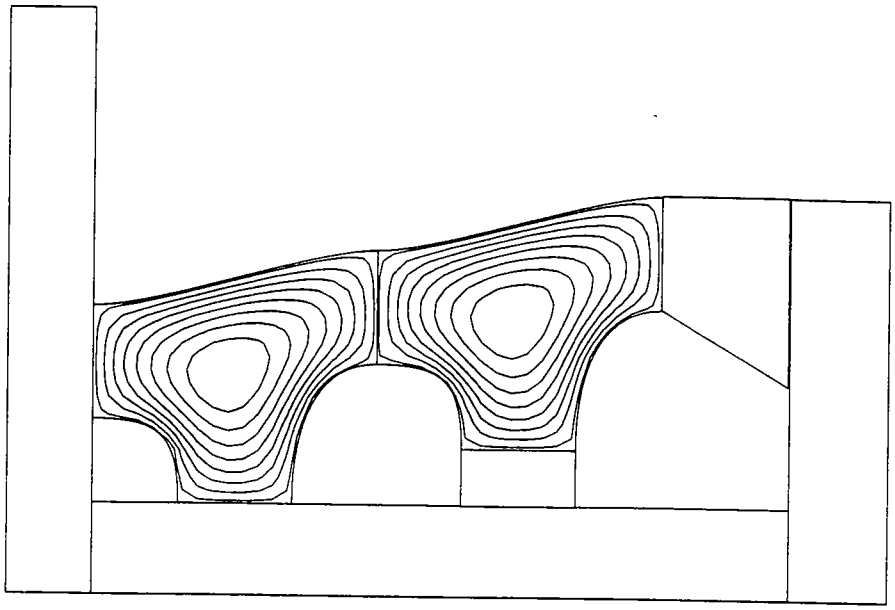


Figure 11: Initial temperature on the domain. Default interface values of zero are used.

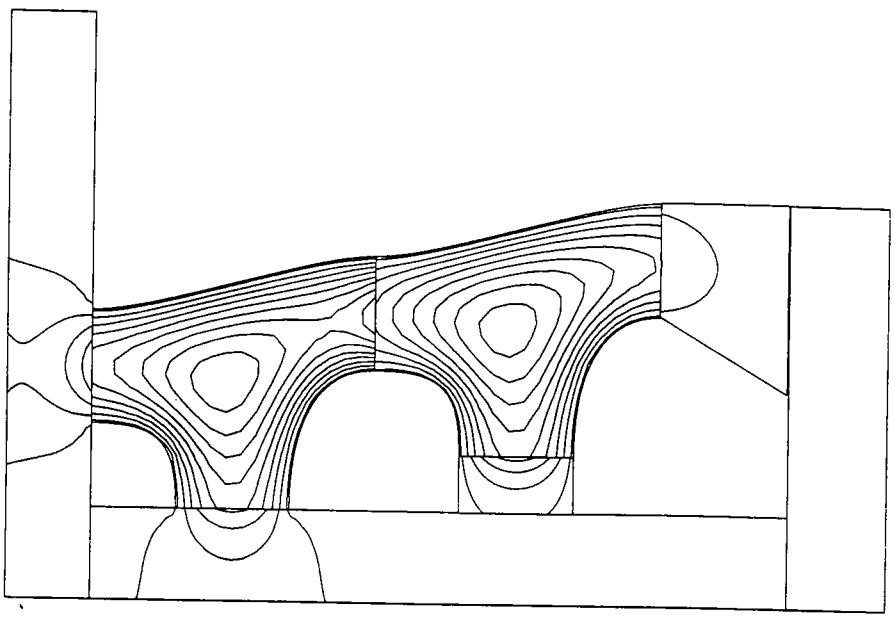


Figure 12: Effect of one iteration of interface relaxation over all the subdomains. The contour plots are produced by each subdomain object independently.

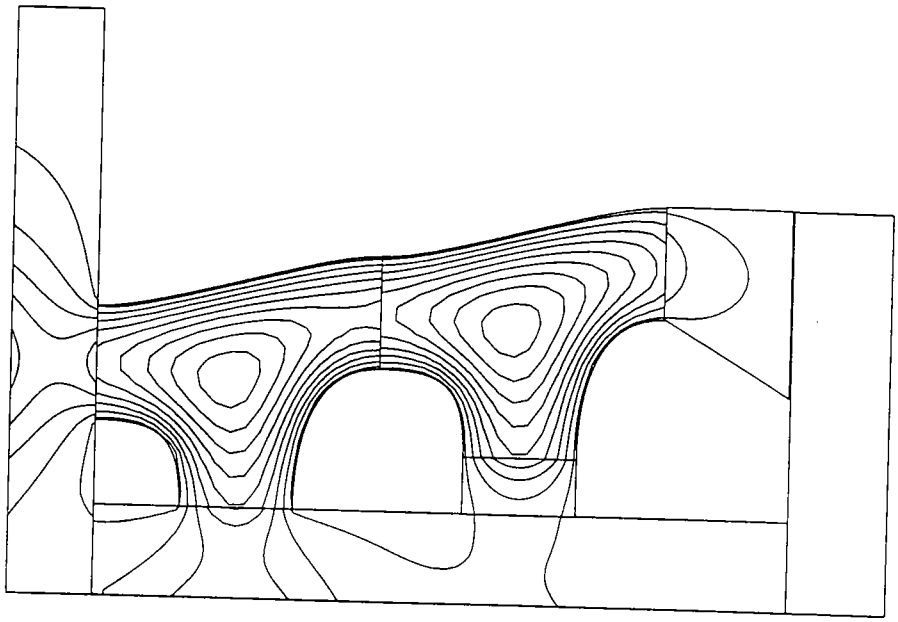


Figure 13: Temperature distribution after three iterations of interface relaxation. The convergence of the method is now apparent.

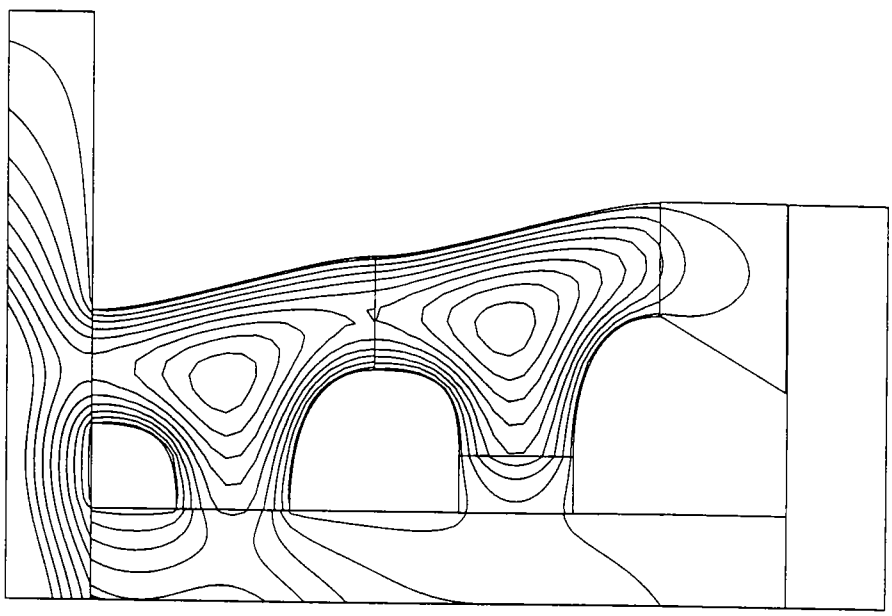


Figure 14: Temperature distribution after 10 iterations.

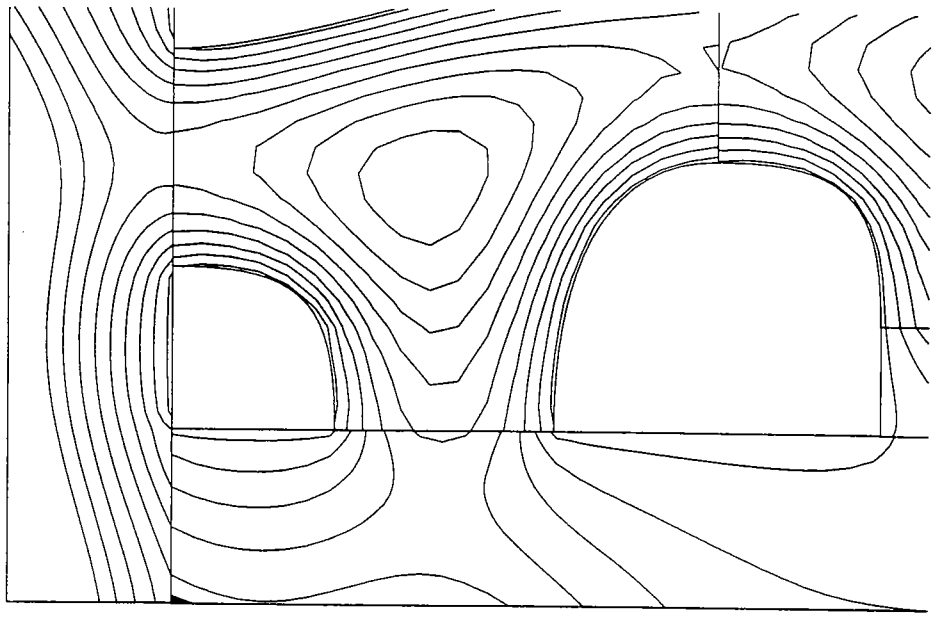


Figure 15: Close up view of the temperature distribution after 13 iterations. The discontinuities of the contours are due to the contour plotting method and not to inaccuracies in the PDE solutions.

References

[1] Agoshkov, V.I., Poincaré-Steklov's operator and domain decomposition methods in finite dimensional spaces, (Glowinski, Golub, Meurant and Periaux, eds.), SIAM Publications, Philadelphia, (1988), pp. 73–112.

[2] T. Chan, R. Glowinski, G. Meurant, J. Periaux and O. Widlund, *Decomposition Methods for Partial Differential Equations – II, – III*, SIAM Publications, Philadelphia, PA (1990), (1991).

[3] Chan, T., T. Hou and P.L. Lions, Geometry related convergence results for domain decomposition algorithms, *SIAM J. Numer. Anal.*, **28** (1991), 378–391.

[4] Funarro, O., A. Quarteroni and P. Zanolli, An iterative procedure with interface relaxation for domain decomposition methods, *SIAM J. Numer. Anal.*, **25**, (1988), pp. 1213–1236.

[5] Glowinski, R., G. Golub, G. Meurant, and J. Periaux, *Domain Decompsition Methods for Partial Differential Equations – I*, SIAM Publications, Philadelphia, PA (1988).

[6] Glowinski, R., Y. Kuznetson, G. Meurant, J. Periaux, and O. Widlund, *Domain Decomposition Methods for Partial Differential Equations – IV*, SIAM Publications, Philadelphia, PA (1991).

[7] Lions, P.L., On the Schwarz alternating method, *First Intl. Symposium on Domain Decomposition Methods for PDEs*, (R. Glowinski et al., eds.), SIAM, 1988, pp. 1–42.

[8] McFaddin, H.S., *An Object-Oriented System for Collaborating PDE Solvers*, Ph.D. Thesis, Computer Science Department, Purdue University, 1992.

[9] McFaddin, H.S., and J.R. Rice, RELAX: A software platform for PDE interface relaxation methods, in *Expert Systems for Scientific Computing* (E. Houstis, J. Rice and R. Vichnevetsky, eds), North-Holland (1992) to appear.

[10] Mu, M. and J.R. Rice, Performance of PDE sparse solvers on hypercubes, in *Unstructured Scientific Computation on Multiprocessors*, (P. Mehrotra, J. Sultz and B. Voigt, eds.), MIT Press, (1991), to appear.

[11] Quarteroni, A., and A. Valli, Theory and application of Steklov Poincarè operators in boundary value problems, in *Applied and Industrial Mathematics*, (R. Spigler, ed.), Kluwer Publishing Co., Dordrecht, 1991, pp. 179–203.

[12] Quarteroni, A., Domain decomposition method for the numerical solution of partial differential equations, Univ. Minn. Supercomputer Institute Rpt 90/246, Dec. 1990, 54 pages. Also in *Surveys for Mathematics in Industry*, (XXX, ed.), Springer-Verlag (1991) to appear.

[13] Schwartz, H.A., Ueber einige abbildingsanfgaben, *Jour. f. die reine and angew, Math*, **70**, (1869), pp. 105–120.

[14] Tang, Wei-Pai, *Schwarz Splitting and Template Operators*, Ph.D. Thesis, Department of Computer Science, Stanford University, July 1987.

[15] Tang, Wei-Pai, Relief from the pain of overlap generalized Schwarz splittings, Research Report CS-89-04, Department of Computer Science, Waterloo, Ontario, Jan 1989.