1994

Modeling with Collaborating PDE Solvers - Theory and Practice

Mo Mu

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

Report Number:
94-056

MODELING WITH COLLABORATING PDE SOLVERS -THEORY AND PRACTICE

Mo Mu
John R. Rice

CSD-TR-94-056
August 1994

MODELING WITH COLLABORATING PDE SOLVERS -THEORY AND PRACTICE

MO MU* AND JOHN R. RICE†

Abstract. We consider the problem of modeling very complex physical systems by a network of collaborating PDE solvers. Various aspects of this problem are examined from the points of view of real applications, modern computer science technologies, and their impact on numerical methods. The related methodologies include *network of collaborating software modules*, *object-oriented programming* and *domain decomposition*. We present an approach which combines independent PDE solvers for simple domains which collaborate using interface relaxation to solve complex problems. The mathematical properties and application examples are discussed. A software system RELAX is described which is implemented as a platform to test various relaxers and to solve complex problems using this approach. Both theory and practice show that this is a promising approach for efficiently solving complicated problems on modern computer environments.

Key words. domain decomposition, iterative methods, relaxation, partial differential equations, parallel computation, networks, object-oriented-methods

AMS(MOS) subject classifications. 65N55, 65F10, 65Y05, 65c20

1. Introduction.

Modeling physical phenomena with scientific computing is an interdisciplinary effort involving engineers, mathematicians and computer scientists. Practical physical systems are often mathematically modeled by complicated partial differential equations (*PDEs*). The computer simulation of these systems requires very large software systems. The numerical solution relies on large-scale computation using high performance computers and efficient algorithms. Therefore, the design of numerical PDE algorithms must balance these factors. In practical applications, one should be able to handle the complexity and generality of PDE problems. The numerical methods used must be fast and accurate. Among the major concerns for software development are software productivity, complexity, reusability, maintenance, portability, and other quality issues. High-performance computers are parallel which leads to issues such as parallel algorithms, communication cost, and scalability. Obviously, many of these objectives conflict with each other. The principal trade off is software development effort versus execution time efficiency. We examine various aspects of the simulation problem from the practical point of view. These considerations lead to the domain decomposition approach which we call *collaborating PDE solvers*. It aims to solve complex physical problems using modern computer science technologies combined with a classical relaxation idea of iteratively solving local problems and adjusting interface conditions. A software system RELAX has been implemented as a platform to support this approach. One can use this system to model complex physical objects, specify mathematical problems and test various interface relaxation schemes. It is shown that this approach is promising in both theory and practice. A shorter version of this paper has been published earlier [7].

* Department of Mathematics, The Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong, mamu@uxmail.ust.hk

† Computer Sciences Department, Purdue University, West Lafayette, IN 47907 U.S.A., rice@cs.purdue.edu. Work supported in part by the Air Force Office of Scientific Research grants, 88-0243, F49620-92-J-0069, Strategic Defense Initiative through Army Research Office contract DAAL03-86-K-0106, and the Advanced Research Projects Agency through Army Research Office contract DAAH04-94-G-0010.

2. Collaborating PDE solvers. A physical system in the real world normally consists of a large number of components which have different shapes, obey different physical laws, and interact with each other through interface conditions. Mathematically, it corresponds to a very complicated PDE problem with various formulations for the geometry, PDE, and interface/boundary conditions in many different regions. As a typical example an automobile engine system is shown in Fig. 2.1. Many other applications are given in [9] where different types of PDE's on different domains are coupled with suitable interface conditions, typically the velocity continuity and flux balance. One can imagine the great difficulty in creating a software system to model accurately such a complicated real problem. Therefore, one needs an effective software development mechanism which first, is applicable to a wide variety of practical problems, second, allows for the use of advanced software technologies in order to achieve high productivity and quality, and finally, is suitable for some reasonably fast numerical methods.

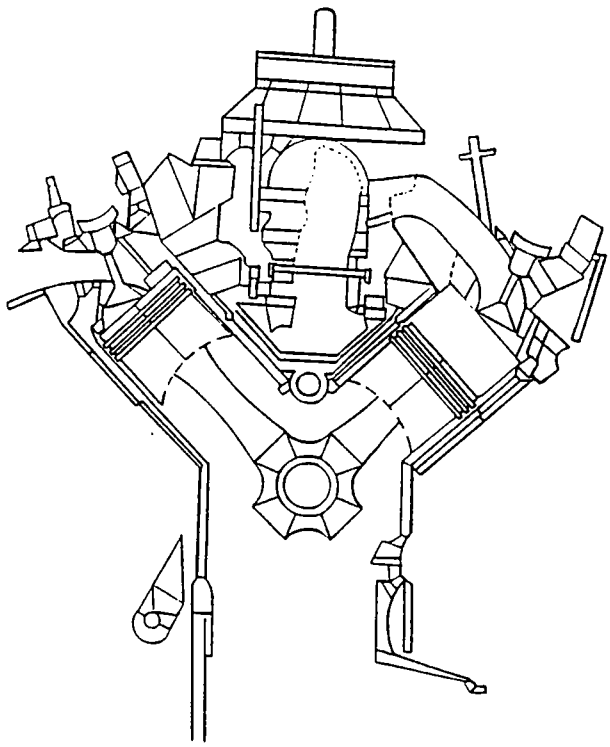

FIG. 2.1. *An automobile engine system consisting of many different parts.*

Most physical systems in practical applications can be modeled as a mathematical *network*. Here a network is a directed graph consisting of a set of nodes and edges. If we represent each physical component in a system by a node, then a pair of neighboring components are linked by an edge in the graph, with the edge directions used to indicate the transmission of interface information. Each node in the network contains a mathematical model for the local physical law for the corresponding component. For numerical relaxation one may also assign certain weights to each edge in order to provide detailed control for the interface adjustments, such as for boundary values and their jumps across the interfaces. Fig. 2.2 illustrates a four-node network corresponding to a heat flow problem.

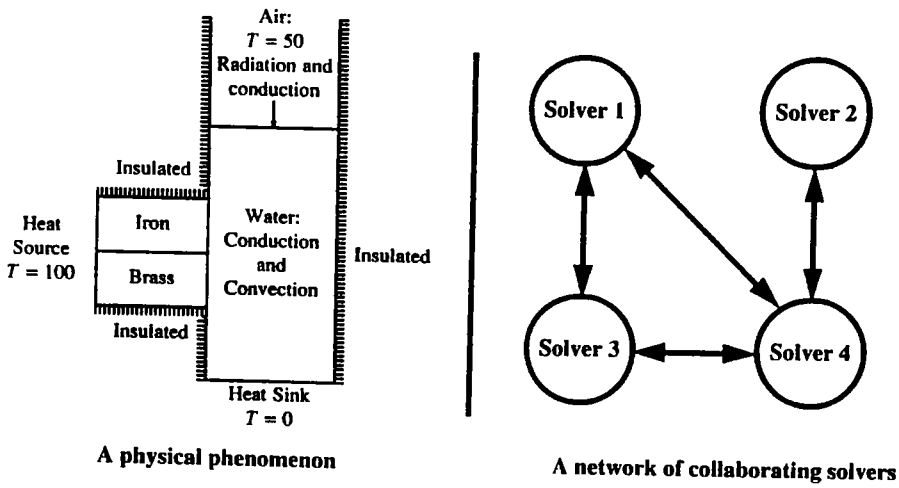

FIG. 2.2. *A four-node network corresponding to a heat flow problem.*

Usually, individual components are simple enough so that each node corresponds to a simple PDE problem with a single PDE defined on a regular geometry. There exist many standard, reliable PDE solvers that can be applied to these local node problems. To solve the global problem, we let these local solvers collaborate with each other to relax the interface conditions. An interface controller collects boundary values from neighboring subdomains and adjusts interface conditions according to the network specifications. Therefore, the network abstraction of a physical system allows us to build a software system which is a network of collaborating PDE solvers. These networks can be very big for major applications. There are normally about 5 interfaces per subdomain. For a highly accurate weather prediction, for example, one needs 3 billion variables in a simulation with continuous input at 50 million places. This assumes a 3-D adaptive grid, otherwise the computation is much larger. Very optimistically, if one needs a new forecast every 2-3 hours, the answer is 100 gigabytes in size and requires 80 mega-giga FLOPs to compute. An "answer" is a data set that allows one to display an accurate approximate solution at any point. It is much smaller than the computed numerical solution. Such a network roughly consists of 3,000 subdomains and 15,000 interfaces. Another example to consider is a realistic vehicle simulation, where there are perhaps 100 million variables and many different time scales. This problem has very complex geometry and is very non-homogeneous. The answer is 20 gigabytes in size and requires about 10 tera FLOPs to compute. The network has 10,000 subdomains and 35,000 interfaces.

A software network of this type is a natural mapping of a physical system. It simulates how the real world evolves and thus normally produces a reasonable solution. It allows various advanced software technologies to be applied to create a high quality system in a very productive way. For instance, one can apply the networking technology to efficiently integrate a collection of software components into an entire system and to implement a neat and flexible system architecture for the model and its interface connections. This allows the use of the software parts technology (object-oriented programming) that is the natural evolution of the software library idea with the addition

of software standards. It allows software reuse for easy software update and evolution which are extremely important in practice. The real world is so complicated and diverse that we believe it is impractical to build monolithic, universal solvers for such problems. Without software reuse, it is impractical for anyone to create on his own a large software system for a reasonably complicated application. For example, automobile manufacturers frequently change automobile models. Each change normally results in a new software system. Recreating such a system could easily take several months or years. In contrast, the execution time to perform the required computation might only be a few hours. Notice that such a physical change usually corresponds to replacing, adding, or deleting a few nodes in the network with a corresponding change in interface conditions. These are simple manipulations on a network without affecting the rest part of the entire system and can thus be easily done. This approach can and should be implemented using the object-oriented programming methodology. In this application each physical component can be viewed both as a physical object and as a software object. Actions and interactions of objects are clearly defined by the network. Two basic principles of object-oriented programming are data structure abstraction and information hiding for each object. These principles are expressed here by the local solvers and the interface conditions. In addition, this network approach is naturally suitable for parallel computing as it exploits the potential parallelism in physical systems. One can easily handle issues like data partition, assignment, and load balancing on the physics level by the structure of a given physical system. Synchronization and communication are controlled by the network specification and restricted to interfaces of subdomains, which results in a coarse-grained computational problem. This is especially suitable for today's advanced parallel supercomputer architectures. The network approach also allows high scalability. Finally, this network approach naturally fits into the mathematical domain-decomposition framework with the overall geometry being viewed as automatically partitioned into a collection of subdomains and interfaces which simply correspond to the network nodes and edges, respectively.

Many types of domain decomposition methods have been proposed over the past decade, see [1, 3, 4, 12] for general references. However, not all of them are suitable for, or directly applicable to, this network framework. First, many methods use artificial subdomain overlapping for mathematical convergence purposes and this obviously violates the basic principles of object-orientation. Each software object should correspond to a natural physical component without knowing part of the local data structures of neighboring objects. Second, it is not practical to apply the algebraic type of domain decomposition methods that first discretize a PDE problem on an entire domain and then partition the discrete system according to the geometric decomposition. In fact, the network framework implies that the problem partition is made on the continuous problem level so that PDE solution techniques in different regions may be totally independent depending on local properties. One may use finite differences for one subdomain, and finite elements or even an analytic solution for another. In addition, the subdomain PDE operators are not necessarily extensible to interfaces so that global discretization is not always applicable. More importantly, the efficiency of most of these methods relies on finding a good preconditioner for the interface Schur complement matrix, which is very difficult to do in practice for a complicated physical system. Another well-known class of methods are motivated by observing that in many physical applications the global solution U on an entire

domain satisfies certain continuity or flux balance conditions on interfaces involving U and $\partial U/\partial n$. A method which tries to match the continuity for both U and $\partial U/\partial n$ is as follows. One starts with an initial guess for U and $\partial U/\partial n$ on interfaces and then takes them as boundary data to solve a Dirichlet or Neumann boundary value problem on all the subdomains. These solutions are then used to update the interface vaue for U and $\partial U/\partial n$ and one iterates until convergence. This is referred to as the *subdomain-iteration* approach in the literature. There are two common alternatives, called alternating Dirichlet-Neumann in space and in time, respectively. The former imposes Dirichlet boundary condition on one side of an interface and Neumann condition on the other side. The latter imposes the same type of boundary condition on both sides of an interface in one iteration, but then alternates the Dirichlet and Neumann types in the next iteration step. These are non-overlapping methods. In certain cases, it can be shown [10] that the Dirichlet-Neumann approach corresponds to a preconditioned Richardson scheme applied to the reduced interface problem. In general, it is difficult to analyze the convergence, especially when cross points are present on interfaces. The choice of the convergence parameter in this approach is also not easy. In addition, interface conditions in practical applications are usually more complicated. Nevertheless, this subdomain-iteration based approach best fits into the network of collaborating PDE solvers framework and is thus most promising from the practical point of view. The challenge is then to extend it to general interface conditions and to guarantee its fast convergence.

3. Interface Relaxation. We now present a general mathematical formulation of the problem and define a general subdomain-iteration approach based on the classical relaxation idea.

Let Γ_{ij} be a typical interface, that is, the common boundary piece of two neighboring subdomains Ω_i and Ω_j, i.e., $\Gamma_{ij} = \partial\Omega_i \cap \partial\Omega_j$. Each subdomain obeys a physical law locally. Namely, there is a PDE L_l and function U_l defined in each Ω_l so that

$$L_l U_l = f_l \quad \text{in } \Omega_l \quad \textit{for } l = i, j. \tag{3.1}$$

There is an interface condition on Γ_{ij} which can usually be specified in the form

$$g_{ij}(U_i, U_j, \frac{\partial U_i}{\partial n}, \frac{\partial U_j}{\partial n}) \;=\; 0. \tag{3.2}$$

In some applications, the left-hand side of (3.2) may also involve higher order derivatives but we only consider first order derivatives here. For example, for the continuity conditions of the solution and its normal derivative discussed earlier, (3.2) takes the form

$$(U_i - U_j)^2 + \left(\frac{\partial U_i}{\partial n} - \frac{\partial U_j}{\partial n}\right)^2 \;=\; 0. \tag{3.3}$$

Note how satisfying two conditions can be formulated in the form (3.2). For some physical phenomena we might have different conditions to be satisfied on opposite sides of the interface so that the interface conditions need not be symmetric, i.e., we can have $g_{ij} \neq g_{ji}$. Denote by $BV(U_i, U_j) \equiv \{U_i,\ U_j,\ \frac{\partial U_i}{\partial n},\ \frac{\partial U_j}{\partial n}\}|_{\Gamma_{ij}}$ the data set of

boundary and derivative values of local solutions U_i and U_j on Γ_{ij}. Equation (3.2) can then be viewed as a constraint on $BV(U_i, U_j)$. The general PDE problem to be solved is thus

$$\begin{aligned} L_l U_l &= f_l \quad \text{in } \Omega_l \text{ for all } l \\ g_{ij}\left(U_i, U_j, \tfrac{\partial U_i}{\partial n}, \tfrac{\partial U_j}{\partial n}\right) &= 0 \quad \text{on } \Gamma_{ij} \text{ for all } i, j \end{aligned} \tag{3.4}$$

We now describe a general subdomain-iteration procedure as follows. Suppose that we have an initial guess for BV, denoted by BV^{old}, which satisfies the constraint (3.2) for all interfaces. For each subdomain, we solve the boundary value problems with the corresponding PDEs in (3.1) and by using part of BV^{old} as the boundary data. With the newly computed local solutions, denoted by U_l^{new} for Ω_l, we then evaluate their boundary values to get $BV(U_i^{new}, U_j^{new})$ for all Γ_{ij}, which is denoted by BV' for brevity. In general, BV' does not satisfy the constraint (3.2) although part of BV^{old} may be preserved in BV' as the boundary data used in the local solve. The relaxation idea is to further change, i.e., to *relax*, certain components in BV' to obtain a new data set BV^{new} that (better) satisfies the constraint (3.2). This leads to solving equation (3.2) for the corresponding boundary components as the unknowns. The above two-phase procedure, consisting of local PDE solve and constraint relaxation, defines a mapping from BV^{old} to BV^{new}. Iterating this procedure until convergence, we then obtain the global solution that satisfies both the local PDEs and interface constraints.

It is easy to have an object-oriented implementation of this relaxation procedure. The actions defined on a subdomain object are (a) solving a PDE boundary value problem with the provided boundary data in BV from interfaces and (b) evaluating boundary values of the resulting local solution. The actions defined on an interface object are (a) collecting boundary values from neighboring subdomains, (b) checking for convergence by examining the interface constraints, (c) relaxing the constraint to update BV, and (d) invoking local solvers for neighboring subdomains.

There are various choices possible for the relaxation, depending on the boundary condition type for each subdomain solve and the way of relaxing the interface constraint. The alternating Dirichlet-Neumann approaches described in Section 2 can be viewed as two examples. An interesting alternative is to apply a relaxing or smoothing procedure along an interface, which blends the neighboring solutions to better satisfy the interface constraints along the interface. This alternative is more general but it converges more slowly on model problems where additional properties can be exploited. It is also possible to apply least squares to perform an overdetermined interface constraint relaxation rather than an exact relaxation. As usual in iterative methods, one may use a multi-step type of relaxer using certain relaxation parameters and taking a weighted average of previous and updated iterates. In addition, preconditioning and other acceleration techniques may also be combined with relaxation.

We consider the following class of relaxers which has been found effective in experiments. First, we consider only *stationary relaxers*, those that use the same relaxation and PDE solution techniques at every iteration. There are non-stationary relaxers of serious interest, such as those that alternate between satisfying Neumann and Dirichlet conditions. Second, we consider only relaxers that use values and derivatives of PDE solutions along interfaces. That is, at each iteration a PDE is solved for U_l in

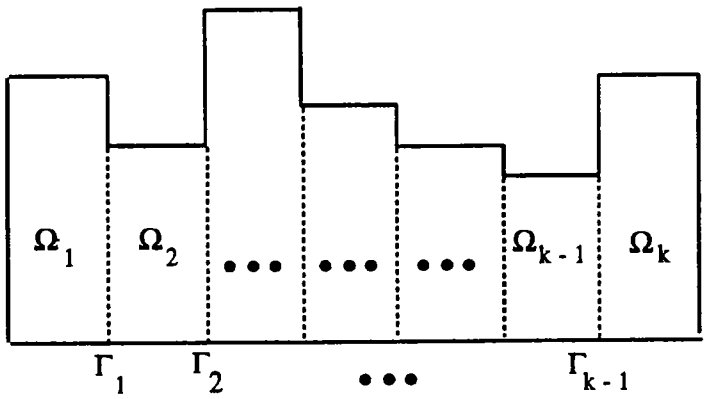

FIG. 3.1. *A "one dimensional" composite domain* Ω.

Ω_l and the boundary values of U_l and its derivatives are the input to the relaxers. Discrete versions of the relaxers may involve differences or combinations of U_l values on or near interfaces instead of derivatives of U_l.

We define this class of relaxers precisely as follows. Let $I(l)$ be the indices of those subdomains that are neighbors of subdomain l. Let the PDE problem that is solved on Ω_l be

$$
\begin{aligned}
& L_l U_l^{new} = f_l && \text{in } \Omega_l, \\
& B_{lj} U_l^{new} = b_{lj} && \text{on } \Gamma_{lj} \text{ for } j \in I(l), \\
& U_l^{new} \text{ satisfies the global boundary conditions on } \partial\Omega,
\end{aligned}
\tag{3.5}
$$

where B_{lj} is a usual boundary condition operator and b_{lj} is defined as part of the relaxer as follows. Let $\vec{X}_{lj}^{old}$ be the vector of values $(U_l^{old}, \frac{\partial U_l^{old}}{\partial n})$ which approximate the solution and its derivative on Γ_{lj} for $j \in I(l)$. The length of the vector $\vec{X}$ would be longer than 2 if the interface conditions involved higher derivatives. Then a *relaxer* is a procedure that maps U_l^{old}, $\vec{X}_{lj}^{old}$ for $j \in I(l)$, $\vec{X}_{jl}^{old}$ for $j \in I(l)$ into b_{lj}.

Note that this definition of relaxers makes them domain-based and not interface-based. That is, it is possible that $b_{ij} \neq b_{ji}$ even if Γ_{ij} and Γ_{ji} geometrically represent the same interface segment. It is more complicated at a cross point where several interfaces meet.

To be more specific and for the sake of simplicity, let us assume that Ω is as in Fig. 3.1 and denote $\Gamma_{i,i+1}$ simply by Γ_i. Furthermore, without loss of generality, we assume that the global solution vanishes on $\partial\Omega$, and the interior interface condition is (3.3). Suppose that we impose Dirichlet conditions on each Γ_{lj} for $j \in I(l)$ in (3.5). In this simplified example, the solutions on both sides of any interface Γ_i have the same boundary values on Γ_i, denoted by X_i, at each iteration. Thus we have $U_i - U_{i+1} = 0$ and Equation (3.3) is then reduced to

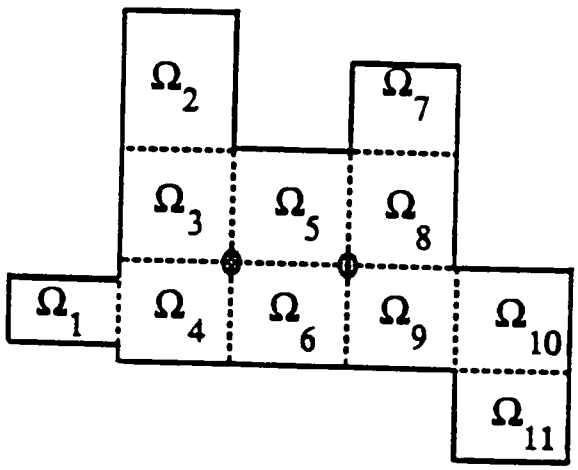

FIG. 3.2. *A general "two dimensional" composite domain with interior cross points (marked by "circles").*

$$\frac{\partial U_i}{\partial n} = \frac{\partial U_{i+1}}{\partial n} \quad \text{on } \Gamma_i \quad for \;\; i = 1, 2, \ldots, k-1. \tag{3.6}$$

The PDE solvers must handle derivatives discretely in some way, so we know that the derivatives in (3.6) at the interface point x_k are replaced by a difference approximation of some type, i.e.,

$$\begin{aligned} \tfrac{\partial U_i(x_k)}{\partial n} &= \alpha_0 X_{i-}^{new}(x_k) + \sum_m \alpha_m U_{i+}^{new}(x_m) \\ \tfrac{\partial U_{i+1}(x_k)}{\partial n} &= \beta_0 X_{i+}^{new}(x_k) + \sum_n \beta_n U_{i+1}^{new}(x_n) \end{aligned} \tag{3.7}$$

The points x_m and x_n are somewhere in the two domains. In (3.7) the coefficients $\alpha_0, \alpha_1, \ldots$ and $\beta_0, \beta_1, \ldots$ depend on the geometry of the subdomain, the meshes and discretizations used in each subdomain, and the choice of difference approximation. The discrete forms of equations (3.6) are thus

$$\begin{gathered} X_{i-}^{new} = X_{i+}^{new} \equiv X_i^{new} \\ \alpha_0 X_{i-}^{new} + \textstyle\sum \alpha_m U_i^{new} = \beta_0 X_{i+}^{new} + \sum \beta_n U_{i+1}^{new} \end{gathered} \tag{3.8}$$

or

$$X_i^{new} = \textstyle\sum \alpha_m U_i^{new}/(\alpha_0 - \beta_0) + \sum \beta_n U_{i+1}^{new}/(\alpha_0 - \beta_0) \tag{3.9}$$

Note that α_0 and β_0 will have opposite signs since the difference approximations in (3.7) are on opposite sides of the interface. As in general relaxation methods, one can further introduce some relaxation paramters or make use of other U_i^{new} and

U_{i+1}^{new} values or use previous values X_i^{old}, U_i^{old}, etc., in order to accelerate the overall convergence. For example, one can define X_i^{new} by

$$X_i^{new} = \omega X_i^{old} + (1-\omega) X_i^{temp} \tag{3.10}$$

where X_i^{temp} is the value obtained in (3.9) and ω is a relaxation parameter. Recall that $X_i^{old} = U_i^{new}|_{\Gamma_i} = U_{i+1}^{new}|_{\Gamma_i}$ in this example. In general, we see that a *linear relaxer* can be expressed as

$$X_i^{new} = \varphi_i(U_i^{new}, U_{i+1}^{new}) \quad \text{for } i = 1, 2, \ldots, k-1, \tag{3.11}$$

where φ_i is a linear combination of U_i^{new} or U_{i+1}^{new} restricted to grid lines near to the interface Γ_i with certain weights. The choice of φ_i depends on the interface condition (3.2), the approximation accuracy of the finite differences to the normal derivatives, the relaxation techniques, and so on. If preconditioning is used, φ_i is defined by solving a linear system of equations.

We may combine solving (3.5) for $\{U_l^{new}\}_{l=1}^k$ with (3.11) to obtain the matrix representation of $\{X_i^{new}\}$ in terms of $\{X_i^{old}\}$. The convergence analysis of the relaxation process is then reduced to the standard spectral analysis of the corresponding iteration matrix. We show in [8] that this iteration converges for the class of relaxers as described above and for general elliptic PDE problems and the domain decomposition with cross interface points as shown in Fig. 3.2. Furthermore, under certain model problem assumptions for a rectangular domain as decomposed in Fig. 3.1, an explicit expression is obtained for the spectrum of the iteration matrix so that the convergence mechanism is fully understood. In addition, the optimal relaxation parameters are also determined. Extensive numerical experiments are reported in [8] to support the theoretical convergence analysis and to demonstrate the practicality of the method.

4. RELAX problem solving environment. In this section, we describe the problem solving environment RELAX [5] that is implemented as a platform to support the collaborating PDE solvers approach. RELAX provides both a computational and user interface environment. The computational environment coordinates teams of single-domain PDE solvers, which collaboratively solve composite PDE systems like (3.4) that model complex physical systems. The user interface environment coordinates multiple interactive user interface components, called editors, which display or alter any feature of a composite PDE problem. Editors may be both text-oriented (e.g., equation editors) and graphics-oriented (e.g., solution plotters).

RELAX is implemented using the object-oriented programming technology. The system architecture is based upon a set of inter-communication software components. Editors and single-domain PDE solvers are examples of RELAX components - these particular ones are externally supplied (perhaps from libraries or other software systems). RELAX provides a message passing mechanism for supporting the inter-component communication. Is is capable of integrating existing scientific software for PDEs into a broader problem solving environment. It also has the capability of using pre-existing display and interaction components to form a flexible, dynamic user interface. Fig. 4.1 illustrates the arrangement of the components of the RELAX architecture.

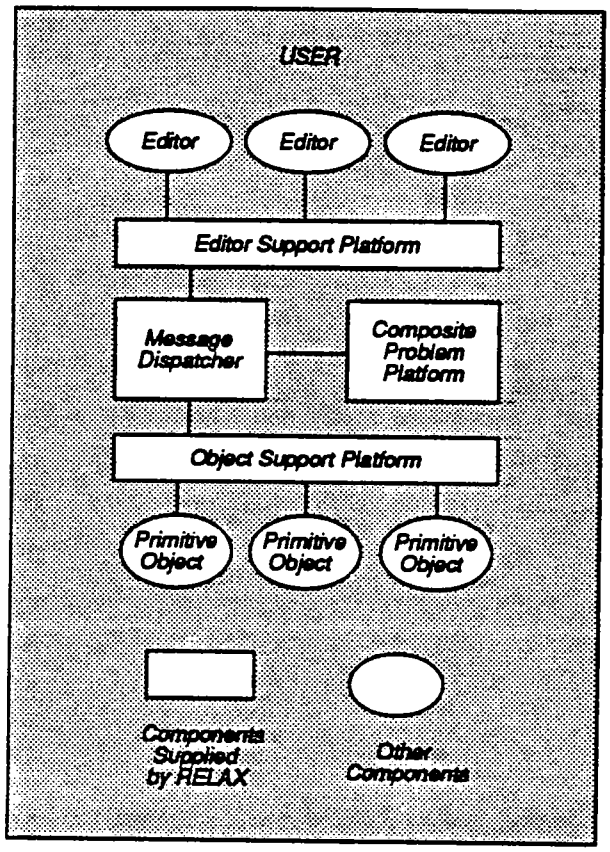

FIG. 4.1. *Architectural arrangement of RELAX. The elliptic shapes represent components which are supplied externally. The rectangular shapes represent system components.*

We briefly outline the function of each type of components and refer to [5, 6] for more details:

- *Primitive Objects*: These are externally supplied components which model and solve primitive PDE problems. Primitive objects are responsible for all aspects of solving a single-domain PDE problem, including the generation of numerical meshes, discretization of the PDE, and solving systems of equations.
- *Editors*: These are externally supplied components which provide an interface between the user and some feature of the system. The editor component is responsible for the complete presentation of the user interface, including all communication with the window system and/or graphics package.
- *Message Dispatcher*: This is a system supplied component which handles all transmission of messages within the system. The message dispatcher also registers RELAX objects and can assist editors in locating them.
- *Composite Problem Platform*: This is a system component which maintains the data structures defining a composite PDE problem. For example, the composite problem platform stores topological information about which primitive objects share geometric interfaces, as well as equations defining the interface conditions along those interfaces. Additionally, the composite problem platform maintains data structures defining a global solution iteration, and is capable of executing such iterations. Finally, the composite problem platform is capable of defining composite PDE problems hierarchically.
- *Object Support Platform*: This is a system component responsible for integrating primitive objects into the system. The object support platform provides

the attachment point for primitive objects (external components) that are dynamically attached to the running system. The object support platform relays messages between primitive objects and the message dispatcher.

- *Editor Support Platform*: This is a system component which provides an attachment point and communication interface for editors. The editor support platform relays messages to and from editors, and is also capable of parameterizing and controlling the message flow, for example, by copying and buffering messages.

With this environment, one can easily describe primitive PDE problems and interface conditions to compose a complex mathematical system, specify local solvers and relaxers to define an iterative procedure, and display the computed solution in various ways. Fig. 4.2 is a typical RELAX screen showing the user interface of some of the editors built to test and use the prototype system. Fig. 4.3 shows the user interface of an editor modifying the parameters of joining up two primitive objects called *box1* (a rectangular region) and *joint2* (a curved region). As an application example, we solve a physical heat flow problem as shown in Fig. 4.4 by the RELAX system. The complex object consists of seven simple subdomains with nine interfaces. The radiation conditions allow heat to leave on the left and bottom while the temperature U is zero on all the other boundaries. The mounting regions have heat dissipated. The interface conditions are continuity of temperature U and its derivative. Fig. 4.5 shows the solution computed after 15 iterations, where the initial guess is zero and the relaxer used is as described in Section 3 with the relaxation parameter $\omega = 0$.

5. Interface Conditions for Composite PDE Problems. One tool needed for this approach is an interface condition handler. The mathematical formulation (3.4) of the problem is rather simple but, once the problem is discretized, the discrete formulations usually becomes quite messy. That is, the coefficients in (3.7) are routine but very tedious to derive. It is, of course, true that any PDE solver method must be able to develop information about the PDE solution and its first derivatives at any point. However, this capability may be buried inside the PDE solver implementation and difficult to locate. In [11] it is shown how programs to provide values for $\frac{\partial U}{\partial n}$ at arbitrary points can be derived from many PDE solver codes. In [2] the tools are further developed for any PDE solver that can provide a "nearest points" procedure. That is, given (x, y) coordinates and an integer k, this procedure returns the k points nearest to (x, y) where an approxiamte solution to U is generated. This tool greatly eases the work in generating derivative values along interfaces and simplifies the relaxation implementation.

6. Conclusions. We examine in this paper various aspects of the real world simulation of complex PDE based applications with the emphasis on software productivity. Application of modern software technologies and the impact on numerical PDE methods are considered. We present a general approach for modeling complex physical systems by a network of collaborating PDE solvers. The related methodologies include *networks of collaborating software modules*, *object-oriented programming* and *domain decomposition* which lead to a subdomain-iteration procedure with interface relaxation. Various types of relaxers are discussed. A software system RELAX is described which is implemented as a platform to test various relaxers and to solve complex problems using the network of collaborating PDE solvers approach. Both

theory and practice show that this is a very promising approach for efficiently solving complicated problems on modern computer environments.

Acknowledgement. The RELAX environment is implemented by Dr. Scott McFaddin. We would like to thank him very much for providing the implementation details.

REFERENCES

[1] T.F. Chan, R. Glowinski, J. Periaux and D. Widlund, *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM Pubs., Philadelphia, PA. (1990).

[2] Tzvetan Drashansky and John R. Rice, Processing PDE interface conditions II, Tech. Rpt. CSD-TR-94-0XX, Dept. Computer Science, Purdue University, (August 1994).

[3] R. Glowinski, G. Golub, G. Meurant and J. Periaux, *First Intl. Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM Pubs., Philadelphia, PA (1988).

[4] D.E. Keyes, T.F. Chan, G. Meurant, J.S. Scroggs, and K.G. Voigt, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM Pubs., Philadelphia, PA (1992).

[5] Scott McFaddin and John R. Rice, RELAX: A platform for software relaxation, in *Expert Systems for Scientific Computing* (Houstis, Rice and Vichnevetsky, eds.) North-Holland, Amsterdam (1992), pp. 125–194.

[6] Scott McFaddin and John R. Rice, Collaborating PDE solvers, *Applied Numerical Mathematics*, **10** (1992), pp. 279–295.

[7] Mo Mu and John R. Rice, Modeling with collaborating PDE solvers: Theory and practice, *AMS Contemporary Mathematics*, (1994).

[8] Mo Mu and John R. Rice, Collaborating PDE solvers with interface relaxation, to appear.

[9] A. Quarteroni, F. Pasquarelli, and A. Valli, Heterogeneous domain decomposition: Principles, algorithms, applications, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equartions*, D. Keyes et. al., eds, SIAM Pubs., Philadelphia, PA (1992) pp. 129–150.

[10] A. Quanteroni and A. Valli, Theory and application of Steklov-Poicare operators for boundary value problems: The heterogeneous operator case, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, et. al., eds., SIAM, Philadelphia, 1991, pp. 58–81.

[11] John R. Rice, Processing PDE interface conditions, Tech. Rpt. CSD-TR-94-041, Dept. Computer Science, Purdue University, (June 1994).

[12] Jinchao Xu, Iteration methods by space decomposition and subspace correction, *SIAM Review* **34** (1992), pp. 581–613.

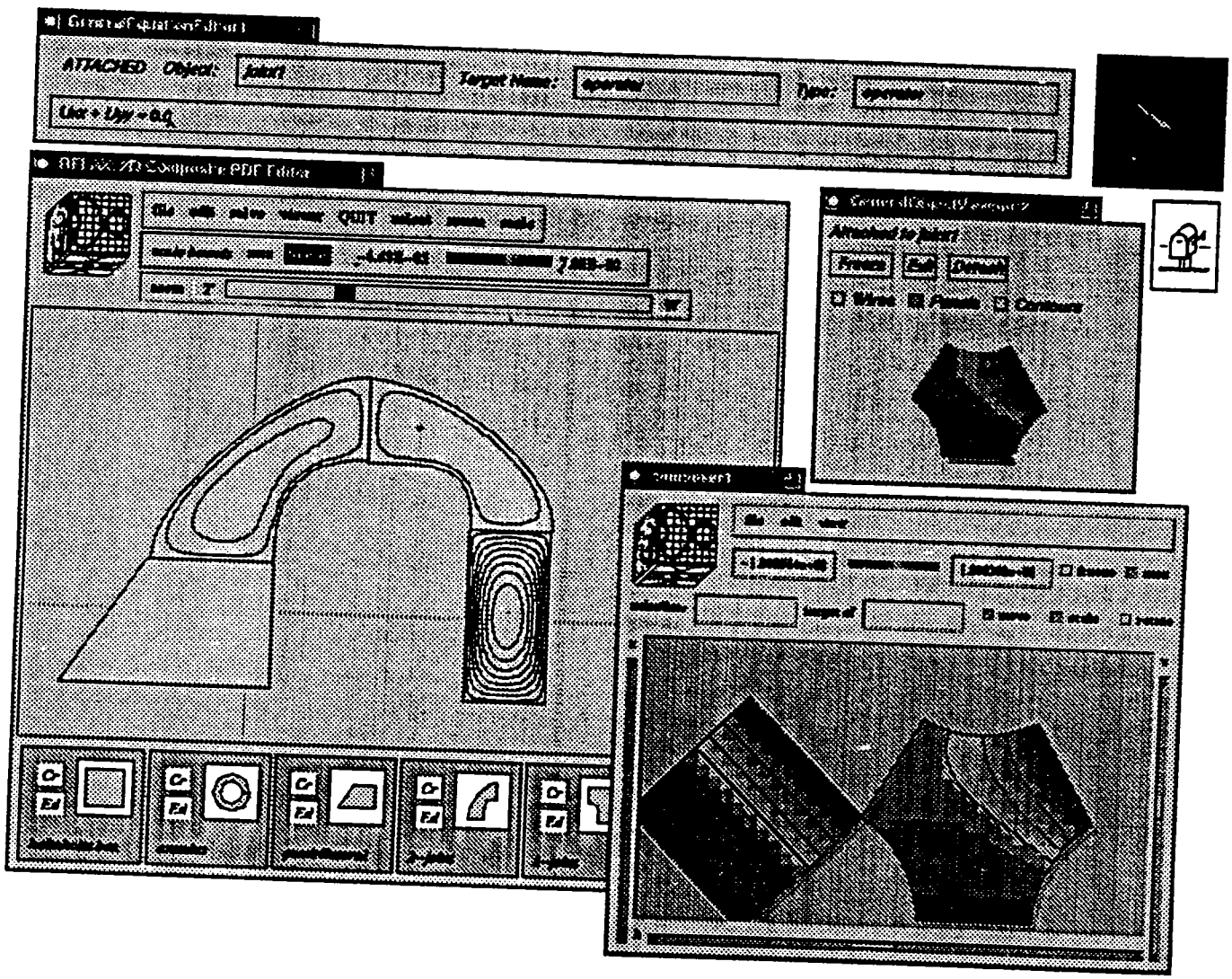

FIG. 4.2. *The RELAX multi-editor environment. A typical RELAX screen showing the user interface of some of the editors built to test and use the prototype system.*

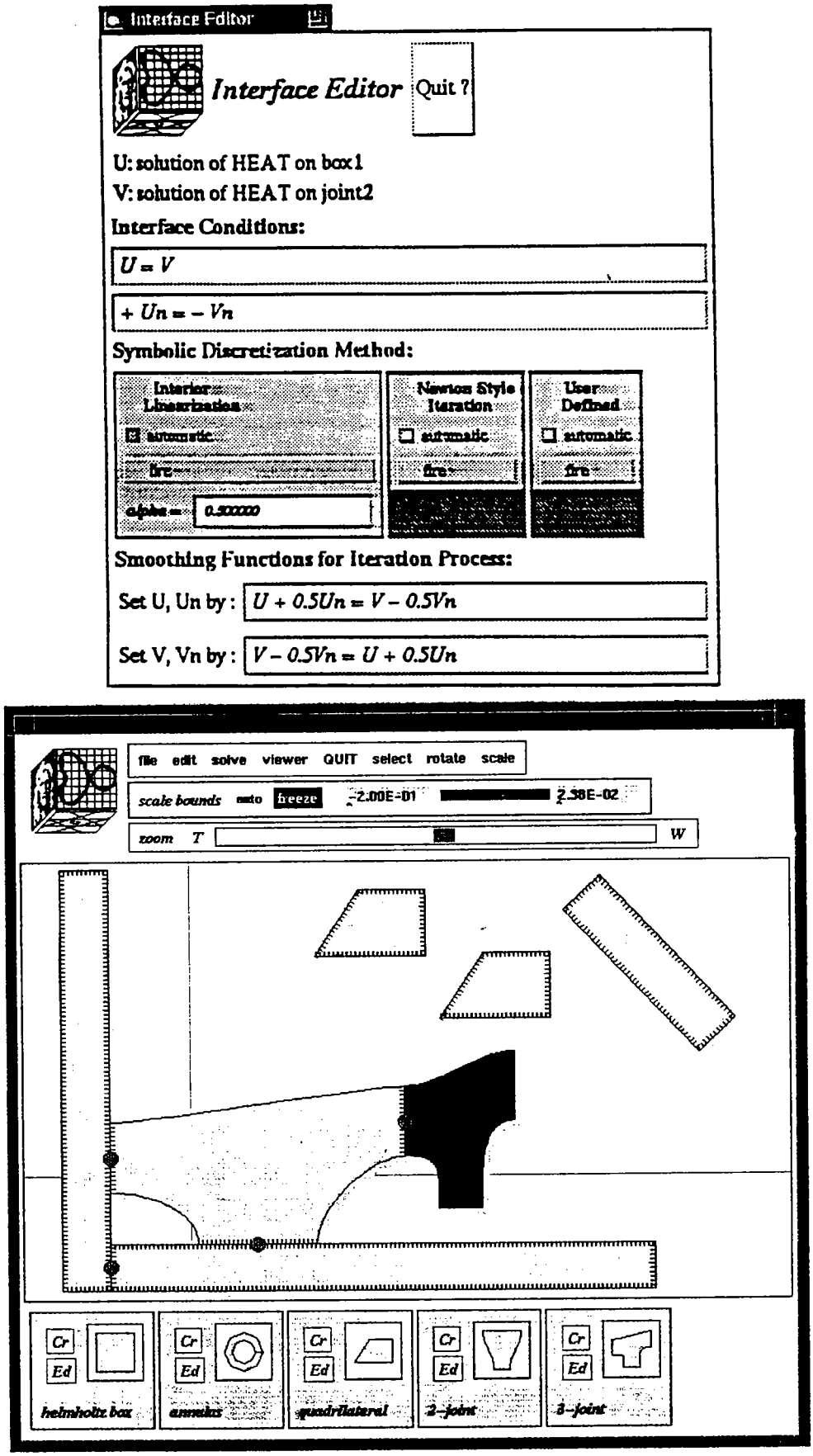

FIG. 4.3. *The user interface of an editor modifying the parameters of an adjacency. The adjacency is assigned to the intersection of geometric interfaces of primitive objects called box1 (a rectangular region) and joint2 (a curved region).*

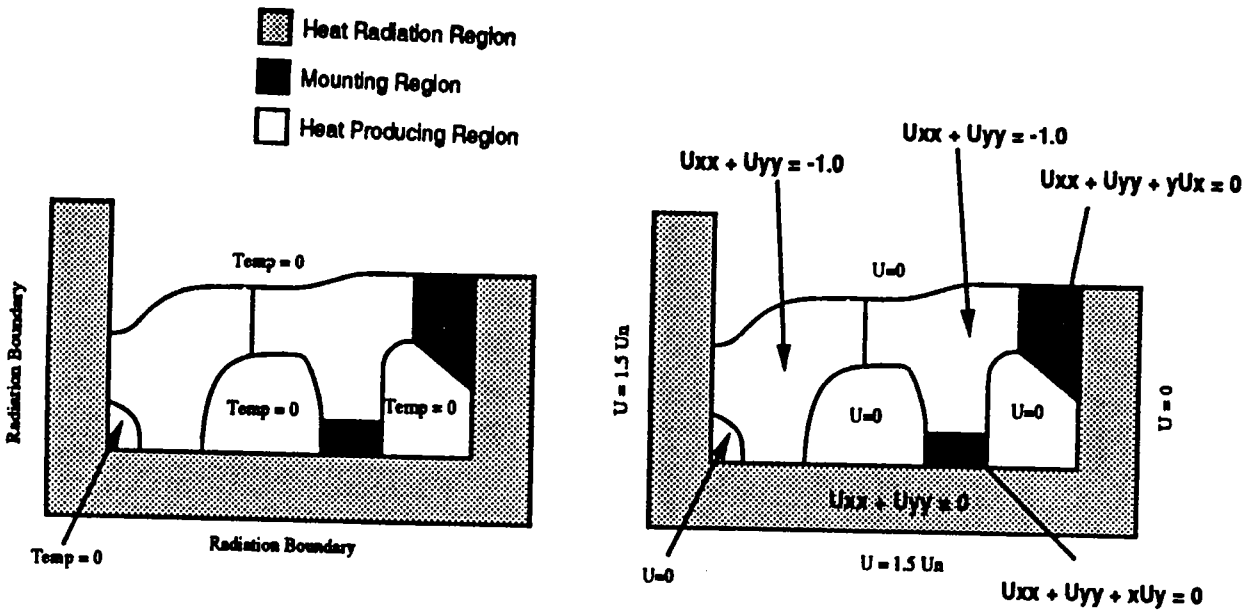

FIG. 4.4. *A physical heat flow problem for a complex domain along with the physical and mathematical descriptions.*

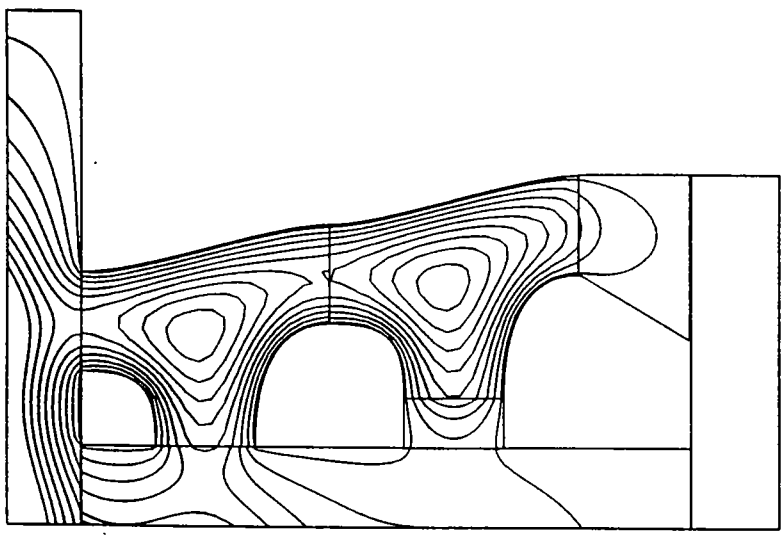

FIG. 4.5. *The contour plot of the solution computed after 15 iterations for the problem in Fig. 4.4.*