from *SIAM News*, Volume 31, Number 6

An Agent-based Architecture for Solving Partial Differential Equations

By John R. Rice

APPLICATIONS ON ADVANCED ARCHITECTURE COMPUTERS

Greg Astfalk, Editor

Composite or multi-physics partial differential equations arise in models involving multiple physical phenomena. A model of a pan of water coming to a boil, for example, involves heat conduction in a solid, plus both heat conduction and heat convection in water and in air. Figure 1 shows a device similar to a pan of boiling water, with these same physical phenomena. While the individual physical phenomena are modeled by PDEs in the usual way, their solutions must satisfy interface conditions between the domains of the phenomena. These interface conditions also model the physical behavior at the interfaces. In our simple example, the interface conditions correspond to the continuity of temperature and heat flux. If the water starts to boil, then the air–water interface becomes much more complicated as mass (steam) transfers from the water into the air, carrying heat with it. One of the challenges at the frontier of simulation technology is to solve the composite PDEs that arise in modeling more complicated devices (e.g., a complete internal combustion, or gas turbine, engine).

This article describes the *interface relaxation method* for solving composite PDEs and then presents a (virtual) architecture and software system, SciAgents [3], that uses this method and architecture. It is assumed that we can solve exactly any single PDE on any simple domain or, more realistically, that given such a PDE problem, we can select an exact solver for it from our library. The interface relaxation method uses a library of "single, simple-domain, exact" PDE solvers to solve composite PDE problems. It is an iterative method of the classical type, based on relaxation, as follows:

1. *Guess solution values (and derivatives if needed) on all interfaces.*
2. *Solve all single PDEs exactly and independently with these values as boundary conditions.*
3. *Compare and improve the values on all interfaces using a relaxer (discussed below).*
4. *Return to Step 2 until satisfactory accuracy has been achieved.*

Interface relaxation is thus in the spirit of Southwell's idea, as applied to the linear algebraic equations used in the 1930s. The simplest relaxers just take some sort of "average" of values on the interfaces. Averaging is a good mental model for a relaxation formula.

The attraction of the SciAgents architecture is threefold. First, it allows the accurate coupling of independent models and the reuse of PDE software that handles single-phenomenon models. Second, it uncouples the parallelism of the computation somewhat from that of the machines used. Finally, it is intuitively consistent with a person's view of the geometry and physical models of a composite PDE problem.

The Interface Relaxation Method

Interface relaxation is an iteration defined at the continuum (mathematical) level; its convergence properties are a question of mathematical analysis, not of numerical analysis. Let U_N and V_N be the PDE solution values at iteration N on opposite sides of an interface. Assume for simplicity that the interface conditions to be satisfied are continuity of value ($U_N = V_N$) and normal derivative ($\partial U_N / \partial n = -\partial V_N / \partial n$). Then two simple relaxation formulas for U and V are:

$$V_{N+1} = U_{N+1} = (U_N + V_N)/2 - f(\partial U_N/\partial n + \partial V_N/\partial n)$$

$$V_{N+1} = U_{N+1} = \omega U_N + (1-\omega)\left[\alpha(U_N - V_N)^2 + \beta(\partial U_N/\partial n + \partial V_N/\partial n)^2\right],$$

where f, ω, α, and β are relaxation parameters.

Early relaxation formulas were proposed in the context of domain decomposition [5, 6]. Approximately 10 relaxation formulas are catalogued in [9]. One of the important open questions concerns the comparative performance of these methods and procedures for computing good relaxation parameters. Of course, actual implementations of interface relaxation use numerical methods since, generally, the

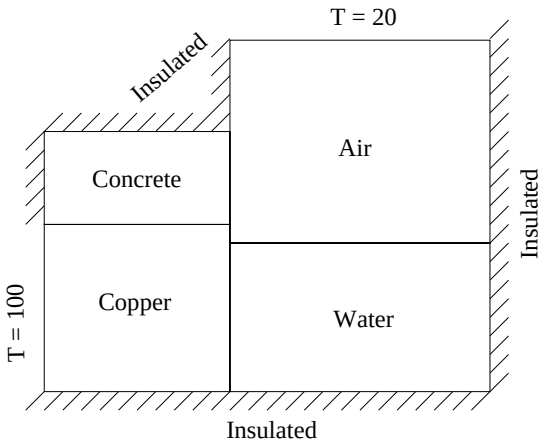

Figure 1. *A simple composite PDE problem with three phenomena, four simple domains, and five interfaces. Heat is transmitted across the interfaces according to standard models of heat flux and temperature continuity.*

exact analytic PDE solvers do not exist. The level of accuracy produced by the numerical solvers should be visualized as substantially higher than that of the relaxation iteration. There is negligible correlation between the accuracy of interface relaxation and that of the numerical solvers. This is similar to assuming that computer arithmetic is exact even though it is not (and this can sometimes cause problems). Interface relaxation looks like domain decomposition. It is different in that domain decomposition has a single underlying PDE for all the domains and the interface conditions are mathematical smoothness (continuous value and derivative) or some numerical equivalent.

The convergence analysis of interface relaxation presents formidable mathematical challenges; almost any question asked will be both hard and open. Even for the single-PDE case—one global PDE or domain decomposition—work on convergence analysis has appeared rarely, starting about 10 years ago [6, 8] and then more recently in 1992 [2, 7]. The analysis in the latter case is greatly simplified by the existence of a rich convergence theory for solving the single PDE that can be applied.

The difference between the two convergence problems can be illustrated as follows: Let U_{Nh} and U_{hN} be the solutions produced at iteration N by discretization h for interface relaxation and domain decomposition, respectively (for interface relaxation, h is merely a placeholder as there is no concept of a global discretization variable). The two methods and their convergence questions are then:

Domain Decomposition	***Interface Relaxation***
1. Discretize PDE using h to define U_h	1. Decompose problem into subdomains
2. Decompose problem into subdomains	2. Use interface conditions and iteration method to define U_N
3. Use interface conditions and iteration method to define U_{hN}	3. Discretize each PDE using h to define U_{Nh}
4. Iterate 3 on N until convergence	4. Let $h \rightarrow 0$ in 3 until convergence
5. Let $h \rightarrow 0$ in 1–4 until convergence	5. Iterate 2–4 on N until convergence
$\lim_{h \rightarrow 0} (\lim_{N \rightarrow \infty} U_{hN})$ = ??	$\lim_{N \rightarrow \infty} (\lim_{h \rightarrow 0} U_{Nh})$ = ??

For domain decomposition, if $\lim_{N \rightarrow \infty}$ can be handled, then the other limit can be handled by existing theory. This is not the case for interface relaxation as there is no convergence theory for multiple (or even one) discretization method applied to composite PDEs.

Given that theoretical analysis is intractable for the moment, we should use experiments to provide guidance and insight for interface relaxation. Numerous experiments done in recent years indicate that interface relaxation converges for a wide variety of problems and relaxers. The convergence is sometimes very fast, other times not. There is reason to be hopeful that, as we better understand interface relaxation, it can become a very useful method for solving composite PDEs. (A crude form of interface relaxation already in fairly widespread use simply involves “trading” current values across interfaces without any relaxation. This method makes the most sense in time-varying problems, but we are not aware of any attempts to analyze the effects of the errors involved.)

Implementation in High Performance Environments

Interface relaxation is naturally suited for distributed high performance computing. The method defines a mathematical network with a single PDE solver at each node (representing a domain) and relaxers connecting the nodes. The single (and usually different) PDE solvers are distributed to high performance machines (since many PDE problems have to be solved), and the relaxers “control” the computation.

We can extend the concept by making each node a solver agent capable of solving its PDE, with a goal of doing so whenever new boundary (interface) conditions are given to it. We can make each relaxer connection into a mediator agent capable of accepting values from solvers, applying relaxation formulas, and returning improved values to its two solvers. The goal of the mediator agent is to apply these formulas and to decide whether convergence has taken place (i.e., whether the values it receives satisfy the interface conditions sufficiently well). The entire method is placed in an agent-based framework through the introduction of a control agent that sets policies for the other agents—e.g., What are the tolerances for local convergence? Should a solver agent resolve its PDE when it gets each new interface value or wait until it gets all new interface values?

This agent-based framework creates a virtual architecture for each particular composite PDE problem. The virtual architecture must be mapped onto a real architecture, which brings other issues, e.g., load balancing, communication latency, into play. If one of the nodes has a dominating PDE solution time, then traditional domain decomposition methods can be used to partition this local PDE problem. An attraction of the agent-based approach, however, is that it cleanly separates the algorithmic and software issues from the hardware being used.

SciAgents is an agent-based implementation of interface relaxation built on the PDE-solving infrastructure of Parallel ELLPACK (PELLPACK) [5]. SciAgents integrates about a dozen PDE- solving packages (e.g., CADSOL, ELLPACK, FIDSOL, NSC2KE, PDECOL, VECFEM) into a uniform problem-solving environment; some of these packages, in turn, contain many individual solvers, so that the overall system contains over 1.5 million lines of code. For any particular PDE, of course, only one solver is used. While PELLPACK has one-, two-, and three-dimensional solvers, SciAgents is currently restricted to two-dimensional problems. A wrapper has been put around PELLPACK to make it into an agent for SciAgents. This wrapper involves adding or changing about 1000 lines of code in PELLPACK, an almost negligible amount given the

size of the entire code.

SciAgents has a facility for creating the network of solver and mediator agents for a composite PDE problem. With the templates for mediator agents, the user can opt for a default relaxer, select a relaxer from a menu, or (for experts) program a new relaxation formula. Thus, the creation facility relies heavily on the user interfaces (GUI) of the solver and mediator agents. The PELLPACK solver GUI is elaborate and provides many options for defining PDE problems, selecting among the solvers, and visualizing the solution. The SciAgents mediator GUI is much simpler, allowing the user to select one of eight built-in relaxation formulas or to program a new one.

To create a SciAgents network, the user makes a sketch of the composite PDE problem in order to identify (and number) the domains and interfaces, and then uses the agent GUI to define the network, to provide equations and parameters, and so forth. An example from the end of the process is shown in Figure 2, where the domain is displayed at the lower left. The window shows the (somewhat cryptic) text of SciAgents, in which the network data are consolidated. At the bottom of this text, machine names are given for each agent; as yet, SciAgents does not provide any automatic load-balancing or automatic machine-assignment facilities.

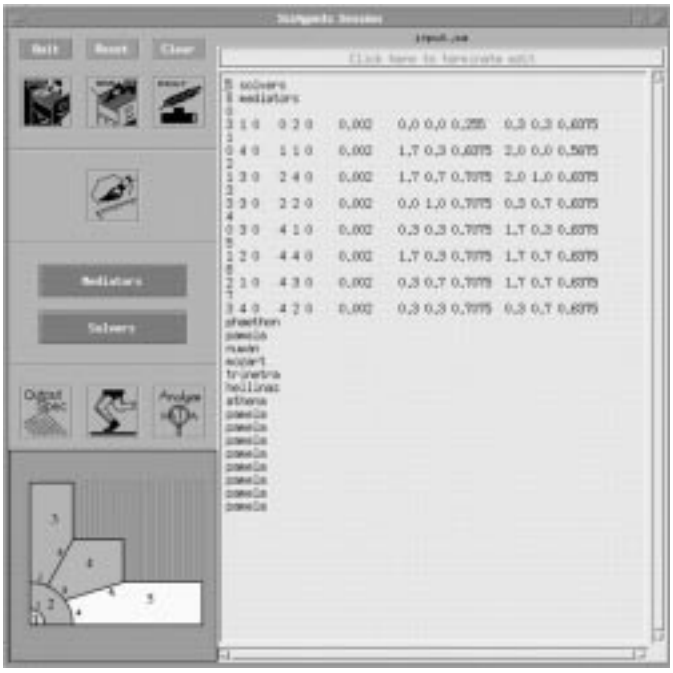

Figure 2. *The SciAgents control window.*

The KQML system [4] for information exchange, which has been used for a number of agent-based applications in the AI community, is used for the implementation of SciAgents. The SciAgents application stresses KQML in two respects, both of which would be expected to be common in scientific agent-based systems. First, the size of the information packets quickly exceeds the KQML limits, and a "work-around" must be implemented. Second, the KQML system degrades and then crashes as the number of inter-agent messages grows. Thus, the KQML-based implementation of SciAgents becomes unreliable at about 8 to 10 domains. We plan to use a more robust KQML implementation or a substitute system in the future. The SciAgents architecture for the composite PDE problem of Figure 1 is shown in Figure 3. The system components are shown, along with the problem-solving agents created for this application.

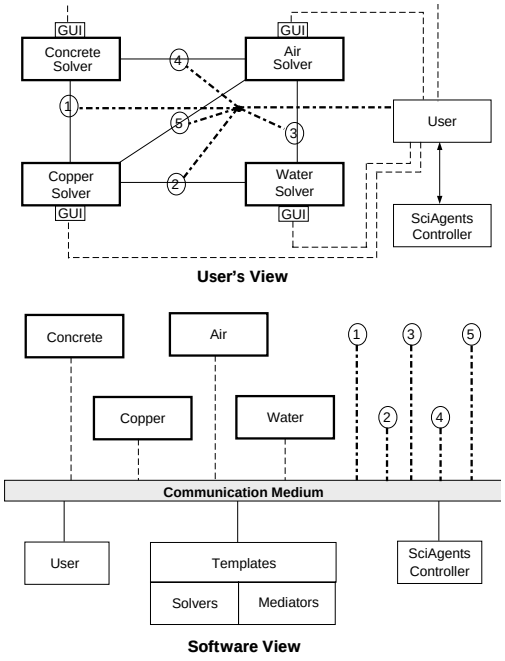

Figure 3. *The SciAgents architecture as seen by the user (top) and by the SciAgents software (bottom). The solver agents are in boxes with heavy borders, and the mediator agents are in ovals with numbers.*

Results

To date, we have solved approximately 200 two-dimensional composite PDEs using interface relaxation. Perhaps 150 of these problems are of the domain decomposition type, i.e., the same elliptic PDE is used on all domains. In these experiments we have varied the PDE, the number of domains (up to 500 have been used), the shapes of the domains, the relaxation formulas, and so forth. The other 50 have been truly composite PDE problems, usually linear elliptic PDEs. Again, the objective has been to explore the range of applicability of the interface relaxation method.

A handful of the test problems are nonlinear, including the following, which was solved on the geometry shown in the lower left of Figure 2 (and in which there are singularities at the reentrant corners):

$$UU_{xx} + (1 + U)U_{yy} + aU(1 + U) = b(x^2 + y^2 - 2) \qquad \text{Domain 1}$$
$$U_{xx}/(1 + (x - y)^2) + U_{yy}/(1 + (4x - 5y)^2) + cU/(101 + U) = 0 \qquad \text{Domains 2 \& 4}$$
$$U_{xx} + U_{yy} - d(U_x + U_y) + cU = 0 \qquad \text{Domain 3}$$

$$U_{xx} + U_{yy} + ae^{x + y + U/500} = b(x^2 + y^2 - 2) \qquad \text{Domain 1}$$
$$UU_{xx} + UU_{yy} + (U_x + 20)U_y + 2(U_x - 20)U_x = 0 \qquad \text{Domains 2 \& 4}$$
$$U_{xx} + U_{yy} - b(U_x + U_y) + aU = 0 \qquad \text{Domain 3}$$

The total human time required to solve a composite system on the domain shown in Figure 2 is about three hours. This

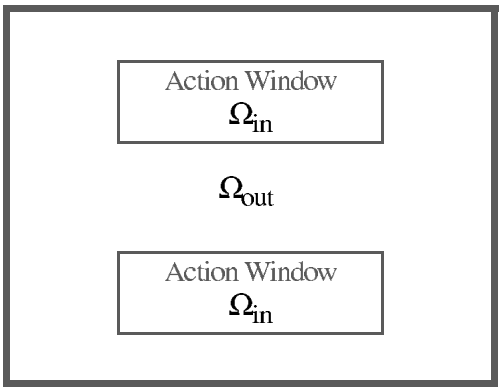

Figure 4. *Typical configuration of a two-window Josephson junction. The solution is specified on the external boundary, and two interface conditions are specified on the internal boundaries.*

includes the time to create the SciAgents solver, i.e., to make the sketch, to define completely four PDE problems with PELLPACK, to define the initial guess, which must be continuous along all the interfaces and equal to external boundary conditions, to create five mediators, to select relaxation formulas along with their parameters, and to assign all the agents to workstations on the local network. The solver agents are assigned to separate workstations, and everything else is assigned to another workstation. Sun SPARCstations 5, 10, and 20 have been used. This creation effort accounts for about half the time.

Once the composite PDE solution is started, there are significant waits (30 seconds to 2 minutes), depending on the time needed by a particular workstation to solve a particular PDE (four of them, of course, could be working simultaneously). For engineering accuracy, the solutions of the problems discussed here require a maximum of 53 iterations; because of the loosely coupled, agent-oriented control mechanism of SciAgents, not all the solvers are executed the same number of times. The singularities at the reentrant corners account in part for the slow convergence.

During this time the user can adjust the relaxation parameters. Once the global solution is obtained, some time is required to collect, save, patch together, and view its four pieces. The number of "iterations" needed for engineering accuracy usually ranges from 15 to 100; the number is not easily predicted and is strongly influenced by relaxation parameter values. We have also solved several hundred one-dimensional composite PDEs using interface relaxation.

Real-world applications in progress include a model for a window Josephson junction in superconducting films [1]. In this problem Ω_{in} is an action window region (see Figure 4) of a Josephson junction embedded in a global domain Ω; $\Omega_{out} = \Omega \setminus \Omega_{in}$ is the region without superconductivity. The phase difference $U(x,y)$ of the order parameter in the superconducting films satisfies the nonlinear sine-Gordon equation inside Ω_{in} and is harmonic outside:

$$\begin{cases} \nabla^2 U_{in} = \sin(U_{in}) & \text{in } \Omega_{in} \\ \nabla^2 U_{out} = 0 & \text{in } \Omega_{out} \end{cases}$$

These solutions are subject to the following interface and boundary conditions:

$$\begin{cases} U_{in} = U_{out} & \text{on } \partial\Omega_{in} \\ \dfrac{1}{L_{in}} \dfrac{\partial U_{in}}{\partial n} = \dfrac{1}{L_{out}} \dfrac{\partial U_{out}}{\partial n} & \text{on } \partial\Omega_{in} \\ \dfrac{\partial U_{out}}{\partial n} = g & \text{on } \partial\Omega \end{cases}$$

where the surface inductances L_{in} and L_{out} are, in general, different. The domain Ω_{in} can consist of several disjoint junction windows. The shape of these windows plays an important role in the stability of the junction; oval or "bowtie" shapes are usually the best.

Conclusions

These experiments lead us to believe that interface relaxation has high potential as a general method for composite PDE problems. At this time there is considerable uncertainty about the scope of applicability of the method, about procedures for choosing good parameters for relaxers (the method is analogous in this respect to iterative methods for linear systems), and about its implementation for time-dependent problems.

The focus of this article, and of our work, has not been on algorithm or code performance, because the principal uncertainty is not how fast interface relaxation works, but *whether* it works. For the composite nonlinear PDEs discussed earlier, there are no "established" alternative solution methods. Parallelism is an inherent property of interface relaxation, and the SciAgents system is slowed down significantly if all the agents are placed on a single computer.

Interface relaxation is very relevant to traditional domain decomposition. In early analyses and implementations, a single PDE was solved. The method could be viewed as a "continuous Schur complement iteration" method. In any case, it is likely to lead to new insight and ideas for domain decomposition. Our experiments have not exhibited marked differences in performance for single- and composite-PDE problems. The analysis of a simple model problem in [7] shows that the convergence rate is good and independent of the number of domains; this has been confirmed experimentally with up to 500 domains. We conjecture that convergence will eventually be shown to be independent of both the number of domains and the numerical method used for broad classes of composite PDEs.

Acknowledgments

The work reported here has been led by T. Drashansky, E. Houstis, A. Joshi, S. McFaddin, M. Mu, J. Rice, and E. Vavalis, with substantial assistance from A. Catlin, P. Tsompanopoulou, and S. Weerawarana.

Additional information about the SciAgents system can be found at the Web site http://www.cecs.missouri.edu/~joshi/sciag/.

References

[1] A. Barone and G. Paterno, *Physics and Applications of the Josephson Effect*, John Wiley, New York, 1982.

[2] J. Douglas and C. Huang, *An accelerated domain decomposition procedure based on Robin transmission conditions*, Purdue University, Department of Mathematics, Technical Report 289, 1997.

[3] T.T. Drashansky, *An agent based approach to building multiple disciplinary problem solving environments*, PhD thesis, Department of Computer Science, Purdue University, 1996.

[4] R. Fritzson, T. Finn, D. McKay, and R. McEntire, *KQML—A language and protocol for knowledge and information exchange*, Proceedings of the 13th International Distributed Intelligence Workshop, Springer-Verlag, New York, 1994.

[5] E. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, P. Papachiou, K.-Y. Wang, and M. Gaitatzes, *PELLPACK: A problem solving environment for PDE based applications on multicomputer platforms*, ACM Trans. Math. Software, in press, 23 (1998).

[6] P.L. Lions, "On the Schwarz alternating method III: A variant for nonoverlapping subdomains," *Domain Decomposition Methods for PDEs*, T. Chan et al., eds., SIAM, Philadelphia, 1990.

[7] M. Mu, *Solving composite problems with interface relaxation*, SIAM J. Sci. Comp., in press, 1998.

[8] A. Quarteroni, F. Pasquarelli, and A. Valli, *Heterogeneous domain decomposition: Principles, algorithms, applications*, Proceedings of the Fifth International Symposium on Domain Decomposition Methods for PDEs, D. Keyes et al., eds., SIAM, Philadelphia, 1992.

[9] J.R. Rice, *Collaborating PDE solvers*, available at http://www.cs.purdue.edu/people/jrr/slides/Collaborating/index.html.

John R. Rice is the W. Brooks Fortune Distinguished Professor of Computer Science at Purdue University.