CHAPTER 32

The Performance of an Explicit/Implicit Domain Decomposition Procedure for Parabolic Equations on an Intel Hypercube

Clint N. Dawson*

Abstract. A domain decomposition procedure for parabolic equations is described. In this procedure, the computational domain is divided into nonoverlapping subdomains. The equation is discretized by finite differences in time, and in space, a Galerkin finite element method is used on each subdomain. Subdomain solutions are related by an explicit flux calculation on the interfaces between subdomains. The interface fluxes are calculated in a stable and accurate manner, thus no iterations between the interface and subdomains are required. The method has been implemented on an Intel iPSC/860 Hypercube, and comparisons between domain decomposition solutions and a fully implicit Galerkin solution are presented for a set of test problems.

1. Introduction. In this paper, we discuss a domain decomposition algorithm, first presented in [Dawson and Dupont, 90], for solving equations of the form

$$u_t - \nabla \cdot (a \nabla u) + bu = f, \quad \text{on } \Omega \times (0,T], \tag{1}$$

$$u(x,0) = u^0(x), \qquad \text{on } \Omega, \tag{2}$$

on parallel processing computers. The domain $\Omega \subset \mathbf{R}^d$ is assumed to have a piecewise uniformly smooth, Lipschitz boundary, $\partial\Omega$. The coefficients a and b are assumed to be smooth and uniformly bounded, with $a > 0$. For simplicity, we assume homogeneous Neumann boundary conditions are given,

$$\frac{\partial u}{\partial n_\Omega} = 0, \quad \text{on } \partial\Omega \times (0,T], \tag{3}$$

where n_Ω is the outward normal to $\partial\Omega$; however, any standard boundary condition is allowed.

In the approach considered here, the domain Ω is divided into nonoverlapping subdomains, separated by interfaces which are smooth $d-1$-dimensional manifolds. Equation (1)

* Mathematical Sciences Dept., Rice University, Houston, TX 77251. The author wishes to acknowledge Lawrence Cowsar for his assistance, and the National Science Foundation Center for Research in Parallel Computation, for the use of their computing facilities.

is discretized by finite differences in time. At each discrete time level, an approximation to the normal derivative along each interface is computed, using the current approximate solution. This information is then used as Neumann boundary data for implicit subdomain problems. In each subdomain, the solution is updated to the new time level using a standard Galerkin finite element procedure.

The approximate normal derivatives along the interfaces are calculated in an accurate and stable manner; thus, no iterations between subdomains and interfaces, and no overlapping of subdomains are required. The price to be paid for this freedom is a constraint involving an interface discretization parameter and the time step.

The rest of the paper is organized as follows. First, we present the algorithm in a simple geometric configuration, and state an error estimate. In Section 3, numerical results generated on an Intel Hypercube are presented for two-dimensional test problems.

2. The basic procedure. In this section we will describe the method for a simple case, a more general treatment is given in [Dawson and Dupont, 90].

Take $\Omega = (0,1) \times (0,1)$ and $a = b = 1$. Assume Ω is divided into two subdomains, Ω_1 and Ω_2, where the interface $\Gamma = \{1/2\} \times (0,1)$.

Denote by $(\cdot,\cdot)_\Sigma$ the $L^2(\Sigma)$ inner product. When Σ is Ω, we drop the subscript. Let $\Delta t = T/M$ for some positive integer M, $t^n = n\Delta t$, $n = 0,\ldots,M$, and $g^n = g(t^n)$. Also, let $\partial_t g^n = (g^n - g^{n-1})/\Delta t$.

For functions ψ and ρ with restrictions in $H^1(\Omega_j)$, let

$$D(\psi,\rho) = \sum_{j=1}^{2} \left[(\nabla\psi, \nabla\rho)_{\Omega_j} + (\psi,\rho)_{\Omega_j} \right]. \tag{4}$$

Note that such a function ψ can have jumps in values across Γ, which we denote by $[\psi]$. For definiteness, let $[\psi]$ denote the trace from Ω_2 minus the trace from Ω_1.

Given an approximate solution at time t^n, the first step in updating the solution to time t^{n+1} is the calculation of an approximate normal derivative on Γ. Define a function $\phi_2(x)$ by

$$\phi_2(x) = \begin{cases} 1-x, & 0 \le x \le 1, \\ x+1, & -1 \le x \le 0, \\ 0, & \text{otherwise.} \end{cases}$$

Let $H \in (0,1/2)$, and set $\phi(x) = \phi_2((x-1/2)/H)/H$. Define

$$B(\psi(\cdot,y)) = -\int_0^1 \psi(x,y)\phi'(x)dx. \tag{5}$$

It can easily be shown that for ψ sufficiently smooth,

$$\left\| B(\psi) - \psi_x(\frac{1}{2},\cdot) \right\|_{L^2(\Gamma)} \le CH^2.$$

A domain decomposition procedure can now be described as follows. Let $\mathcal{M}_j$ denote a finite dimensional subspace of $H^1(\Omega_j)$, $j = 1,2$, and let $\mathcal{M}$ be the set of $L^2(\Omega)$ functions whose restrictions to Ω_j belong to $\mathcal{M}_j$. For $0 = t^0 < t^1 < \ldots < t^M = T$, and $U^0 \in \mathcal{M}$ given, define $U^1,\ldots,U^M$ by

$$(\partial_t U^n, v) + D(U^n, v) + (B(U^{n-1}), [v])_\Gamma = 0, \quad v \in \mathcal{M}. \tag{6}$$

Note that, since the boundary flux $B(U^{n-1})$ is known and the spaces $\mathcal{M}_1$ and $\mathcal{M}_2$ are independent, (6) decomposes into two problems corresponding to Ω_1 and Ω_2, which can be solved in parallel.

In order to state an error estimate for (6), define the elliptic projection $W(\cdot,t) \in \mathcal{M}$ at each $t \in [0,T]$ by

$$D((u-W)(\cdot,t),v) = 0, \quad v \in \mathcal{M}. \tag{7}$$

Let the error in this projection be denoted by

$$\eta = u - W. \tag{8}$$

The following theorem is proven in [Dawson and Dupont, 90],

THEOREM 2.1. *Suppose that the solution u to (1)-(3) is sufficiently smooth and that $U^0 = W(\cdot,0)$. Then there exists a constant C, independent of the spaces $\mathcal{M}_j$, such that*

$$\max_n ||(u-U)(\cdot,t^n)|| \le C\left\{\Delta t + H^{2.5} + \int_0^T ||\eta_t(\cdot,t)||dt + H^{-\frac{1}{2}}||\eta||_{L^\infty(\Omega\times(0,T))}\right\},$$

provided

$$\Delta t \le \frac{H^2}{4}. \tag{9}$$

An approximation to the interface flux which is fourth order in H can be obtained by defining ϕ in (5) by $\phi(x) = \phi_4((x-1/2)/H)/H$, where $H \in (0,1/4)$, and

$$\phi_4(x) = \begin{cases} (x-2)/12, & 1 \le x \le 2, \\ -5x/4 + 7/6, & 0 \le x \le 1, \\ 5x/4 + 7/6, & -1 \le x \le 0, \\ -(x+2)/12, & -2 \le x \le -1, \\ 0, & \text{otherwise.} \end{cases}$$

In this case, the $H^{2.5}$ term in the estimate given in Theorem 2.1 is replaced by $H^{4.5}$, provided that $\Delta t \le H^2/5.15$.

We note that in this theorem there are no assumptions that require $\mathcal{M}_1$ and $\mathcal{M}_2$ to be compatible in some way. Also, the parameter H for the operator B is not necessarily related to any aspect of the spaces $\mathcal{M}_j$; in particular, it is not required that ϕ restricted to Ω_j have any relation to $\mathcal{M}_j$.

The estimate given in Theorem 2.1 appears to be suboptimal. In the example above, if we choose $\mathcal{M}_j$ to be the space of continuous, piecewise bilinears on Ω_j, $j = 1,2$, the theorem states that

$$\max_n ||(u-U)(\cdot,t^n)|| \le C(\Delta t + H^{2.5} + h^2 + H^{-1/2}h^2 \ln|h|).$$

Numerically, we have observed convergence of order $\Delta t + h^2 + H^3$ for certain test problems [Dawson and Dupont, 90]. This rate can be proven in one space dimension, and for certain types of discretizations in rectangular geometry. In general, however, we do not know how to improve the estimate given in Theorem 2.1.

3. Numerical results. We now present some numerical results for the scheme described in Section 2. The algorithm has been implemented on an Intel iPSC/860 Hypercube. The experiments described below were performed on a 32 processor machine located at the Center for Research in Parallel Computation at Rice University.

We will compare domain decomposition solutions and a fully implicit Galerkin finite element solution (no domain decomposition) for three test problems:

Test Problem 1:

$$\begin{aligned} u_t - \Delta u &= 0, \\ u^0(x,y) &= \cos(\pi x)\cos(\pi y), \\ \frac{\partial u}{\partial n_\Omega} &= 0. \end{aligned}$$

This problem has the solution $u(x,y,t) = e^{-2\pi^2 t}u^0(x,y)$.

Test Problem 2:

$$\begin{aligned} u_t - \nabla\cdot((1+x)\nabla u) &= f, \\ u^0(x,y) &= \cos(\pi x)\cos(\pi y), \\ \frac{\partial u}{\partial n_\Omega} &= 0, \end{aligned}$$

where f is chosen so that the solution $u(x,y,t) = e^{-2\pi^2 t}u^0(x,y)$.

Test Problem 3:

$$\begin{aligned} u_t - \nabla\cdot(.01(1+e^{2x}e^{3y})\nabla u) &= f, \\ u^0(x,y) &= \cos(\pi x)\cos(\pi y), \\ \frac{\partial u}{\partial n_\Omega} &= 0, \end{aligned}$$

where f is again chosen so that the solution $u(x,y,t) = e^{-2\pi^2 t}u^0(x,y)$.

The underlying discretization is an 80×80 uniform grid, $H = .10$, $\Delta t = .0025$, and $T = .10$. On each subdomain, U^n lies in the space of continuous, piecewise bilinear functions, and in the operator B, ϕ is constructed from ϕ_2.

As seen in (6), on each subdomain an implicit method is used. Thus, a linear algebraic system of equations must be solved at each time step on each subdomain. We have used a preconditioned conjugate gradient procedure for solving this system, where the preconditioner consists of simply scaling each row by its diagonal entry. This method was chosen for convenience; clearly any reasonable linear algebraic solver may be applied to the subdomain problems.

In the tables below, we compare fully implicit Galerkin solutions with various domain decomposition solutions. The notation $p \times q$ refers to a decomposition of Ω into $p \cdot q$ subdomains, where the subdomains are rectangular regions constructed by dividing Ω into p uniform subdomains in the x direction and q uniform subdomains in the y direction. When $p = q = 1$, this represents the fully implicit Galerkin solution. A decomposition into $p \cdot q$ subdomains means $p \cdot q$ processors were used to generate the solution. Thus $p \cdot q \leq 32$ for the machine used here.

In Table 1, we give clock time and number of conjugate gradient iterations for various domain decompositions applied to Problem 1. The column labeled *CGI* represents the average number of conjugate gradient iterations needed to "solve" the linear algebraic system of equations at each time step. By "solve" we mean reduce the residual below 10^{-6}. When

Decomposition	Clock Time (sec)	CGI
1x1	48.709	5
2x1	106.650	41
2x2	55.010	48
4x2	27.819	47
4x4	10.642	37

TABLE 1
Table 1: Results for Test Problem 1

Decomposition	Clock time (sec)	CGI	L^2 error
1x1	438.698	61	$2.25 * 10^{-3}$
2x1	174.379	41	$1.74 * 10^{-3}$
2x2	105.720	63	$1.31 * 10^{-3}$
4x2	51.335	58	$8.51 * 10^{-4}$
4x4	22.496	46	$5.87 * 10^{-4}$
8x4	10.459	38	$1.07 * 10^{-3}$

TABLE 2
Results for Test Problem 2

multiple subdomains are used, the number CGI represents the maximum average over all of the subdomains. As can be seen in Table 1, the fully implicit method converges in only five conjugate gradient iterations per time step for Test Problem 1. This seems to be due to the fact that the problem is homogeneous and $a \equiv 1$. The domain decomposition procedure introduces nonhomogeneous boundary data into the system and hence the number of conjugate gradient iterations needed to converge at each time step is substantially larger. Comparing clock times for the domain decomposition solutions generated on 2, 4, 8, and 16 processors, we note that as the number of processors is doubled, a speed-up of essentially a factor of two is obtained. This factor increases as we go from 8 to 16 processors, because CGI decreases, probably due to the smaller problem size.

Test Problems 2 and 3 represent more difficult problems, due to the variable coefficient and nonzero source term. Results for these problems are given in Tables 2 and 3, respectively. In these tables we also compare the L^2 error, $||u - U||$, for each discretization at final time $T = .10$. It is interesting to note that for both problems, the domain decomposition solutions are more accurate than the fully implicit solution. Comparing clock times for the different decompositions, we note that domain decomposition with 2 processors requires less than half the clock time of the fully implicit scheme for both problems. In fact, in each case a speed-up factor of near N is obtained when comparing domain decomposition with N processors to the fully implicit case, and in some cases, the speed-up factor is greater than N. In particular, the factor is 41.7 for 32 processors in Problem 2, and 35 for 32 processors

Decomposition	Clock time (sec)	CGI	L^2 error
1x1	596.931	46	$8.40 * 10^{-3}$
2x1	293.157	45	$8.18 * 10^{-3}$
2x2	140.894	43	$8.00 * 10^{-3}$
4x2	73.169	42	$7.81 * 10^{-3}$
4x4	35.681	39	$7.66 * 10^{-3}$
4x8	17.094	32	$7.34 * 10^{-3}$

TABLE 3
Results for Test Problem 3

in Problem 3. Thus, we are obtaining better than linear speed-up based on this type of comparison.

It is clear that the timings given in Tables 1-3 depend strongly on the particular method used to solve the linear algebraic systems which arise at each time step. If we had used a direct method to solve these systems, it is expected that near linear speed-up would have been obtained in all cases. Thus, the timing comparisons between domain decomposition and fully implicit Galerkin for Problem 1 would likely have been more favorable, but the "superlinear" speed-up seen in Problems 2 and 3 would not have occurred. If the subdomain problems are large, however, it is generally preferable to use an iterative solver rather than a direct solver, and conjugate gradient procedures such as the one used here are often employed. Therefore, the consequences of using iterative linear solvers to solve the subdomain problems should be examined.

As noted above, the domain decomposition procedure gives L^2 errors which compare favorably to the errors for the fully implicit solutions for Test Problems 2 and 3. In the domain decomposition method we are possibly committing substantial errors in calculating fluxes at the interfaces between subdomains; therefore, it is of interest to examine how this error effects the spatial distribution of the error in the solution. For this study, we consider Test Problem 1 and apply a 1x2 decomposition. In Figure 1, the error function $e(x, y, t) = |U(x, y, t) - u(x, y, t)|$ at time $t = .005$ is contoured. The function e has been scaled so that the contour values lie between zero and one. In these runs, a uniform 40×40 grid was used, $\Delta t = .0025$, and $H = .10$. Thus, Figure 1 represents the error after two time steps. As can be seen in the figure, the maximum errors do occur in a neighborhood of the interface, indicating that the errors in the interface flux are polluting the solution. In Figure 2, we plot the error after ten time steps, $e(x, y, .05)$. Examining this figure we see that there are still substantial errors near the interface; however, the error is more evenly distributed and the maximum errors are actually located in the corners of the domain. A similar error distribution is seen for the fully implicit method. In Figure 3, we plot $e(x, y, .05)$ for the fully implicit Galerkin scheme applied on the same mesh. The figure indicates that the maximum errors for this scheme are also located in the corners of the domain. In summary, it appears that for this test problem, as the simulation proceeds, the interface errors dissipate and the domain decomposition solution behaves very similarly to the fully implicit solution.

4. Conclusions. In conclusion, it should be noted that much numerical testing on the method described here remains to be done. The results obtained so far are encouraging, however, and the method shows promise for solving large-scale problems in parallel.

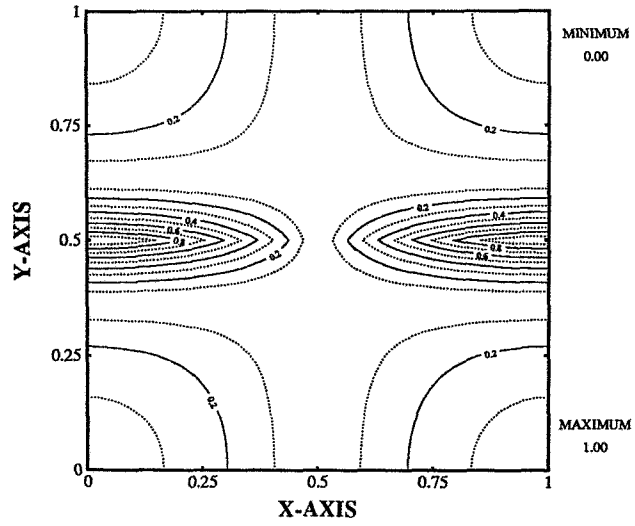

FIG. 1. *Contour plot of error in domain decomposition solution at* $t = .005$

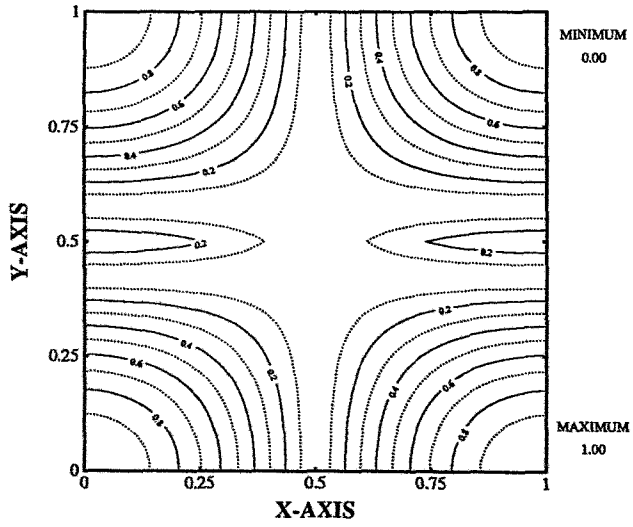

FIG. 2. *Contour plot of error in domain decomposition solution at* $t = .005$

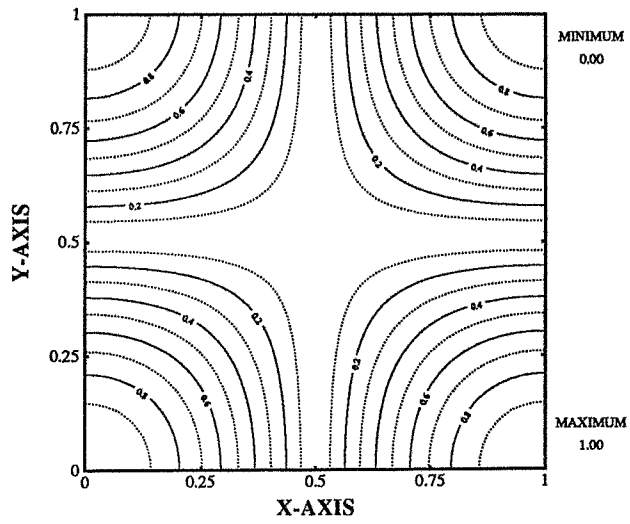

FIG. 3. *Contour plot of error in fully implicit solution at* $t = .005$

REFERENCES

Dawson, C. N. and T. F. Dupont, *An explicit/implicit, conservative, Galerkin domain decomposition procedure for parabolic equations*, Rice Technical Report TR90-26, Dept. of Mathematical Sciences, Rice University, and to appear in Math. Comp.