A PARALLEL IMPLEMENTATION OF AN ITERATIVE SUBSTRUCTURING ALGORITHM FOR PROBLEMS IN THREE DIMENSIONS

BARRY F. SMITH*

Mathematics and Computer Science Division Preprint MCS-P249-0791
Argonne National Laboratory
August 1, 1991

Abstract. Numerical results from a parallel implementation of an iterative substructuring algorithm are reported. The algorithm is for solving scalar, self-adjoint elliptic partial differential equations in three dimensions. Results are given for two variants of the algorithm. In the first variant, exact interior solvers are used; in the second, one multigrid V-cycle is used to approximately solve the interior problems. The results are compared to theoretical behavior of the algorithm reported in previous work.

Key words. domain decomposition, finite elements, iterative substructuring, parallel computing.

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

Much work on domain decomposition algorithms has focused on the abstract analysis of the algorithms, with less discussion of implementation issues and few nontrivial numerical results. This paper focuses on the implementation, on a state-of-the-art parallel machine, of several iterative substructuring algorithms for elliptic partial differential equations in three dimensions. Full analysis of some of the algorithms, using standard domain decomposition techniques, can be found in Smith [19] and Dryja, Smith, and Widlund [11]. The underlying algorithm was first introduced in Smith [20].

Iterative substructuring algorithms are domain decomposition algorithms in which nonoverlapping subdomains are used. A preconditioned conjugate gradient method is used to solve the linear system obtained by a finite element discretization of the partial differential equation. The preconditioner is obtained by separately solving linear systems associated with the interiors of the subdomains, the faces between subdomains, and a system that provides global coupling between the subdomains.

There is a fundamental difference in the nature of finite element solutions of elliptic problems in two and three dimensions, certain results do not carry over from two to three dimensions. Hence new analysis and numerical experiments must be carried out for problems in three dimensions. Earlier theoretical work on the subject which has strongly influenced our work can be found in Bramble, Pasciak, and Schatz [4], Dryja [10], Dryja and Widlund [12], Mandel [18], and Smith [19]. For a modern treatment of iterative substructuring algorithms in three dimensions, see Dryja, Smith, and Widlund

* Argonne National Laboratory, 9700 South Cass Ave., Argonne, IL 60439-4844. E-mail: bsmith@mcs.anl.gov. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

[11]. Other large-scale experimental work in domain decomposition is described in Bjørstad and Hvidsten [1], Bjørstad, Moe, and Skogen [2], Keyes and Gropp [14], [15], De Roeck [8], De Roeck and Le Tallec [9], Le Tallec, De Roeck, and Vidrascu [16], and Mandel [17].

Any good iterative substructuring algorithm with exact interior solvers can be reformulated to use approximate interior solvers. This approach has been analyzed with some success in Börgers [3] and Haase, Langer, and Meyer [13]. In this paper we report on experiments in which both approximate and exact interior solvers are used. We use multigrid for the approximate interior solver in our experiments.

This paper is organized as follows. In Section 2 we introduce the algorithm using matrix notation. In Section 3 we discuss the implementation of the algorithm on distributed-memory machines. In Section 4, we present numerical results for some piecewise constant coefficient problems. In Section 5, we discuss future work which will focus on the application of the algorithms to more difficult multicomponent elliptic partial differential equations.

2. Matrix Form of Preconditioner.

Consider a scalar, second-order, self-adjoint, H^1-coercive, bilinear form $a_\Omega(u,v)$ on $\Omega \subset R^3$ and impose a homogeneous Dirichlet boundary condition on $\Gamma_0 \subset \partial\Omega$ and a Neumann boundary condition on $\partial\Omega \setminus \Gamma_0$. We assume that the underlying elliptic operator has no zero-order terms. Let $H^1_{\Gamma_0}(\Omega)$ be the subspace of functions in $H^1(\Omega)$ that vanish on Γ_0. The variational problem is to find $u \in H^1_{\Gamma_0}(\Omega)$ such that

$$a_\Omega(u,v) = (f,v), \quad \forall\, v \in H^1_{\Gamma_0}(\Omega).$$

We triangulate the domain Ω using the usual rules for finite element triangulations. Let $V^h(\Omega) \subset H^1_{\Gamma_0}(\Omega)$ be the space of continuous, piecewise linear, functions on the triangulation that vanish on Γ_0. In addition, for the construction of the preconditioner, we assume that the set of elements is partitioned into disjoint substructures Ω_i. Let H be the characteristic diameter of the substructures; that is, assume that there exist constants c and C independent of h and H such that for all substructures $cH \leq \mathrm{diam}(\Omega_i) \leq CH$. In the experiments reported here, the substructures are always brick-shaped, but this is not necessary for the algorithm.

The discrete problem is to find $u^h \in V^h(\Omega)$ such that

$$a_\Omega(u^h,v^h) = (f,v^h), \quad \forall\, v^h \in V^h(\Omega). \tag{1}$$

If u^h is expanded in the standard nodal basis, $u^h = \sum_k u_k \phi_k$, the variational problem (1) can be written as the linear system

$$K\underline{u} = \underline{f}.$$

In previous work [19], we constructed preconditioners for this system that involve separately solving linear systems associated with the interiors of the subdomains, the faces shared by pairs of subdomains, and a system associated with the remaining degrees of freedom. The application of our preconditioner results in a convergence rate that is independent of the number of subdomains and is independent of jumps in the coefficients of the partial differential equation between subdomains.

Convergence of the preconditioned conjugate gradient method is determined by the distribution of the eigenvalues of the preconditioned matrix, $B^{-1}K$. In particular, the number of iterations needed to achieve a fixed accuracy of the solution can be bounded by a constant times the square root of the condition number of the matrix, $\kappa(B^{-1}K)$. Most of the theoretical work on domain decomposition algorithms focuses on obtaining bounds on the condition number. In this work, we report on condition numbers, iteration counts, and total solution time and compare these to the behavior predicted by theory.

We partition the unknown coefficients into those associated with the interiors of the subdomains, $\underline{u}_I$, those associated with the faces shared by exactly two subdomains, $\underline{u}_F$, and those shared by more than two subdomains (the wirebasket), $\underline{u}_W$. We use $\underline{u}_B$ to denote the vector of coefficients $(\underline{u}_F, \underline{u}_W)$. In addition, we let $\underline{u}^{(i)}$ represent the coefficients associated with the closure of subdomain Ω_i.

We express the inverse of the stiffness matrix in partially factored form.

$$\begin{pmatrix} I & -K_{II}^{-1}K_{IF} & -K_{II}^{-1}K_{IW} \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix}\begin{pmatrix} I & 0 & 0 \\ 0 & I & -S_{FF}^{-1}S_{FW} \\ 0 & 0 & I \end{pmatrix}\begin{pmatrix} K_{II}^{-1} & 0 & 0 \\ 0 & S_{FF}^{-1} & 0 \\ 0 & 0 & S_{WW}^{-1} \end{pmatrix}\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & -S_{FW}^{T}S_{FF}^{-1} & I \end{pmatrix}\begin{pmatrix} I & 0 & 0 \\ -K_{IF}^{T}K_{II}^{-1} & I & 0 \\ -K_{IW}^{T}K_{II}^{-1} & 0 & I \end{pmatrix}.$$

The matrix S_{FF} represents that part of the Schur complement, after the interior nodes have been eliminated, that is associated with the coupling between the nodes on the faces of the subdomains. S_{WW} is the Schur complement associated with the wirebasket once the unknowns of the interior and the faces have been eliminated. We do not explicitly form these matrices; instead, the preconditioner is constructed by replacing various blocks with more computationally attractive matrices. We note that K_{II}^{-1}, S_{FF}^{-1}, $S_{FF}^{-1}S_{FW}$, and S_{WW}^{-1} may potentially be replaced.

The K_{II} is a block diagonal matrix with a block for each subdomain interior. For K_{II}^{-1} we use either a sparse factorization (in particular, the Yale Sparse Matrix Package which uses the minimum-degree algorithm to reorder to reduce fill-in) or one or several multigrid V-cycles to approximate the action of the inverse. In the latter case, the approximate inverse can be written as $\tilde{K}_{II}^{-1} = (I - M_\nu)K_{II}^{-1}$. M_ν is the symmetric error iteration matrix for ν multigrid V-cycles, and $\rho(M_\nu) < 1$. We note that there is considerable freedom in choosing the multigrid solver and the number of V-cycles. When exact interior solvers are used, we can eliminate the unknowns $\underline{u}_I$ initially and iterate on only $\underline{u}_B$, using one solver involving K_{II} per iteration. Once $\underline{u}_B$ is known, we can backsolve for $\underline{u}_I$. If an approximate solver is used for the interior, then two interior solvers are needed per iteration, and all of the variables are present in the iterative process.

We replace the Schur complement S_{FF} with a block diagonal matrix with one block for each face. Let $\tilde{S}_{FF}$ be the generic substitution. Several candidates exist for the matrix blocks. They are all derived by extending earlier results for problems in two dimensions.

1. We can explicitly use the blocks from the Schur complement. The advantage of this approach is that the block is automatically well adapted for each differential equation. Unfortunately, it is quite expensive to calculate the blocks except for small subdomains. This method may be needed for extremely ill-conditioned problems where no good substitutes exist.
2. We can use a suitable scaling of the J operator. The J operator is the square root of the two-dimensional discrete Laplacian on a regular, rectangular mesh. See Bramble, Pasciak, and Schatz [4] for a discussion of why this leads to good results. It has been shown that J is spectrally equivalent to the explicit block of the Schur complement. It is computationally cheap to apply the action of J^{-1} to a vector. It does not, however, adapt to the particular partial differential equation.
3. For constant-coefficient elliptic partial differential equations on rectangular subdomains with a uniform finite difference mesh, we can exactly diagonalize the Schur complement associated with a face, using fast sine transforms. For two dimensions, Chan and Hou [5] have proposed the use of this fast spectral decomposition to approximate the actual Schur complement. This approach could be extended to three dimensions. Again, as with the J operator, this requires that we use a regular, rectangular mesh on brick-shaped subdomains.
4. We could use a multilevel preconditioner. This is an extension to three dimensions of the hierarchical Schur complement preconditioner considered in Smith and Widlund [21]; see also Haase, Langer, and Meyer [13].
5. Another approach is to use the tangential component of the original operator restricted to that face. It can be obtained easily and adapts reasonably well to the partial differential equation. This approach is taken in Chan and Keyes [6] and Keyes and Gropp [15]. It does not perform well, however, when the components of the operator that are normal to the face dominate.
6. The method of probing (see Chan and Keyes [6] and Chan and Mathew [7]) could be used to calculate diagonal or band diagonal approximations to the Schur complement on each face.

We observe that the operator $S_{FF}^{-1}S_{FW}$ maps values from the boundaries of the faces to the faces. It is known that the most important property of the mapping is that it maps a constant value on the boundary of the face onto the face as the same constant. We use a simple mapping that preserves this property. We map the average of the unknowns on the boundary of the face onto the face; see Smith [19] for the underlying theory. Let T^T denote this mapping. More sophisticated interpolation schemes are also possible and may be needed for more difficult problems.

For S_{WW}^{-1}, inspired by Mandel [18], we use the matrix defined by the following minimization problem:

$$\min_{\underline{u}} \sum \min_{\bar{w}^{(i)}} \delta_i (H/h)(\underline{u}_W^{(i)} - \bar{w}^{(i)} z^{(i)})^T I(\underline{u}_W^{(i)} - \bar{w}^{(i)} z^{(i)}) - \underline{u}_W^T f_W. \tag{2}$$

The $z^{(i)}$ is a vector of all ones of the same dimension as $\underline{u}_W^{(i)}$. We let G_{WW} denote the matrix defined by the minimization given above. For nontrivial problems, a diagonal, or block diagonal, matrix, that is adapted to the particular partial differential equation may be substituted in place of the identity matrix in the above formula. A simpler choice for this part of the preconditioner would be the block diagonal part of the original stiffness matrix associated with the wirebasket. This choice, however, results in a preconditioned system whose condition number grows faster than C/H^2, while the former choice results in a condition number bounded by $(1+\log(H/h))^2$, see Smith [19].

We write the generic form of the inverse of the preconditioner as

$$\begin{pmatrix} I & -\tilde{K}_{II}^{-1}K_{IF} & -\tilde{K}_{II}^{-1}K_{IW} \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix}\begin{pmatrix} I & 0 & 0 \\ 0 & I & -T^T \\ 0 & 0 & I \end{pmatrix}\begin{pmatrix} \tilde{K}_{II}^{-1} & 0 & 0 \\ 0 & \tilde{S}_{FF}^{-1} & 0 \\ 0 & 0 & \tilde{S}_{WW}^{-1} \end{pmatrix}\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & -T & I \end{pmatrix}\begin{pmatrix} I & 0 & 0 \\ -K_{IF}^T\tilde{K}_{II}^{-1} & I & 0 \\ -K_{IW}^T\tilde{K}_{II}^{-1} & 0 & I \end{pmatrix}.$$

We consider five specific preconditioners in the numerical studies.

1. Diagonal preconditioning of the original stiffness matrix. We denote the preconditioner by D.
2. Two preconditioners that use exact interior solvers.

   a) The first version involves solving the wirebasket problem with the technique introduced above and in Smith [19]. We can express the preconditioner as

$$B_G^{-1} = \begin{pmatrix} T^T \\ I \end{pmatrix} G_{WW}^{-1} \begin{pmatrix} T & I \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{S}_{FF}^{-1} \begin{pmatrix} I & 0 \end{pmatrix}.$$

   b) In the second version, we solve the wirebasket problem using a diagonal matrix

$$B_D^{-1} = \begin{pmatrix} 0 \\ I \end{pmatrix} D_{WW}^{-1} \begin{pmatrix} 0 & I \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{S}_{FF}^{-1} \begin{pmatrix} I & 0 \end{pmatrix}.$$

3. The next two versions use approximate solvers on the interior subproblems.

   a)

$$B_{G^A}^{-1} = \begin{pmatrix} I & -\tilde{K}_{II}^{-1}K_{IB} \\ 0 & I \end{pmatrix}\begin{pmatrix} \tilde{K}_{II}^{-1} & 0 \\ 0 & B_G^{-1} \end{pmatrix}\begin{pmatrix} I & 0 \\ -K_{IB}^T\tilde{K}_{II}^{-1} & I \end{pmatrix}.$$

   b)

$$B_{D^A}^{-1} = \begin{pmatrix} I & -\tilde{K}_{II}^{-1}K_{IB} \\ 0 & I \end{pmatrix}\begin{pmatrix} \tilde{K}_{II}^{-1} & 0 \\ 0 & B_D^{-1} \end{pmatrix}\begin{pmatrix} I & 0 \\ -K_{IB}^T\tilde{K}_{II}^{-1} & I \end{pmatrix}.$$

For the piecewise constant coefficient problems considered in this paper, we use a multiple of the J operator as our face preconditioner. For our model problems this approach works almost as well as the computationally more expensive explicit Schur complement.

3. Implementation Issues. We have implemented the algorithm on a distributed-memory machine. Each processor has its own local memory and can communicate, either directly or indirectly, with all other processors by using explicit message passing. The specific architecture is that of the Intel iPSC/860 and Touchstone DELTA System. These machines have from 8 to 528 Intel i860 processors, each of which is capable of sustained rates of more than 4 megaflops with compiled Fortran or C. Their peak performance for hand-coded assembler is considerably higher. Each processor has between 8 and 16 megabytes of local memory. The Intel iPSC/860 machine has a hypercube connection between the nodes, while the Touchstone DELTA has a two-dimensional mesh connection.

Communication time on these machines is slow compared to the floating-point speed. Hence, data locality and the minimization of communication are vital.

3.1. Basic Description of Our Implementation. The particular implementation of the algorithm introduced above is closely related to the work of Keyes and Gropp [14] for problems in two dimensions. Keyes and Gropp subdivide the domain into rectangular tiles. Each tile is then discretized by using a regular mesh. This is a very natural approach combining flexibility of the domain with regular subdomains and the possibility of local uniform mesh refinement.

The three-dimensional domain is partitioned into brick-shaped subdomains each of which is assigned a uniform finite element or finite difference mesh. To simplify the coding, we require that adjacent subdomains share an entire face, entire edge, or a vertex; see Fig. 3.1. This requirement is necessary for our implementation, but not for the underlying mathematical algorithms.

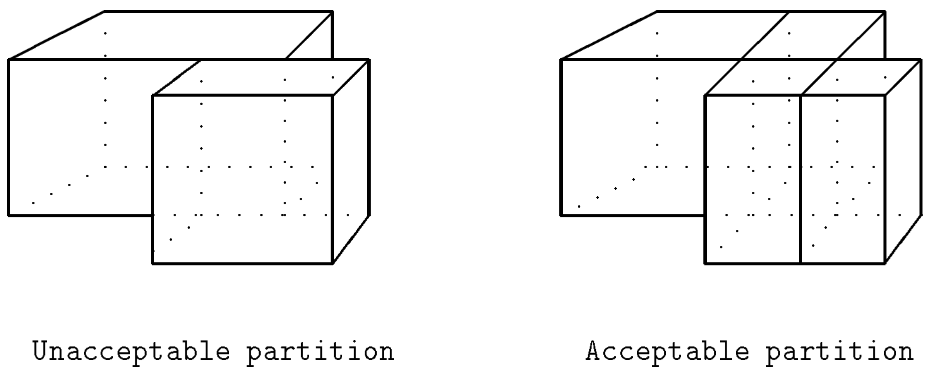

FIG. 1. *Acceptable and unacceptable partitions of a domain*

Each processor is assigned one or more subdomains. The information pertaining to the interior of the subdomain is uniquely owned by that processor and is not directly available to any other processor. Each face, edge, and vertex is jointly owned by several subdomains and hence potentially by several processors. Because of this joint ownership, whenever a change is made to the part of the solution associated with a face, edge, or vertex of one subdomain, this information must be conveyed to the other joint owners by using explicit message passing. We refer to this process as *merging of partial data.* For each face, vertex, and edge we designate one of the joint owners as the main owner and the others as auxiliary owners.

Essentially three types of communication between processors are required when the preconditioned conjugate gradient method is used to solve the linear system. The first is multiplication by the stiffness matrix. After the calculation of the local contribution to the matrix multiplication, the parts of the product vector that are shared by two or more processors must be merged. This merging of partial results can be performed in several ways. At this time a naive approach is used. The partial sums on each face, edge, and vertex are accumulated by the main owner and then sent out to the joint owners. For large problems, when using the full preconditioner, we find that less than five percent of the time is spent doing communication related to the matrix multiplication. With diagonal preconditioning, the matrix multiply dominates the entire solution time. Hence, optimizing the communication in the matrix multiply becomes important.

The application of the preconditioner is the most expensive operation in terms of communication. The principal reason is that the preconditioner is designed to provide for global communication of information in each step of the iteration process. When less communication is provided, more iterations are needed, however, at a lower cost per iteration. With simple diagonal preconditioning, for instance, no cross-processor communication is needed. This fact suggests that for many well-conditioned problems diagonal scaling is the optimal approach for parallel computing systems of the type considered in this paper.

We list below the steps currently used in the application of the preconditioner B_G^{-1}. The steps in braces are the additional steps needed when approximate interior solvers are used.

1. {Approximate solvers on interior problems.}
2. {Merge results.}
3. Interpolate face averages onto the edges.
4. Merge results.
5. Calculate an average on the wirebasket for each subdomain, and send off to coarse solver.
6. Solve face problems and coarse problem simultaneously.
7. Interpolate coarse solution to the wirebasket.
8. Merge results.
9. {Approximate solvers on interior problems.}

The conjugate gradient method requires several inner products per iteration. When possible, we use a direct call to a low-level implementation of a cross–processor inner product.

4. Experimental Results for Piecewise Constant Coefficient Problems.

In this section we report on experiments with scalar elliptic problems with piecewise constant coefficients. The reason for examining such problems is threefold: we can compare the well-developed theory with the numerical results, we can obtain a lower bound on how well the algorithm performs for more difficult problems, and we can resolve questions about the optimal scaling of different parts of the preconditioner.

The total solution time depends on the number of iterations needed and the average amount of time needed per iteration. Information useful to compare different algorithms is provided by the number of iterations needed to obtain a fixed accuracy of the solution. The square root of condition number of the preconditioned system gives a bound on the number of iterations needed.

4.1. On the Local Bounds. In this algorithm, as with most iterative substructuring algorithms (see Dryja, Smith, and Widlund [11]), it is possible to bound the condition number of the preconditioned matrix by bounds obtained locally, that is,

$$\kappa(B_G^{-1}S) \leq \frac{\max C_i}{\min c_i},$$

where the c_i and C_i satisfy

$$c_i \underline{u}^{(i)^T} B_G^{(i)} \underline{u}^{(i)} \leq \underline{u}^{(i)^T} S^{(i)} \underline{u}^{(i)} \leq C_i \underline{u}^{(i)^T} B_G^{(i)} \underline{u}^{(i)}, \qquad \forall \underline{u}^{(i)}.$$

We wish to determine how close the local bounds are to the actual condition numbers as a function of the number of subdomains. We have performed two sets of experiments, one using the exact blocks of the Schur complement, and the other using the J operator as the face preconditioner. We make two observations from Table 1:

- The condition numbers when using either the explicit Schur complement or the J operator are virtually identical for the Laplace operator.
- The bounds obtained from the local analysis quite closely predict the condition numbers even for a relatively small number of subdomains.

The positions in the table denoted by a dash are cases for which experiments were not carried out because of time or memory constraints.

Table 1
Condition numbers and local bounds

	Explicit Schur Complement					J Operator				
	Local	Number of Subdomains				Local	Number of Subdomains			
H/h	Bound	27	64	125	216	Bound	27	64	125	216
4	9.66	8.33	8.77	8.82	9.22	10.25	8.72	9.41	9.35	9.92
5	11.18	9.57	10.28	10.20	10.68	12.10	9.86	10.78	10.61	11.30
6	12.40	10.87	11.52	11.43	–	13.64	11.05	12.07	11.85	12.89
7	14.04	11.83	12.63	–	–	15.07	11.96	13.56	13.30	14.51
8	15.86	12.83	14.05	–	–	16.31	12.87	15.00	14.70	16.06
9	17.59	13.62	–	–	–	17.54	13.63	16.37	–	–
10	19.23	–	–	–	–	19.26	14.38	17.67	17.61	18.91
16	–	–	–	–	–	–	21.12	24.20	24.16	25.98
20	–	–	–	–	–	–	24.15	27.88	27.78	29.83

4.2. On the Scaling of the Coarse Problem. We can express the preconditioned problem when using exact interior solvers as

$$B_G^{-1} = \begin{pmatrix} T^T \\ I \end{pmatrix} G_{WW}^{-1} \begin{pmatrix} T & I \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{S}_{FF}^{-1} \begin{pmatrix} I & 0 \end{pmatrix}.$$

The mathematical analysis of the algorithm (see Smith [19] and Dryja, Smith, and Widlund [11]) tells us that asymptotically, for large H/h, we should scale G_{WW} by a factor $\delta_i(H/h) = C(1 + \log(H/h))$. The analysis gives no information, however, about the selection of the constant C nor whether the scaling is important for relatively small values of H/h. We shall refer to the case with $\delta_i(H/h) = 1$ as the natural scaling. In our experiments, we determine for each mesh size the optimal scaling $\delta_i(H/h)$ using a simple bisection method and compare the condition number to that obtained using the natural scaling. The results are presented in Table 2.

A related question is whether, in the construction of G_{WW} (see equation (2)), we should scale the diagonal elements for the nodes associated with the vertices of the subdomains differently from those nodes associated with the edges. The most natural choice is to scale the former elements by 1/2, since those nodes are contained in exactly twice as many subdomains. We refer to the resulting choice as a weighted G_{WW}. We make the following conclusions from Table 2:

- For the range of computationally practical meshes on the subdomains (i.e., $H/h \leq 32$), the natural scaling is only trivially worse than the optimal scaling.
- Using the weighted G_{WW} results in only a trivial improvement in the condition number.

TABLE 2
Natural vs. optimal scaling of coarse problem: condition numbers

$G^{(i)}$	Scaling	H/h							
		4	8	12	16	20	24	28	32
Natural	Natural	9.4	15.0	20.1	24.2	27.9	31.1	33.9	36.5
	Optimal	8.8	14.0	19.3	23.8	27.6	30.9	33.9	36.5
Weighted	Natural	–	14.2	19.3	23.6	27.2	–	–	–
	Optimal	–	13.6	18.8	23.3	27.1	–	–	–

4.3. The Growth of the Condition Numbers. Mathematical analysis predicts the growth in the condition number as a function of the mesh refinement, H/h, and the number of subdomains, but it does not give good estimates of the actual numerical values. Our results are given in Tables 3 and 4. For the preconditioner labeled B_{G^A} we have used one multigrid V–cycle to solve the subdomain problems approximately. The difference in the condition number between using one multigrid V–cycle, two multigrid V–cycles, and an exact interior solver is very small. Similar results have previously been noted by Börgers [3] and Haase, Langer, and Meyer [13] for problems in two dimensions but have they not yet been fully explained theoretically.

TABLE 3
Growth in condition numbers for 64 subdomains ($H = 1/4$)

H/h	Unknowns	K	S	$B_G^{-1}S$	$B_{GA}^{-1}K$	$B_D^{-1}S$	$B_{DA}^{-1}K$
4	3,375	103	53.81	9.4	9.4	67.6	67.7
8	29,791	414	122	15.0	15.0	107	107
12	103,823	933	192	20.1	20.1	131	133
16	250,047	1,656	261	24.2	24.4	150	152
20	493,039	2,593	331	27.9	28.1	165	166
24	857,375	3,734	401	31.1	31.3	–	–
28	1,367,631	5,083	–	33.9	34.2	–	–
32	2,048,383	6,640	–	36.5	36.9	–	–
Observed Growth		$(1/h)^2$	$1/h$	$(1+\log(H/h))^2$		$(1/H^2)(1+\log(H/h))^2$	

TABLE 4
Growth in condition numbers for 216 subdomains ($H = 1/6$)

H/h	Unknowns	K	S	$B_G^{-1}S$	$B_{GA}^{-1}K$	$B_D^{-1}S$	$B_{DA}^{-1}K$
4	12,167	232	119	9.9	9.9	155	149
8	103,823	933	269	16.1	16.1	230	232
12	357,911	2,099	421	21.5	21.5	281	283
16	857,375	3,734	573	26.0	26.1	318	321
20	1,685,159	5,835	726	29.8	30.0	436	350
Observed Growth		$(1/h)^2$	$1/h$	$(1+\log(H/h))^2$		$(1/H^2)(1+\log(H/h))^2$	

4.4. A Comparison with a Bramble, Pasciak, and Schatz Algorithm. Since our basic algorithm is similar to one of the important algorithms introduced by Bramble, Pasciak, and Schatz [4], we have reproduced the experiments reported in their paper using the preconditioner B_G^{-1}. The first set of experiments is for a unit cube divided into eight subcubes. The stiffness matrix is derived from the usual finite difference discretization for the Laplace operator. The second problem is for a unit cube divided in 27 subcubes with a different constant coefficient on each subcube; see [4] for the values used. We see from Tables 5 and 6 that for this class of problem, the two preconditioners produce similar condition numbers. We note that these are relatively small problems and that diagonal scaling also works well.

TABLE 5
Comparison with BPS IV: Laplacian operator

	Condition numbers			
H/h	Diagonal	BPS	$B_G^{-1}S$	Unknowns
4	25.3	13.9	10.3	343
8	103	17.7	12.9	3,375
16	414	23	18.4	29,791

TABLE 6
Comparison with BPS IV: coefficients with jumps

	Condition numbers			
H/h	Diagonal	BPS	$B_G^{-1}S$	Unknowns
4	63.4	14.1	9.0	1,331
8	265.4	18.3	14.5	12,167

4.5. Timings. We next present timing results on a 32-node Intel iPSC/860 hypercube for a set of intentionally simple examples. We consider three problems. The first two problems are on the unit cube; the third is on a more complicated region depicted in Fig. 4.5. The unit cube is uniformly divided into subcubes Ω_{ijk}. In the third problem we use 244 subdomains which are not cubes; their aspect ratios are 4:5:20.

- **Problem 1.** Find u^h such that

$$\sum_{ijk} \int_{\Omega_{ijk}} \epsilon_{ijk}(\nabla u^h, \nabla v^h) = \int_\Omega f v^h, \qquad \forall v^h \in V^h.$$

  The boundary conditions are given by $u^h = 0$ on $\partial\Omega$. The coefficients ϵ_{ijk} are constant on each subdomain and have large jumps between neighboring subdomains. Specifically, $\epsilon_{ijk} = \sin^2(16z)(e^{18x\sin(4y)} + e^{15(1-x)}) + 1$, where (x, y, z) is the center of Ω_{ijk}. The right-hand side is given by $f(x, y, z) = ze^x \sin(y)$.
- **Problem 2.** Find u^h such that

$$\int_\Omega (\nabla u^h, \nabla v^h) = \int_\Omega f v^h, \qquad \forall v^h \in V^h.$$

  The solution u^h is constrained to be zero on one face of the cube and is free on the rest of the boundary. The right-hand side is the same as in Problem 1.
- **Problem 3.** This problem is the same as in Problem 2 except that the domain is as depicted in Fig. 4.5. The solution u^h is fixed on the bottom of the object and free on the rest of the object's boundary.

All the results are for one multigrid V-cycle sweep as an approximate solver for the interior problems, that is, two sweeps per subdomain per iteration. This choice was made because additional multigrid sweeps did not result in a decrease in the number of outer iterations. The times needed when exact interior solvers are used (i.e. with $B_G^{-1}S$) are much higher than those for the approximate solver. In addition, we cannot run the large problems when using exact interior solvers, as the sparse factors take a large percentage of the available space. For instance, for Problem 2 with 64 subdomains, the largest problem we could solve using the sparse interior solver was with a mesh of $H/h = 16$, while with multigrid we could solve problems with meshes up to $H/h = 32$. This fact suggests that for well-behaved problems like the Poisson problem, exact interior solvers, such as banded or sparse linear system solvers, are too expensive to be competitive. For more difficult problems, we do not yet know which approach is superior. We used the ordering routines in the Yale Sparse Matrix Package to order the unknowns for the sparse interior solvers. The nested dissection ordering might be a better choice. The results for Problems 1 and 2 are given in Tables 7 and 8, respectively.

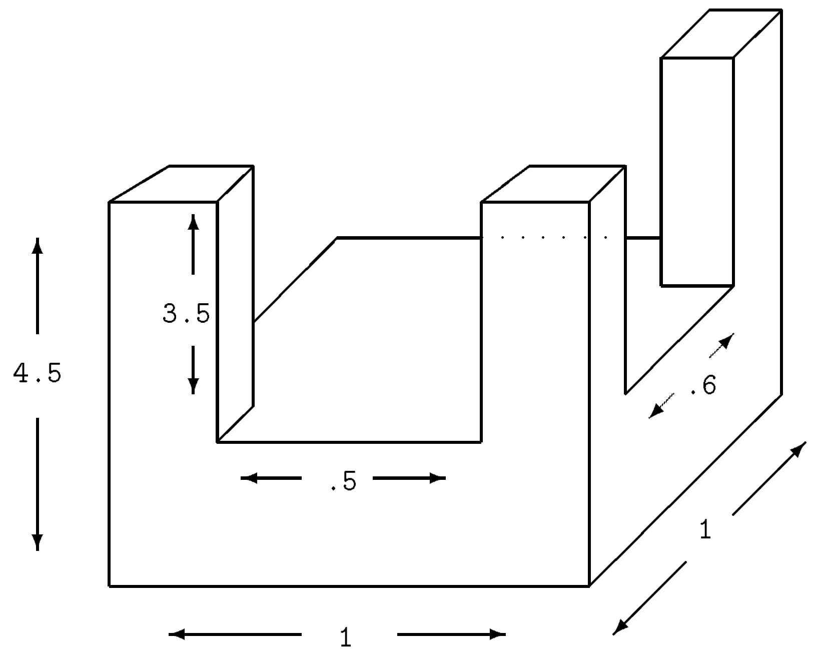

FIG. 2. *Domain for problem 3*

For Problem 3 (see Table 9), the iteration counts are slightly higher than for Problem 2. It is well known that increasing aspect ratios cause a decay in the convergence rate of domain decomposition algorithms. In Problem 3 the algorithm with a diagonal scaling as the wirebasket problem performs poorly; see the column labeled $B_{DA}^{-1}K$ in Table 9. This poor performance is because the condition number of the preconditioned problem grows like $1/H^2$, so the preconditioner becomes less effective when a large number of subdomains is used.

4.6. Speed of Computational Kernels. Most large numerical codes have a few routines that perform the bulk of the numerical calculations and use most of the CPU time. We refer to these routines as the computational kernels. The best-known computational kernels are the BLAS and FFT. The computational kernels involve no communication with other processes and should ideally vectorize and pipeline well. It is also important that they use the data and instruction caches well. Since the computational kernels dominate the time of the entire calculation, their optimization is important. On certain processors, the Intel i860 for example, the replacement of Fortran or C computational kernels with assembler language kernels can result in large decreases in the time of the calculation at the expense of a great deal of careful hand-coding of assembler code. We note that improvement in the speeds of computational kernels is a *local* optimization and does not involve communication or parallelization.

The present code is all written with standard Fortran and C computational kernels. We have observed the following floating-point speeds; see Fig. 4.6.

- Dot product; 4.9 million floating-point operations per second (MFLOPS).
- Matrix multiply; 7.8 MFLOPS.

Table 7

Problem 1 with 64 subdomains (time in seconds)

H/h	Number Unknowns	Number Processors	Diagonal	$B_{G^A}^{-1}K$	$B_{D^A}^{-1}K$
8	29,791	Number of Iterations	86	19	42
		Condition Numbers	221	14.1	62
		4	14.1	14.8	23.0
		8	8.3	8.8	12.8
		16	5.7	5.4	8.0
		32	3.9	3.7	5.0
16	250,047	Number of Iterations	169	24	49
		Condition Numbers	885	23.1	89
		4	94.2	85.6	155.1
		8	51.7	44.9	80.9
		16	31.2	25.9	44.3
		32	17.0	14.3	23.2
20	493,039	Number of Iterations	212	26	50
		Condition Numbers	1,379	26.6	99
		8	113.0	109.4	187.7
		16	65.2	58.7	98.9
		32	34.2	30.4	50.7
24	857,375	Number of Iterations	256	28	53
		Condition Numbers	1,984	29.5	107
		8	193.7	163.5	290.5
		16	110.1	86.8	151.9
		32	57.4	44.8	77.6
32	2,048,383	Number of Iterations	343	30	55
		Condition Numbers	3,525	34.0	119
		32	153.9	119.3	207.3

- DAXPY; 4.8 MFLOPS.
- Multigrid solver; 3.4 MFLOPS.
- Diagonal preconditioner; 2.7 MFLOPS.
- Sparse factorization; 3.8 MFLOPS.
- Sparse triangular solvers; 4.9 MFLOPS.

These results were obtained from the largest set of problems listed in Table 8. They do not fit completely in cache.

In the problem with 2,130,048 unknowns listed in Table 8, the per-processor flop rate for the entire calculation (from distributing the geometric information to the nodes, to solving the system) was 4.0 MFLOPS for the diagonal preconditioner and 3.1 MFLOPS for the more sophisticated preconditioner. Yet the diagonal preconditioner took more than twice as much time. The lower overall flop rate for the sophisticated

TABLE 8

Problem 2 with 64 subdomains (time in seconds)

H/h	Number Unknowns	Number Processors	Diagonal	$B_{G^A}^{-1}K$	$B_{D^A}^{-1}K$
8	34,848	Number of Iterations	129	17	65
		Condition Numbers	4,972	16.9	1,200
		4	24.1	16.6	40.7
		8	15.5	9.0	22.2
		16	8.5	5.5	13.0
		32	5.4	3.9	7.5
16	270,400	Number of Iterations	262	23	78
		Condition Numbers	19,916	27.2	1,644
		4	159.8	92.9	261.7
		8	86.9	48.2	136.1
		16	49.7	26.1	72.2
		32	26.6	14.1	37.6
20	524,880	Number of Iterations	325	25	83
		Condition Numbers	31,121	31.6	1,788
		8	168.0	89.9	265.3
		16	94.1	48.2	138.4
		32	49.8	25.3	71.4
24	903,264	Number of Iterations	388	28	89
		Condition Numbers	44,817	38.8	1,911
		8	304.2	170.3	481.6
		16	168.2	89.5	248.5
		32	87.6	46.8	129.1
32	2,130,048	Number of Iterations	522	32	91
		Condition Numbers	79,682	51.0	2,101
		32	233.4	130.8	347.8

preconditioner can be explained by the much lower flop rate of the multigrid solver.

The per-processor flop rate was obtained by taking the total number of floating-point operations performed on the processor and dividing by the total time the processor was in operation, including the time it is communicating with the other processors. While this number is a useful indicator of how well the processor is being utilized, it should not be overemphasized. The goal is to *minimize total computation time.* The best algorithm is the one that does exactly that, even if its per-processor flop rate is lower than that for other algorithms.

In Figures 4.6 and 4.6, we graph the percentage of total wall-clock time spent in each portion of the code for the diagonal preconditioned and fully preconditioned problems. This gives a clear indication of what in the code can most fruitfully be optimized. We also graph, in Fig. 4.6, the percentage of wall-clock time spent in various parts of the

TABLE 9

Problem 3 with 244 subdomains (time in seconds)

H/h	Number Unknowns	Number Processors	Diagonal	$B_{G^A}^{-1}K$	$B_{D^A}^{-1}K$
8	132,792	Number of Iterations	309	20	152
		Condition Numbers	19,657	20.8	4,710
		8	143.2	54.9	217.4
		16	78.2	35.9	114.9
		32	48.3	26.6	66.4
16	1,030,512	Number of Iterations	617	35	179
		Condition Numbers	78,486	74.1	6,428
		16	427.1	155.1	629.0
		32	237.2	88.5	332.1
20	2,000,460	Number of Iterations	772	39	187
		Condition Numbers	122,582	93.9	6,981
		32	453.4	157.5	622.7

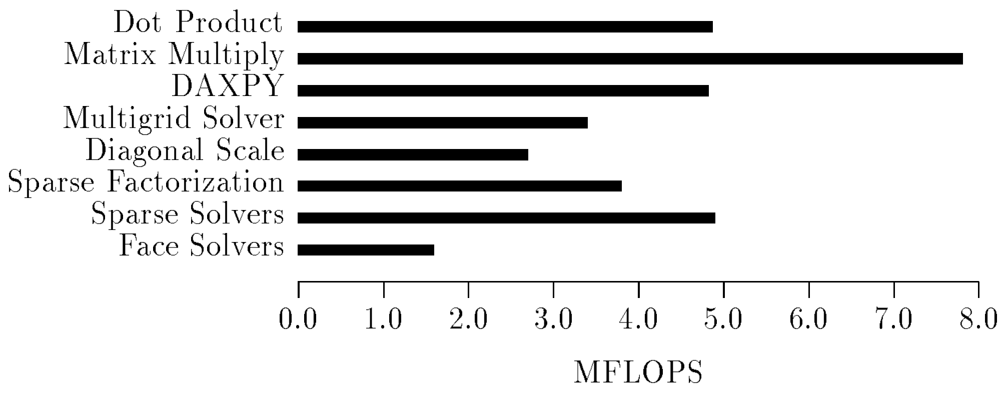

FIG. 3. *Flop rate in the computational kernels*

code for Problem 2 when the sparse interior solver is used. This is for 64 subdomains with a mesh of $H/h = 16$, the largest problem we could fit onto 32 processor nodes while using the sparse direct solver to solve the interior problems. The overall flop rate obtained here was 3.8 MFLOPS.

For the largest problem for which the sparse solver was used, 4.7% of the total computation time was spent on interprocessor communication. When multigrid was used to approximately solve the interior problems, the communication time increased to 11.7% of the total time. With diagonal preconditioning, 26.2% of the time was devoted to interprocessor communication.

5. Conclusions and Future Directions. We believe that our results indicate that the iterative substructuring approach is a viable technique for the solution of elliptic partial differential equations in three dimensions on modern distributed-memory machines. For model problems, the iterative substructuring algorithm performs better

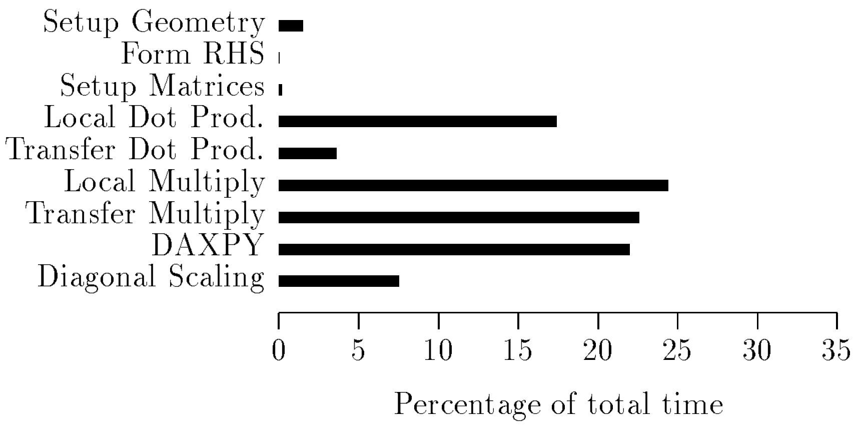

FIG. 4. *Percentage of time in different states, diagonal preconditioner*

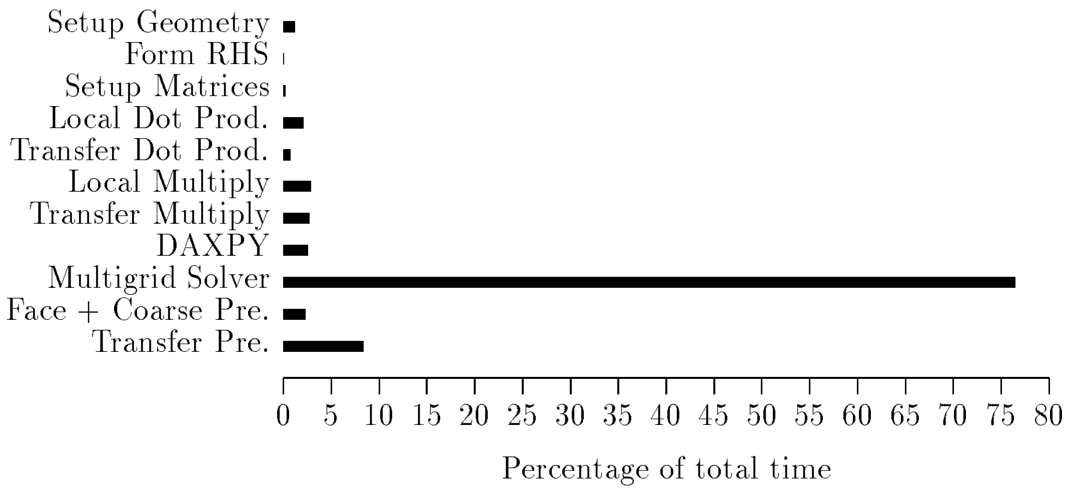

FIG. 5. *Percentage of time in different states, full preconditioner*

than diagonal scaling, but not by an enormous factor. In fact, it would appear that is not enough to justify the extra burden imposed by coding the algorithms. However, we believe that for nontrivial problems, the difference between the two approaches in terms of computational time will increase.

We also note that for solving extremely large problems, it is important to use approximate solvers for the interior problems. The fill from a band or sparse solver begins to dominate the memory usage, making it impossible to solve extremely large problems. For difficult problems, we may not be able to find an iterative solver for the interior problems that performs well enough to serve as replacement for the direct solver, and this fact might limit the size of the problems that we could solve.

The iterative substructuring algorithm considered here works well on simple, piecewise, constant coefficient problems. To be useful in practice, it must be adaptable to a wide range of multicomponent elliptic partial differential equations. We therefore plan to focus on adapting each piece of the algorithm to a wide range of differential equations. The parts of the algorithm that must be generalized are the face preconditioners, the

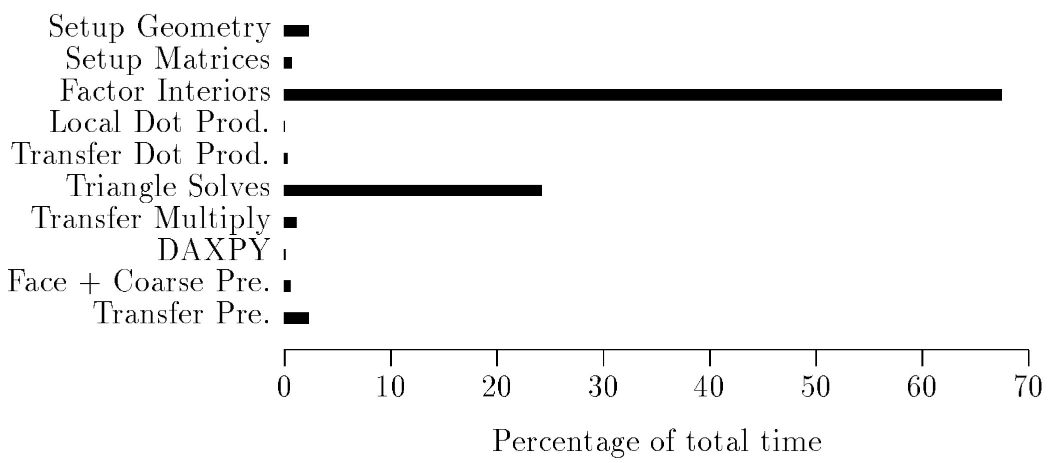

FIG. 6. *Percentage of time in different states, sparse solver*

wirebasket coarse problem, and the interpolation onto the faces from the wirebasket. In addition, we shall consider other approaches to building interior iterative solvers, such as incomplete factorizations. Finally, we shall consider nonsymmetric problems.

Acknowledgments: I thank Bill Gropp for his excellent advice on the development of this implementation of the algorithm. In addition, I thank Olof Widlund for his assistance in revising this paper.

REFERENCES

[1] P. E. BJØRSTAD AND A. HVIDSTEN, *Iterative methods for substructured elasticity problems in structural analysis*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., SIAM, Philadelphia, 1988, pp. 301–312.

[2] P. E. BJØRSTAD, R. MOE, AND M. SKOGEN, *Parallel domain decomposition and iterative refinement algorithms*, in Parallel Algorithms for PDEs, Proceedings of the 6th GAMM-Seminar held in Kiel, Germany, January 19–21, 1990, W. Hackbusch, ed., Vieweg-Verlag, Braunschweig, Wiesbaden, 1990.

[3] C. BÖRGERS, *The Neumann–Dirichlet domain decomposition method with inexact solvers on the subdomains*, Numer. Math., 55 (1989), pp. 123–136.

[4] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring, IV*, Math. Comp., 53 (1989), pp. 1–24.

[5] T. F. CHAN AND T. Y. HOU, *Domain decomposition interface preconditioners for general second order elliptic problems*, Tech. Rep. CAM 88-16, Department of Mathematics, UCLA, 1988.

[6] T. F. CHAN AND D. F. KEYES, *Interface preconditioning for domain-decomposed convection-diffusion operators*, Tech. Rep. CAM 89-28, Department of Mathematics, UCLA, 1989.

[7] T. F. CHAN AND T. P. MATHEW, *The interface probing technique in domain decomposition*, Tech. Rep. CAM 91-02, Department of Mathematics, UCLA, 1991.

[8] Y.-H. DE ROECK, *A local preconditioner in a domain-decomposed method*, Tech. Rep. TR89/10, Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France, 1989.

[9] Y. DE ROECK AND P. LE TALLEC, *Analysis and test of a local domain decomposition preconditioner*, in Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, Y. Kuznetsov, G. Meurant, J. Périaux, and O. Widlund, eds., SIAM, Philadelphia, 1991, pp. 112–128.

[10] M. Dryja, *A method of domain decomposition for 3-D finite element problems*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., SIAM, Philadelphia, 1988, pp. 43–61.

[11] M. Dryja, B. F. Smith, and O. B. Widlund, *Schwarz analysis of iterative substructuring algorithms for problems in three dimensions*, Tech. Rep., Department of Computer Science, Courant Institute, 1991. Also Mathematics and Computer Science Preprint MCS-P250-0791, Argonne National Laboratory, 1991.

[12] M. Dryja and O. B. Widlund, *Some domain decomposition algorithms for elliptic problems*, in Iterative Methods for Large Linear Systems, Academic Press, San Diego, California, 1989, pp. 273–291.

[13] G. Haase, U. Langer, and A. Meyer, *A new approach to the Dirichlet domain decomposition method*, Tech. Rep., Technical University of Chemnitz, 1990.

[14] D. E. Keyes and W. D. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s166–s202.

[15] ——, *Domain decomposition techniques for nonsymmetric systems equations: Examples from computational fluid dynamics*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., SIAM, Philadelphia, 1989, pp. 312–339.

[16] P. Le Tallec, Y.-H. De Roeck, and M. Vidrascu, *Domain-decomposition methods for large linearly elliptic three dimensional problems*, Tech. Rep. TR/PA/90/20, Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France, 1990.

[17] J. Mandel, *Hierarchical preconditioning and partial orthogonalization for the p-version finite element method*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., SIAM, Philadelphia, 1990, pp. 141–156.

[18] ——, *Two-level domain decomposition preconditioning for the p-version finite element version in three dimensions*, Internat. J. Numer. Methods Engrg., 29 (1990), pp. 1095–1108.

[19] B. F. Smith, *A domain decomposition algorithm for elliptic problems in three dimensions*, Tech. Rep. 519, Department of Computer Science, Courant Institute, October 1990. Also Mathematics and Computer Science Division Preprint MCS-P185-1090, Argonne National Laboratory, 1990. To appear in Numer. Math.

[20] ——, *Domain Decomposition Algorithms for the Partial Differential Equations of Linear Elasticity*, Ph.D. thesis, Courant Institute of Mathematical Sciences, September 1990. Tech. Rep. 517, Department of Computer Science, Courant Institute, 1990.

[21] B. F. Smith and O. B. Widlund, *A domain decomposition algorithm using a hierarchical basis*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 1212–1220.