OVERLAPPING DOMAIN DECOMPOSITION ALGORITHMS FOR GENERAL SPARSE MATRICES *

XIAO-CHUAN CAI† AND YOUCEF SAAD‡

Abstract. Domain decomposition methods for Finite Element problems using a partition based on the underlying finite element mesh have been extensively studied. In this paper, we discuss algebraic extensions of the class of overlapping domain decomposition algorithms for general sparse matrices. The subproblems are created with an overlapping partition of the graph corresponding to the sparsity structure of the matrix. These algebraic domain decomposition methods are especially useful for unstructured mesh problems. We also discuss some difficulties encountered in the algebraic extension, particularly the issues related to the coarse solver.

Key words. Sparse matrix, iterative methods, preconditioning, graph partitioning, domain decomposition.

1. Introduction.

The aim of this paper is to develop parallel preconditioned iterative methods for solving general large sparse linear systems that arise from the discretization of partial differential equations, more particularly on unstructured meshes. We are interested in the class of overlapping Schwarz domain decomposition preconditioners that were previously introduced in the context of variational solution of partial differential equations; see [5] and references therein. According to the divide-and-conquer philosophy underlying the domain decomposition approach, the domain of definition of the partial differential equation is partitioned into a set of subdomains whose union is the original domain and the partial differential equations are then discretized on each of the subdomains. The solution of the original PDE is obtained typically by a Krylov space type iterative method, such as the generalized minimal residual algorithm (GMRES) [15], which is preconditioned by an operator which typically incorporates the solutions of the subproblems.

Our goal in this paper is to extend the framework of the overlapping domain decomposition approach to general sparse linear systems. The fundamental principle underlying this extension is to replace the *domain of definition* of the problem by the *adjacency graph* of the sparse matrix, i.e., the graph that represents its non-zero pattern. We note that by switching from a domain to a graph the concept of Euclidean distance,

* Appeared as Preprint 93-027, Army High Performance Computing Research Center, University of Minnesota, 1993. (SIAM J. Sci. Comp., submitted)

† Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309. `cai@cs.colorado.edu`. The work of this author was supported in part by the National Science Foundation and the Kentucky EPSCoR Program under grant STI-9108764 and in part by a contract between the Army Research Office and the University of Minnesota for the Army High Performance Computing Research Center under grant number DAAL03-89-C-0038, while in residence at the University of Minnesota.

‡ Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. `saad@cs.umn.edu`. The work of this author was supported in part by DARPA under grant number 60NANB2D1272 and in part by a contract between the Army Research Office and the University of Minnesota for the Army High Performance Computing Research Center under grant number DAAL03-89-C-0038.

which plays an important role in the optimality analysis of these domain decomposition methods, is lost. We show in this paper, mostly by means of numerical experiments, that the efficiency of the overlapping methods can be preserved to some extent with certain well-balanced overlapping graph decomposition. Other preconditioned iterative methods can also be used to solve this class of sparse systems and the interested reader should refer to [1, 6, 13] for further references.

In the practical implementation of the algebraic Schwarz algorithms, a crucial step resides in the non-numerical preprocessing of the problem, e.g., graph partitioning, graph coloring, etc.. As is well-known in graph theory many of the problems that arise in this context, such as the perfect graph coloring problem, are NP-hard. However, there are often very inexpensive heuristic algorithms that deliver more than adequate results. In this paper we will restrict our attention to such heuristics.

The paper is organized as follows. In Section 2, we will briefly review that classical (variational) Schwarz algorithms, which motivate the current paper. We then introduce, in Section 3, an overlapping graph partitioning scheme based on which we define the algebraic Schwarz algorithms. Both additive and multiplicative versions will be discussed. In Section 4, we discuss a rather difficult issue regarding the use of coarse solvers to speed up convergence. Section 5 is devoted to the discussion and description of some useful tools for graph decompositions, graph coloring, etc. Finally, in Section 6, we present some preliminary numerical experiments.

2. Review of Variational Schwarz Algorithms. Before presenting the algebraic formulation of the Schwarz algorithms, we give a brief review of the classic Schwarz algorithms, including the additive and multiplicative versions. Details of the classic versions can be found in [3, 4, 5]. To illustrate the ideas of Schwarz type algorithms, we consider a homogeneous Dirichlet boundary value problem:

$$\left\{ \begin{array}{rcll} Lu &=& f & \text{in} \quad \Omega, \\ u &=& 0 & \text{on} \quad \partial\Omega, \end{array} \right. \tag{1}$$

where Ω is a two- or three-dimensional domain with boundary $\partial\Omega$. Using a Green's formula, we obtain the weak form of the continuous and discrete problems: Find $u \in V$, such that

$$a(u,v) = f(v), \quad \forall\ v \in V \tag{2}$$

and find $u_h \in V_h$, such that

$$a(u_h, v_h) = f(v_h), \quad \forall\ v_h \in V_h \tag{3}$$

respectively. Here V^h is a finite dimensional subspace of the Sobolev space $V = H_0^1(\Omega)$ and $a(\cdot,\cdot)$ is the usual bilinear form associated with the elliptic operator L. Following the Dryja-Widlund construction of the overlapping decomposition of V^h (cf. [5]), the triangulation of Ω is introduced as follows. The region is first divided into nonoverlapping substructures Ω_i, $i = 1, \cdots, N$. Then all the substructures Ω_i, which have diameter of order H, are divided into elements of size h. The common assumption, in finite element theory, that all elements are shape regular is adopted. To obtain an overlapping decomposition of the domain, we extend each subregion Ω_i to a larger region Ω_i', i.e. $\Omega_i \subset \Omega_i'$. We assume that the overlap is uniform and $V_i \subset V^h$ is the usual finite element space over Ω_i'. It clear that

$$\Omega = \bigcup_i \Omega_i'$$

and

$$V^h = V_1 + \cdots + V_N.$$

Equation (3) yields a large, sparse, linear system of equations,

$$Au = f \tag{4}$$

which is usually not well-conditioned. Therefore, a good preconditioner plays an essential role in the success of any iterative methods used to solve it. For Schwarz methods, cf. Dryja and Widlund [5], the preconditioner is constructed by solving a sequence of subdomain problems of the form: Find $T_i e \in V_i$, such that

$$a(T_i e, v) = b(e, v), \quad \forall v \in V_i.$$

$T_i e$ is a projection of the error onto the subspace V_i. There are quite a few different ways to construct preconditioners with the operators T_i, see for example[2]. For simplicity, we discuss only the additive and the multiplicative Schwarz algorithms. The additive Schwarz preconditioned system can be written as

$$M^{-1}Au \equiv (T_1 + \cdots + T_N)u = g. \tag{5}$$

We note that the action of T_i on a vector u can be carried out simultaneously in parallel. The preconditioned system for the multiplicative Schwarz can be written as

$$M^{-1}Au \equiv (I - (I - T_N) \cdots (I - T_1))u = g. \tag{6}$$

We remark that the operator T_i can be expressed in matrix form as $T_i = R_i^T A_i^{-1} R_i A$, where R_i is the restriction matrix and A_i^{-1} is the subdomain problem solve. In practice, for the additive algorithm, we usually let the number N of subdomains be equal to the number of available processors, and the size of subproblems, including the size of overlap, be determined by the available memory on each processor. For the multiplicative algorithm, in order to improve parallelism, the subdomains are usually colored and before calling the multiplicative Schwarz algorithms the subdomain i, therefore T_i, is redefined as the union of the original disconnected subdomains that share the same color i.

Both algorithms discussed above are one-level algorithms and the convergence rates deteriorate, linearly in the number of T_is, as the number of subdomains increases. A coarse solver is usually included to prevent the loss of optimal convergence. We refer to [2, 3, 4, 5] for further discussions of this issue.

3. Algebraic Schwarz Algorithms. We consider a linear system of the form,

$$Au = f \tag{7}$$

where A is a sparse matrix, having a symmetric pattern. To describe a model algebraic Schwarz algorithm, let us define the graph $G = (W, E)$, where the set of vertices $W = \{1, \cdots, n\}$, represents the n unknowns and the edge set $E = \{(i,j) \mid a_{i,j} \neq 0\}$ represents the pairs of vertices that are coupled by a nonzero element in A. Here n is the size of the matrix. Since we assume that the non-zero pattern is symmetric the adjacency graph G is indirected. There are several algorithms described in the literature for partitioning a graph into subgraphs [10, 11, 14]. The approach described in [10] is a form of nested dissection algorithm, and the author proposes a number of strategies to find good separators. In [11] a spectral analysis of Discrete Laplacian is exploited. In section 5 we give a brief description of a technique presented in [14]. In summary, the algorithm consists of two phases. The first phase finds a set of n_0 initial vertices that are reasonably well spread apart in the graph. Ideally, n_0 should be equal to the number of processors, but it is typically larger, and this issue is discussed in Section 5. We will refer to these nodes as *centers*. Once these centers are found we proceed with a level-set expansion from each of them to build the subdomains. Each subdomain initially consists of one node only, namely the center. At each step of the expansion from a center, we add each unmarked node of the next level set. We recall that a level set is defined recursively as the set all unmarked neighbors of all the nodes of a previous level set. As soon as a level set is traversed its nodes are added to a subdomain and they are marked. When all nodes are marked, a nonoverlapping partition of G into n_0 subgraphs is obtained. To generate an overlapping partition of G, we further expand each subgraph by a certain number, denoted as `ovlp`, of level sets as if all nodes are unmarked. A detailed description of the graph partition algorithm will be given in Section 5.

For the remaining discussion, we will assume that the graph partitioning has been applied and has resulting in a number p of subsets W_i whose union is W,

$$W = \bigcup_i W_i.$$

Here p is generally smaller than n_0 as some of the subgraphs may be combined. The result of the partitioning algorithm is illustrated with a simple example in Figure 1.

3.1. Additive Schwarz. We will denote by L_i the vector space spanned by the set W_i in $\mathbf{R}^n$ and by m_i its dimension. For each subspace L_i we define the orthogonal projector onto L_i. In matrix terms, this is defined by the sub-identity matrix I_i of size $n \times n$ whose diagonal elements are set to one if the corresponding node belongs to W_i and to zero otherwise. With this we define the matrix,

$$A_i = I_i A I_i \ ,$$

which is an extension to the whole subspace, of the restriction of A to L_i. This is sometimes termed the *section* of A on L_i. Its action on a vector is to project it on

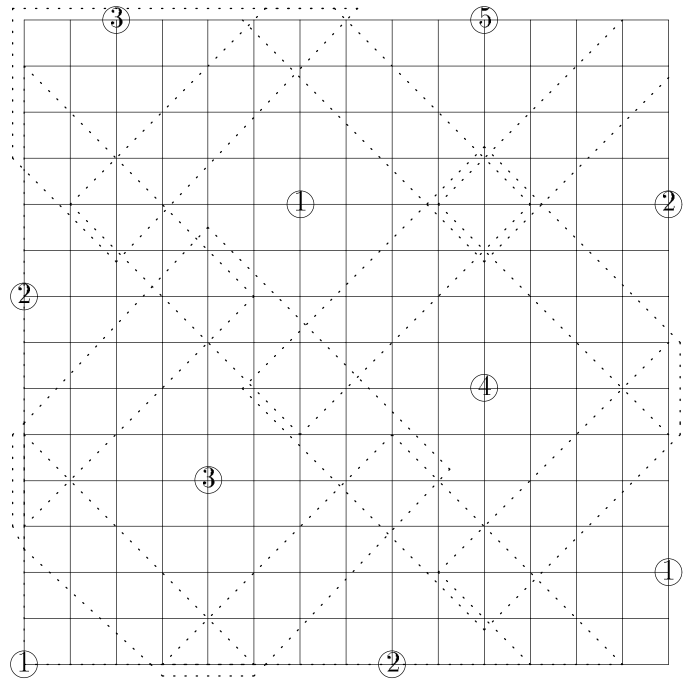

FIG. 1. *The graph partitioning algorithm in action for a 15×15 grid on the unit square. The circled nodes are the initial nodes of S_0, output of the coarsen algorithm. The ten subgraphs are bounded by the dotted lines. The numbers, 1-5, in the circles indicate the color of the subdomains found by the Coloring Algorithm.*

L_i, then apply A to the result and finally project the result back onto L_i. Note that although A_i is not invertible, we can invert its restriction to the subspace spanned by W_i, and define

$$A_i^{-1} \equiv I_i \left((A_i)_{|L_i} \right)^{-1} I_i$$

With this definition, the additive Schwarz algorithm can now be simply described as follows

ALGORITHM 1 (ADDITIVE SCHWARZ). *Solve the equation*

$$M^{-1} A u = M^{-1} f$$

by a Krylov subspace method, where the preconditioning M is defined by

$$M^{-1} = A_1^{-1} + \cdots + A_N^{-1} \ .$$

We note that the particular case where there is no overlapping, i.e., when the W_i's form an actual *partition* of W, then the Additive Schwarz algorithm is nothing but a block Jacobi preconditioned Krylov subspace iteration.

3.2. Multiplicative Schwarz. We next define the multiplicative Schwarz algorithm. It can be seen easily that if the multiplicative Schwarz algorithm is used as in the form of (6), then it is a purely sequential algorithm, however, if we color and regroup the subgraphs such that each is a union of several disconnected subgraphs then a great deal of parallelism can be introduced (see discussion in [4]), i.e. the subproblems defined on the disconnected subgraphs can be solved independently in parallel. Simple greedy heuristic subgraph coloring algorithms have been discussed in the literature, see for example, [13]. For the completeness of this paper, we will give a description of one such coloring algorithm in Section 5.

Let us define the matrix $\hat{A}_i$ as the sum of all the matrices A_i that share the same color. Here the color of the matrix is the same as that of its subgraph. $\hat{A}_i^{-1}$ can also be defined accordingly. The multiplicative Schwarz algorithm can now be defined as

ALGORITHM 2 (MULTIPLICATIVE SCHWARZ). *Solve the equation*

$$M^{-1}Au = M^{-1}f$$

by a Krylov subspace method, where the preconditioning operation $w := M^{-1}v$ is defined by the sequence of operations,

$$\begin{aligned} v_1 &= \hat{A}_1^{-1}v \\ v_j &= v_{j-1} + \hat{A}_j^{-1}(v - Av_{j-1}) \ , \quad \textit{for} \quad j = 2, \ldots, J. \\ w &= v_J. \end{aligned}$$

We note that the independent subproblems within each of the above sequential steps can be solved in parallel and that the preconditioned matrix has the form,

$$M^{-1}A = I - (I - \hat{A}_1^{-1}A) \cdots (I - \hat{A}_J^{-1}A).$$

With a straightforward implementation, the Multiplicative Schwarz algorithm will require a number of sequential steps equal to the number of colors, since only one color set can be active at a time. This may limit the efficiency of the algorithm. In such a case it would be important for the coloring algorithm to use as small a number of colors as possible, in order to minimize the number of sequential steps. In addition, we note that the convergence rate of the multiplicative Schwarz depends inversely on the number of colors, cf [4]. As a result, minimizing the number of colors will not only increase the parallelism but it will also improve the intrinsic convergence rate. We found that the greedy coloring algorithm just mentioned does give satisfactory results in practice. A overlapping subdomain example with ten subdomains, can be found in Fig. 1, where the colors are indicated by the circled numbers.

However, regarding the reduction in efficiency we note that *there are many possible remedies*. The simplest of these remedies is to have more subdomains than there are processors and to assign at least one subdomain from each color to each processor. In practice, this is quite simple to achieve by performing a two-level graph partitioning.

First we proceed as before and use the domain partitioning algorithm described earlier to get the actual node to processor mapping. Then each subdomain is further subdivided independently into a small number of sub-subdomains, normally just four. We then color the corresponding global subdomain partition with the assumption that all sub-subdomains in the same subdomain are connected. This assumption guarantees that all sub-subdomain colors in the same subdomain are different, and will decrease the likelihood that a processor is ever idle during the algorithm. Note that we do not know how the additional partitioning will affect the total number of colors. There are several other alternatives which we do not consider here.

Finally, we remark that, in the algorithms discussed above, all subproblems are assumed to be solved exactly; usually with Gaussian elimination. However, our numerical experiments show that this is not really necessary. In the Numerical Experiments section of this paper, we shall present some examples in which the subproblems are solved with the incomplete LU factorization ILU(k).

3.3. The Symmetric Case. In this paper, we do not assume that A is symmetric or positive definite, therefore the Conjugate Gradient (CG) Algorithm was not mentioned as a possible iterative method for solving the preconditioned systems. In fact, if the matrix A is symmetric and positive definite, then the additive Schwarz preconditioned system is also symmetric and positive definite with respect to the energy inner product defined as $(A\cdot,\cdot)$, hence the standard CG can be used. A better known alternative is to use the M-inner product for the preconditioned system $M^{-1}Au = M^{-1}f$ which is again self-adjoint with respect to this inner product.

In the case of multiplicative Schwarz preconditioning, a symmetrization technique has to be used in order to obtain an A-self-adjoint preconditioned system, we refer to [4] for further discussions. The symmetrization corresponds to doing a forward and backward sweep, as is usually done with SSOR, i.e., the preconditioning operation $w = M^{-1}v$ is now defined through the sequence of operations,

$$\begin{aligned}
v_1 &= \hat{A}_1^{-1}v \\
v_j &= v_{j-1} + \hat{A}_j^{-1}(v - Av_{j-1}) \ , \quad \text{for} \quad j = 2, \ldots, J. \\
v_j &= v_{j+1} + \hat{A}_j^{-1}(v - Av_{j+1}) \ , \quad \text{for} \quad j = J-1, \ldots, 1 \\
w &= v_1
\end{aligned}$$

4. Global Coarse Solvers. All the algorithms discussed in the previous section are one-level algorithms, in that they are simply direct extensions of the block Jacobi (additive) and the block Gauss-Seidel (multiplicative) preconditioners. As is known, the convergence rates of these algorithms deteriorate as the number of subdomains increases. This problem is successfully handled for the cases of variational Schwarz algorithms by inserting a solve with an extra coarse mesh space to the preconditioner. This allows to incorporate the needed communication between the almost decoupled local subspaces and prevents deterioration in convergence rates, see [5] for a theoretically analysis. However, for the case of general sparse matrices, defining the analogue of a 'coarse problem' and whether a similar cure to the deterioration of convergence rates can be achieved, are still almost entirely open issues.

Here we discuss a semi-automatic method that may speed up the convergence in some practical cases. In this method, we require the user to supply (i) a set of *cross points*, $C = \{c_i, i = 1, \cdots, m\} \subset W$; and (ii) a *cross-to-fine mapping* matrix $E = E_{m\times n}$. Supposing that (i)-(ii) are given, we define an $m \times m$ coarse matrix as

$$M_0 = EAE^T,$$

which can be obtained by two sparse matrix-matrix products. Assuming that M_0 is nonsingular, we therefore can define the coarse preconditioner as

$$A_0^{-1} = E^T (M_0)^{-1} E$$

and the additive and multiplicative Schwarz algorithms can be modified as

$$(A_0^{-1} + A_1^{-1} + \cdots + A_N^{-1})Ax = \hat{b}$$

and

$$\left[I - (I - A_0^{-1}A)(I - \hat{A}_1^{-1}A) \cdots (I - \hat{A}_J^{-1}A)\right] x = \hat{b}$$

respectively. Of course, $\hat{b}$ must also be modified.

5. Tools for Graph Decomposition Methods . In this section, we discuss several useful algorithms for the graph decomposition algorithm described in the previous section. Let `(a, ja, ia)` be the usual Compressed Sparse Row format(CSR), see [12], of the sparse matrix A [1] The graph $G = (W, E)$ of A is completely determined by the two one dimensional arrays `ia` and `ja`. Indeed the adjacency list for node i is simply the set of nodes $ja(k_1), ja(k_1) + 1, \ldots, ja(k_2)$ with $k_1 = ia(i), k_2 = ia(i + 1) - 1$.

5.1. Graph Partitioning. The graph partitioning algorithm described in [14] consists of two phases. The purpose of the first phase is to provide a subset S_0 of W consisting of points that are at nearly equal distance to each other. Note that here distance is understood in a graph theory sense, i.e., the distance between two nodes is the smallest number of edges needed to reach one node from the other. The way in which this "coarsening" phase works is by a recursive algorithm which defines a graph with far fewer nodes from the previous one until a graph with a satisfactory number of nodes is found. An essential step in the algorithm is to find an *independent set of a graph.* Given a graph $G = (W, E)$ an *independent set* S is a subset of the vertex set W such that

$$\text{if } x \in S \quad \text{then} \quad (x, y) \in E \ \text{ or } \ (y, x) \in E \rightarrow y \notin S.$$

[1] The array `a` contains the nonzero elements A_{ij} of A row by row; the integer array `ja` contains the column indices of the elements A_{ij} in the array `a`; and the integer array `ia` contains the pointers to the beginning of each row in `ja`

In other words, elements of S are not allowed to be coupled with other elements of S by incoming or outgoing edges. Finding such sets is relatively easy. An *independent set* is maximal if it cannot be augmented by elements in its complement to form a larger independent set. In what follows an independent set is always meant in the sense of a maximal independent set. Algorithms for finding such sets are described in [13]. The 'coarsening' algorithm can be described as follows.

ALGORITHM 3 (COARSENING ALGORITHM:).

Start: *Select a threshold number nodes* n_w.

Set $\hat{W} = W, \hat{E} = E$.

Coarsening Loop:

While $|\hat{W}| \geq n_w$ *do*

Find $S \subset \hat{W}$, *an independent set in* $\hat{W}, \hat{E}$.

On the set S *construct an edge set* F *by the rule:*

$(i,j) \in F$ *iff* $i \in S, j \in S, \exists\, k \in \hat{W},\ (i,k) \in \hat{E}$ *and* $(k,j) \in \hat{E}$

Define $\hat{W} := S,\ \hat{E} := F$.

EndWhile

Let us call S_0 the final set $\hat{W}$ obtained from the algorithm. Once the set S_0 of initial nodes has been found, we will perform a level-set expansion from each node in S_0. This is achieved by the following algorithm.

ALGORITHM 4. *Automatic Graph Partitioning Algorithm*

Start:

Find an initial set S_0 *of ndom 'coarse mesh' vertices,* $v_1, \ldots, v_{ndom}$

For $i = 1, 2, \ldots, ndom$ *Do:* $label(v_i) := i$.

Define $levset := \{v_1, \ldots, v_{ndom}\}$ *and* $nodes = ndom$

Loop: *While* $(nodes < n)$ *Do*

$Next\_levset = \phi$

For each v_j *in levset Do*

for each neighbor v_k *of* v_j *s.t.* $label(v_k) = 0$ *Do*

$Next\_levset := Next\_levset \bigcup \{v_k\}$

$label(v_k) := label(v_j)$

$nodes = nodes + 1$

EndFor

EndFor

$levset := Next\_levset$

EndWhile

The above two algorithms provide a basic way of partioning an arbitrary graph into subgraphs. There are several additions and improvements which we now briefly discuss.

Getting overlapping subgraphs. In Domain Decomposition, it is very common and desirable to obtain subdomains that have a small overlapping region. In the graph decomposition context discussed here we can achieve this quite easily by adding more level-sets in the level-set expansion of the the graph decomposition algorithm.

Obtaining a desired number of subdomains. As was already mentioned the number of subgraphs obtained by the Coarsening algorithm, is rarely equal to the desired number of subgraphs entered. This is due to the fact that from step to the other the number of coarse mesh points can be divided by a factor of 3 or 4 for typical graphs (roughly speaking, we are taking every other points in each direction). The result is that the number n_0 center nodes obtained may sometimes 3 or 4 times as large as the desired number n_w. The simplest cure is simply to take the first n_w points obtained in the set S_0 and ignore the others. The disadvantage of this strategy is that it will typically yield subdomains that are unbalanced, since the size of the subdomains is effected by the distance between the different initial nodes.

Obtaining well-balanced subgraphs. The sizes of the different subgraphs may vary by a factor of 2 to 3 with the simple implementation of Algorithm 4. In reality, it is easy to add a load-balancing criterion to the algorithm. After step 11 in Algorithm 4, we can update a counter to record the number of nodes that have been acquired by each subgraph. Then, we can set a priority rule by have the subgraph with fewer nodes have priority over the others. A simpler alternative is not to allow a subgraph to get more nodes if its size exceeds a given target size.

Parallel implementations. We mentioned earlier that Algorithm 4 is actually a parallel algorithm, since the level expansions can be performed from each node independently. The parallel version is best implementation with a host or master node which serves the role of arbitrator between the processors when there is contention as to which processor will acquire a vertex. On the other hand the graph-coarsening algorithm is not as trivially parallelizable. There is ample underlying parallelism however since nodes that are not adjacent can be treated at the same time.

5.2. Graph Coloring and independent set orderings. The general idea of graph-coloring has been successfully exploited by numerical analysts in many different ways. Whenever we have a graph $G = (W, E)$, we can color its nodes in such a way that no two adjacent vertices have the same color. Let $w_i, i = 1, \ldots, n$ be the elements of W. We can formulate a trivial heuristic for graph coloring as follows.

Algorithm 5. **Greedy multicoloring algorithm** .

1. **Start.** *For* $i = 1, \ldots, n$ *Do:* $Color(w_i) = 0$.
2. **Coloring Loop.** *For* $i = 1, 2, \ldots, n$ *Do:*

$$Color(w_i) = \min\{k > 0 \mid k \neq Color(w_j), \forall\ w_j \in \mathrm{Adj}(w_i))\}.$$

Here $\mathrm{Adj}(i)$ represents the set of nodes that are adjacent to w_i. The order in which the nodes w_i are listed, i.e., the order of traversal in the algorithm, may have an important effect on the number of colors found. The color assigned to node i in step 2 is simply the smallest allowable color number which can be assigned to node i, where *allowable* means different from the colors of the nearest neighbors and positive. The procedure is illustrated in Figure 2.

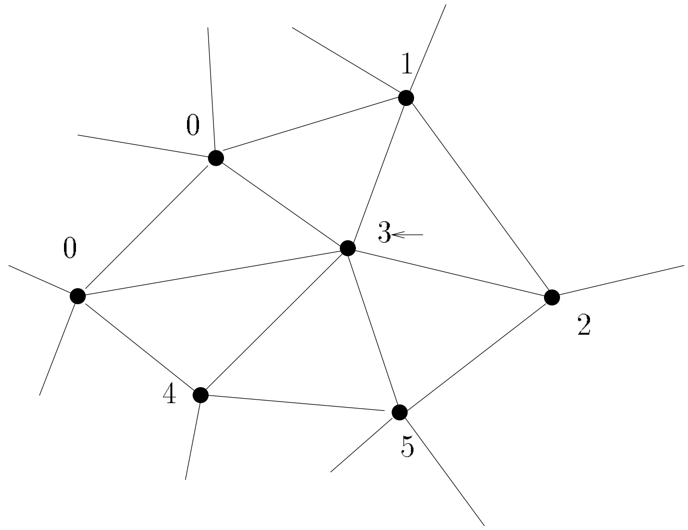

FIG. 2. *The greedy multicoloring algorithm. The node being colored is indicated by an arrow. It will be assigned color number 3, the smallest positive integers different from 1, 2, 4, 5.*

This algorithm can be used to color the subdomains in a partitioning. The vertices of the corresponding graph in this case represent the domains and the adjacency list for each vertex (i.e., subdomain) is the list of neighboring subdomains, i.e., subdomains that share common interface points.

6. Numerical Experiments. In this section, we present some preliminary numerical examples for the algorithms developed in the previous sections. There are quite a few parameters that can affect the overall performance of the algorithms, such as the initial number of subgraphs, initial ordering of the sparse matrices, number of colors, number of overlaps, as well as the methods used to solve the subdomain problems. In the section, we only discuss those parameters that are of interest to us currently. For a fixed preconditioner, there are also many accelerators which can be used such as GMRES [15], BiCGSTAB [16], TFQMR [7], etc.. However, we shall restrict ourselves to the use of the restarted GMRES algorithm. We consider the following two model problems.

Problem 0. We consider the Poisson equation

$$- \triangle u = f$$

with Dirichlet boundary condition on the unit square in R^2. The equation is discretized with the usual 5-point finite difference scheme on a uniform 128×128 grid.

Problem 1. We consider the convection-diffusion equation

$$- \triangle u + \gamma \left(\frac{\partial e^{xy} u}{\partial x} + \frac{\partial e^{-xy} u}{\partial y} \right) + \alpha u = f,$$

with Dirichlet boundary condition on the unit cube in R^3 and $\gamma = 10$, $\alpha = -10$. The equation is discretized with the usual 7 point centered difference scheme on an $15 \times 15 \times 15$ uniform mesh.

For all the above matrices, we construct the right-hand side artificially of the form $b = Ae$ such that the solution e is a random vector. The initial guess is always zero. All the testing problems are discretized on uniform meshes. However, this is not exploited except in one case where a coarse solver is defined with the mesh information in mind. The GMRES method, restarted at the 20^{th} iteration, is used for all of the preconditioned linear systems. The stopping criterion is the reduction of the initial (preconditioned) residual by five orders of magnitude in the L^2 norm, namely

$$(r_k, r_k)^{1/2} \leq 10^{-5}(r_0, r_0)^{1/2},$$

where $r_k = M^{-1}(b - Ax_k)$, for $k \geq 0$, and M^{-1} is one of the preconditioners discussed previously. All experiments were done on a Sparcstation 2 in double precision. The number of overlapping level-sets is denoted by *ovlp*. m_0 is the initial seed for the number of subgraphs.

The testing codes used in the experiments are developed with the help of two pieces of software, namely the SPARSKIT of Saad [12] and the package PETSc of W. Gropp and B. Smith [9].

6.1. One-level algorithms. The first suit of tests are for the additive Schwarz algorithms, see Tables 1 and 2. In these two tables, all the subdomain problems are solved exactly with Gaussian elimination. In Table 1, the column $ovlp = 0$ corresponds to the usual block diagonal preconditioning, or block Jacobian method. We note that the iteration counts have a sudden jump from the column $ovlp = 0$ to the column $ovlp = 1$ due to the introducing of the first level of overlap. The decreases of the iteration counts after the first level of overlap are not as noticeable as the first one.

The second set of tests are for the multiplicative Schwarz algorithms, see Tables 3 and 4, also with exact subdomain problem solves. We observe that the number of colors, which equals the number of sequential steps in the multiplicative Schwarz algorithms, is almost independent of the number of subdomains. However, due to the increase of data dependency in three dimensional problems when compared with two dimensional problems, the number of colors is indeed higher in the 3-D case, see Table 4, than in the 2-D case, see Table 3.

As mentioned previously, in practice, the subdomain problems usually need not to be solved exactly as shown in Tables 1-4. Some inexact solution techniques, such as incomplete LU(k), can be used instead. This can sometime save some CPU time both at the pre-iteration step, such as the factorization of the subdomain matrices, and during the iterations. In Fig. 3, we compare the iteration histories when we use the exact LU and ILU(k), with $k = 0, 1, 2$ and 3. Clearly, the exact LU offers the minimal number of iterations to reach the desired accuracy. However, the story changes when comparing the curves of CPU time versus iteration count. To reach the same accuracy, ILU(2) takes the least amount of CPU time in the iteration process. A similar conclusion has

Table 1

Iteration counts for the **additive Schwarz** *preconditioned GMRES(20) for* **Problem 0** *with* $n = 128 \times 128$ *and no coarse solver. The subdomain problems are solved by Gaussian elimination.*

m_0	# of subgraphs	$ovlp = 0$	$ovlp = 1$	$ovlp = 2$	$ovlp = 3$
2	2	19	15	13	12
4	5	21	17	16	14
8	13	28	24	20	19
16	41	47	32	25	21
32	41	47	32	25	21

Table 2

Iteration counts for the **additive Schwarz** *preconditioned GMRES(20) for* **Problem 1** *with* $n = 15 \times 15 \times 15 = 3375$, $nnz = 22275$ *and no coarse solver. The subdomain problems are solved by Gaussian elimination.*

m_0	# of subgraphs	$ovlp = 0$	$ovlp = 1$	$ovlp = 2$
2	2	9	8	7
4	9	21	18	18
8	9	21	18	18
16	40	29	28	26

also been reached in [13], where global ILU(k) preconditioners, among others, were discussed. We note that the most popularly used ILU(0) is not a good choice for this example.

6.2. Two-level algorithms. When the matrix A is obtained from the discretization, with a mesh parameter h, of a continuous differential equation, it has been shown, in [2, 8], that a two-level method which utilizes a coarse mesh performs considerly better than the corresponding one-level method. The coarse mesh matrix is usually obtained by using the same discretization scheme but with a much larger mesh parameter H. For general sparse matrices, we found that to obtain a good coarse matrix without knowing the underlying mesh structure is rather difficult.

Let us now assume that we are provided with some information about the matrix A that is sufficient to define a coarse matrix and its corresponding interpolation operator E. This coarse matrix is usually generated with the initial grid that is typically input to some mesh generation tool. To illustrate the idea, we present an example here in Table 5. We divide the unit square into an 4×4 triangular mesh and define the matrix $E_{16 \times n}$ row by row such that each row corresponds to the nodal values of the usual finite element hat function centered at that coarse grid point. The coarse matrix is thus obtained by two sparse BLAS routine calls, i.e., the sparse matrix-matrix products to compute EAE^T. In Table 5, we show what a difference a coarse solver can make for

TABLE 3
Iteration counts for the **multiplicative Schwarz** *preconditioned GMRES(20) for* **Problem 0** *with $n = 128 \times 128$ and no coarse solver. The subdomain problems are solved by Gaussian elimination.*

m_0	# of subgraphs	# of colors	$ovlp = 1$	$ovlp = 2$	$ovlp = 3$
2	2	2	8	7	7
4	5	3	8	7	6
8	13	4	10	9	8
16	41	4	12	10	9

TABLE 4
Iteration counts for the **multiplicative Schwarz** *preconditioned GMRES(20) for* **Problem 1** *with $n = 15 \times 15 \times 15$ and no coarse solver. The subdomain problems are solved by Gaussian elimination.*

m_0	# of subgraphs	# of colors	$ovlp = 1$	$ovlp = 2$	$ovlp = 3$
2	2	2	4	3	3
4	9	7	6	5	5
8	9	7	6	5	5
16	40	9	6	5	5

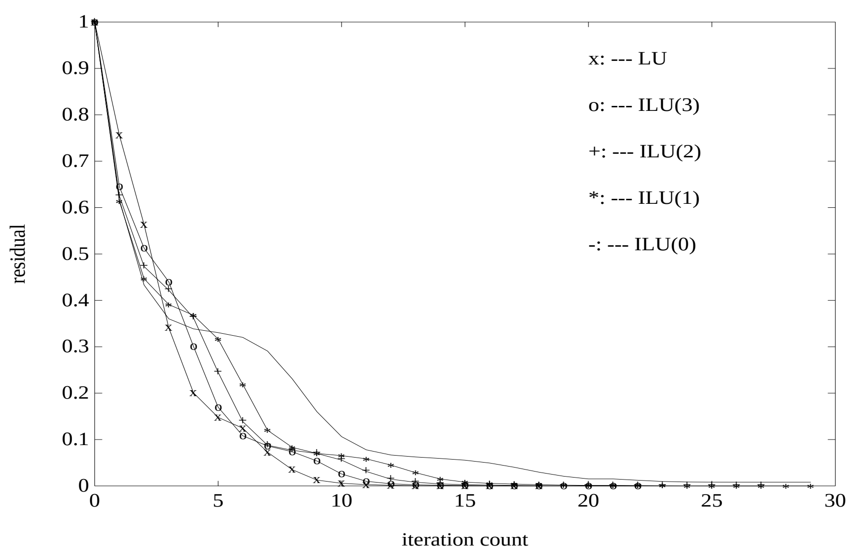

FIG. 3. *Residual curves for* **Problem 1**, *with the additive Schwarz preconditioner. The subdomain problems are solved by LU and ILU(k). The number of subdomains is 9 and overlap is 1.*

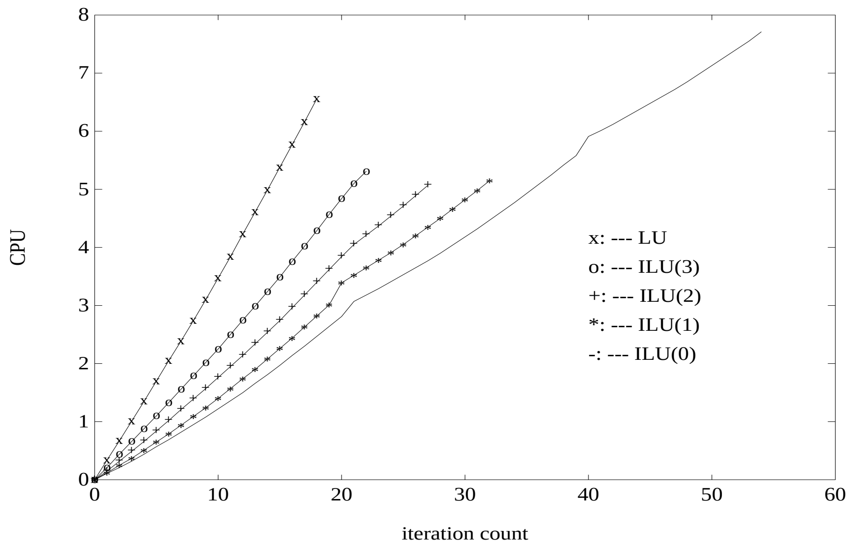

FIG. 4. *The curves for the CPU time(sec) versus iteration. The test problem and parameters are the same as in Fig. 3.*

the additive Schwarz methods when applied to the model problem. We can save more than half the number of iterations, with not too large a coarse grid solver. We remark that the subgraph problems are obtained as previously without the knowledge of the grid structure.

There are other ways to generate the matrix E, such as by a piecewise constant interpolation, or a piecewise linear-in-level set interpolation. However, our numerical experiments do not show any advantages of using these techniques.

TABLE 5

Iteration counts for the **additive Schwarz***, with a regular coarse grid solver, preconditioned GMRES(20) for* **Problem 0** *with* $n = 127 \times 127$*. All subproblems are solved exactly with Gaussian elimination.*

		coarse mesh 0×0	coarse mesh 4×4	coarse mesh 8×8
$m_0 = 4$ # of subgraphs $= 10$	ovlp=0	30	21	16
	ovlp=1	27	18	14
	ovlp=2	24	16	13
$m_0 = 50$ # of subgraphs $= 136$	ovlp=0	59	33	20
	ovlp=1	57	28	17
	ovlp=2	33	24	15

REFERENCES

[1] L. Adams and J. Ortega, *A multi-color SOR method for parallel computers*, Proceedings of Int. Conf. Par. Proc., 1982. pp. 53-56.

[2] X.- C. Cai, W. D. Gropp and D. E. Keyes, *A comparison of some domain decomposition algorithms for nonsymmetric elliptic problems,* J. Numer. Lin. Alg. Appl. 1993.

[3] X. -C. Cai and O. B. Widlund, *Domain decomposition algorithms for indefinite elliptic problems,* SIAM J. Sci. Stat. Comp. 13 (1992), pp. 243-258.

[4] X. -C. Cai and O. B. Widlund, *Multiplicative Schwarz algorithms for nonsymmetric and indefinite elliptic problems*, SIAM J. Numer. Anal. 30 (1993), pp. 936-952.

[5] M. Dryja and O. B. Widlund, *Towards a unified theory of domain decomposition algorithms for elliptic problems*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., SIAM, Philadelphia (1990).

[6] H. C. Elman and E. Agron, *Ordering techniques for the preconditioning conjugate gradient method on parallel computers*, UMIACS-TR-88-53, UMIACS, Univ. of Maryland, 1988.

[7] R. W. Freund G. H. and N. M. Nachtigal, *Iterative solution of linear systems*, Acta Numerica 1991, pp. 57-100.

[8] W. Gropp and B. Smith, *Experiences with domain decomposition in three dimensions: Overlapping Schwarz methods* in Sixth International Symposium on Domain Decomposition Methods for Partial Differential Equations, A. Quarteroni, et al. eds., AMS, (1993). To appear.

[9] W. Gropp and B. Smith, *Simplified linear equation solvers: User's manual,* Mathematics and Computer Science Division, Argonne National Laboratory, 1993.

[10] J. W. H. Liu, *A graph partitioning algorithm by node separators*, ACM Transactions on Mathematical Software, 15, (1989), pp. 198-219.

[11] A. Pothen, H. D. Simon and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl. 11 (1990), pp. 430-452.

[12] Y. Saad, SPARSKIT: A basic tool kit for sparse matrix computations, TR 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.

[13] Y. Saad, *Highly parallel preconditioners for general sparse matrices*, Preprint 92-087, Army High Performance Computing Research Center, University of Minnesota, 1992.

[14] Y. Saad, *Krylov subspace methods in distributed computing environments*, Preprint 92-126, Army High Performance Computing Research Center, University of Minnesota, 1992.

[15] Y. Saad and M. H. Schultz, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems,* SIAM J. Sci. Stat. Comp. 7 (1986), pp. 865-869.

[16] H. A. Van der Vorst, *Bi-CGSTAB: A more smoothly converging variant of CG-S for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp. 13 (1992), pp. 631-644.