

A New Parallel Domain Decomposition Preconditioner II: Generalization to a Mesh-Free Parallel Solver

Randolph E. Bank*and Peter K. Jimack†

Abstract: This paper continues and further develops some of the ideas previously introduced in [5]. In particular, it is shown that the main parallel solution technique developed in [5] may be generalized to allow the parallel solution of an arbitrary sparse matrix. This generalization requires the matrix to be partitioned into p blocks and then coarsened (preferably in parallel) so that each of p different processors stores an entire submatrix plus a coarsening of the rest of the matrix. The linear problems with these new matrices may then be solved concurrently in order to obtain approximations to the solution of the full problem which may then be combined together in an appropriate way to define a general parallel preconditioner. As well as providing an overview of this new algorithm the paper also addresses the issues associated with partitioning the sparse matrix and coarsening certain blocks of its rows and columns. The paper concludes with the presentation and discussion of some preliminary numerical results.

1 Introduction

In [5] a new parallel domain decomposition preconditioner is introduced for the solution of the sparse linear systems that arise from the parallel adaptive finite element discretization of a class of self-adjoint linear elliptic partial differential equations (with extensions to nonlinear and non-self-adjoint problems also discussed (see, in addition, [6])). In this paper we consider further extensions of this work for the solution of an arbitrary sparse positive-definite (PD) linear system of the form

$$K\underline{u} = \underline{b}. \tag{1}$$

No assumptions will be made about the manner in which the matrix K has been obtained (although for the numerical tests used in Section 5 only matrices which correspond to finite element discretizations of second order elliptic operators are considered), and so we will rely only on its sparsity for the development of our preconditioning algorithm. In this sense the way in which the work presented here relates to that in the previous paper, [5], may be considered to be analogous to the relationship between algebraic multigrid methods and conventional hierarchical mesh multigrid algorithms (see for example [2, 3, 8, 10, 11, 13, 24]). Indeed, many

of the graph coarsening issues which must be addressed in algebraic multigrid codes also arise in this work (and are considered in Section 4 below).

The rest of this paper is organized as follows. Section 2 provides a detailed description of the development of the new preconditioner as a natural extension of that appearing in [5]. This is then followed by brief discussions in Sections 3 and 4 of simple graph partitioning and graph coarsening algorithms respectively: both of these being required by the proposed preconditioner. The paper is then concluded in Section 5 with a presentation of some provisional results and a discussion of this work.

2 Overview of Algorithm

In [5] it is demonstrated that a preconditioner, M say, for a sparse positive-definite linear system of the form (1) may be developed in the following algebraic manner. For $i = 1, \dots, p$ ($p \geq 2$) define matrices $Q_i \in \mathfrak{R}^{m_i \times n}$ and $R_i^T \in \mathfrak{R}^{n \times m_i}$, where $m_i < n$ (in fact m_i should be only a little greater than n/p). Then define K_i ($i = 1, \dots, p$) by

$$K_i = Q_i K Q_i^T \quad (2)$$

and the preconditioner, M , by

$$M^{-1} = \sum_{i=1}^p R_i^T K_i^{-1} Q_i . \quad (3)$$

The precise definitions of Q_i and R_i^T in [5] are based upon the existence of an underlying partial differential equation (PDE) with different finite element meshes covering the problem domain, Ω , on each processor, $i = 1, \dots, p$. There appears to be no reason however why a preconditioner of the same form as (2) and (3) cannot be developed based upon purely algebraic rules, for which no assumptions concerning underlying differential equations or finite element meshes are required. Such a development is undertaken in this section.

In order to generalize the preconditioner introduced in [5] we first consider the special cases where $p = 2$ and the underlying PDE is constrained by Dirichlet conditions on the whole of $\partial\Omega$ (both for simplicity). This allows the system (1) to be written in block-matrix form as

$$\begin{bmatrix} A_1 & B_1 & 0 \\ B_1^T & A_s & B_2^T \\ 0 & B_2 & A_2 \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_s \\ \underline{u}_2 \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_s \\ \underline{b}_2 \end{bmatrix} , \quad (4)$$

where \underline{u}_1 and \underline{u}_2 correspond to the finite element unknowns inside subdomains Ω_1 and Ω_2 respectively (which are non-overlapping), and \underline{u}_s corresponds to the finite element unknowns shared between $\overline{\Omega}_1$ and $\overline{\Omega}_2$ at the subdomain boundaries. The zero blocks are present due to the local nature of the finite element basis functions. Furthermore, recall from [5] that the global fine mesh, containing n vertices with unknown solution values, is defined to be the union of the locally refined meshes on each processor. Let processor i ($i = 1, 2$) have a mesh with ρ_i points in the interior of Ω_i , σ_i points in $\partial\Omega_i - \partial\Omega$ and τ_i points in $\Omega - \overline{\Omega}_i$, then

$\underline{u}_1 \in \mathfrak{R}^{\rho_1}$, $\underline{u}_2 \in \mathfrak{R}^{\rho_2}$, $\underline{u}_s \in \mathfrak{R}^\sigma$ (where $\sigma = \sigma_1 = \sigma_2$) and $n = \rho_1 + \rho_2 + \sigma$. Hence, $A_i \in \mathfrak{R}^{\rho_i \times \rho_i}$, $A_s \in \mathfrak{R}^{\sigma \times \sigma}$ and $B_i \in \mathfrak{R}^{\rho_i \times \sigma}$ for $i = 1, 2$.

In order to define the restriction matrices Q_i used in (2) and (3), the algorithm of [5] requires these to be mappings from the global fine mesh to the mesh which exists on processor i . Hence $Q_i \in \mathfrak{R}^{m_i \times n}$ where $m_i = \rho_i + \sigma_i + \tau_i$ for $i = 1, 2$. Note also that these mappings are such that the blocks A_i , A_s , B_i and B_i^T in (4) are unaltered. (The block A_s is unaltered by each mapping Q_i due to the fact that in [5] the finite element meshes are such that there is always a full layer of refinement immediately outside Ω_i on each processor, so both processors calculate identical (and correct) entries for A_s on their own meshes.) This means that we may express the matrices Q_1 and Q_2 in block form as

$$Q_1 = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & L_1 \end{bmatrix} \quad \text{and} \quad Q_2 = \begin{bmatrix} L_2 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}, \quad (5)$$

where $L_1 \in \mathfrak{R}^{\tau_1 \times \rho_2}$ and $L_2 \in \mathfrak{R}^{\tau_2 \times \rho_1}$. In general we expect that $\tau_1 \ll \rho_2$ and $\tau_2 \ll \rho_1$.

Combining equations (5) with equations (1), (2) and (4) we see that

$$K_1 = \begin{bmatrix} A_1 & B_1 & 0 \\ B_1^T & A_s & B_2^T L_1^T \\ 0 & L_1 B_2 & L_1 A_2 L_1^T \end{bmatrix} \quad \text{and} \quad K_2 = \begin{bmatrix} L_2 A_1 L_2^T & L_2 B_1 & 0 \\ B_1^T L_2^T & A_s & B_2^T \\ 0 & B_2 & A_2 \end{bmatrix}. \quad (6)$$

Hence, when the preconditioner (3) is used to solve the $n \times n$ system

$$M \underline{s} = \underline{r}, \quad (7)$$

the same partitioning of the vectors \underline{s} and \underline{r} as for \underline{u} and \underline{b} in equation (4) gives

$$\begin{bmatrix} \underline{s}_1 \\ \underline{s}_s \\ \underline{s}_2 \end{bmatrix} = \sum_{i=1}^2 R_i^T K_i^{-1} Q_i \begin{bmatrix} \underline{r}_1 \\ \underline{r}_s \\ \underline{r}_2 \end{bmatrix}. \quad (8)$$

Therefore, by (5) and (6), we get

$$\begin{aligned} \begin{bmatrix} \underline{s}_1 \\ \underline{s}_s \\ \underline{s}_2 \end{bmatrix} &= R_1^T \begin{bmatrix} A_1 & B_1 & 0 \\ B_1^T & A_s & B_2^T L_1^T \\ 0 & L_1 B_2 & L_1 A_2 L_1^T \end{bmatrix}^{-1} \begin{bmatrix} \underline{r}_1 \\ \underline{r}_s \\ L_1 \underline{r}_2 \end{bmatrix} \\ &+ R_2^T \begin{bmatrix} L_2 A_1 L_2^T & L_2 B_1 & 0 \\ B_1^T L_2^T & A_s & B_2^T \\ 0 & B_2 & A_2 \end{bmatrix}^{-1} \begin{bmatrix} L_2 \underline{r}_1 \\ \underline{r}_s \\ \underline{r}_2 \end{bmatrix}. \end{aligned} \quad (9)$$

Finally, we observe from [5] that the definition of the prolongation matrices $R_i^T \in \mathfrak{R}^{n \times m_i}$ ($i = 1, \dots, p$) for the case where $p = 2$ reduces to

$$R_1^T = \begin{bmatrix} I & 0 & 0 \\ 0 & \frac{1}{2}I & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad R_2^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2}I & 0 \\ 0 & 0 & I \end{bmatrix}. \quad (10)$$

In view of equations (7), (9) and (10) we may conclude that in order to generalize the preconditioner introduced in [5] so that it may be applied to arbitrary sparse matrices, when $p = 2$ at least, two additional components are required.

1. A means of ordering the unknowns in problem (1) so that it may be written in the block-matrix form of equation (4).
2. A means of obtaining the coarsening matrices L_i that appear in (9).

It may be observed that the partitioning problem described by the first of these requirements may be expressed in terms of the graph, \mathcal{G}_n , of the symmetric sparse matrix K . This graph has n vertices and an edge joining vertices i and j whenever the i - j^{th} entry in K is non-zero. It is therefore necessary to find a small subgraph of \mathcal{G}_n , (of size σ) which forms a separator for two other, larger, subgraphs of approximately equal sizes, ρ_1 and ρ_2 . If the unknowns in the system (1) are ordered corresponding to each of the vertices of the first of the large subgraphs, followed by each of the vertices of the small subgraph, followed by each of the vertices of the other large subgraph, then the required structure of (4) will be obtained. Fig. 1 illustrates a suitable choice of separator for a simple example graph.

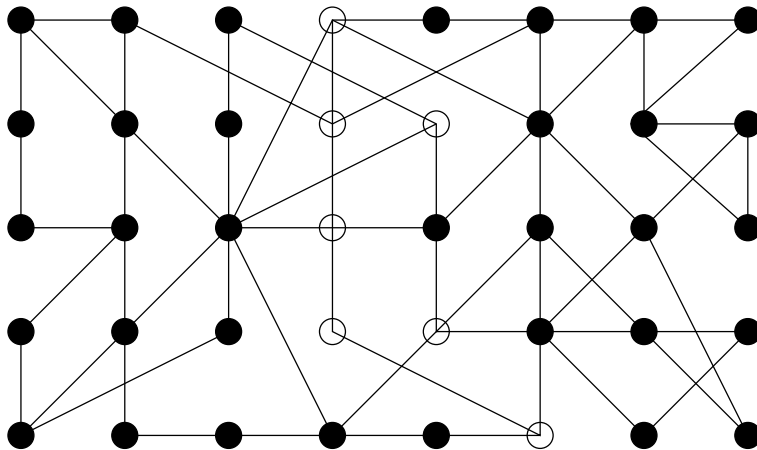


Figure 1: A possible separator set (white vertices) of the type required to give the system (1) the structure shown in (4) for $p = 2$, where the graph represents the sparsity pattern of a symmetric matrix.

In fact, it may be argued that the above graph partitioning problem is rather more complicated than is strictly necessary for our purposes. To illustrate this we now develop a modification of the proposed preconditioner which will allow more standard graph partitioning heuristics to be used (see for example [14, 15, 16, 19, 21, 23]). For the purposes of this description we maintain the simplifying assumption that $p = 2$ for the time-being.

Suppose that we are able to obtain a partition of the graph \mathcal{G}_n into two approximately equally-sized subgraphs which have a small number of edges connecting them. Then, by ordering the unknowns in the system (1) such that those corresponding to vertices in the same subgraph are consecutive, this system may be written in the following form:

$$\begin{bmatrix} A_1 & B^T \\ B & A_2 \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \end{bmatrix}. \quad (11)$$

Here $A_i \in \mathfrak{R}^{\rho_i \times \rho_i}$, $\underline{u}_i \in \mathfrak{R}^{\rho_i}$, $\underline{b}_i \in \mathfrak{R}^{\rho_i}$ and $B \in \mathfrak{R}^{\rho_2 \times \rho_1}$, where ρ_1 and ρ_2 are now defined to be the sizes of the two subgraphs (hence $\rho_1 + \rho_2 = n$) and the matrix blocks A_1 , A_2 and B are all sparse. We may now define $Q_i \in \mathfrak{R}^{m_i \times n}$ (where $m_i = \rho_i + \tau_i$ say) by

$$Q_1 = \begin{bmatrix} I & 0 \\ 0 & L_1 \end{bmatrix} \quad \text{and} \quad Q_2 = \begin{bmatrix} L_2 & 0 \\ 0 & I \end{bmatrix}, \quad (12)$$

for appropriate choices of coarsening matrices $L_1 \in \mathfrak{R}^{\tau_1 \times \rho_2}$ and $L_2 \in \mathfrak{R}^{\tau_2 \times \rho_1}$. Following (2) and (3) this then allows us to define a new preconditioner for K of the form

$$\begin{aligned} M^{-1} &= \sum_{i=1}^2 R_i^T (Q_i K Q_i^T)^{-1} Q_i \\ &= R_1^T \begin{bmatrix} A_1 & B^T L_1^T \\ L_1 B & L_1 A_2 L_1^T \end{bmatrix}^{-1} \begin{bmatrix} I & 0 \\ 0 & L_1 \end{bmatrix} \\ &\quad + R_2^T \begin{bmatrix} L_2 A_1 L_2^T & B^T L_2^T \\ L_2 B & A_2 \end{bmatrix}^{-1} \begin{bmatrix} L_2 & 0 \\ 0 & I \end{bmatrix}. \end{aligned} \quad (13)$$

Hence, when this preconditioner is used to solve an $n \times n$ system of the form (7), one gets

$$\begin{bmatrix} \underline{s}_1 \\ \underline{s}_2 \end{bmatrix} = R_1^T \begin{bmatrix} A_1 & B^T L_1^T \\ L_1 B & L_1 A_2 L_1^T \end{bmatrix}^{-1} \begin{bmatrix} \underline{r}_1 \\ L_1 \underline{r}_2 \end{bmatrix} + R_2^T \begin{bmatrix} L_2 A_1 L_2^T & B^T L_2^T \\ L_2 B & A_2 \end{bmatrix}^{-1} \begin{bmatrix} L_2 \underline{r}_1 \\ \underline{r}_2 \end{bmatrix} \quad (14)$$

(where \underline{s} and \underline{r} in (7) have been partitioned in the same manner as \underline{u} and \underline{b} in (11)). Again following the philosophy behind the non-symmetric preconditioner in [5] we now define

$$R_1^T = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad R_2^T = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix}, \quad (15)$$

in order to obtain a parallel preconditioner when $p = 2$.

The generalization of the preconditioner (13) for an arbitrary number of processors, p , is quite straightforward. First it is necessary to partition the graph, \mathcal{G}_n of the $n \times n$ sparse matrix K into p approximately equally-sized subgraphs (each with ρ_i vertices, for $i = 1, \dots, p$, say). This partition should be such that the number of edges which connect subgraph i to any of the other subgraphs (for $i = 1, \dots, p$) is as small as possible — so that as many of the non-zero entries of K as possible will be in the diagonal blocks when the unknowns are ordered according to this partition. On processor i (for any $i \in \{1, \dots, p\}$) it is then possible to express the system (1) in the block-matrix form

$$\begin{bmatrix} A_i & B_i^T \\ B_i & \bar{A}_i \end{bmatrix} \begin{bmatrix} \underline{u}_i \\ \underline{\bar{u}}_i \end{bmatrix} = \begin{bmatrix} \underline{b}_i \\ \underline{\bar{b}}_i \end{bmatrix}, \quad (16)$$

where $A_i \in \mathfrak{R}^{\rho_i \times \rho_i}$ (and corresponds to the i^{th} subgraph of \mathcal{G}_n), $\bar{A}_i \in \mathfrak{R}^{(n-\rho_i) \times (n-\rho_i)}$ and $B_i \in \mathfrak{R}^{(n-\rho_i) \times \rho_i}$. Now define $Q_i \in \mathfrak{R}^{m_i \times n}$ (where $m_i = \rho_i + \tau_i$ say) by

$$Q_i = \begin{bmatrix} I & 0 \\ 0 & L_i \end{bmatrix}, \quad (17)$$

for an appropriate choice of coarsening matrix $L_i \in \mathfrak{R}^{\tau_i \times (n - \rho_i)}$ (where, typically, we expect $\tau_i \ll n - \rho_i$). Hence, the contribution to the preconditioner M from processor i , when solving the $n \times n$ system (7), is given by

$$\begin{bmatrix} \underline{s}_i \\ \underline{\bar{s}}_i \end{bmatrix} = R_i^T \begin{bmatrix} A_i & B_i^T L_i^T \\ L_i B_i & L_i \bar{A}_i L_i^T \end{bmatrix}^{-1} \begin{bmatrix} \underline{r}_i \\ L_i \bar{\underline{r}}_i \end{bmatrix}, \quad (18)$$

which requires the solution of an $m_i \times m_i$ linear system on processor i . If we now choose

$$R_i^T = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \quad (19)$$

then the values of $\underline{s}_i \in \mathfrak{R}^{\rho_i}$ in (18) for $i = 1, \dots, p$ (which may be computed concurrently) define the vector \underline{s} , the solution of the preconditioning system (7) (note that $n = \sum_{i=1}^p \rho_i$).

To summarize the above discussion therefore, we have argued that the action of a purely algebraic preconditioner for an arbitrary sparse PD system (1) may be defined by (18) and (19) for $i = 1, \dots, p$, provided the following two requirements can be fulfilled.

1. We have a partition of the graph, \mathcal{G}_n , of the matrix K into p subgraphs of approximately equal size (ρ_i for $i = 1, \dots, p$) with a small number of edges of \mathcal{G}_n connecting vertices belonging to different subgraphs.
2. We have p matrices $L_i \in \mathfrak{R}^{\tau_i \times (n - \rho_i)}$ for some $\tau_i \ll n - \rho_i$ ($i = 1, \dots, p$).

Furthermore, it is clear that each of the vectors \underline{s}_i which make up the solution, \underline{s} , of the preconditioning problem (7) may be computed independently (and therefore concurrently) on processors $i = 1, \dots, p$ once the above partition and coarsening matrices have been defined.

It is important to emphasize that the above argument only provides a mechanism for obtaining a parallel algebraic preconditioner. It tells us nothing about the quality of such a preconditioner, which will depend upon the particular partition of \mathcal{G}_n that is chosen and the specific choices of L_i that are made. For example, when each $\tau_i = 0$, M reduces to a simple block-diagonal preconditioner on the particular partition that is being used. This is unlikely to be very effective in general, unless each of the blocks B_i in (16) are much sparser than (or the non-zero entries are much smaller in magnitude than for) the other blocks A_i and \bar{A}_i (in which case the original system, (1), is almost block diagonal).

When the system (1) is not close to being block diagonal then the choice of coarsening matrices, L_i , will be very important. In order to get a good preconditioner these matrices should be chosen so that, where possible,

$$L_i^T (L_i \bar{A}_i L_i^T)^{-1} L_i \approx \bar{A}_i^{-1}, \quad (20)$$

for $i = 1, \dots, p$. To see why this is the case observe that, from (16), we have

$$(A_i - B_i^T \bar{A}_i^{-1} B_i) \underline{u}_i = \underline{b}_i - B_i^T \bar{A}_i^{-1} \bar{\underline{b}}. \quad (21)$$

Also note that the corresponding part of the solution of the preconditioning system

$$\begin{bmatrix} A_i & B_i^T L_i^T \\ L_i B_i & L_i \bar{A}_i L_i^T \end{bmatrix} \begin{bmatrix} \underline{u}_i \\ \tilde{\underline{u}}_i \end{bmatrix} = \begin{bmatrix} \underline{b}_i \\ L_i \bar{\underline{b}}_i \end{bmatrix} \quad (22)$$

solved on processor i , satisfies

$$\left(A_i - B_i^T \left[L_i^T (L_i \bar{A}_i L_i^T)^{-1} L_i \right] B_i \right) \underline{u}_i = \underline{b}_i - B_i^T \left[L_i^T (L_i \bar{A}_i L_i^T)^{-1} L_i \right] \bar{\underline{b}}_i. \quad (23)$$

Hence, if $L_i^T (L_i \bar{A}_i L_i^T)^{-1} L_i = \bar{A}_i^{-1}$ for $i = 1, \dots, p$, the preconditioner will correspond to an exact solve. If (20) holds therefore, the preconditioner should prove to be very effective.

Further details concerning the choice of the matrices L_i are given in Section 4, which also provides a brief discussion of the general graph coarsening problem. Before this however we consider, in the next section, the first of the two requirements enumerated above: which concerns obtaining a good p -way partition of the graph \mathcal{G}_n of the symmetric sparse matrix K .

3 Partitioning the Graph of the Sparse Matrix

As outlined in the previous section, an important component of the parallel algebraic preconditioner that we propose involves obtaining a partition of the graph, \mathcal{G}_n , of the sparse PD matrix K into p subgraphs of approximately equal size. Note that this is a different partitioning problem to that originally described in Section 2 (and illustrated in Fig. 1) since it is no longer necessary to obtain a separator set of vertices when partitioning \mathcal{G}_n .

Fig. 2 illustrates how the same example graph used in Fig. 1 might now be partitioned when $p = 2$. Note that an important feature of this partition is that the number of edges which connect subgraph i with the rest of \mathcal{G}_n is small (for $i = 1, \dots, p$). This ensures that the number of non-zero entries in the coupling matrices, B_i , in (16) is approximately minimized. Comparison of equations (21) and (23) also shows that making the norms of these coupling matrices as small as possible is likely to be beneficial for the performance of the preconditioner. This suggests that it is appropriate to assign positive weights to the edges of \mathcal{G}_n , equal to the magnitude of the corresponding entries in the symmetric matrix K , and then to attempt to minimize the total weight of the edges which connect subgraph i with the rest of \mathcal{G}_n (again, subject to the constraint that $\rho_i \approx n/p$ for $i = 1, \dots, p$). We will refer to this total weight as the *cut-weight* of subgraph i .

The problem of finding a p -way partition of a weighted graph for which the size of each subgraph is approximately equal, and the total cut-weight is minimized, is well-known. Although this problem belongs to the class NP there are many heuristics which are known to provide reasonably good solutions for a wide variety of graphs, \mathcal{G}_n , in polynomial time. It is beyond the scope of this short paper to discuss these numerous heuristics in any detail, however we do note that in many cases there are efficient software implementations which are generally available (see [14, 16, 19, 23] for example). In some cases there are also

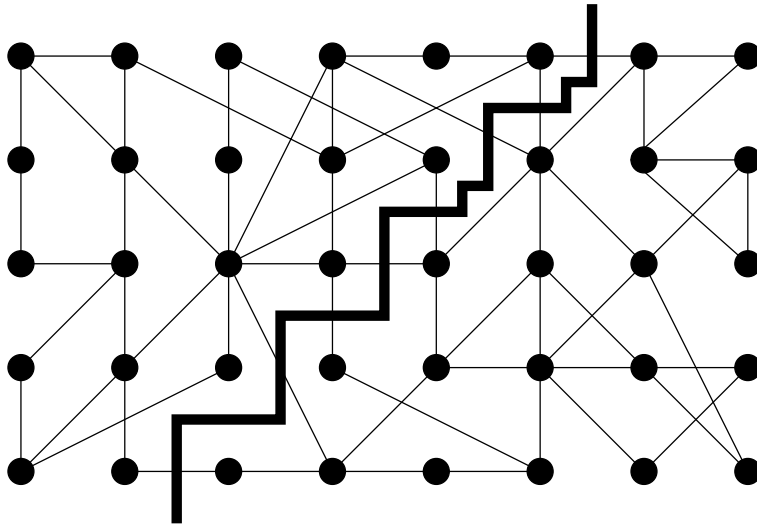


Figure 2: A possible partition of the graph \mathcal{G}_n of a sparse matrix which will allow a system such as (1) to be written with the structure shown in (16) for $i = 1, \dots, p$ (here $p = 2$).

parallel implementations (e.g. [17, 22]) however, for this application, the graph partitioning may be viewed as a pre-processing step which can be undertaken sequentially if necessary.

For the preliminary results presented in Section 5 a simple geometric partitioning rule (similar to recursive coordinate bisection [21]) has been used. This is possible because each of the examples in this paper are derived from finite element discretizations of second order elliptic PDEs. More general heuristics, such as in [14, 16, 19, 23], could also be used and are likely to perform just as well, if not better.

4 Graph Coarsening Algorithms

Having briefly discussed the graph partitioning aspect of the parallel algebraic preconditioner suggested in Section 2, we now consider the other requirement enumerated in that section. It is also necessary to produce matrices $L_i \in \mathfrak{R}^{\tau_i \times (n - \rho_i)}$, where $\tau_i \ll n - \rho_i$, for $i = 1, \dots, p$, such that (20) holds. Throughout Section 2 the L_i 's are referred to as “coarsening matrices” since their role is to allow the $(n - \rho_i) \times (n - \rho_i)$ matrices \bar{A}_i to be approximated by the (much smaller) $\tau_i \times \tau_i$ matrices $L_i \bar{A}_i L_i^T$.

For the particular preconditioner used in [5] (Version 3) L_i represents a restriction operator from a finite element space on a fine mesh to a smaller finite element space on a coarser mesh. Conversely, L_i^T represents an interpolation operator from the coarser mesh to the fine mesh. Since the meshes used in [5] are generated by regular hierarchical refinement (as in [4, 18] for example), these restrictions and interpolations are calculated, in practice, without the explicit construction of the L_i matrices: it is sufficient to make use of the parent-child

relations between nodes in the mesh hierarchy. When computing restrictions, for example, this is achieved by starting with the nodes at the finest level of the final fine mesh, which are not present in the computational mesh on processor i , and adding half of their nodal value to each of their parents — which are the two nodes at the ends of the edge which was bisected when creating the node at the finer level. This process is repeated at coarser and coarser levels until the only nodes remaining are the ones in the computational mesh on processor i . The interpolation operation is essentially the opposite of this: starting at the coarsest mesh each new node introduced at the next level up is given a value which is the average of its two parents' values.

Unlike in [5] however, in this work we do not wish to assume either that the matrix K in (1) (which is also written in block form on processor i as (16)) is derived from a discretization of a particular PDE, or that there is a hierarchy of finite element spaces that can be used to coarsen the blocks of \bar{A}_i in (16). In this sense, the problem that we face, of constructing the matrices $L_i \in \mathfrak{R}^{\tau_i \times (n - \rho_i)}$, is quite similar to that encountered in algebraic multigrid (AMG) algorithms when constructing suitable restriction and prolongation matrices. We would like to simulate the coarsening process used in [5] but only making use of the entries of the matrix blocks \bar{A}_i (just as AMG algorithms seek to simulate the restriction and prolongation behaviour of regular multigrid algorithms by only using the sparse matrix entries). It is instructional therefore to briefly consider some of the ideas that have been successfully used in algebraic multigrid algorithms in recent years.

Typically, early AMG algorithms, such as in [11] for example, still make assumptions concerning the relationship between the sparse matrix and an underlying PDE. In particular, in [11] it is assumed that the matrix problem arises from the discretization of a particular class of PDEs on a structured rectangular mesh. By developing coarsening algorithms for this structured mesh a grid hierarchy may be defined and then standard multigrid techniques applied. In [9] this mesh coarsening idea is generalized significantly to allow sequences of coarsened meshes to be conveniently defined from an initial unstructured mesh in 2-d. In the context of this work, once such a sequence is obtained on each processor, $i = 1, \dots, p$, it is possible to apply essentially the same technique for defining the matrices L_i as is used in [5] (and outlined above).

Although the mesh coarsening approach doesn't rely on the existence of a mesh hierarchy *a priori* it does require that the matrix system results from a (known) discretization of some differential equation. More recent AMG algorithms, such as in [10] for example, attempt to drop such assumptions completely. In [10], Braess works with the graph of the sparse symmetric matrix for the linear system to be solved. He uses the same definition for this graph as given above but prefers to use a relative, rather than an absolute, weight for each edge: the weight of the edge joining vertices i and j being given by $K_{ij}^2 / |K_{ii}| |K_{jj}|$. Now, instead of coarsening an underlying mesh, the algorithm in [10] coarsens this graph directly: taking into account both the graph's topology and the strengths of the couplings between the different rows and columns of the matrix (i.e. the relative edge weights defined above).

Another algebraic multilevel approach is described in [7]. This too makes use of the graph of the sparse matrix in order to develop an appropriate coarsening strategy. The implementation of this strategy takes the form of a reordering plus the assignment of zero or more parents to each vertex. One level of coarsening is accomplished by first computing a quality function for each vertex (the higher the value the more suitable it is for elimination from the graph at this level). The vertex with the highest value then has a set of (one or more) parents assigned to it and is eliminated. The parents are each assigned a new quality value of zero, and the process is repeated for the vertex which now has the highest quality. Termination occurs when all of the remaining vertices have a quality value of zero: these are the vertices that are carried forward to the next level. The process may be repeated for a number of levels, and the final ordering of the vertices is defined by the order in which they were eliminated. The process that we use for coarsening the graphs of the blocks \overline{A}_i is very similar to one level of this algorithm, however it is much simplified and also attempts to reflect the definitions of the L_i matrices in [5] as much as possible.

In [5] the mesh on processor i which covers the subregion $\Omega - \Omega_i$ is defined to be very fine in a small neighbourhood of $\partial\Omega_i$ and becomes gradually coarser as it gets further from $\partial\Omega_i$. We attempt to simulate this in our algebraic approach in the following manner. Let \mathcal{G}_i and $\overline{\mathcal{G}}_i$ be the graphs of the matrices A_i and \overline{A}_i on processor i respectively. By considering these as subgraphs of \mathcal{G}_n , it is possible to define the distance of each vertex within $\overline{\mathcal{G}}_i$ from \mathcal{G}_i to be the number of edges in the shortest path from that vertex to any vertex of \mathcal{G}_i . One may now define a coarsening of $\overline{\mathcal{G}}_i$ in which all of the vertices at a distance of 1 from \mathcal{G}_i are kept, half of the vertices at a distance 2 from \mathcal{G}_i are kept, a quarter of the vertices at distance 3 are kept, etc. In this coarsening we choose, for simplicity, that each eliminated vertex has just one parent, and that this is always at the same distance from \mathcal{G}_i as its child (or children). The action of the coarsening matrix, L_i , on a vector, $\overline{\mathbf{r}}_i$ say, is then defined by adding each entry of $\overline{\mathbf{r}}_i$ which corresponds to an eliminated vertex of $\overline{\mathcal{G}}_i$ to the entry that corresponds to its parent. Similarly, $L_i B_i$ ($B_i^T L_i^T$) may be obtained by adding the eliminated rows (columns) of B_i (B_i^T) to the row (column) of their parent, and the construction of $L_i \overline{A}_i L_i^T$ follows in the same manner.

In deciding which of the vertices of $\overline{\mathcal{G}}_i$ at a given distance from \mathcal{G}_i should be eliminated and which should be their parents it seems likely that it would be best to make use of both the connectivity of $\overline{\mathcal{G}}_i$ and the magnitudes of the non-zero entries of \overline{A}_i . In the computational experiments undertaken for the next section however we apply the following, simpler, algorithm which only uses the topology of $\overline{\mathcal{G}}_i$. Its advantages and disadvantages, along with some possible alternatives, are discussed in Section 5.

1. Order the vertices of $\overline{\mathcal{G}}_i$ ($j = 1, \dots, n - \rho_i$) by distance, $d(j)$ say, from \mathcal{G}_i .
2. Set $\nu(\delta) :=$ the number of vertices of $\overline{\mathcal{G}}_i$ at a distance $\leq \delta$ from \mathcal{G}_i .
3. Choose $d_{\max} > 1$.
4. For j from $\nu(d_{\max} + 1)$ to $n - \rho_i$:

- (a) Set $d(j) := d_{\max}$.
- 5. Set $\nu(d_{\max}) := n - \rho_i$.
- 6. For ℓ from 2 to d_{\max} :
 - (a) Set $f := 2^{(\ell-1)}$.
 - (b) For j from $\nu(\ell - 1) + 1$ to $\nu(\ell)$:
 - i. Set $k := j - \nu(\ell - 1)$.
 - ii. If $(k \bmod f = 1)$ then
 - Set $d(j) := 1$;
 - Update ordering of vertices by $d(j)$;
 - Set $\text{parent}(j) := 0$;
 - Set $\pi := j$
 - Else
 - Set $\text{parent}(j) := \pi$.

Note that in the above algorithm all of the vertices of $\overline{\mathcal{G}}_i$ that are a distance of greater than d_{\max} from \mathcal{G}_i are assigned a modified distance value of d_{\max} . This done to ensure that a minimum percentage of the nodes of $\overline{\mathcal{G}}_i$ that are far from \mathcal{G}_i are retained. Hence, increasing the value of d_{\max} has a similar effect to reducing the size of the underlying coarse mesh for the algorithm in [5]. For example, the choice of $d_{\max} = 8$ (that is used in the following section), ensures that approximately 1% of the vertices which are far from \mathcal{G}_i are retained.

5 Provisional Results and Discussion

In order to assess the quality and potential of the family of algebraic preconditioners that we have proposed, it is necessary to select a test set of sparse matrices. Ideally, these matrices should have a wide variety of sparsity patterns and spectral properties, and should come from a number of different sources. However, in this section we present some provisional numerical results using just two different classes of matrix: these are the matrices generated when piecewise linear finite elements are used to discretize the following two partial differential equations.

Problem 1

$$\begin{aligned} -\underline{\nabla} \cdot (\underline{\nabla} u) &= f \quad \forall \underline{x} \in \Omega \equiv (0, 1) \times (0, 1), \\ u &= g \quad \forall \underline{x} \in \partial\Omega. \end{aligned}$$

Problem 2

$$\begin{aligned} -\underline{\nabla} \cdot \left(\begin{pmatrix} 10^2 & 0 \\ 0 & 1 \end{pmatrix} \underline{\nabla} u \right) &= f \quad \forall \underline{x} \in \Omega \equiv (0, 1) \times (0, 1), \\ u &= g \quad \forall \underline{x} \in \partial\Omega. \end{aligned}$$

Note that these are the same test problems that are used in [5] (with the same choices of f and g), and we make use of the same sequences of meshes as in [5] also. For example, a mesh with 4096 elements yields a 1985×1985 sparse matrix, and when there are 262144 elements, $n = 130561$.

Table 1 shows the number of iterations required by our algorithm to reduce the 2-norm of the residual by a factor of 10^6 when solving these two sequences of problems. Since the original matrix, K , is always symmetric and positive-definite, the “no preconditioning” column shows the number of iterations required by a conjugate gradient solver for each problem. The iteration counts in the other columns are obtained using GMRES with the preconditioner outlined over the previous sections. (See, for example, [1, 12, 20] for a description of conjugate gradients, GMRES and similar iterative methods.)

n	Problem 1				
	No precon.	$p = 2$	$p = 4$	$p = 8$	$p = 16$
1985	73	12	12	16	17
8065	143	15	15	20	21
32513	282	18	18	24	26
130561	549	22	23	29	34
n	Problem 2				
	No precon.	$p = 2$	$p = 4$	$p = 8$	$p = 16$
1985	223	15	17	25	26
8065	435	19	21	31	34
32513	818	23	26	40	44
130561	1512	29	31	50	54

Table 1: Iteration counts, using the proposed algebraic preconditioner, for the sequences of matrices obtained from piecewise linear discretizations of Problems 1 and 2.

Note that without preconditioning the number of iterations doubles whenever n is quadrupled — implying that the condition numbers for these two sequences of matrices are approximately proportional to n (as expected since, here, $n = O(h^{-2})$). In contrast to this, the number of iterations required by the preconditioned GMRES algorithm appears to be increasing by an approximately constant amount each time n is quadrupled. This suggests that the condition number of the preconditioned problems may be increasing only logarithmically with n — a significant improvement.

Despite the encouragement of seeing only slow growth in the condition number for these two problems as n is increased, comparison with the corresponding results in [5] shows that this algebraic approach is still some way behind the, less general, mesh-based algorithm. Some significant improvements in performance may easily be obtained however. Table 2 shows equivalent results to those in Table 1 but using a slightly different (less aggressive) coarsening algorithm. Here, step 6(a) has been modified to

$$\text{Set } f := 2^{((\ell-1)/2)}$$

(where integer division is used in the exponent) and d_{\max} has been doubled (to 16). This has the effect of creating a wider transition layer between the immediate neighbourhood of \mathcal{G}_i , where all vertices of $\overline{\mathcal{G}}_i$ are retained, and the part of $\overline{\mathcal{G}}_i$ that is far from \mathcal{G}_i , where less than 1% of the vertices are retained. It is still the case that $\tau_i \ll n - \rho_i$ however.

n	Problem 1				
	No precon.	$p = 2$	$p = 4$	$p = 8$	$p = 16$
1985	73	9	10	13	14
8065	143	12	13	16	17
32513	282	14	16	20	23
130561	549	17	20	24	29
n	Problem 2				
	No precon.	$p = 2$	$p = 4$	$p = 8$	$p = 16$
1985	223	12	13	19	20
8065	435	15	17	24	27
32513	818	19	21	31	34
130561	1512	23	25	38	43

Table 2: Iteration counts, using the modified algebraic preconditioner, for the sequences of matrices obtained from piecewise linear discretizations of Problems 1 and 2.

Other straightforward improvements to the coarsening algorithm could also be made, and need to be investigated. In particular, the current version uses an essentially arbitrary choice of parent for each deleted vertex (based upon the numbering of the unknowns in (16)). A better approach would be to use the connectivity of $\overline{\mathcal{G}}_i$ when making this choice. Better still, the actual entries of the sub-matrices \overline{A}_i should be used in the coarsening algorithm, as they are in [7, 8, 10] for example.

Slightly more elaborate improvements to the coarsening algorithm should also be considered. For example, by keeping d_{\max} and f quite small not too many vertices will be eliminated from each $\overline{\mathcal{G}}_i$ — however the coarsening algorithm could then be applied on the remaining graph in a recursive manner. This would still yield final values of τ_i as small as those obtained at present, but with a more sophisticated multilevel coarsening. Further enhancements could be achieved by allowing eliminated vertices to have more than one parent and weighting each of these parents with a value in $[0, 1]$. These weights would appear in the matrices L_i , which would no longer have entries of just zero and one.

Finally, we remark on some aspects of the practical implementation of the algorithm outlined in the previous section. The first of these concerns the ordering of the vertices of $\overline{\mathcal{G}}_i$ by their distance from \mathcal{G}_i (step 1). This ordering may be achieved at the same time as the distance values, $d(j)$, are calculated by making a breadth-first traversal of $\overline{\mathcal{G}}_i$. Provided the matrices, K , are such that the maximum number of non-zero entries per row is independent of n this is an $O(n)$ algorithm. It may even be possible to improve the efficiency of this

step by only traversing $\overline{\mathcal{G}}_i$ to a depth of d_{\max} from \mathcal{G}_i , however we have yet to experiment with such an approach.

The most serious challenge that must be overcome in order to obtain a truly scalable parallel implementation of the overall algorithm considered in this paper concerns a different aspect of the coarsening process. So far it has been implicitly assumed that it will be possible for each processor, $i = 1, \dots, p$, to store the whole of the matrix K and then to coarsen the entire submatrix \overline{A}_i of dimension $n - \rho_i \approx (p-1)n/p$. Clearly, as the problem size grows, even if p grows at the same rate as n , the size of \overline{A}_i will still be $O(n)$ on each processor. This is extremely undesirable, both in terms of the memory requirement (for distributed rather than shared memory computing) and the computational overhead. A better approach, for parallel scalability at least, would be to emulate [5] once more by allowing each processor to coarsen its own subgraph, \mathcal{G}_i , in some way, and then to communicate the result of this coarsening to the other processors for use in the construction of their L_i matrices. This is an issue requiring further investigation.

6 Conclusions

In this paper we have introduced an algebraic preconditioner which generalizes the new domain decomposition approach described in [5]. The key requirements for this preconditioner to perform well are a good p -way partition of the graph, \mathcal{G}_n , of the sparse matrix, K , in (1), and the construction on each processor, $i = 1, \dots, p$, of coarsening matrices L_i which satisfy (20) (given the block-matrix form (16)). The partitioning problem is quite standard and numerous heuristics are known to perform well. The coarsening problem, whilst less standard, is also shown to occur in other applications, such as algebraic multigrid for example. (It is also interesting to note that many graph partitioning algorithms, such as [16, 23] for example, also make significant use of graph reduction (i.e. coarsening) techniques.) A very simple algorithm for graph coarsening, based upon the experience of [5], has been used with some success in the examples presented in Section 5. In these preliminary results, for problems in which the condition number is known to be proportional to n , it appears that the preconditioned systems have condition numbers which grow much more slowly.

Acknowledgements

The work of REB was supported by the National Science Foundation under contract DMS-9706090 and the work of PKJ, whilst visiting UCSD, was supported by a Research Grant from the Leverhulme Trust.

References

- [1] S.F. Ashby, T.A. Manteuffel and P.E. Taylor, “A *Taxonomy for Conjugate Gradient Methods*”, SIAM J. on Numerical Analysis, 27, 1542–1568, 1990.

- [2] O. Axelsson and P. Vassilevski, “*Algebraic Multilevel Preconditioning Methods, Part I*”, Numerische Mathematik, 56, 157–177, 1989.
- [3] O. Axelsson and P. Vassilevski, “*Algebraic Multilevel Preconditioning Methods, Part II*”, SIAM J. on Numerical Analysis, 27, 1569–1590, 1990.
- [4] R.E. Bank, “*PLTMG Users’ Guide 8.0*”, SIAM, Philadelphia, 1998.
- [5] R.E. Bank and P.K. Jimack, “*A New Parallel Domain Decomposition Preconditioner I: Application with an Adaptive Parallel Finite Element Solver*”, in this book.
- [6] R.E. Bank and P.K. Jimack, “*A New Parallel Domain Decomposition Method for the Adaptive Finite Element Solution of Elliptic Partial Differential Equations*”, in preparation, 1999.
- [7] R.E. Bank and R.K. Smith, “*The Incomplete Factorization Multigraph Algorithm*”, to appear in SIAM J. on Scientific Computing, 1999.
- [8] R.E. Bank and C. Wagner, “*Multilevel ILU Decomposition*”, to appear in Numerische Mathematik, 1999.
- [9] R.E. Bank and J. Xu, “*An Algorithm for Coarsening Unstructured Meshes*”, Numerische Mathematik, 73, 1–36, 1996.
- [10] D. Braess, “*Towards Algebraic Multigrid for Elliptic Problems of Second Order*”, Computing, 55, 379–393, 1995.
- [11] J.E. Dendy, “*Black Box Multigrid*”, J. Comput. Phys., 48, 366–386, 1982.
- [12] G.H. Golub and C.F. Van Loan, “*Matrix Computations*”, John Hopkins Press, 3rd edition, 1996.
- [13] W. Hackbusch, “*Iterative Solution of Large Sparse Systems*”, Springer, New York, 1994.
- [14] B. Hendrickson and R. Leland, “*A Multilevel Algorithm for Partitioning Graphs*”, Technical Report SAND 93-1301, Sandia National Laboratories, 1993.
- [15] D.C. Hodgson and P.K. Jimack, “*Efficient Mesh Partitioning for Parallel Elliptic Differential Equation Solvers*”, Computing Systems in Engineering, 6, 1–12, 1995.
- [16] G. Karypis and V. Kumar, “*A Coarse Grain Parallel Formulation of Multilevel k -way Graph Partitioning Algorithm*”, in Proc. of the Eighth SIAM Conf. on Parallel Processing for Scientific Computing (M. Heath *et al*, eds.), SIAM, Philadelphia, 1997.
- [17] G. Karypis, K. Schloegel and V. Kumar, “*ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 2.0*”, Department of Computer Science, University of Minnesota, 1998.

- [18] R. Lohner, “*An Adaptive Finite Element Scheme for Transient Problems in CFD*”, *Comp. Meth. in Appl. Mech. and Eng.*, 61, 323–338, 1987.
- [19] B. Monien and R. Diekmann, “*A Local Graph Partitioning Heuristic Meeting Bisection Bounds*”, in *Proc. of the Eighth SIAM Conf. on Parallel Processing for Scientific Computing* (M. Heath *et al*, eds.), SIAM, Philadelphia, 1997.
- [20] Y. Saad and M. Schultz, “*GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*”, *SIAM J. on Scientific Computing*, 7, 856–869, 1986.
- [21] H.D. Simon, “*Partitioning of Unstructured Problems for Parallel Processing*”, *Computing Systems in Engineering*, 2, 135–148, 1991.
- [22] C. Walshaw, M. Cross and M.G. Everett, “*Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes*”, *J. Par. Dist. Comput.*, 47, 102–108, 1997.
- [23] C. Walshaw, M. Cross, M.G. Everett, S. Johnson and K. McManus, “*Partitioning and Mapping of Unstructured Meshes to Parallel Machine Topologies*”, in *Irregular '95: Parallel Algorithms for Irregularly Structured Problems* (A. Ferreira and J. Rolim, eds.), Springer, 1995.
- [24] P.M. De Zeeuw, “*Matrix-Dependent Prolongations and Restrictions in a Black Box Multigrid Solver*”, *J. Compu. Appl. Math.*, 33, 1–27, 1990.