

Domain Decomposition Preconditioning for Parallel PDE Software

Peter K. Jimack
School of Computing
University of Leeds, LS2 9JT, UK.

Abstract

Domain decomposition methods have been applied to the solution of engineering problems for many years. Over the past two decades however the growth in the use of parallel computing platforms has ensured that interest in these methods, which offer the possibility of parallelism in a very natural manner, has become greater than ever. This interest has led to research that has yielded significant advances in both the theoretical understanding of the underlying mathematical structure behind domain decomposition methods and in the variety of domain decomposition algorithms that are available for use by the engineering community. In this paper we provide a brief overview of some of the main categories of domain decomposition algorithm and then focus on a particular variant of the overlapping Schwarz algorithm that is based upon the use of a hierarchy of finite element grids. Throughout the paper we consider domain decomposition methods as preconditioners for standard, Krylov subspace, iterative solvers however they may also be used directly as iterative methods in their own right. All of the theoretical results that are described apply equally in both cases.

Keywords: partial differential equations, domain decomposition, parallel computing

1 Introduction

The majority of this paper considers the parallel finite element (FE) solution of the following two-dimensional variational problem, which is derived from a second order self-adjoint partial differential equation (PDE). All of the results and algorithms described can be generalized to three-dimensional problems however, and in section 4 practical extensions to a class of non-self-adjoint problems are also considered.

Problem 1.1 Find $u \in H_E^1(\Omega)$ such that

$$\mathcal{A}(u, v) = \mathcal{F}(v), \quad \forall v \in H_0^1(\Omega), \quad (1)$$

where $\Omega \in \mathbb{R}^2$ is the problem domain,

$$H_E^1(\Omega) = \{u \in H^1(\Omega) : u|_{\partial\Omega_E} = u_E(\underline{x})\} \quad (2)$$

and

$$H_0^1(\Omega) = \{u \in H^1(\Omega) : u|_{\partial\Omega_E} = 0\}. \quad (3)$$

Here $\partial\Omega_E$ is the (non-empty) part of the boundary, $\partial\Omega$, upon which essential (Dirichlet) boundary conditions are imposed and $\mathcal{A}(\cdot, \cdot)$ and $\mathcal{F}(\cdot)$ are the bilinear and linear forms

$$\mathcal{A}(u, v) = \int_{\Omega} (A(\underline{x})\nabla u) \cdot \nabla v \, d\underline{x} \quad \text{and} \quad \mathcal{F}(v) = \int_{\Omega} f v \, d\underline{x} + \int_{\partial\Omega_N} g v \, ds, \quad (4)$$

where $A(\underline{x})$ is symmetric and strictly positive-definite, and $\partial\Omega_N = \partial\Omega - \partial\Omega_E$ is the part of the boundary subject to Neumann boundary conditions: $\underline{n} \cdot (A(\underline{x})\nabla u) = g(\underline{x})$.

In order to approximate this solution from a finite dimensional space of trial functions, $S^h(\Omega)$ say, it is necessary to solve the following discrete problem.

Problem 1.2 Find $u^h \in S^h(\Omega) \cap H_E^1(\Omega)$ such that

$$\mathcal{A}(u^h, v^h) = \mathcal{F}(v^h), \quad \forall v^h \in S^h(\Omega) \cap H_0^1(\Omega). \quad (5)$$

This problem may in turn be expressed as the matrix equation

$$K\underline{u} = \underline{b}, \quad (6)$$

where K is the stiffness matrix, \underline{b} is the load vector and \underline{u} is a vector of nodal displacements which is to be determined. Note that the matrix K is strictly positive-definite and, for the usual choices of FE trial space and basis (e.g. [33, 49]), sparse. Hence an iterative solution method for (6), such as the conjugate gradient (CG) method, [23], is most appropriate.

In fact, for the FE method it is well known that when $S^h(\Omega)$ is a space of piecewise polynomial functions defined on a mesh of elements covering Ω with edge size h , \mathcal{T}^h say, the condition number of K grows like $O(h^{-2})$ as $h \rightarrow 0$ (see [33] for example). For this reason it is necessary to apply a preconditioned version of the CG algorithm, or other iterative solver, for realistic mesh sizes h (again see [23]). The majority of this paper is concerned with this preconditioning step however for the rest of this section we consider the other major issues associated with solving (6) in parallel using an iterative method.

1.1 Mesh Partitioning and Parallel Finite Element Assembly

Let us assume that the problem domain Ω is a bounded polygonal region which is discretized into a non-overlapping set of triangles \mathcal{T} . Furthermore, consider the simplest FE trial space consisting of piecewise linear functions on \mathcal{T} . Hence u^h takes the form

$$u^h = \sum_{i=1}^n u_i N_i(\underline{x}) + \sum_{i=n+1}^{n+m} u_i N_i(\underline{x}), \quad (7)$$

where $N_i(\underline{x})$ are the usual piecewise linear basis functions on \mathcal{T} (which has n interior vertices and m on the Dirichlet boundary), and u_{n+1}, \dots, u_{n+m} are known to equal u_E from the Dirichlet boundary condition. The terms in the matrix equation (6) may therefore be written explicitly as:

$$K_{ji} = \int_{\Omega} A(\underline{x}) \underline{\nabla} N_j \cdot \underline{\nabla} N_i \, d\underline{x}, \quad (8)$$

$$b_j = \int_{\Omega} f N_j \, d\underline{x} + \int_{\partial\Omega_N} g N_j \, ds - \sum_{i=n+1}^{n+m} u_i \int_{\Omega} A(\underline{x}) \underline{\nabla} N_j \cdot \underline{\nabla} N_i \, d\underline{x}. \quad (9)$$

Before being able to solve this problem in parallel it is first necessary to form the system of equations (6) in parallel. This may be achieved by partitioning the triangulation \mathcal{T} so that each processor works only with a subset of \mathcal{T} . In particular it is necessary to produce a set of sub-triangulations, $\{\mathcal{T}_1, \dots, \mathcal{T}_p\}$ say, such that

$$\bigcup_{i=1}^p \mathcal{T}_i = \mathcal{T} \quad (10)$$

and

$$\mathcal{T}_i \cap \mathcal{T}_j = \phi \quad \text{when } i \neq j. \quad (11)$$

It is then possible to assemble the contributions to the matrix K and the vector \underline{b} in (6) independently (and concurrently) on p different processors, $i = 1, \dots, p$, with processor i working only on \mathcal{T}_i . There are many possible algorithms for producing a suitable partition of \mathcal{T} however a description of these is beyond the scope of this paper. The main requirements of the partition are that $|\mathcal{T}_i| \approx |\mathcal{T}|/p$ (i.e. each processor deals with an approximately equal number of elements¹) and the number of vertices lying on the partition boundary is as small as possible (this will minimize communication overheads). Further details of partitioning algorithms may be found in, for example, [16, 19, 22, 24, 25, 34, 44, 50, 52], and the papers cited therein.

Suppose that, once the triangulation \mathcal{T} has been partitioned, the unknowns, \underline{u} , are ordered in the following manner. Each of the interior vertices in \mathcal{T}_1 (\underline{u}_1 say), followed by each of the interior vertices in \mathcal{T}_2 (\underline{u}_2 say), etc., up to each of the interior vertices

¹We assume here that a homogeneous parallel computing system is being used for which each processor has the same performance characteristics.

in $\mathcal{T}_p(\underline{u}_p)$, followed by each of the vertices lying on the partition boundary (\underline{u}_s say). It then follows that the system (6) may be written in block matrix form as

$$\begin{bmatrix} A_1 & & & B_1 \\ & A_2 & & B_2 \\ & & \ddots & \vdots \\ & & & A_p & B_p \\ B_1^T & B_2^T & \cdots & B_p^T & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_p \\ \underline{b}_s \end{bmatrix}. \quad (12)$$

In (12) the block-arrowhead structure of the matrix K stems from the local support of the FE basis functions. Moreover, each of the blocks A_i , B_i (and therefore B_i^T) and \underline{b}_i may be computed entirely by processor i (which works only with sub-mesh \mathcal{T}_i), for $i = 1, \dots, p$. The blocks A_s and \underline{b}_s both have contributions from elements in all of the sub-meshes however it is possible for each processor to assemble the contributions to each of these only from those elements on its own sub-mesh. On processor i we will refer to these contributions as $A_{s(i)}$ and $\underline{b}_{s(i)}$ respectively.

Once each processor has (concurrently) assembled each of the blocks A_i , B_i , \underline{b}_i , $A_{s(i)}$ and $\underline{b}_{s(i)}$, the system (12) is stored in a distributed manner. It is then ready to be solved. As indicated above the type of solver that we consider here is one based upon Krylov subspace iterations, such as the CG method. Apart from the preconditioning step, the main two computational tasks that must be undertaken within each iteration of such an algorithm are the formation of at least one matrix-vector product and at least one inner product of two vectors. The manner in which these may be undertaken in parallel is now described.

1.2 Parallel Matrix-Vector Products and Inner Products

Consider the calculation of the following matrix-vector product: $\underline{q} = K\underline{r}$, where K is the stiffness matrix that appears in (6). Such a product must be calculated at least once per iteration when using a Krylov subspace solver such as CG. This product may be written in the block matrix notation of (12), to give

$$\begin{bmatrix} \underline{q}_1 \\ \underline{q}_2 \\ \vdots \\ \underline{q}_p \\ \underline{q}_s \end{bmatrix} = \begin{bmatrix} A_1 & & & B_1 \\ & A_2 & & B_2 \\ & & \ddots & \vdots \\ & & & A_p & B_p \\ B_1^T & B_2^T & \cdots & B_p^T & A_s \end{bmatrix} \begin{bmatrix} \underline{r}_1 \\ \underline{r}_2 \\ \vdots \\ \underline{r}_p \\ \underline{r}_s \end{bmatrix}. \quad (13)$$

From this it is clear that, for $i = 1, \dots, p$,

$$\underline{q}_i = A_i \underline{r}_i + B_i \underline{r}_s \quad (14)$$

and

$$\begin{aligned}\underline{q}_s &= \sum_{i=1}^p B_i^T \underline{r}_i + A_s \underline{r}_s \\ &= \sum_{i=1}^p (B_i^T \underline{r}_i + A_{s(i)} \underline{r}_s) .\end{aligned}\tag{15}$$

Given that A_i , B_i and $A_{s(i)}$ are all stored on processor i , equations (14) and (15) show that, provided \underline{r}_i is stored on processor i and \underline{r}_s is stored on all processors, the matrix-vector product may be computed with only a single communication (corresponding to the summation in (15)). In fact, if we store \underline{q}_s in a distributed pattern, where $\underline{q}_{s(i)} = B_i^T \underline{r}_i + A_{s(i)} \underline{r}_s$ is stored on processor i , then no inter-processor communication is required when forming a matrix-vector product.

In order to form the inner product of two vectors, \underline{r} and \underline{q} say, it is again helpful to use the block notation of (12). Thus

$$\tau = \underline{r} \cdot \underline{q} = \begin{bmatrix} \underline{r}_1 \\ \underline{r}_2 \\ \vdots \\ \underline{r}_p \\ \underline{r}_s \end{bmatrix} \cdot \begin{bmatrix} \underline{q}_1 \\ \underline{q}_2 \\ \vdots \\ \underline{q}_p \\ \underline{q}_s \end{bmatrix} .\tag{16}$$

If we again assume that \underline{r}_i and \underline{q}_i are stored on processor i , and if we further assume that \underline{r}_s is stored on all processors but that $\underline{q}_{s(i)}$ is stored on processor i , then we may write (16) as:

$$\tau = \sum_{i=1}^p (\underline{r}_i \cdot \underline{q}_i + \underline{r}_s \cdot \underline{q}_{s(i)}) .\tag{17}$$

It is clear from this that each inner product may be calculated with the need for just one inter-processor communication.

Full details of the parallel implementation of the multiplication operations considered in this subsection, along with further details on the parallel implementation of a CG solver and the FE assembly, may be found in [32]. This reference makes use of the library of parallel communication functions called MPI (Message Passing Interface), [38].

1.3 Schur Complement Methods

Before moving on to consider domain decomposition (DD) preconditioning for the iterative solution of (12) it is worth noting an alternative strategy for the solution of linear systems of this form. This strategy is based upon elimination of all of the interior unknowns $\underline{u}_1, \dots, \underline{u}_p$ in (12). This is achieved by noting that

$$A_i \underline{u}_i + B_i \underline{u}_s = \underline{b}_i \quad \text{for } i = 1, \dots, p\tag{18}$$

and

$$\sum_{i=1}^p B_i^T \underline{u}_i + A_s \underline{u}_s = \underline{b}_s . \quad (19)$$

Combining these two expressions gives

$$(A_s - \sum_{i=1}^p B_i^T A_i^{-1} B_i) \underline{u}_s = \underline{b}_s - \sum_{i=1}^p B_i^T A_i^{-1} \underline{b}_i \quad (20)$$

which may be written in the form

$$S \underline{u}_s = \underline{g} , \quad (21)$$

where

$$S = A_s - \sum_{i=1}^p B_i^T A_i^{-1} B_i \quad (22)$$

and the right-hand-side vector, \underline{g} , is given by

$$\underline{g} = \underline{b}_s - \sum_{i=1}^p B_i^T A_i^{-1} \underline{b}_i . \quad (23)$$

This system is known as the Schur complement, or Capacitance, system and the matrix S is known as the Schur complement, or Capacitance, matrix. Note that if one is able to obtain a solution to (21) for the interface unknowns \underline{u}_s , then (18) may be used to solve for the remaining unknowns \underline{u}_i for $i = 1, \dots, p$.

The solution of (21) is usually undertaken iteratively. This is because, although the matrix S is generally dense, it is computationally expensive to form S explicitly. If an iterative method, such as the CG algorithm is applied to solve (21) then all that is required is the product of S with a vector, $\underline{q}_s = S \underline{r}_s$ say. Hence the matrix S itself is not required explicitly. Furthermore, note that

$$S \underline{r}_s = \sum_{i=1}^p A_i \underline{r}_s - \sum_{i=1}^p B_i^T (A_i^{-1} (B_i \underline{r}_s)) . \quad (24)$$

Hence each product may be obtained using only local matrix-vector products and interior subdomain solves independently on each subdomain (followed by a single communication step). If an efficient sequential solver is available on each processor then this technique can be highly competitive. For very large problems however it is usually necessary to solve the subdomain problems iteratively and so the the Schur complement approach becomes less attractive. We do not consider it further in this paper but refer the reader to works such as [1, 2, 18, 21, 28, 35, 45, 46] for further details.

2 Preconditioning

There are many possible ways in which the system (6) can be preconditioned. Some of these are purely algebraic, such as incomplete Cholesky factorization [6, 37] or using sparse approximate inverses [7, 15], whilst others make use of the underlying FE derivation of the system, such as element-by-element preconditioning [27, 51] or domain decomposition preconditioners, which are the subject of this paper. The essential idea behind any preconditioning strategy for (6) is to find a positive-definite matrix, M say, that has two properties.

1. The matrix $M^{-1}K$ should have a small condition number.
2. The system $M\underline{s} = \underline{r}$ should be computationally cheap to solve.

(In fact the above properties refer to what is known as left preconditioning, where the system (6) is expressed as

$$(M^{-1}K)\underline{u} = M^{-1}\underline{b}. \quad (25)$$

This is the form of preconditioning that is considered in this paper, however all of the issues discussed apply equally to symmetric or right preconditioning.) When seeking to solve the system (6) on a parallel computer there is an additional requirement.

3. The system $M\underline{s} = \underline{r}$ should be easy to solve in parallel.

These properties are required because the rate of convergence of the preconditioned conjugate gradient (PCG) algorithm is dependent upon the condition number of the preconditioned matrix $M^{-1}K$, and the major computational step in this algorithm at each iteration is the solution of a system of the form $M\underline{s} = \underline{r}$ ([23]). As we will see, domain decomposition preconditioning can satisfy all three of these requirements.

In this section we provide an overview of the two main classes of DD preconditioner: iterative substructuring methods and Schwarz methods. There are other variants in the literature too, perhaps most notably the FETI approach of Farhat et al, [21], which is a variant of the substructuring approach based upon the use of Lagrange multipliers. Details of such variants however are beyond the scope of this paper.

2.1 Iterative Substructuring

These techniques are based upon a partition of the FE triangulation \mathcal{T} into a set of non-overlapping sub-triangulations $\{\mathcal{T}_1, \dots, \mathcal{T}_p\}$, such that not only do (10) and (11) both hold, but also the interface between subdomains forms a valid (coarse) triangulation of the domain Ω . Figure 1 illustrates an allowable decomposition of an example FE triangulation \mathcal{T} .

Given an allowable decomposition of the mesh it is possible to write the algebraic system (6) in the block form (12). Now suppose that the nodes on the interface, whose solution values are identified by the vector \underline{u}_s , are broken down into two types: those

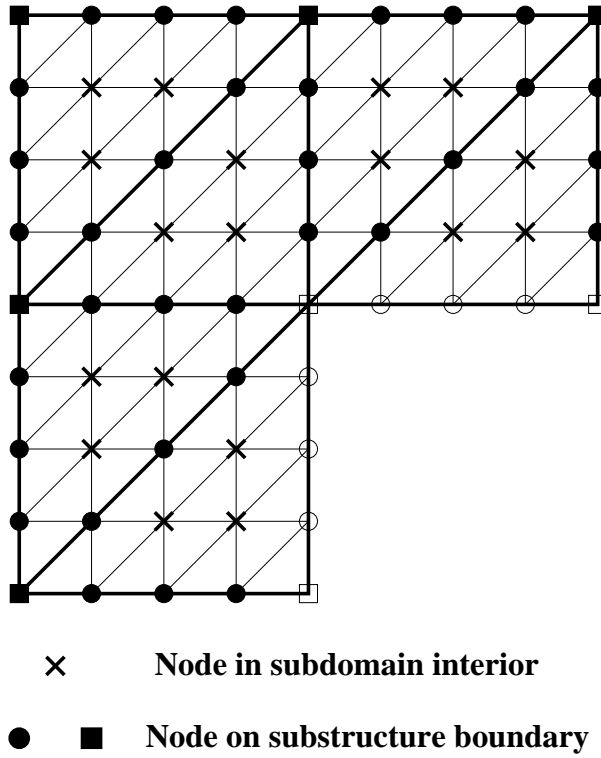


Figure 2: The different types of unknown in a typical mesh.

of the substructure edges. That is,

$$\text{block}(A_{EE}) = \begin{bmatrix} A_{EE(1)} & & & \\ & A_{EE(2)} & & \\ & & \ddots & \\ & & & A_{EE(N)} \end{bmatrix}, \quad (28)$$

where N is the number of edges in the coarse substructure not lying on the Dirichlet part of the boundary. In [9] Bramble, Pasciak and Schatz analyze their own preconditioner which is based upon this idea of separating the internal, the substructure edge and the substructure vertex unknowns. This is the first of a series of papers analyzing the construction of preconditioners by substructuring [9, 10, 11, 12], and in it they prove that the condition number of their preconditioned problem grows in proportion to $(1 + \log(H/h))^2$, where H represents the size of elements in the coarse mesh (substructure).

To see how such a preconditioner may be constructed recall that the preconditioning step in the PCG algorithm ([23]) requires the solution of the system $M\underline{s} = \underline{r}$ at each

which overlap are coloured differently. This then allows solves to be performed in all of the subdomains of the same colour in parallel (see [26, 39, 47] for details). This is known as multiplicative additive Schwarz preconditioning.

An alternative way of introducing parallelism into the above preconditioner is to decouple the subproblems so that all of the subdomain solves may be performed simultaneously. This may be viewed as an overlapping block Jacobi preconditioner as opposed to an overlapping block Gauss-Seidel preconditioner ([39]). In [17] however, Dryja and Widlund describe this decoupling more algebraically. Following [36], they define the overlapping Schwarz method in terms of the product of a number of projections onto subspaces (corresponding to the subregion problems). They then define the decoupled approach to preconditioning in terms of sums of these projection operators, referring to such techniques as additive (as opposed to multiplicative) Schwarz methods (see below).

With both additive and multiplicative Schwarz methods the number of iterations required for convergence grows significantly with the number of subdomains. Hence, in [18], Dryja and Widlund introduce an extra step to the additive preconditioner (i.e. in addition to the decoupled overlapping subdomain solves). This step requires the additional solution of a problem on a much coarser triangulation than \mathcal{T} , whose elements are of size H say. The solution of this coarse grid problem is rather like the substructure solves given by \tilde{K}^{-1} in the previous subsection, however the elements of coarse grid do not need to correspond to the subdomains in this case (generally each subdomain will be made up of a number of coarse grid elements). Although this modification places a restriction on the way in which the domain Ω may be decomposed into subdomains it has the advantage that, provided the overlap distance, δ , is bounded below by a fixed fraction of the coarse grid size, H , the condition number of the preconditioned system is independent of both H and h . These arguments may be formalized by the following theoretical results.

Suppose that we have a coarse grid \mathcal{T}^0 and the fine grid \mathcal{T} may be obtained through a refinement of this grid². Then the FE space \mathcal{V}_0 , made up of all possible piecewise linear (say) polynomials on \mathcal{T}^0 is a subspace of \mathcal{V} , the FE space of all possible piecewise linear functions on \mathcal{T} . Furthermore, suppose that \mathcal{T}_i is the subset of \mathcal{T} that covers the subdomain Ω_i , for $i = 1, \dots, p$ (where the subdomains, and therefore the triangulations \mathcal{T}_i , may overlap). Then each \mathcal{V}_i , the FE space of all possible piecewise linear functions on \mathcal{T}_i , is also a subspace of \mathcal{V} . Given the usual FE basis for each of the spaces $\mathcal{V}, \mathcal{V}_0, \mathcal{V}_1, \dots, \mathcal{V}_p$ it is possible to define rectangular matrices, R_i say, corresponding to the L_2 projections from \mathcal{V} to \mathcal{V}_i for $i = 0, 1, \dots, p$. The additive Schwarz (AS) preconditioner is then given by the following algebraic expression,

$$M^{-1} = \sum_{i=0}^p R_i^T K_i^{-1} R_i, \quad (31)$$

²This restriction that the fine grid should be a refinement of the coarse grid is not strictly necessary but it does simplify the description that follows sufficiently to justify being made an assumption throughout this paper. See, for example [47], for a discussion of the use of non-nested grids.

where $K_i = R_i K R_i^T$ for $i = 0, 1, \dots, p$. An alternative way of viewing K_i is that it is the FE stiffness matrix derived from \mathcal{A}_i , the restriction of \mathcal{A} (from (1) and (4)) to $\mathcal{V}_i \times \mathcal{V}_i$ given by:

$$\mathcal{A}_i(u_i, v_i) = \mathcal{A}(u_i, v_i), \quad \forall u_i, v_i \in \mathcal{V}_i. \quad (32)$$

The following theorem which is proved in [53] for example (or see [47] for a slightly more general form), provides the main theoretical justification for this preconditioner.

Theorem 2.1 *The matrix M defined by (31) is symmetric and positive-definite. Furthermore, if we assume that there is some constant $C > 0$ such that: for all $v \in \mathcal{V}$ there are $v_i \in \mathcal{V}_i$ such that $v = \sum_{i=0}^p v_i$ and*

$$\sum_{i=0}^p \mathcal{A}_i(v_i, v_i) \leq C \mathcal{A}(v, v), \quad (33)$$

then the spectral condition number of $M^{-1}K$ is given by

$$\kappa(M^{-1}K) \leq n_c C, \quad (34)$$

where n_c is the minimum number of colours required to colour the subdomains Ω_i in such a way that no neighbours are the same colour.

Furthermore the following result, which is also proved in [47] and [53], demonstrates that the condition (33) is satisfied for a constant C that is independent of h , H and p provided that the overlap between the subdomains Ω_i is always proportional to the size of elements in the coarse triangulation. This is often referred to as a ‘‘generous’’ overlap, and the preconditioner is said to be ‘‘optimal’’ in this case.

Theorem 2.2 *Provided the overlap between the subdomains Ω_i is of size $O(H)$, where H represents the mesh size of \mathcal{T}^0 , then there exists $C > 0$, which is independent of h , H and p , such that for any $v \in \mathcal{V}$ there are $v_i \in \mathcal{V}_i$ such that $v = \sum_{i=0}^p v_i$ and*

$$\sum_{i=0}^p \mathcal{A}_i(v_i, v_i) \leq C \mathcal{A}(v, v). \quad (35)$$

The requirement for a generous overlap between subdomains in order to obtain an optimal preconditioner is extremely demanding. In two dimensions, as \mathcal{T} is refined (assuming uniform global refinement for simplicity), the number of elements of \mathcal{T} in the overlap regions is $O(h^{-2})$, and in three dimensions it is $O(h^{-3})$. In practice therefore this requirement is usually dropped in one of two different ways. The simplest option is to stick to a fixed number of layers of overlapping elements of the triangulation \mathcal{T} as it is refined. This means that the overlap region decreases as $h \rightarrow 0$ and so the condition number of the preconditioned system grows. Fortunately however this growth is typically quite slow and so the additional iterations required by the

PCG solver are justified by the reduced cost of applying the preconditioner at each iteration (since there are less unknowns associated with each subdomain problem in comparison to the optimal preconditioner with an $O(H)$ overlap), [47]. Nevertheless, the preconditioner is no longer optimal.

The other common approach taken to avoid the need for a generous overlap is to make use of a sequence of nested grids (as opposed to just a coarse grid and a fine grid). The coarse grid problem in (31) is then replaced by the solution of a problem on a grid which is only one level of refinement less than the fine grid problem. This problem is then solved using a two level approximation based upon the grid that is two levels of refinement less than the fine grid problem. Repetition of this two level approach at repeatedly coarser levels leads to a “multilevel” Schwarz algorithm that is rather more complex than the two level algorithm given by (31) but which yields an optimal preconditioner with only a minimal amount of overlap. Again see, for example, [47] for further details.

The main drawback with the multilevel AS algorithm is the need for a series of projections from one grid to the next at each preconditioning step. On a regular sequence of grids obtained through uniform refinement this is relatively straightforward to manage in parallel, however on a sequence of grids generated with local, rather than global, mesh refinement this can become a complex programming task. Throughout the rest of this paper therefore we describe an alternative to the multilevel approach that is a two level AS algorithm based upon the use of a hierarchical sequence of grids. As such, this may be considered to be a cross between the classical two level AS algorithm with a generous overlap and the classical multilevel AS algorithm.

3 A Hierarchical Two Level Schwarz Algorithm

This section introduces an alternative two level AS algorithm based upon the use of a “weakly overlapping” hierarchy of nested grids, as described in [5]. An overview of the parallel implementation is also included.

3.1 A Weakly Overlapping Additive Schwarz Preconditioner

As before, let \mathcal{T}^0 be a coarse triangulation of Ω and suppose that \mathcal{T}^0 consists of N_0 triangular elements, $\tau_j^{(0)}$, each of size $O(H)$. Now assume that \mathcal{T}^0 is partitioned into p *non-overlapping* subdomains Ω_i such that:

$$\bar{\Omega} = \bigcup_{i=1}^p \bar{\Omega}_i, \quad (36)$$

$$\Omega_i \cap \Omega_j = \phi \quad (i \neq j), \quad (37)$$

$$\bar{\Omega}_i = \bigcup_{j \in I_i} \tau_j^{(0)} \quad \text{where } I_i \subset \{1, \dots, N_0\} \quad (I_i \neq \phi). \quad (38)$$

Here the overbar is used to denote the closure of a set of points.

Now permit \mathcal{T}^0 to be refined several times, to produce a family of triangulations, $\mathcal{T}^0, \dots, \mathcal{T}^J$, where each triangulation, \mathcal{T}^k , consists of N_k elements, $\tau_j^{(k)}$, such that

$$\overline{\Omega} = \bigcup_{j=1}^{N_k} \tau_j^{(k)} \quad \text{and} \quad \mathcal{T}^k = \{\tau_j^{(k)}\}_{j=1}^{N_k}. \quad (39)$$

The successive mesh refinements that define this sequence of triangulations need not be global and may be non-conforming, however they must satisfy a number of conditions, as in [8] for example:

1. $\tau \in \mathcal{T}^{k+1}$ implies that either
 - (a) $\tau \in \mathcal{T}^k$, or
 - (b) τ has been generated as a refinement of an element of \mathcal{T}^k into four similar children,
2. the level of any triangles which share a common point can differ by at most one,
3. only triangles at level k may be refined in the transition from \mathcal{T}^k to \mathcal{T}^{k+1} .

(Here the level of a triangle is defined to be the least value of k for which that triangle is an element of \mathcal{T}^k .) In addition to the above it is also necessary that:

4. in the final mesh, \mathcal{T}^J , all pairs of triangles on either side of the boundary of each subdomain Ω_i have the same level as each other.

Having defined a decomposition of Ω into subdomains and a nested sequence of triangulations of Ω , [5] next defines the restrictions of each of these triangulations onto each subdomain by

$$\Omega_{i,k} = \{\tau_j^{(k)} : \tau_j^{(k)} \subset \overline{\Omega}_i\}, \quad (40)$$

and, in order to introduce a certain amount of overlap between neighbouring subdomains,

$$\tilde{\Omega}_{i,k} = \{\tau_j^{(k)} : \tau_j^{(k)} \text{ has a common point with } \overline{\Omega}_i\}. \quad (41)$$

Following this, finite element spaces associated with these local triangulations are introduced. Let G be some triangulation and denote by $\mathcal{S}(G)$ the space of continuous piecewise linear functions on G . Then the following definitions can now be made:

$$\mathcal{V} = \mathcal{S}(\mathcal{T}^J) \quad (42)$$

$$\mathcal{V}_0 = \mathcal{S}(\mathcal{T}^0) \quad (43)$$

$$\mathcal{V}_{i,k} = \mathcal{S}(\Omega_{i,k}) \quad (44)$$

$$\tilde{\mathcal{V}}_{i,k} = \mathcal{S}(\tilde{\Omega}_{i,k}) \quad (45)$$

$$\mathcal{V}_i = \tilde{\mathcal{V}}_{i,0} + \dots + \tilde{\mathcal{V}}_{i,J}. \quad (46)$$

Note that the spaces \mathcal{V} and \mathcal{V}_0 are the same as those defined in the previous section but that the spaces \mathcal{V}_i , for $i = 1, \dots, p$ are different (since they now correspond to an

overlap of just one element at each level of the mesh hierarchy). Nevertheless, it is still evident that

$$\mathcal{V} = \mathcal{V}_0 + \mathcal{V}_1 + \dots + \mathcal{V}_p . \quad (47)$$

This means that for all $v \in \mathcal{V}$ there are $v_i \in \mathcal{V}_i$ such that $v = \sum_{i=0}^p v_i$, and this is the decomposition of \mathcal{V} that [5] proposes for the alternative two level AS preconditioner of the form (31).

In [5] the following theorem is proved for problems in both two and three dimensions. When combined with Theorem 2.1, this shows that the preconditioner (31), with the rectangular matrices now representing projections from \mathcal{V} to the new spaces \mathcal{V}_i , is optimal.

Theorem 3.1 *Let \mathcal{V} , \mathcal{V}_0 and \mathcal{V}_i (for $i = 1, \dots, p$) be defined by (42), (43) and (46) respectively. Then there exists $C > 0$, which is independent of h , H and p , such that for any $v \in \mathcal{V}$ there are $v_i \in \mathcal{V}_i$ such that $v = \sum_{i=0}^p v_i$ and*

$$\sum_{i=0}^p \mathcal{A}_i(v_i, v_i) \leq C \mathcal{A}(v, v) . \quad (48)$$

3.2 Implementation

In order to implement the above AS preconditioner in parallel, [5] combines the coarse grid solve associated with K_0^{-1} in (31) with each of the subdomain solves. This is achieved by assigning a copy of the entire coarse mesh, \mathcal{T}^0 , to each processor but only allowing processor i to refine this mesh inside $\tilde{\Omega}_{i,k-1}$ at step k of the refinement process (i.e. from \mathcal{T}^{k-1} to \mathcal{T}^k). The continuous piecewise linear FE spaces on the resulting meshes are then given by

$$\tilde{\mathcal{V}}_i = \mathcal{V}_0 \cup \mathcal{V}_i \quad (49)$$

for $i = 1, \dots, p$. Corresponding meshes (with $p = 3$) are illustrated for a simple 2-d example in Figure 3. Note that in this figure there are a number of ‘‘slave’’ nodes in each processor’s mesh which cause these meshes to be non-conforming. The solution values at these nodes are not free: they are determined by the nodal values at the ends of the edges on which the slave nodes lie. For a practical implementation it turns out to be simpler to allow the solution values at these nodes to be free by performing an interior refinement of those elements on the unrefined sides of the edges that have ‘‘hanging’’ nodes on them. In the 2-d example of Figure 3 this simply involves bisecting all triangular elements containing a hanging node and for 3-d problems a similar intermediate refinement strategy may be used (see [48] for example).

Having obtained a mesh on each processor it is possible for processor i to assemble the stiffness matrix, K_i , for its own mesh independently of the other processors. These

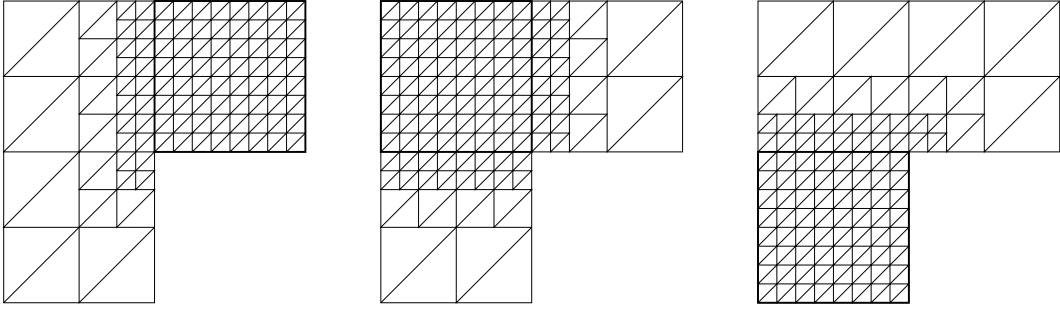


Figure 3: An example of the meshes produced when $p = 3$ and $J = 2$ for a coarse mesh containing 24 elements.

	$p = 2$	$p = 4$	$p = 8$	$p = 16$
$J = 2$	6	9	12	14
$J = 3$	6	8	12	13
$J = 4$	6	8	12	13
$J = 5$	6	7	11	12
$J = 6$	6	7	11	12

Table 1: The number of PCG iterations required to reduce the residual by a factor of 10^6 when solving Poisson's equation on a coarse mesh of 256 elements (taken from [5]).

are the matrices that are used in a modified version of the preconditioner (31) which takes into account the merging of \mathcal{V}_0 with \mathcal{V}_i :

$$M^{-1} = \sum_{i=1}^p \tilde{R}_i^T K_i^{-1} \tilde{R}_i . \quad (50)$$

In (50) each \tilde{R}_i is the rectangular matrix that represents the projection from \mathcal{V} to $\tilde{\mathcal{V}}_i$. Note that the application of these projections (and the corresponding prolongations \tilde{R}_i^T) requires interprocessor communications but that the subdomain solves (corresponding to K_i^{-1}) may be undertaken independently and concurrently. A number of results are presented in [5] however Table 1 shows a sample of these for solving Poisson's equation on a square domain in two dimensions on between 2 and 16 subdomains, using between 2 and 6 levels of uniform refinement. As predicted by the theory, the number of iterations taken appears to be bounded independently of h and p for a fixed choice of H .

4 Extensions

In this section two possible extensions of the work of [5], outlined in the previous section, are considered. The first of these is based upon an observation made by Cai

and Sarkis in [13] concerning traditional AS preconditioners of the form (31). The second extension is to a larger class of problem than those given by Problem 1.1. In addition to the straightforward generalization to three dimensional problems (which is included in [5]), it is possible to apply the preconditioning techniques reviewed in this paper to non-self-adjoint problems. This is described for convection-diffusion equations, based upon the work appearing in [4, 30, 31, 40, 41].

4.1 A Restricted Version of the Preconditioner

In [13] an AS preconditioner of the form (31) is considered. It is noted that for each subdomain i , for $i = 1, \dots, p$, the operations R_i and R_i^T each require a certain amount of computational work in order to restrict vectors between \mathcal{T} and \mathcal{T}_i , and prolongate vectors from \mathcal{T}_i to \mathcal{T} , respectively. Furthermore, in a parallel implementation on p processors, each of these operations requires interprocessor communication between neighbouring processors. In [13] therefore, a restricted AS preconditioner is proposed in which R_i^T is replaced by a different prolongation matrix, \hat{R}_i^T say, for $i = 1, \dots, p$. This matrix represents a simpler prolongation to \mathcal{T} from those nodes of \mathcal{T}_i in the region of Ω_i that does not overlap with neighbours, with overlapping nodal values set to zero. Hence no interprocessor communication is required for the prolongation steps and the total communication cost of each preconditioning step is halved. Interestingly, [13] reports that, with this restricted AS preconditioner, iteration counts are actually slightly lower than those obtained using the full AS preconditioner (31). It should be noted however that this new preconditioner is not symmetric positive-definite (SPD) and so the PCG algorithm can no longer be used.

In [4] and [41] results are presented (in two and three dimensions respectively) using a GMRES solver (see, for example, [3, 23, 42]) with a preconditioner based upon a restricted version of the new preconditioner (50):

$$M^{-1} = \sum_{i=1}^p \hat{R}_i^T K_i^{-1} \tilde{R}_i. \quad (51)$$

Here \tilde{R}_i and K_i are as in (50) and \hat{R}_i^T is a rectangular prolongation matrix that maps $\underline{\zeta} \in \mathfrak{R}^{\tilde{n}_i}$ to $\underline{z} \in \mathfrak{R}^n$ in the following manner (where the index j is used to enumerate the n nodes in the fine mesh \mathcal{T}).

$$\begin{aligned} z_j &= \zeta_k && \text{when node } j \text{ is in the interior of} \\ & && \text{the subregion owned by processor } i \\ & && \text{(and is numbered } k \text{ on processor } i), \\ z_j &= \zeta_k / \nu_k && \text{when node } j \text{ is on the boundary of} \\ & && \text{the subregion owned by processor } i, \\ & && \text{(and is numbered } k \text{ on processor } i), \\ z_j &= 0 && \text{otherwise.} \end{aligned}$$

	AS preconditioner (50)				Reduced AS preconditioner (51)			
	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
$J = 1$	6	11	16	19	3	3	4	4
$J = 2$	7	12	17	19	3	5	6	6
$J = 3$	8	13	19	20	4	6	7	8
$J = 4$	8	13	19	20	4	8	9	9

Table 2: The number of preconditioned GMRES iterations required to reduce the residual by a factor of 10^5 when solving Poisson’s equation on a coarse mesh of 384 tetrahedral elements (taken from [41]).

Here ν_k represents the total number of processors for which node k of mesh i lies on their subregion boundary and \tilde{n}_i denotes the dimension of $\tilde{\mathcal{V}}_i$ as defined by (49).

As with the restricted version of (31) reported in [13], the preconditioner (51) typically performs better than its symmetric counterpart. For example, Table 2 illustrates results included in [41] for the solution of Poisson’s equation on a cube-like domain in three dimensions using between 2 and 16 subdomains and 1 to 4 levels of uniform refinement. It is clear that for the same partitions into 2, 4, 8 and 16 weakly overlapping subdomains the restricted preconditioner performs considerably better. The original preconditioner is optimal but the upper bound on the number of iterations appears to be much higher than for the restricted preconditioner. In the latter case there is, as yet, no theoretical proof of the optimality of the preconditioner however numerical evidence, such as that presented in Table 2 and [4, 40, 41], suggests that it is the better choice in practice.

4.2 Generalization to Convection-Diffusion Problems

So far this paper has only considered the application of DD techniques to self-adjoint PDEs that may be expressed in the form given by Problem 1.1. When discretized this type of equation naturally leads to a matrix system (6) which is SPD. In practice however, many practical problems are not self-adjoint and so it is desirable to be able to apply DD techniques to a wider class of equation. In this subsection we follow [4, 30, 31, 40, 41] in considering the extension of the restricted weakly overlapping AS preconditioner (51) to convection-diffusion problems of the following form.

Problem 4.1 Find $u \in H_E^1(\Omega)$ such that

$$\mathcal{A}(u, v) + \mathcal{C}(u, v) = \mathcal{F}(v), \quad \forall v \in H_0^1(\Omega), \quad (52)$$

where $\Omega \in \mathbb{R}^2$ or \mathbb{R}^3 and $H_E^1(\Omega)$ and $H_0^1(\Omega)$ are as in (2) and (3) respectively.

Here $\mathcal{A}(u, v)$ and $\mathcal{F}(v)$ are as in (4) and

$$\mathcal{C}(u, v) = \int_{\Omega} \underline{b} \cdot \nabla uv \, d\underline{x} \quad (53)$$

	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$J = 2$	3	4	4	4	5
$J = 3$	3	4	5	5	5
$J = 4$	3	4	5	5	5
$J = 5$	3	4	5	5	5
$J = 6$	3	4	5	5	5
$J = 7$	3	4	5	5	5

Table 3: The number of preconditioned GMRES iterations required to reduce the residual by a factor of 10^6 when solving (56) on a coarse mesh of 64 elements (taken from ([4]).

for some $\underline{b} \neq \underline{0}$. The standard Galerkin FE discretization of this problem leads to a matrix system (6) which is no longer SPD. In fact, each entry of the stiffness matrix K is now given by

$$K_{ji} = \int_{\Omega} A(\underline{x}) \underline{\nabla} N_j \cdot \underline{\nabla} N_i \, d\underline{x} + \int_{\Omega} N_j \underline{b} \cdot \underline{\nabla} N_i \, d\underline{x}, \quad (54)$$

with entries of the load vector (9) similarly modified (from the Dirichlet boundary terms). Following the block matrix notation of (12) the matrix system may now be written as

$$\begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \cdots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_p \\ \underline{b}_s \end{bmatrix}, \quad (55)$$

and a Krylov subspace solver, such as GMRES, applied in parallel. All of the observations made in Subsections 1.1 and 1.2 concerning parallel assembly and parallel matrix-vector products still apply to this new system but with B_i^T replaced by C_i for $i = 1, \dots, p$.

When it comes to preconditioning the system (55) it is natural to use the restricted preconditioner (51) rather than (50) since (55) is already non-symmetric. Results, taken from [4], are presented in Table 3 for the solution of the PDE

$$-\underline{\nabla}(\underline{\nabla}u) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \underline{\nabla}u = f \quad (56)$$

on a square domain in two dimensions. As before we see that, for a given coarse grid \mathcal{T}^0 , the number of preconditioned GMRES iterations appears to be bounded independently of h (equivalently J) and p .

In [30, 31, 40, 41] Problem 4.1 is considered in the case where the convection term dominates the diffusion term, i.e. $\|A(\underline{x})\| \ll \|\underline{b}\|$ (with $A(\underline{x})$ and \underline{b} as in (4) and (53) respectively). In this situation the Galerkin method is known to be unstable unless the

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
$J = 1$	2	3	3	3	2	3	3	3
$J = 2$	3	3	4	4	3	4	4	4
$J = 3$	3	4	4	5	3	4	4	4
$J = 4$	4	5	6	6	3	4	4	4

Table 4: The number of preconditioned GMRES iterations required to reduce the residual by a factor of 10^5 when solving Problem 4.1 with $A(\underline{x}) = \varepsilon I$, $\underline{b} = (1, 0, 0)^T$, and a coarse mesh of 384 tetrahedral elements (taken from [41]).

mesh Peclet number (essentially the ratio of h to $\|A(\underline{x})\|/\|\underline{b}\|$) is sufficiently small (see [33] for example). In order to be able to solve such problems on realistic meshes therefore, a more stable FE discretization is required and so, as in [30, 31, 40, 41], the streamline-diffusion FE method may be considered. This again alters the definition of the matrix K in (6) so that, assuming piecewise linear FE elements are used for simplicity,

$$K_{ji} = \int_{\Omega} A(\underline{x}) \underline{\nabla} N_j \cdot \underline{\nabla} N_i \, d\underline{x} + \int_{\Omega} N_j \underline{b} \cdot \underline{\nabla} N_i \, d\underline{x} + \alpha \int_{\Omega} (\underline{b} \cdot \underline{\nabla} N_j) (\underline{b} \cdot \underline{\nabla} N_i) \, d\underline{x}. \quad (57)$$

Here, α is a streamline-diffusion parameter which also appears in a similarly modified form of the load vector \underline{b} .

It is apparent that the AS techniques described in this paper, and in particular the restricted weakly overlapping AS preconditioner (51), may be trivially adapted for the case where the stiffness matrix K is given by (57) as opposed to (8) or (54). Results are presented in [30, 31, 40, 41] showing that this again leads to an apparently optimal preconditioner, although no formal proof of this is offered. Table 4 shows a typical set of iteration counts, taken from [30], where $A(\underline{x}) = \varepsilon I$, $\underline{b} = (1, 0, 0)^T$, $\Omega = (0, 2) \times (0, 1) \times (0, 1)$ and the coarse grid \mathcal{T}^0 contains 768 tetrahedral elements.

4.3 Some Parallel Performance Results

We conclude this section with a small number of sample parallel results taken from [4] and [40]. It should be noted that there are a number of important factors that affect the quality of these results and so they should be regarded as illustrative only. Further details may be found in [4, 40].

The main factors that affect the parallel performance of the preconditioner (51) may be summarized as follows.

- The way in which Ω is decomposed into p subdomains. In the two examples below a simple recursive coordinate bisection algorithm, [44], has been used on \mathcal{T}^0 , but for more complex problems a more sophisticated strategy should

be used. Furthermore, for convection-dominated problems there may be some advantage to be had in aligning the subdomains with the convection direction where this is possible.

- The accuracy of the subproblem solves (corresponding to K^{-1} in (51)) on each processor. If these are solved too accurately then unnecessary computational effort is expended, however if they are not solved sufficiently accurately the number of iterations taken by the preconditioned GMRES solver may grow. In practice a relative residual reduction of between 10^1 and 10^2 appears to be best.
- The efficiency of the parallel implementation. Where possible all communication should be overlapped with computation for example. Idle processor time should be kept to a minimum through a good load-balancing strategy: see the note on the decomposition of Ω above.
- The quality of the DD solver when compared to the best available sequential solver. In the tables below all timings are contrasted against those of a fast sequential algebraic multilevel ILU preconditioned solver, see [6] for example.

The timings given in Tables 5 and 6 are taken from [4] and [40] respectively. In each case the ‘‘Speedup’’ row compares the parallel solution time (on a SGI Origin 2000 computer) with the best sequential solution time, whereas the ‘‘Parallel speedup’’ row compares the parallel solution time on p processors with the sequential solution time for the p subdomain algorithm.

Table 5 shows timings for the solution of the two-dimensional problem (56) using the Galerkin FE method on a square domain for which \mathcal{T}^0 has 256 elements and a uniform refinement level of $J = 6$ (hence $\mathcal{T}^J = \mathcal{T}^6$ and contains 1048576 elements). Table 6 shows timings for the solution of the three-dimensional problem

$$-\frac{1}{1000}\nabla(\nabla u) + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \nabla u = f \quad (58)$$

using the streamline-diffusion FE method. Here $\Omega = (0, 2) \times (0, 1) \times (0, 1)$, \mathcal{T}^0 contains 768 elements and a uniform refinement level of $J = 4$ is used (hence $\mathcal{T}^J = \mathcal{T}^4$ and contains 3145728 elements). It can be seen that in both cases excellent parallel speedups and very useful speedups are achieved.

5 Summary

The aim of this paper has been to introduce the reader to some of the main aspects of domain decomposition preconditioning. This includes a motivation for DD methods through their applicability to the solution of PDEs using parallel computing architectures. Given that this is the main motivation for using these methods the paper also provides a short overview of the parallel assembly of finite element systems of

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
Sequential time	234.4	351.0	301.2	271.0	262.6
Parallel time	—	191.0	83.1	42.9	20.6
Speedup	—	1.2	2.8	5.5	11.4
Parallel speedup	—	1.8	3.6	6.3	12.7

Table 5: Solution times (in seconds) and speedups for the restricted weakly overlapping algorithm on the two-dimensional problem (56) (taken from [4]).

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
Sequential time	559.7	634.1	760.6	868.3	941.6
Parallel time	—	323.4	196.6	114.2	65.8
Speedup	—	1.7	2.8	4.9	8.5
Parallel speedup	—	2.0	3.9	7.6	14.3

Table 6: Solution times (in seconds) and speedups for the restricted weakly overlapping algorithm on the three-dimensional problem (58) (taken from [40]).

equations based upon a geometric decomposition of the problem. This decomposition requires that the FE grid be partitioned and that this partition be mapped onto the processor network in some way.

Before discussing preconditioning algorithms the introductory material in Section 1 also describes the main computational steps that are required by a typical iterative solution algorithm such as CG or GMRES. The assumption is made that the FE equations have been assembled in parallel and that a parallel solution is required. This requires the ability to undertake distributed matrix-vector multiplications in parallel and to compute distributed inner products in parallel. Both of these operations are considered. The final part of the introductory section is included for completeness and is not built upon in the rest of the paper. This describes the Schur complement approach to the solution of block-arrowhead systems of the form (13). Although not discussed here, it should be noted that there are many similarities between the domain decomposition methods that may be used to precondition the Schur complement system (21) and the full system (13): see, for example, [35] for further details.

Section 2 of the paper introduces the notion of preconditioning for the iterative solution of linear systems of equations. The motivation for the need to precondition comes from the earlier observation that the condition number of the stiffness matrix K grows like $O(h^{-2})$ as $h \rightarrow 0$. The main properties required for a preconditioning matrix are discussed and then two possible classes of DD preconditioner are considered in turn.

A simple example of an iterative substructuring technique is described both in terms of the construction of a preconditioning matrix and the action of the multiplication of a vector by the inverse of this matrix (i.e. the solution of a linear system). The main

feature of this type of preconditioner is that the subdomains themselves act as a very coarse grid upon which a solution is required as part of the preconditioning process. In a parallel implementation this tends to lead to there being more than one subdomain per processor in order to obtain a substructure that is not too coarse. Since this method also requires frequent subdomain solves (one per iteration of the PCG solver) having many subdomains per processor means that these subdomain problems are smaller, although there are more of them.

The rest of the paper is concerned with Schwarz preconditioners. The distinction between additive and multiplicative Schwarz methods is made and a simple additive Schwarz preconditioner, (31), is introduced. The potential use of subdomain colouring in order to implement a multiplicative algorithm in parallel is commented upon but otherwise the focus is on the properties and parallel implementation of (31), and similar AS preconditioners, with one subdomain per processor. In particular two fundamental theorems are presented concerning the quality of (31). The first of these, Theorem 2.1, states that, provided the finite element subspaces that are associated with each subdomain form a stable splitting (as defined by (33)) of the global finite element space, then the condition number of the preconditioned system is bounded. The second result, Theorem 2.2, then shows that if there is a generous overlap between the subdomains then the subdomain spaces do indeed form a stable splitting of the global space (provided a coarse grid solve is included as part of the preconditioner).

Unfortunately the computational cost associated with maintaining an overlap that is independent of h as the finest mesh is refined is not justified in terms of practical performance, and so the use of a smaller overlap and the use of multilevel methods are both discussed. Instead of these approaches a new alternative is then proposed which combines many of the features of the optimal two level algorithm with those of optimal multilevel algorithms. This is based upon a two level preconditioner of the form (31) but with a different splitting into subdomain problems which makes use of a nested hierarchy of finite element grids. In this splitting each subdomain problem has an overlap layer of precisely one element at each level of the refinement. This has many fewer elements in the overlap layer than the conventional generous overlap splitting however, as demonstrated by Theorem 3.1, it still yields an optimal preconditioner.

The practical implementation of this weakly overlapping preconditioner is then discussed in Subsection 3.2. This involves maintaining a copy of the entire coarse grid \mathcal{T}^0 on each subdomain and then building the subdomain problems on top of this on each processor by only refining the grids in, or immediately next to, the subdomain owned by that processor. The combination of the coarse grid problem with each subdomain problem is described algebraically by (49) and the modified form of the AS preconditioner is given by (50). Results, taken from [5], are provided to illustrate the practical behaviour of the algorithm on a simple two-dimensional test problem. The immediate extension to three dimensions is also noted.

The final main section of the paper discusses two extensions to the weakly overlapping AS preconditioner. The first of these is to introduce a restricted version of this preconditioner, (51), following the work of [13] for conventional AS preconditioners.

This is shown not only to require less communication when implemented in parallel but also to significantly reduce the number of iterations required over the preconditioner (50). The only potential drawback of the restricted approach is that symmetry is lost and so the PCG algorithm has to be replaced by a more general iterative method such as GMRES. It is clear however that the small additional cost per iteration of applying GMRES is more than offset by the reduced number of iterations taken.

The second extension considered takes the loss of symmetry one step further by applying the restricted weakly overlapping technique to the solution of problems that are themselves non-symmetric. In particular convection-diffusion problems are considered. Two discretization methods are discussed: standard Galerkin FE and streamline-diffusion FE (the latter being necessary for the stability of the solution of convection-dominated problems). In both cases, and in both two and three dimensions, the observed numerical results suggest that the preconditioner is still optimal (or close to optimal) even though no theoretical proof of this is available.

Section 4 concludes by presenting some typical parallel performance results, as reported in [4] and [40]. These show that the DD technique parallelizes very efficiently but that the speed of the sequential version of the p subdomain algorithm is generally less than that of the best available sequential algorithm. This imposes a restriction on the overall efficiency of the algorithm when compared to the best available sequential algorithm. Nevertheless, the parallel timings are extremely encouraging and demonstrate that it is possible to obtain practical results that are in line with theoretical expectations.

References

- [1] M. Ainsworth, “A Preconditioner Based on Domain Decomposition for h - p Finite Element Approximation on Quasi-Uniform Meshes”, *SIAM J. on Numerical Analysis*, 33, 1358–1376, 1996.
- [2] M. Ainsworth, “A Hierarchical Domain Decomposition Preconditioner for h - p Finite Element Approximation on Locally Refined Meshes”, *SIAM J. on Scientific Computing*, 17, 1395–1413, 1996.
- [3] 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.
- [4] R.E. Bank and P.K. Jimack, “A New Parallel Domain Decomposition Method for the Adaptive Finite Element Solution of Elliptic Partial Differential Equations”, *Concurrency and Computation: Practice and Experience*, 13, 327–350, 2001.
- [5] R.E. Bank, P.K. Jimack, S.A. Nadeem and S.V. Nepomnyaschikh, “A Weakly Overlapping Domain Decomposition Preconditioner for the Finite Element Solution of Elliptic Partial Differential Equations”, *SIAM J. on Scientific Computing*, 23, 1818–1842, 2002.
- [6] R.E. Bank and R.K. Smith, “An Incomplete Factorization Multigraph Algorithm”, *SIAM J. on Scientific Computing*, 20, 1349–1364, 1999.

- [7] M. Benzi, J.K. Cullum and M. Tuma, “*Parallel Implementation and Practical use of Sparse Approximate Inverse Preconditioners with A Priori Sparsity Patterns*”, SIAM J. on Scientific Computing, 22, 1318–1332, 2000.
- [8] V. Bornemann and H. Yserentant, “*A Basic Norm Equivalence for the Theory of Multilevel Methods*, Numerische Mathematik, 64), 455–476, 1993.
- [9] J. Bramble, J. Pasciak and A.H. Schatz, “*The Construction of Preconditioners for Elliptic Problems by Substructuring, I*”, Mathematics of Computation, 47, 103–134, 1986.
- [10] J. Bramble, J. Pasciak and A.H. Schatz, “*The Construction of Preconditioners for Elliptic Problems by Substructuring, II*”, Mathematics of Computation, 49, 1–16, 1987.
- [11] J. Bramble, J. Pasciak and A.H. Schatz, “*The Construction of Preconditioners for Elliptic Problems by Substructuring, III*”, Mathematics of Computation, 51, 415–430, 1988.
- [12] J. Bramble, J. Pasciak and A.H. Schatz, “*The Construction of Preconditioners for Elliptic Problems by Substructuring, IV*”, Mathematics of Computation, 53, 1–24, 1989.
- [13] X.-C. Cai and M. Sarkis, “*An Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems*”, SIAM J. on Scientific Computing, 21, 792–797, 1999.
- [14] T.F. Chan and J.P. Shao, “*Parallel Complexity of Domain Decomposition Methods and Optimal Coarse Grid Size*”, Parallel Computing, 21, 1033–1049, 1995.
- [15] E. Chow, “*A Priori Sparsity Patterns for Parallel Sparse Approximate Inverse Preconditioners*”, SIAM J. on Scientific Computing, 21, 1804–1822, 2000.
- [16] G. Cybenko, “*Dynamic Load Balancing for Distributed Memory Multiprocessors*”, J. of Parallel and Distributed Computing, 7, 279–301, 1989.
- [17] 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 (T.F. Chan *et al.*, eds.), SIAM publications, Philadelphia, 1990.
- [18] M. Dryja and O.B. Widlund, “*Some Domain Decomposition Algorithms for Elliptic Problems*”, in Iterative Methods for Large Linear Systems, Academic Press, 1990.
- [19] C. Farhat, “*A Simple and Efficient Automatic FEM Domain Decomposer*”, Computers and Structures, 28, 579–602, 1988.
- [20] C. Farhat, P.-S. Chen and P. Stern, “*Towards the Ultimate Iterative Substructuring Method: Combined Numerical and Parallel Scalability, and Multiple Load Cases*”, Computing Systems in Engineering, 5, 337–350, 1994.
- [21] C. Farhat, J. Mandel and F.X. Roux, “*Optimal Convergence Properties of the FETI Domain Decomposition Method*”, Computer Methods for Applied Mechanics and Engineering, 115, 365–385, 1994.
- [22] G. Globisch, “*PARMESH: A Parallel Mesh Generator*”, Parallel Computing, 21, 509–524, 1995.
- [23] G.H. Golub and C.F. Van Loan, “*Matrix Computations*”, John Hopkins Press,

- 2nd edition, 1989.
- [24] B. Hendrickson and R. Leland, “*An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*”, SIAM J. on Scientific Computing, 16, 452–469, 1995.
 - [25] D.C. Hodgson and P.K. Jimack, “*Efficient Mesh Partitioning for Parallel Elliptic Differential Equation Solvers*”, Computing Systems in Engineering, 6, 1–12, 1995.
 - [26] K.H. Hoffman and J. Zou, “*Parallel Efficiency of Domain Decomposition Methods*”, Parallel Computing, 19, 1375–1391, 1993.
 - [27] T.J.R. Hughes, I. Levit and J. Winget, “*An Element-by-Element Solution Algorithm for Problems of Structural and Solid Mechanics*”, Computer Methods for Applied Mechanics and Engineering, 36, 241–254, 1983.
 - [28] D.C. Hodgson and P.K. Jimack, “*A Domain Decomposition Preconditioner for a Parallel Finite Element Solver on Distributed Unstructured Grids*”, Parallel Computing, 23, 1157–1181, 1997.
 - [29] P.K. Jimack and D.C. Hodgson, “*Parallel Preconditioners Based Upon Domain Decomposition*”, in Parallel and Distributed Processing for Computational Mechanics: Systems and Tools, ed. B.H.V. Topping (Saxe-Coburg Publications), 207–223, 1999.
 - [30] P.K. Jimack and S.A. Nadeem, “*Parallel Application of a Novel Domain Decomposition Preconditioner for the Stable Finite Element Solution of Three-Dimensional Convection-Dominated PDEs*”, in Euro-Par 2001 Parallel Processing: 7th International Euro-Par Conference Manchester, UK, August 2001 Proceedings, ed. R. Sakellariou et al. (Lecture Notes in Computer Science 2150, Springer), 592–601, 2001.
 - [31] P.K. Jimack and S.A. Nadeem, “*A Parallel Domain Decomposition Algorithm for the Adaptive Finite Element Solution of 3-D Convection-Diffusion Problems*”, in Computational Science – ICCS 2002: International Conference Amsterdam, The Netherlands, April 2002 Proceedings Part II, ed. P.M.A. Sloot et al. (Lecture Notes in Computer Science 2330, Springer), 797–805, 2002.
 - [32] P.K. Jimack and N. Touheed, “*Developing Parallel Finite Element Software Using MPI*”, in High Performance Computing for Computational Mechanics, ed. B.H.V. Topping and L. Lammer (Saxe-Coburg Publications), 15–38, 2000.
 - [33] C. Johnson “*Numerical Solution of Partial Differential Equations by the Finite Element Method*”, Cambridge University Press, 1987.
 - [34] G. Karypis and V. Kumar, “*Parallel Multilevel k -way Partition Scheme for Irregular Graphs*”, SIAM Review, 41, 278–300, 1999.
 - [35] D.E. Keyes and W.D. Gropp, “*A Comparison of Domain Decomposition Techniques for Elliptic PDEs and their Parallel Implementation*”, SIAM J. on Scientific and Statistical Computing, 8, 166–202, 1987.
 - [36] P.L. Lions, “*Interprétation Stochastique de la Méthode Alternée de Schwarz*”, C. R. Acad. Sci. Paris, 268, 325–328, 1978.
 - [37] T.A. Mantueffel, “*Shifted Incomplete Cholesky Factorization*”, in Sparse Matrix Proceedings (I.S. Duff and G.W. Stewart eds.), SIAM Publications, Philadelphia,

- 1978.
- [38] Message Passing Interface Forum, “*MPI: A Message-Passing Interface Standard*”, International Journal of Supercomputer Applications, 8, No. 3/4, 1994.
 - [39] G. Meurant, “*Domain Decomposition Methods for PDEs on Parallel Computers*”, Int. J. Supercomputer Applications, 2, 5–12, 1988.
 - [40] S.A. Nadeem “*Parallel Domain Decomposition Preconditioning for the Adaptive Finite Element Solution of Elliptic Problems in Three Dimensions*”, Ph.D. thesis, University of Leeds, 2001.
 - [41] S.A. Nadeem and P.K. Jimack, “*Parallel Implementation of an Optimal Two Level Additive Schwarz Preconditioner for the 3-D Finite Element Solution of Elliptic Partial Differential Equations*”, to appear in Int. J. Num. Meth. Fluids, 2002.
 - [42] 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.
 - [43] H.A. Schwarz, “*Über Einige Abbildungsaufgaben*”, J. für Die Reine und Angewandte Mathematik, 70, 105–120, 1869.
 - [44] H.D. Simon, “*Partitioning of Unstructured Problems for Parallel Processing*”, Computing Systems in Engineering, 2, 135–148, 1991.
 - [45] B.F. Smith, “*An Optimal Domain Decomposition Preconditioner for the Finite Element Solution of Linear Elasticity Problems*”, SIAM J. on Scientific Computing, 13, 364–378, 1992.
 - [46] B.F. Smith, “*A Parallel Implementation of an Iterative Substructuring Algorithm for Problems in Three Dimensions*”, SIAM J. on Scientific Computing, 14, 406–423, 1993.
 - [47] B. Smith, P. Bjorstad and W. Gropp, “*Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*”, Cambridge University Press, 1996.
 - [48] W. Speares and M. Berzins, “*A 3-d Unstructured Mesh Adaptation Algorithm for Time-Dependent Shock Dominated Problems*”, International Journal for Numerical Methods in Fluids, 25, 81–104, 1997.
 - [49] G. Strang and G.J. Fix, “*An Analysis of the Finite Element Method*”, Prentice-Hall, Englewood-Cliffs, 1973.
 - [50] C. Walshaw and M. Cross, “*Parallel Optimisation Algorithms for Multilevel Mesh Partitioning*”, Parallel Computing, 26, 1635–1660, 2000.
 - [51] A.J. Wathen, “*An Analysis of some Element-by-Element Techniques*”, Computer Methods for Applied Mechanics and Engineering, 74, 271–287, 1989.
 - [52] R.D. Williams, “*Performance of Dynamic Load Balancing for Unstructured Mesh Calculations*”, Concurrency: Practice and Experience, 3, 457–481, 1991.
 - [53] J. Xu, “*Iterative Methods by Space Decomposition and Subspace Correction*”, SIAM Review, 34, 581–613, 1992.