CONCURRENCY—PRACTICE AND EXPERIENCE
Concurrency: Pract. Exper. 2001; **00**:1–17 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

Parallel application of a novel domain decomposition preconditioner for the adaptive finite element solution of three-dimensional convection-dominated PDEs

P. K. Jimack*and S. A. Nadeem†

Computational PDEs Unit, School of Computing, University of Leeds, Leeds, LS2 9JT, U.K.

SUMMARY

We describe and analyze the parallel implementation of a novel domain decomposition preconditioner for the fast iterative solution of linear systems of algebraic equations arising from the discretization of elliptic partial differential equations (PDEs) in three dimensions. In previous theoretical work this preconditioner has been proved to be optimal for symmetric positive-definite (SPD) linear systems. In this paper we provide details of our 3-d parallel implementation and demonstrate that the technique may be generalized to the solution of non-symmetric algebraic systems, such as those arising when convection-diffusion problems are discretized using either Galerkin or stabilized finite element methods (FEMs). Furthermore we illustrate the potential of the preconditioner for use within an adaptive finite element framework by successfully solving convection-dominated problems on locally, rather than globally, refined meshes.

KEY WORDS: Domain decomposition; additive Schwarz; weakly overlapping; convection-diffusion.

1. INTRODUCTION

Domain decomposition (DD) techniques for the solution of sparse linear algebraic systems arising from the discretization of PDEs have become extremely popular in recent years due to their obvious potential for parallel implementation. Typically, two main approaches have been followed: generating and solving systems of equations on the subdomain interfaces (e.g. [7, 10],

*Correspondence to: pkj@comp.leeds.ac.uk
†The work of SAN was supported by the Government of Pakistan through a Quaid-e-Azam scholarship.

Received 21 December 2001

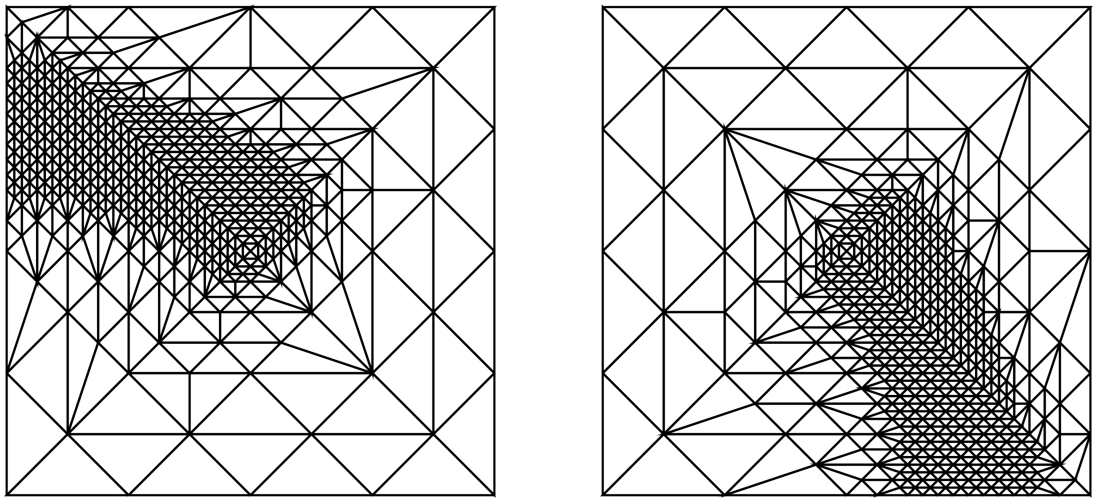
Figure 1. An example (in 2-d for clarity) of two weakly overlapping finite element meshes generated from a coarse grid of 64 elements with three levels of hierarchical refinement.

which each require exact subdomain solves at each iteration) or solving the complete system as a partitioned matrix (e.g. [6, 9]). In this work we focus on a recently proposed method of the second type ([3, 4]) which we refer to as a weakly overlapping additive Schwarz (AS) preconditioner.

Typical AS preconditioners (see, for example, [16]) require a fixed amount of overlap between subdomains in order to guarantee that the preconditioned linear systems which arise following discretization have a condition number which is independent of the mesh size h[†]. For practical applications therefore this optimality property is usually discarded in favour of keeping a fixed number of mesh layers in the overlap region (which, in three dimensions, therefore results in an overlap of $O(h^2)$ elements, as opposed to $O(h^3)$, as $h \to 0$). In [3] a hierarchical finite element technique is introduced which defines the solution space on each subdomain to consist of a global coarse grid plus a single layer of overlap at each level of refinement in the mesh hierarchy (see Fig. 1 for a two-dimensional illustration). It is then proved in [4] that for certain symmetric self-adjoint operators the resulting additive Schwarz preconditioner is still optimal, despite only having $O(h^2)$ elements in the overlap ($O(h)$ in 2-d).

To illustrate the technique algebraically consider solving a self-adjoint problem with just two subdomains (one on each of two processors say), as illustrated in 2-d in Fig. 1 (where the

[†]It is also necessary to add the solution of a restricted coarse grid problem at each iteration for such an optimal preconditioner (again see [16], or [6]).

subdomains lie above and below diagonal from the bottom left to the top right of the domain). Following the usual parallel finite element approach (e.g. [10]), a distributed global stiffness matrix may be assembled in parallel on the two processors by permitting processor i to assemble contributions from those fine mesh elements inside subdomain i only. The corresponding linear system of finite element equations may then be represented in the following block matrix form:

$$\begin{bmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ B_1^T & B_2^T & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{f}_1 \\ \underline{f}_2 \\ \underline{f}_s \end{bmatrix} . \tag{1}$$

Here $\underline{u}_i$ is the vector of unknown nodal values for nodes strictly inside subdomain i $(i = 1, 2)$ and $\underline{u}_s$ is the vector of unknown nodal values for nodes on the interface between subdomains. Moreover, each block A_i, B_i and $\underline{f}_i$ may be computed (and stored) independently on processor i $(i = 1, 2)$. Finally, we may express

$$A_s = A_{s(1)} + A_{s(2)} \quad \text{and} \quad \underline{f}_s = \underline{f}_{s(1)} + \underline{f}_{s(2)} , \tag{2}$$

where $A_{s(i)}$ and $\underline{f}_{s(i)}$ are the components of A_s and $\underline{f}_s$ respectively that may be calculated (and stored) independently on processor i. It is now quite straightforward to implement an iterative solver such as the conjugate gradient (CG) method ([1]) in parallel since distributed matrix-vector products may be computed with very little parallel overhead and distributed inner products may be computed with just a single global reduction operation (see, for example, [9]).

Parallel application of the weakly overlapping AS preconditioner, $\tilde{A}$ say, may now be described by considering the action of $\underline{z} = \tilde{A}^{-1}\underline{r}$ in the block matrix notation of (1) as follows. On processor 1 solve the system

$$\begin{bmatrix} A_1 & 0 & B_1 \\ 0 & \tilde{A}_2 & \tilde{B}_2 \\ B_1^T & \tilde{B}_2^T & A_s \end{bmatrix} \begin{bmatrix} \underline{z}_{1,1} \\ \underline{z}_{2,1} \\ \underline{z}_{s,1} \end{bmatrix} = \begin{bmatrix} \underline{r}_1 \\ M_2\underline{r}_2 \\ \underline{r}_s \end{bmatrix} \tag{3}$$

and on processor 2 solve the system

$$\begin{bmatrix} \tilde{A}_1 & 0 & \tilde{B}_1 \\ 0 & A_2 & B_2 \\ \tilde{B}_1^T & B_2^T & A_s \end{bmatrix} \begin{bmatrix} \underline{z}_{1,2} \\ \underline{z}_{2,2} \\ \underline{z}_{s,2} \end{bmatrix} = \begin{bmatrix} M_1\underline{r}_1 \\ \underline{r}_2 \\ \underline{r}_s \end{bmatrix} , \tag{4}$$

then set

$$\begin{bmatrix} \underline{z}_1 \\ \underline{z}_2 \\ \underline{z}_s \end{bmatrix} = \begin{bmatrix} \underline{z}_{1,1} + M_1^T\underline{z}_{1,2} \\ M_2^T\underline{z}_{2,1} + \underline{z}_{2,2} \\ \underline{z}_{s,1} + \underline{z}_{s,2} \end{bmatrix} . \tag{5}$$

In the above notation, the blocks $\tilde{A}_2$ and $\tilde{B}_2$ (resp. $\tilde{A}_1$ and $\tilde{B}_1$) are the assembled components of the stiffness matrix for the part of the mesh on processor 1 (resp. 2) that covers subdomain 2 (resp. 1). These may be computed and stored without communication. Moreover, because of the single layer of overlap in the refined regions of the meshes, A_s may be computed and stored on each processor without communication. Finally, the rectangular matrix M_1 (resp.

M_2) represents the restriction operator from the fine mesh covering subdomain 1 (resp. 2) on processor 1 (resp. 2) to the coarser mesh covering subdomain 1 (resp. 2) on processor 2 (resp. 1). This is the usual hierarchical restriction operator that is used in most multigrid algorithms (see, for example, [13]) and requires the communication of data between the processors.

It is easy to verify that the above preconditioner is symmetric and may be generalized from 2 to p subdomains (see [3] or [4] for details). It should also be noted that each of the local problems ((3) and (4) in the two-subdomain example above) combines its own subspace solve with the coarse grid solve and so we are effectively repeating this coarse grid correction on each processor at each iteration. This is not a significant overhead however since the coarse grid is generally very much smaller than the refined grid so most two level parallel codes (e.g. [10]) solve this problem sequentially on a single processor anyway. Furthermore, as $h \to 0$, each of these local problems tends to exactly $\frac{1}{p}$ times the size of the full problem, even with the coarse grid included.

2. SOLUTION OF CONVECTION-DIFFUSION PROBLEMS

Whilst the theoretical results of [4] demonstrate that the preconditioner given by (3) to (5) (when $p = 2$) is optimal for a class of linear self-adjoint PDEs (leading to SPD linear systems), it is clear that many important practical problems cannot be realistically modelled by such equations. One of the most important class of problem that comes into this category involves convection-diffusion equations of the form

$$-\varepsilon \underline{\nabla}^2 u + \underline{b} \cdot \underline{\nabla} u = f(\underline{x}) \ . \tag{6}$$

Provided $\varepsilon > 0$ this is an elliptic problem but, when $\underline{b} \neq \underline{0}$, it is not self-adjoint. When ε is small (relative to $||\underline{b}||$) the equation is said to be convection-dominated. Such problems arise frequently in fluid mechanics, heat and mass transfer, environmental modelling, etc. and, when discretized by the standard Galerkin FEM (see, for example, [11]), lead to a non-symmetric linear system.

When considering how to generalize the preconditioner introduced in the previous section to non-symmetric systems an important clue may be obtained from observations made in [3] and [5]. Both of these papers make the empirical observation that setting the M_i^T terms in (5) to zero (and scaling the interface terms accordingly) not only has the effect of reducing the communication cost of each iteration but, provided an appropriate solver is used, also leads to a reduction in the number of iterations required to converge[‡]. In the case where $p = 2$ equation (5) therefore becomes

$$\left[\begin{array}{c} \underline{z}_1 \\ \underline{z}_2 \\ \underline{z}_s \end{array} \right] = \left[\begin{array}{c} \underline{z}_{1,1} \\ \underline{z}_{2,2} \\ \frac{1}{2}(\underline{z}_{s,1} + \underline{z}_{s,2}) \end{array} \right] , \tag{7}$$

[‡]In [5] the observation is of course described for conventional AS preconditioning rather than the weakly overlapping modification being considered here.

which means that the preconditioner is no longer SPD. Hence, even for a self-adjoint differential operator, the CG algorithm can no longer be used and must be replaced by a more general alternative such as QMR or GMRES for example (see [1], [8] or [15] for details). In this work we use the GMRES algorithm, as implemented in [14]. Although the cost per iteration is greater for GMRES than CG, the reduced inter-processor communication at each iteration and the decrease in the number of iterations required always appear to more than make up for this (see [3] for specific comparisons in 2-d and [5] for corresponding remarks concerning conventional AS preconditioning).

With a parallel preconditioner for GMRES given by (3), (4) and (7) (still using the special case $p = 2$ for simplicity) it is clear that our algorithm may easily generalize to non-symmetric problems such as that obtained from a finite element discretization of (6).

2.1. Parallel implementation

Generalizing the above discussion to the solution of a non-symmetric linear system on p processors we may write the action ($\underline{z} = \tilde{A}^{-1}\underline{r}$) of the preconditioner in terms of the computations required on each processor i (from 1 to p) as follows.
(i) Solve

$$\begin{bmatrix} A_i & 0 & B_i \\ 0 & \bar{A}_i & \bar{B}_i \\ C_i & \bar{C}_i & A_{i,s} \end{bmatrix} \begin{bmatrix} \underline{z}_i \\ \underline{\bar{z}}_i \\ \underline{z}_{i,s} \end{bmatrix} = \begin{bmatrix} \underline{r}_i \\ \bar{M}_i \underline{\bar{r}}_i \\ \underline{r}_{i,s} \end{bmatrix} . \tag{8}$$

(ii) Average each entry of of $\underline{z}_{i,s}$ over all corresponding entries on neighbouring processors.

In (8) A_i, B_i and C_i are assembled components of the stiffness matrix for the elements of the mesh in subdomain i, $\bar{A}_i$, $\bar{B}_i$ and $\bar{C}_i$ are components for the elements of this mesh outside subdomain i and $A_{i,s}$ stores the components of the stiffness matrix where both the row and column correspond to nodes on the interface of subdomain i. A similar partition of the vector $\underline{r}$ is provided into components $\underline{r}_i$ inside subdomain i, $\underline{\bar{r}}_i$ outside subdomain i and $\underline{r}_{i,s}$ on its interface. $\bar{M}_i$ represents the hierarchical restriction operator from the fine mesh on each processor other than i to the coarser mesh outside of subdomain i on processor i (therefore requiring an all-to-one communication to compute its action for each i).

The main implementation issue that needs to be addressed is that of computing the action of each of the restriction operations $\bar{M}_i\underline{\bar{r}}_i$ efficiently at each iteration. Note that because the preconditioner is not symmetric we do not need to evaluate the corresponding prolongation at the end of each preconditioning step. The evaluation of $\bar{M}_i\underline{\bar{r}}_i$ is completed in two phases: a set-up phase which occurs before the first iteration, and a communication phase which occurs at each iteration. All of our implementations have been in ANSI C using the MPI communication library, [12].

In the set-up phase each processor, i say, sends to each other processor, j say, a list of the nodes of mesh i which lie in, or on the boundary of, subdomain j. Processor i then receives from each other processor, j say, a list of all nodes of mesh j which lie in, or on the boundary of, subdomain i. For each of these lists processor i then matches each of the nodes in this list with the corresponding node on mesh i. This is achieved very efficiently by using the mesh hierarchy that is present on processor i (see [18] for a description of the hierarchical refinement of tetrahedral grids that is used on each processor).

At each iteration processor i then contributes to the restriction operation $\bar{M}_j \underline{\bar{r}}_j$ for each $j \neq i$. The part of the vector $\underline{r}$ which is stored on processor i ($\underline{r}_i$) corresponds to all nodes of mesh i in subdomain i or on its boundary. For each j this sub-vector, $\underline{r}_i$, may be restricted to the nodes of mesh j which lie in subdomain i or on its boundary (which are known from the set-up phase). This restriction uses the mesh hierarchy in the standard multilevel manner (as described in [13] for example). These restricted vectors may then each be sent to the corresponding processor, j. Following this, processor i should receive a list of its own restricted vectors from each of the other processors. These are then put together on processor i to produce the required vector $\bar{M}_i \underline{\bar{r}}_i$ before the solution to (8) is found locally. Note that in MPI ([12]) all of the above message passing may be implemented as a single all-to-all communication. Given the high cost of such a communication we again see the value of only requiring one of these per iteration (as opposed to two for the original symmetric preconditioner).

The final stage in computing the action of $\underline{z} = \tilde{A}^{-1} \underline{r}$ requires only neighbour-to-neighbour communication between processors sharing a subdomain boundary. This allows the $\underline{z}_{i,s}$ vectors to be updated on each processor i, as required for step (ii) above.

2.2. Numerical results

In order to assess the performance of the proposed parallel preconditioner on typical convection-diffusion equations we consider here two specific test problems of the form (6). In both cases the equation is solved on the domain $\Omega = (0,2) \times (0,1) \times (0,1)$ with $\underline{b} = (1,0,0)^T$.

Problem 1

$$f(\underline{x}) = 0 \ ,$$

subject to the Dirichlet boundary condition $u|_{\partial\Omega} = u^*|_{\partial\Omega}$, where

$$u^* = \frac{e^{-x/\varepsilon} - 1}{e^{-x/\varepsilon} - e^{(2-x)/\varepsilon}}$$

is the exact solution of (6) with this choice of f.

Problem 2

$$f(\underline{x}) = 2\varepsilon \left(x - \frac{2(1 - e^{x/\varepsilon})}{(1 - e^{2/\varepsilon})} \right) (y(1-y) + z(1-z)) + y(1-y)z(1-z) \ ,$$

subject to the Dirichlet boundary condition $u|_{\partial\Omega} = u^*|_{\partial\Omega}$, where

$$u^* = \left(x - \frac{2(1 - e^{x/\varepsilon})}{(1 - e^{2/\varepsilon})} \right) y(1-y)z(1-z)$$

is the exact solution of (6) with this choice of f.

Table I. The performance of the proposed algorithm on Problem 1 for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
6144	4	4	5	6	11	14	15	31
49152	2	3	3	4	6	7	4	4
393216	1	2	3	3	3	4	4	5
3145728	2	2	3	3	1	3	6	6

Table II. The performance of the proposed algorithm on Problem 2 for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
6144	4	4	5	8	11	15	16	32
49152	4	4	5	6	7	8	8	12
393216	3	4	5	7	6	6	6	9
3145728	3	5	6	8	5	5	5	8

In both cases the exact solutions, u^*, exhibit steep layers of width $O(\varepsilon)$ near to the boundary $x = 2$ when $0 < \varepsilon << ||\underline{b}|| = 1$.

Tables I and II show the number of iterations required to solve the Galerkin finite element discretizations of the above problems to a moderately high level of accuracy using GMRES with our parallel implementation of the weakly overlapping DD preconditioner. Results are presented for a sequence of meshes which represent between one and four levels of refinement of a coarse tetrahedral mesh containing 768 elements. At each level of refinement each tetrahedron is subdivided into eight children, as described in [18]. It may be observed that, as the mesh is refined or the number of processors (subdomains) is increased, the total number of GMRES iterations appears to be bounded. Such a result is proved in [4] but only for the symmetric version of the preconditioner applied to certain SPD problems.

It is interesting to note that for both Tables I and II more iterations are required on the coarse grids, especially when $\varepsilon = 10^{-3}$. This is a highly convection-dominated problem for which the Galerkin method is known to be unstable when the grid is not sufficiently fine. For this reason we consider the application of our preconditioner with a stabilized finite element discretization in the following section. When the finite element grid is refined we note that the preconditioner is still effective with the Galerkin discretization however. The results in Table I appear to be slightly better than those for the second test problem in Table II. This may be due to the fact that the first test problem is essentially just a one-dimensional problem and so is somewhat easier to solve in practice. It should also be noted that the precise iteration

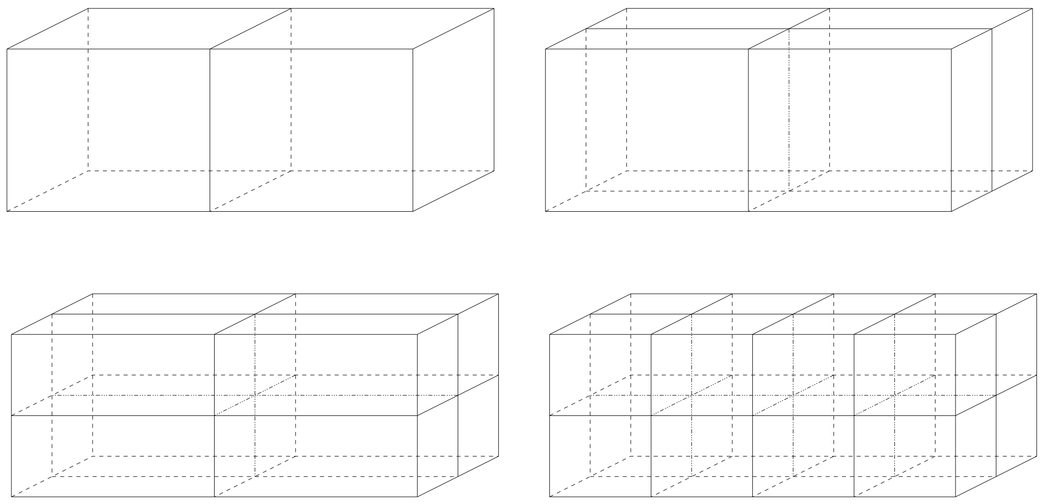
Figure 2. An illustration of the partitioning strategy, based upon RCB, used to obtain 2, 4, 8 and 16 subdomains, where $\Omega = (0,2) \times (0,1) \times (0,1)$.

counts that are obtained depend upon the specific decomposition that is made of the problem domain. For the results presented here the partitions illustrated in Figure 2 are used. These are derived from the use of a simple recursive coordinate bisection (RCB) algorithm, [17], that has the effect of yielding subdomains with a low surface-area to volume ratio.

3. STABILIZED FINITE ELEMENTS FOR CONVECTION-DOMINATED PROBLEMS

The examples of the previous section suggest that the proposed weakly overlapping parallel DD preconditioner works well with a Galerkin finite element discretization of a three-dimensional convection-diffusion problem provided the mesh is sufficiently fine. When the problems become convection-dominated however (i.e. $0 < \varepsilon << ||\underline{b}||$) it is well-known that the elements of the mesh must be very small (i.e. $O(\frac{\varepsilon}{||\underline{b}||})$) in order to prevent the Galerkin solution from becoming oscillatory. In this section we therefore extend our consideration to the solution of convection-dominated problems using a more stable finite element technique based upon streamline-diffusion (see [11], for example, for a full discussion of oscillations and stabilization using streamline-diffusion).

3.1. The streamline-diffusion method

The standard FEM discretization of (6) on a domain Ω seeks an approximation u_h to u from a finite element space $\mathcal{S}_h$ such that

$$\varepsilon \int_\Omega \underline{\nabla} u_h \cdot \underline{\nabla} v \; d\underline{x} + \int_\Omega (\underline{b} \cdot \underline{\nabla} u_h) v \; d\underline{x} = \int_\Omega f(\underline{x}) v \; d\underline{x} \tag{9}$$

for all $v \in \mathcal{S}_h$ (disregarding boundary conditions for simplicity). This in turn yields a non-symmetric linear algebraic system, $A\underline{u} = \underline{f}$, for which a typical entry in A is

$$\varepsilon \int_\Omega \underline{\nabla} \phi_i \cdot \underline{\nabla} \phi_j \; d\underline{x} + \int_\Omega \phi_i (\underline{b} \cdot \underline{\nabla} \phi_j) \; d\underline{x} \; , \tag{10}$$

where $\{\phi_i\}$ is the set of finite element basis functions for $\mathcal{S}_h$.

The streamline-diffusion approach replaces v in (9) by $v + \alpha \underline{b} \cdot \underline{\nabla} v$ to yield

$$\begin{aligned} \varepsilon \int_\Omega \underline{\nabla} u_h \cdot \underline{\nabla}(v + \alpha \underline{b} \cdot \underline{\nabla} v) \; d\underline{x} + \int_\Omega (\underline{b} \cdot \underline{\nabla} u_h)(v + \alpha \underline{b} \cdot \underline{\nabla} v) \; d\underline{x} \quad &= \\ \int_\Omega f(\underline{x})(v + \alpha \underline{b} \cdot \underline{\nabla} v) \; d\underline{x} & \end{aligned} \tag{11}$$

for all $v \in \mathcal{S}_h$.

When $\alpha = 0$ the resulting linear system has a matrix with entries still given by (10) however α is usually chosen to be greater than zero and proportional to the mesh size h. This means that the linear system now being solved is even further from the SPD system analyzed in [4]. Nevertheless, it is possible to apply the same weakly overlapping domain decomposition preconditioning strategy to this stabilized problem. This requires only minor modifications to the code used to produce the results of the previous section (corresponding to the differences between (9) and (11)). The following results demonstrate that this extension also works well in practice.

3.2. Numerical Results

In Tables III and IV we illustrate the improved accuracy of the streamline-diffusion method when Problem 1 and Problem 2 are solved with $\varepsilon = 10^{-2}$. For these convection-dominated calculations we present the infinity norm of the exact error when using the Galerkin and the streamline-diffusion discretizations respectively. As expected, we see that the stabilized FEM provides a less oscillatory solution with a smaller error in each case. If we were to continue to refine the mesh, or solve the same problems with a larger value of ε, the improvement of streamline-diffusion over the Galerkin FEM would be eroded. Conversely, when corresponding results are calculated for $\varepsilon = 10^{-3}$ the relative error in the Galerkin solution is even greater on these meshes, which are far coarser than $O(\frac{\varepsilon}{\|\underline{b}\|})$. In all of the streamline-diffusion calculations α in (11) is taken as $\frac{h}{2}$: it is likely that better choices could be obtained with some fine tuning however.

Next, in Tables V and VI, we provide results corresponding to those given for the Galerkin equations in Tables I and II. From these tables it is clear that, not only does the stabilized

Table III. The infinity norm of the exact error in the two finite element approximations to the solution of Problem 1 when $\varepsilon = 10^{-2}$.

Elements	Error 1 (Galerkin FEM)	Error 2 (stabilized FEM)	$\frac{\text{Error 2}}{\text{Error 1}}$
6144	1.07×10^{0}	6.66×10^{-1}	0.622
49152	9.10×10^{-1}	5.65×10^{-1}	0.620
393216	4.41×10^{-1}	2.57×10^{-1}	0.583
3145728	1.55×10^{-1}	6.70×10^{-2}	0.433

Table IV. The infinity norm of the exact error in the two finite element approximations to the solution of Problem 2 when $\varepsilon = 10^{-2}$.

Elements	Error 1 (Galerkin FEM)	Error 2 (stabilized FEM)	$\frac{\text{Error 2}}{\text{Error 1}}$
6144	1.02×10^{-1}	6.80×10^{-2}	0.667
49152	1.01×10^{-1}	6.24×10^{-2}	0.618
393216	5.22×10^{-2}	2.95×10^{-2}	0.565
3145728	1.88×10^{-2}	8.10×10^{-3}	0.431

Table V. The performance of the proposed algorithm on the stabilized discretization of Problem 1 for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
6144	3	3	3	4	3	4	4	5
49152	2	2	3	3	2	2	3	3
393216	2	2	3	3	2	2	3	3
3145728	1	3	3	3	1	2	2	2

discretization lead to more accurate results but that our weakly overlapping DD preconditioner yields faster convergence than before, especially on the coarser grids. As with Tables I and II we again see that Problem 1 is more straightforward to solve than Problem 2. Once more we have used a partition into subdomains based upon the RCB approach illustrated in Figure 2 and we note that the precise iteration count will change if a different partition is used.

3.3. Local refinement

Having demonstrated the effectiveness of the proposed preconditioner for convection-dominated problems solved using a stabilized FE method on a sequence of uniformly refined

Table VI. The performance of the proposed algorithm on the stabilized discretization of Problem 2 for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
6144	3	4	4	6	5	5	5	7
49152	3	4	4	6	4	5	5	7
393216	3	4	5	7	4	5	5	6
3145728	3	4	6	8	3	4	5	7

Table VII. The performance of the proposed algorithm on the stabilized discretization of Problem 1 using locally refined grids for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
2560	4	5	5	6	5	6	6	6
9728	4	5	5	6	4	5	5	6
38400	4	5	5	6	4	5	5	6
153088	5	6	7	7	4	6	6	7

grids, we now consider the use of local mesh refinement. In practice, problems such as those under consideration in this paper have solutions which do not require a high mesh density everywhere but only in certain regions, such as where boundary layers or shocks occur for example. We now apply a simple *a priori* mesh refinement strategy with the streamline-diffusion discretization. The purpose of this is to illustrate the potential for the DD preconditioner given by (7) on each processor to be used successfully within an adaptive finite element framework. For the particular test problems being considered here we are able to make use of the fact that the only boundary layers in the solutions are known to be next to the boundary $x = 2$, and that the solutions are smooth elsewhere. Hence, beginning with a coarse mesh of 768 elements, we are able to get results of almost identical accuracy to those obtained using uniform mesh refinement by applying local mesh refinement to the same number of levels: only refining elements in the neighbourhood of $x = 2$ at each level. Iteration counts for the DD preconditioner with this mesh refinement strategy are shown in Tables VII and VIII for Problem 1 and Problem 2 respectively.

Comparison of these results with those provided in Tables V and VI show that the use of local refinement leads to a slight increase in the number of iterations required however this still appears to be bounded independently of h and p. Furthermore, for reasons of load-balancing that are discussed in the next section, on parallel performance, different partitions of the domain have been used here from those used with the uniformly refined grids. These

Table VIII. The performance of the proposed algorithm on the stabilized discretization of Problem 2 using locally refined grids for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
2560	4	5	5	6	5	6	6	6
9728	5	5	6	6	5	6	6	6
38400	5	6	6	7	5	6	7	7
153088	6	8	8	8	6	7	7	7

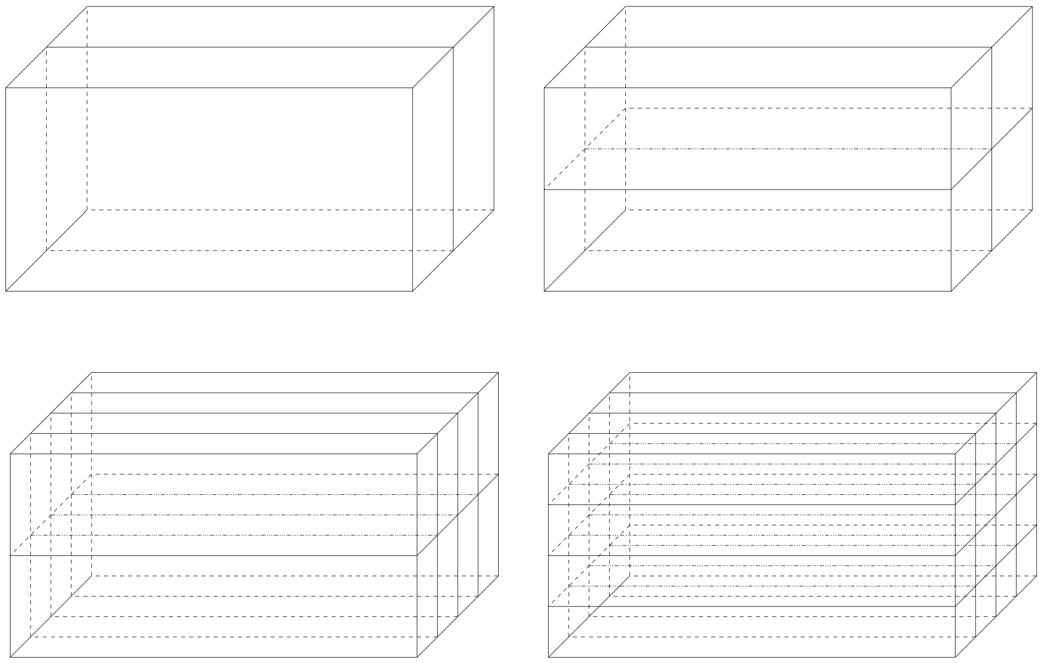
Figure 3. An illustration of the partitioning strategy used to obtain 2, 4, 8 and 16 subdomains when local mesh refinement is undertaken with $\Omega = (0, 2) \times (0, 1) \times (0, 1)$.

new partitions are illustrated in Figure 3 and are such that each subdomain contains an approximately equal proportion of the boundary layer, where most of the elements are found after refinement.

Table IX. Timings for the solution of Problem 1 on the final level globally refined mesh

$\varepsilon = 1.0 \times 10^{-2}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	542.06	323.15	202.34	123.24	70.16
Speedup	–	1.7	2.7	4.4	7.7
Sequential Time	–	640.76	785.34	939.99	989.18
Parallel Speedup	–	2.0	3.9	7.6	14.1

$\varepsilon = 1.0 \times 10^{-3}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	559.70	323.35	196.62	114.20	65.74
Speedup	–	1.7	2.8	4.9	8.5
Sequential Time	–	634.08	760.56	868.29	941.58
Parallel Speedup	–	2.0	3.9	7.6	14.3

4. PARALLEL PERFORMANCE

The calculations described in all of the above tables of iteration counts were performed on a SG Origin 2000 computer which has a non-uniform memory access (NUMA) architecture. The non-uniform nature of the memory access means that timings of a given calculation may vary significantly between runs depending upon how memory has been allocated. For this reason all of the timings quoted in this section represent the best time that was achieved over numerous repetitions of the same computation. Furthermore, there are numerous parameters within the algorithm that affect the overall performance, such as the accuracy to which the systems (8) are solved on each processor at each iteration (for best performance these should only be solved approximately), or the drop tolerance that is used in the sequential ILU preconditioner ([14]) that is used for these systems. Our choices for these parameters, determined empirically to yield the best timings, are still unlikely to be completely optimal.

In Tables IX and X we present parallel timings in seconds for the solution of Problem 1 and Problem 2 respectively, with $\varepsilon = 10^{-2}$ and 10^{-3} on the grid of 3145728 elements obtained using uniform mesh refinement. Corresponding results are also provided in Tables XI and XII for the mesh of 153088 elements obtained using local refinement. In all of these tables we include not only the parallel solution times but also the sequential solution times for different choices of p. We also present two rows of speed-up figures in each table: a regular speed-up which contrasts the parallel solution time with the best sequential solution time, and a parallel speed-up which contrasts the parallel solution time with the sequential solution time of the p subdomain DD solver.

There are at least three factors which affect the parallel performance of our weakly overlapping algorithm. These are the quality of the p subdomain preconditioner contrasted with the best available sequential solver (for which we use [14]), the amount of load imbalance that exists between the processors, and the parallel overheads associated with interprocessor communications.

Table X. Timings for the solution of Problem 2 on the final level globally refined mesh

$\varepsilon = 1.0 \times 10^{-2}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	770.65	497.39	341.78	227.98	113.81
Speedup	–	1.5	2.3	3.4	6.8
Sequential Time	–	984.72	1326.91	1725.23	1597.15
Parallel Speedup	–	2.0	3.9	7.6	14.0

$\varepsilon = 1.0 \times 10^{-3}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	668.12	431.68	271.66	175.48	98.93
Speedup	–	1.5	2.5	3.8	6.8
Sequential Time	–	854.68	1062.95	1342.11	1398.79
Parallel Speedup	–	2.0	3.9	7.6	14.1

Table XI. Timings for the solution of Problem 1 on the final level locally refined mesh

$\varepsilon = 1.0 \times 10^{-2}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	29.19	14.61	8.73	6.30	4.80
Speedup	–	2.0	3.3	4.6	6.1
Sequential Time	–	28.90	33.63	43.72	59.84
Parallel Speedup	–	2.0	3.9	6.9	12.5

$\varepsilon = 1.0 \times 10^{-3}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	20.07	14.02	8.62	6.02	4.87
Speedup	–	1.4	2.3	3.3	4.1
Sequential Time	–	27.64	33.12	42.05	58.88
Parallel Speedup	–	2.0	3.8	7.0	12.1

If we focus initially on the results on the uniformly refined mesh (Tables IX and X) it is quite clear that it is the quality of the preconditioner itself which is the biggest constraint on its efficiency. This is seen not only from the sequential times taken for different choices of p compared to the best sequential time (used for $p = 1$), but also from the fact that the parallel speed-ups are very good compared to the raw speed-ups. Recall that the parallel speed-up is the ratio of the sequential and parallel times for the p subdomain algorithm. It may be possible to improve the basic quality of the p subdomain algorithm by using a better sequential solver (e.g. based upon multigrid) for each subdomain problem, or by optimizing the level of accuracy to which these problems are solved at each iteration. This is an area that undoubtedly requires further investigation.

Table XII. Timings for the solution of Problem 2 on the final level locally refined mesh

$\varepsilon = 1.0 \times 10^{-2}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	26.20	17.54	10.54	7.17	5.35
Speedup	–	1.5	2.9	3.7	4.9
Sequential Time	–	34.64	40.54	50.35	66.96
Parallel Speedup	–	2.0	3.8	7.0	12.5

$\varepsilon = 1.0 \times 10^{-3}$	p=1	p=2	p=4	p=8	p=16
Parallel Time	25.63	17.18	10.03	6.83	5.15
Speedup	–	1.5	2.6	3.8	4.0
Sequential Time	–	33.94	38.75	47.91	63.73
Parallel Speedup	–	2.0	3.9	7.0	12.4

Again considering Tables IX and X it is also clear that load balance plays a more important role in determining efficiency than the communication costs. Note that because there is a single layer of overlap between the subdomains at each level of the mesh hierarchy, the total size of each subdomain problem depends upon the size of the interface of that subdomain with its neighbours. From Figure 2 it is clear that when $p = 2$, 4 or 8 the subdomain problems will be of the same size as each other (but dependent upon p of course). When $p = 16$ however this will not be the case since the 8 subdomains in the middle section of the domain have a larger interface with their neighbours than the other 8 subdomains. Hence these will contain more overlapping elements. This leads to the local problems corresponding to these subdomains each containing approximately 307000 elements, as opposed to just 276000 elements in each of the remainder. The effect of this on the parallel speed-up is quite noticeable since this ranges between 14.0 and 14.3 out of 16 over the four tables. By contrast, on 8 processors the parallel speed-up is about 7.6 in each case: a significantly better efficiency. The small loss of efficiency that is observed in these latter cases (and for $p = 2$ and 4) may be attributed mainly to the communication overheads that are present in forming the right-hand side of (8).

If we now consider Tables XI and XII a similar pattern emerges. The first thing to note however is that the solution times are, as expected, significantly reduced as a result of local, rather than global, refinement. On 16 processors, for example, results of the same accuracy as before are now obtained in between 4.80 and 5.35 seconds. It is also clear that the major factor affecting efficiency is again the sequential time of the p subdomain algorithm compared to the best sequential time that we could achieve for each problem.

The parallel speed-ups for these locally refined problems are still quite good, however load balancing is a much more significant issue here than when the mesh is refined globally. Because we have selected a mesh refinement strategy *a priori* we are able to simplify this load-balancing problem considerably by partitioning the domain as shown in Figure 3. Since the refinement takes place near to the right end boundary we expect a similar number of elements in each subdomain for each particular value of p. However, there is still the further complication of the single layer of overlapping elements at each level of the mesh refinement hierarchy, which causes the load imbalance to grow when $p = 8$ or 16, as may be expected from inspection of Figure 3. This explains why the parallel speed-up falls to 7.0 or below on 8 processors and 12.1 to 12.5 on 16. In the former case the typical number of elements per processor ranges from 27000 to 33000, and in the latter from 15000 to 22000. Given the significant nature of this load imbalance it is clear that the communication overheads are still relatively small when local refinement has been used.

5. DISCUSSION

In this paper we have described the parallel implementation of a weakly overlapping domain decomposition preconditioner and its successfully application to the finite element solution of convection-dominated PDEs in three dimensions. It is demonstrated that excellent convergence rates may be achieved using Galerkin and streamline-diffusion FE discretizations and that the iteration counts appear to be bounded independently of h and p. Furthermore, the algorithm is shown to be similarly effective when local, rather than global, mesh refinement is undertaken. In each of these cases it is demonstrated that very good parallel speed-ups may be obtained however further work is clearly required to improve the underlying speed of the p subdomain algorithm.

As with all parallel algorithms a significant issue is shown to be that of load balance. Even for a uniformly refined grid, subdomains of an equal volume may have local problems of different sizes if they have different numbers of elements overlapping with neighbours. This difficulty becomes more severe when local refinement is undertaken, especially if this were to be within the framework of an adaptive FE strategy for which the refinement pattern would not be known *a priori*. Further research is also required therefore to determine the best strategy for overcoming these difficulties. In [2] it is suggested that it may be possible to use an error estimate on the coarse initial mesh to partition this mesh appropriately at the start of the parallel calculation. Where this type of approach is unsuccessful however it is likely that use will need to be made of dynamic load-balancing techniques, as described in [19, 20] for example.

The specific test problems considered in this paper have a fixed convection direction $\underline{b}$. The particular choice of $\underline{b}$ appears to be of little importance (apart from the implications on load balancing when local refinement is used of course), however we have yet to consider non-constant convection directions, or even nonlinear problems for which $\underline{b}$ is a function of u. The work should also be extended to convection-dominated *systems* of equations, which also arise frequently in scientific modeling.

REFERENCES

1. Ashby, S.F., Manteuffel, T.A., Taylor, P.E.: A taxonomy for conjugate gradient methods. *SIAM J. on Numer. Anal.* **27** (1990) 1542–1568.
2. Bank, R.E., Holst, M.: A new paradigm for parallel adaptive meshing algorithms. *SIAM J. on Sci. Comp.* **22** (2000) 1411–1443.

3. Bank, R.E., Jimack, P.K.: A new parallel domain decomposition method for the adaptive finite element solution of elliptic partial differential equations. *Concurrency and Computation: Practice and Experience* **13** (2001) 327–350.
4. Bank, R.E., Jimack, P.K., Nadeem, S.A., Nepomnyaschikh, S.V.: A weakly overlapping domain decomposition preconditioner for the finite element solution of elliptic partial differential equations. To appear in *SIAM J. on Sci. Comp.* (2002).
5. Cai, X.-C., Sarkis, M.: A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM J. on Sci. Comp.* **21** (1999) 792–797.
6. Chan, T., Mathew, T.: Domain decomposition algorithms. *Acta Numerica* **3** (1994) 61–143.
7. Farhat, C., Mandel, J., Roux, F.X.: Optimal convergence properties of the FETI domain decomposition method. *Comp. Meth. for Appl. Mech. and Eng.* **115** (1994) 365–385.
8. Freund, R.W., Nachtigal, N.M.: QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, **60** (1991) 315–339.
9. Gropp, W.D., Keyes, D.E.: Parallel performance of domain-decomposed preconditioned Krylov methods for PDEs with locally uniform refinement. *SIAM J. on Sci. Comp.* **13** (1992) 128–145.
10. Hodgson, D.C., Jimack, P.K.: A domain decomposition preconditioner for a parallel finite element solver on distributed unstructured grids. *Parallel Computing* **23** (1997) 1157–1181.
11. Johnson, C.: *Numerical Solutions of Partial Differential Equations by the Finite Element Method.* Cambridge University Press (1987).
12. Message Passing Interface Forum: MPI: A message-passing interface standard. *Int. J. Supercomputer Appl.* **8** (1994) no. 3/4.
13. Oswald, P.: *Multilevel Finite Element Approximation: Theory and Applications.* Teubner Skripten zur Numerik, B.G. Teubner (1994).
14. Saad, Y.: SPARSEKIT: A basic tool kit for sparse matrix computations, version 2. Technical Report, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, IL, USA (1994).
15. Saad, Y., Schultz, M.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. on Sci. Comp.* **7** (1986) 856–869.
16. Smith, B., Bjorstad, P., Gropp, W.: *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press (1996).
17. Simon, H.D.: Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering* **2** (1991) 135–148.
18. Speares, W., Berzins, M.: A 3-d unstructured mesh adaptation algorithm for time-dependent shock dominated problems. *Int. J. for Numer. Meth. in Fluids* **25** (1997) 81–104.
19. Touheed, N, Selwood, P, Jimack, P.K., Berzins, M.: A comparison of some dynamic load-balancing algorithms for a parallel adaptive flow solver. *Parallel Computing* **26** (2000) 1535–1554.
20. R.D. Williams: Performance of dynamic load balancing for unstructured mesh calculations. *Concurrency: Practice and Experience* **3** (1991) 457–481.