

# Parallel Application of a Novel Domain Decomposition Preconditioner for the Stable Finite Element Solution of Three-Dimensional Convection-Dominated PDEs

Peter K. Jimack and Sarfraz A. Nadeem

Computational PDEs Unit, School of Computing,  
University of Leeds, Leeds, LS2 9JT, UK  
{pkj, sarfraz}@comp.leeds.ac.uk  
<http://www.comp.leeds.ac.uk/pkj/>

**Abstract.** 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, [3], 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), [9].

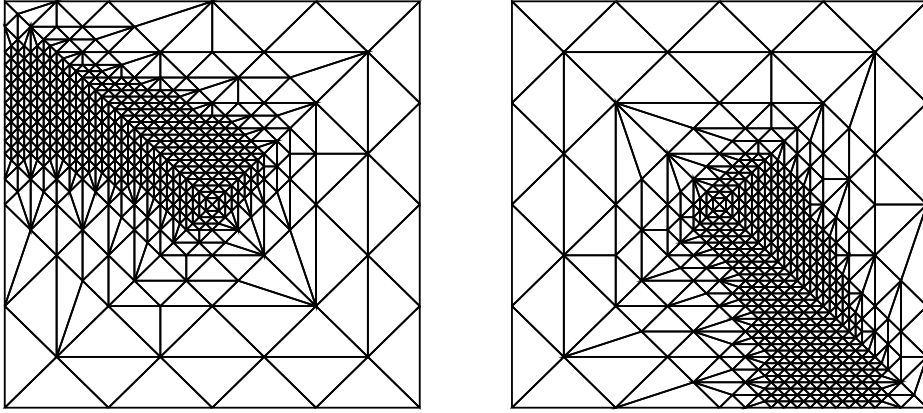
## 1 Introduction

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. [6, 8], which each require exact subdomain solves at each iteration) or solving the complete system as a partitioned matrix (e.g. [5, 7]). In this work we focus on a recently proposed method of the second type ([2, 3]) which we refer to as a weakly overlapping additive Schwarz (AS) preconditioner.

Typical AS preconditioners (see, for example, [13]) 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  $h^1$ . For practical applications therefore this optimality property is usually discarded in favour of keeping a fixed number of

---

<sup>1</sup> It is also necessary to add the solution of a restricted coarse grid problem at each iteration for such an optimal preconditioner (again see [13], or [5]).



**Fig. 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.

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  $h \rightarrow 0$ ). In [2] 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 [3] 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 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. [8]), 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:

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

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

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

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, [7]).

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

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

and on processor 2 solve the system

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

then set

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

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, [11]) 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 [2] or [3] 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 parallel codes (e.g. [8]) solve this problem sequentially on a single processor anyway. Furthermore, as  $h \rightarrow 0$ , 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 [3] 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

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

Provided  $\varepsilon > 0$  this is an elliptic problem but, when  $\underline{b} \neq \underline{0}$ , it is not self-adjoint. When  $\varepsilon$  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, [9]), 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 [2] and [4]. 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<sup>2</sup>. In the case where  $p = 2$  equation (5) therefore becomes

$$\begin{bmatrix} \underline{z}_1 \\ \underline{z}_2 \\ \underline{z}_s \end{bmatrix} = \begin{bmatrix} \underline{z}_{1,1} \\ \underline{z}_{2,2} \\ \frac{1}{2}(\underline{z}_{s,1} + \underline{z}_{s,2}) \end{bmatrix} , \quad (7)$$

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 GMRES (again see [1], or [12] for details of a public domain implementation). 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 [2] for specific comparisons in 2-d and [4] for corresponding remarks concerning conventional AS preconditioning).

With a parallel preconditioner for GMRES given by (3), (4) and (7) (we again focus on 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 ( $\underline{z} = \tilde{A}^{-1} \underline{p}$ ) of the preconditioner in

<sup>2</sup> In [4] the observation is of course described for conventional AS preconditioning rather than the weakly overlapping modification being considered here.

terms of the computations required on each processor  $i$  (from 1 to  $p$ ) as follows.

(i) Solve

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

(ii) Average each entry 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  $\underline{p}$  is provided into components  $\underline{p}_i$  inside subdomain  $i$ ,  $\bar{\underline{p}}_i$  outside subdomain  $i$  and  $\underline{p}_{i,s}$  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  $\bar{M}_i \bar{\underline{p}}_i$  efficiently at each iteration. Note that because the preconditioner is not symmetric we do not need to also evaluate the corresponding prolongation at the end of each preconditioning step. The evaluation of  $\bar{M}_i \bar{\underline{p}}_i$  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, [10].

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 [14] 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  $\bar{M}_j \bar{\underline{p}}_j$  for each  $j \neq i$ . The part of the vector  $\underline{p}$  which is stored on processor  $i$  ( $\underline{p}_i$ ) corresponds to all nodes of mesh  $i$  in subdomain  $i$  or on its boundary. For each  $j$  this sub-vector,  $\underline{p}_i$ , 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 [11] 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  $\bar{M}_i \bar{\underline{p}}_i$  before the solution to (8) is found locally. Note that in MPI ([10]) 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  $\underline{z} = \underline{A}^{-1}\underline{p}$  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 problems we consider here a specific test problem of the form (6). This equation is solved on the domain  $\Omega = (0, 2) \times (0, 1) \times (0, 1)$  with the parameters  $\underline{b} = (1, 0, 0)^T$  and

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

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

$$u^* = \left( x - \frac{2(1 - e^{x/\varepsilon})}{(1 - e^{2/\varepsilon})} \right) y(1 - y)z(1 - z) \quad (9)$$

is the exact solution to this problem. It exhibits a steep layer of size  $O(\varepsilon)$  near to the boundary  $x = 2$  when  $0 < \varepsilon \ll \|\underline{b}\| = 1$ .

Table 1 shows the number of iterations required to solve the discrete finite element system 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 [14]. It may be observed that, as the mesh is refined or the number of processors (subdomains) is increased, the total number of GMRES iterations increases only very slowly. It is not clear however whether this increase will be bounded as the mesh size tends to zero (as is proved in [3] for the symmetric version).

It is also evident from Table 1 that fewer iterations are required to solve the test problem when  $\varepsilon = 10^{-2}$  than when  $\varepsilon = 10^{-1}$ . A similar observation is made when solving the problem sequentially using the package described in [12]. (This package is also used in the parallel preconditioner to solve the system (8) sequentially on each processor.) In this case a preconditioner based upon an incomplete LU (ILU) factorization of the finite element matrix is used, and this is able to exploit the weak coupling between neighbouring nodes with a common edge that is perpendicular to  $\underline{b}$  when the problem is convection-dominated. Hence, although the discrete problem is more non-symmetric in this case, it turns out to be less complex to solve in practice. A discussion of the timings for these calculations follows in the next section.

**Table 1.** The performance of the proposed algorithm on the convection-diffusion test problem for two choices of  $\varepsilon$ : figures quoted represent the number of iterations required to reduce the initial residual by a factor of  $10^5$ .

Elements/Procs.	$\varepsilon = 10^{-1}$				$\varepsilon = 10^{-2}$			
	2	4	8	16	2	4	8	16
6144	3	3	4	5	3	3	4	4
49152	3	4	6	7	3	3	4	4
393216	4	6	7	8	3	4	4	5
3145728	4	7	9	10	4	5	6	6

### 3 Stabilized Finite Elements for Convection-Dominated Problems

The example of the previous section suggests that the proposed weakly overlapping parallel DD preconditioner works well in practice for three-dimensional convection-diffusion problems. When these become convection-dominated however (i.e.  $0 < \varepsilon \ll \|\underline{b}\|$ ) it is well-known that the standard Galerkin FEM becomes oscillatory unless the size of the elements is sufficiently small. 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 [9], 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  $\Omega$  seeks an approximation  $u_h$  to  $u$  from a finite element space  $\mathcal{S}_h$  such that

$$\varepsilon \int_{\Omega} \underline{\nabla} u_h \cdot \underline{\nabla} v \, d\underline{x} + \int_{\Omega} (\underline{b} \cdot \underline{\nabla} u_h) v \, d\underline{x} = \int_{\Omega} f(\underline{x}) v \, d\underline{x} \quad (10)$$

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

$$\varepsilon \int_{\Omega} \underline{\nabla} \phi_i \cdot \underline{\nabla} \phi_j \, d\underline{x} + \int_{\Omega} \phi_i (\underline{b} \cdot \underline{\nabla} \phi_j) \, d\underline{x}, \quad (11)$$

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

The streamline-diffusion approach replaces  $v$  in (10) by  $v + \alpha \underline{b} \cdot \underline{\nabla} v$  to yield

$$\varepsilon \int_{\Omega} \underline{\nabla} u_h \cdot \underline{\nabla} (v + \alpha \underline{b} \cdot \underline{\nabla} v) \, d\underline{x} + \int_{\Omega} (\underline{b} \cdot \underline{\nabla} u_h) (v + \alpha \underline{b} \cdot \underline{\nabla} v) \, d\underline{x} = \int_{\Omega} f(\underline{x}) (v + \alpha \underline{b} \cdot \underline{\nabla} v) \, d\underline{x} \quad (12)$$

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

**Table 2.** The infinity norm of the exact error in the two finite element approximations to (9), the solution of (6), when  $\varepsilon = 10^{-2}$ .

Elements	Error 1 (Galerkin FEM)	Error 2 (stabilized FEM)	$\frac{\text{Error 2}}{\text{Error 1}}$
6144	$1.02 \times 10^{-1}$	$2.41 \times 10^{-2}$	0.236
49152	$1.01 \times 10^{-1}$	$2.18 \times 10^{-2}$	0.216
393216	$5.22 \times 10^{-2}$	$1.74 \times 10^{-2}$	0.333
3145728	$1.88 \times 10^{-2}$	$8.10 \times 10^{-3}$	0.431

When  $\alpha = 0$  the resulting linear system has a matrix with entries still given by (11) however  $\alpha$  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 [3]. 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 (10) and (12)). The following results demonstrate that this extension also works well in practice.

### 3.2 Numerical Results

In Table 2 we illustrate the improved accuracy of the streamline-diffusion method when problem (6) is convection-dominated by considering the infinity norm of the exact error when solving the test problem considered in the previous section. As expected, we see that the stabilized FEM provides a less oscillatory solution with a smaller error, and that the relative improvement over the Galerkin FEM is greatest when the mesh is coarse.

Given that the stabilized method works best for small values of  $\varepsilon$ , in Table 3 we show iteration counts for the parallel preconditioned GMRES algorithm for the cases  $\varepsilon = 10^{-2}$  and  $\varepsilon = 10^{-3}$ . The former permits comparison with the solution of the discrete Galerkin equations of Table 1 (there is little change in the iteration counts), whilst the latter demonstrates the effectiveness of the solver for even smaller  $\varepsilon$ . In each case the effectiveness of the preconditioner is again demonstrated in terms of the small numbers of iterations required (although exact iteration counts do depend upon the precise domain decomposition used).

### 3.3 Parallel Performance

The calculations described in Tables 1 to 3 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 the timings quoted in Table 4 (for solving the

**Table 3.** The performance of the proposed algorithm on the stabilized convection-diffusion test problem for two choices of  $\varepsilon$ : figures quoted represent the number of iterations required to reduce the initial residual by a factor of  $10^5$ .

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

**Table 4.** The performance of the parallel solver for both the Galerkin and stabilized FEM systems when solving the test problem with  $\varepsilon = 10^{-2}$ . The solution times are quoted in seconds and the speed-ups are relative to the best sequential solution time.

Processors	Galerkin FEM					Streamline-Diffusion FEM				
	1	2	4	8	16	1	2	4	8	16
Solution Time	735.5	507.9	359.9	237.5	142.9	731.6	505.7	348.7	235.6	144.7
Speed-Up	—	1.45	2.04	3.10	5.15	—	1.46	2.10	3.11	5.06

test problem in the case  $\varepsilon = 10^{-2}$  using the finest mesh of 3145728 tetrahedral elements) 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, or the drop tolerance that is used in the sequential ILU preconditioner ([12]) that is used for these systems. Our choices for these parameters, determined empirically, may well also contribute to some of the variation between the two sets of results.

Furthermore, there are additional reasons why it is not feasible to obtain efficiencies close to 100% for these calculations. Note that the preconditioner is algebraically different for each choice of  $p$  and that, when  $p > 1$ , the sequential solution time is not generally as good as that for the *best available* sequential algorithm (for which we use [12]). Also, the algorithm itself depends upon the domain decomposition that has been used and so far we have only applied simple variants of recursive coordinate bisection. This could well be improved significantly with further research, although it is unlikely that the efficiency will be such that the use of extremely large numbers of processors will be viable.

## 4 Conclusions

We have described the parallel implementation of a weakly overlapping domain decomposition preconditioner and successfully applied it to the finite element solution of convection-dominated PDEs in three dimensions. Provisional results show that the algorithm is fast and that the parallel implementation provides

moderate speed-ups on up to sixteen processors. Extensions to a wider variety of convection directions to that considered here yield similar results and further improvements to the the mesh partitioning are likely to enhance the parallel performance. The work should also be extended to convection-dominated *systems*, which also arise frequently in scientific modeling.

## Acknowledgments

SAN gratefully acknowledges the funding received from the Government of Pakistan in the form of a Quaid-e-Azam scholarship.

## 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., 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.
3. Bank, R.E., Jimack, P.K., Nadeem, S.A., Nepomnyaschikh, S.V.: A Weakly Overlapping Domain Decomposition for the Adaptive Finite Element Solution of Elliptic Partial Differential Equations. Submitted to *SIAM J. on Sci. Comp.* (2001).
4. Cai, X.-C., Sarkis, M.: A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems. *SIAM J. on Sci. Comp.* **21** (1999) 792–797.
5. Chan, T., Mathew, T.: Domain Decomposition Algorithms. *Acta Numerica* **3** (1994) 61–143.
6. 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.
7. 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.
8. 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.
9. Johnson, C.: *Numerical Solutions of Partial Differential Equations by the Finite Element Method*. Cambridge University Press (1987).
10. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. *Int. J. Supercomputer Appl.* **8** (1994) no. 3/4.
11. Oswald, P.: *Multilevel Finite Element Approximation: Theory and Applications*. Teubner Skripten zur Numerik, B.G. Teubner (1994).
12. 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).
13. Smith, B., Bjorstad, P., Gropp, W.: *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press (1996).
14. 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.