

Parallel Preconditioners Based Upon Domain Decomposition

P.K. Jimack and D.C. Hodgson
Computational PDE Unit
School of Computer Studies
University of Leeds
Leeds LS2 9JT, UK

Abstract: Domain-decomposition methods have been studied and applied to the solution of engineering problems for many years. Recent advances in parallel computing systems have rekindled interest in these methods, at least in part due to the obvious parallelism that is apparent in such algorithms. This paper briefly reviews a number of different domain-decomposition algorithms and shows how they may be used in conjunction with iterative solution techniques for the solution of linear systems of equations arising in the finite element solution of computational mechanics problems. Particular emphasis is placed on so-called iterative substructuring algorithms, in which non-overlapping subdomains are used, and details of how to produce efficient parallel implementations are included.

1 Introduction

In this paper we are concerned with the finite element solution of the following second order model problem.

Find $u \in V \subset H^1(\Omega)$ such that

$$A(u, v) = F(v), \quad \forall v \in V, \quad (1)$$

where $\Omega \in \mathfrak{R}^2$ is the problem domain, $A(\cdot, \cdot)$ and $F(\cdot)$ are the bilinear and linear forms

$$A(u, v) = \int_{\Omega} a(\underline{x}) \underline{\nabla} u \cdot \underline{\nabla} v \, d\underline{x} \quad \text{and} \quad F(v) = \int_{\Omega} f v \, d\underline{x} + \int_{\partial\Omega_t} g v \, ds, \quad (2)$$

with $a(\underline{x}) > 0$ and $\partial\Omega_t$ the part of the boundary subject to Neumann boundary conditions.

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

$$A(u^h, v^h) = F(v^h), \quad \forall v^h \in S^h(\Omega). \quad (3)$$

This problem may in turn be expressed as the matrix equation

$$A\underline{u} = \underline{b}, \tag{4}$$

where A is the stiffness matrix, \underline{b} is the load vector and \underline{u} is a vector of nodal displacements which is to be determined.

In general the matrix A will be strictly positive-definite and sparse, so an iterative solution method for (4), such as the conjugate gradient method, [20], is most appropriate. Unfortunately, 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 , the condition number of A grows like $O(h^{-2})$ as $h \rightarrow 0$ (see [30] for example). For this reason it is necessary to apply a preconditioned version of the conjugate gradient algorithm for realistic mesh sizes h (again see [20]). (Note that, in this context, reference to a mesh size of h means that there exist constants, c_1 and c_2 , such that the length of each edge of every element lies in the interval $[c_1h, c_2h]$.)

2 Domain decomposition preconditioning

There are many possible ways in which the system (4) can be preconditioned. Some of these are purely algebraic, such as incomplete Cholesky factorization [38], whilst others make use of the underlying finite element derivation of the system, such as element-by-element preconditioning [29, 48] or domain decomposition preconditioners, which are the subject of this paper.

The essential idea behind any preconditioning strategy is to find a positive-definite matrix, M say, that has two properties.

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

(In fact the above definition refers to what is known as a left preconditioner: the more general definition requires that the matrix $C^{-1}AC^{-1}$ should be well conditioned (and here $M = C^T C$.) When seeking to solve the system (4) 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 $M^{-1}A$, and the major computational step in this algorithm at each iteration is the solution of a system of the form $M\underline{s} = \underline{r}$ ([20]). As we will see, domain decomposition preconditioning can satisfy all three of these requirements.

2.1 Overlapping Schwartz methods

The original ideas behind domain decomposition techniques may be traced back to the work of Schwartz, who formulated what has become known as the Schwartz alternating method as long ago as the last century [44]. The idea behind this approach is to divide the computational domain Ω into a number of overlapping subdomains (we use two for this illustration), Ω_1 and Ω_2 say. Now let the part of the boundary of Ω_1 that lies inside Ω_2 be labeled Γ_1 , and the part of the boundary of Ω_2 that lies inside Ω_1 be labeled Γ_2 . The Schwartz alternating algorithm is then:

- Guess values of solution unknowns on Γ_1 .
- Repeat steps 1 and 2 until convergence:
 1. Use latest values on Γ_1 to solve problem in Ω_1 ,
 2. Use latest values on Γ_2 to solve problem in Ω_2 .

This idea may easily be incorporated into the PCG algorithm by simply applying steps 1 and 2 above as the preconditioner at each iteration. As it stands however the approach is inherently sequential. If there is a larger number of subdomains then parallelism may be introduced by colouring the subdomains so that any two which overlap are coloured differently. This then allows solves to be performed in all of the subdomains of the same colour in parallel (see [28, 39] for details). Provided the number of subdomains of each colour is much larger than the number of processors in use then good parallel efficiencies may be achieved. Unfortunately however, as noted in [39], the larger the number of subdomains the slower the method converges.

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 [14] however, Dryja and Widlund describe this decoupling more algebraically. Following [36], they define the overlapping Schwartz 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) Schwartz methods. (Note that this idea of producing preconditioners through decomposition of the solution space into subspaces may be extended much further and a thorough review is provided by Xu in [50].)

As with the multiplicative Schwartz methods the number of iterations required for convergence when using additive Schwartz preconditioners also grows significantly with the number of subdomains. Hence, in [15], Dryja and Widlund introduce an extra step to the additive preconditioner (i.e. in addition to the decoupled overlapping subdomain solves). This projection requires the solution of a problem in which the subdomains themselves are treated as elements of a coarse finite element mesh, thus allowing low frequency errors to be damped out quickly. 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 subdomain size, H , the condition number of the preconditioned system is independent of both H and h . A typical result of this type is given in [49] where, provided an exact solver is used for each subdomain,

$$\kappa(M^{-1}A) \leq C(1 + C \max_i \frac{H_i}{\delta_i}), \quad (5)$$

where κ is the condition number, H_i is the diameter of the i^{th} substructure, δ_i is an indicator of the amount of overlap of this substructure and the constant C is independent of each of h , H_i and δ_i . Similar results are also proved for nonconforming elements and for elements subject to h -refinement in [9, 10] and [24, 43] respectively.

Despite these apparently attractive theoretical results, numerical experiments (e.g. [11, 12]) suggest that using a large overlap δ is not always the best strategy. This is because, although the condition number may be superior, the amount of work required per iteration is much greater when δ is large. In addition, from a distributed parallel computing viewpoint, a large amount of overlap often leads to an increase in the total amount of communication that is required at each iteration. Hence a more popular approach in practice is to allow only a single layer of mesh points to be common to pairs of neighbouring subdomains (i.e. with minimal overlap), [49]. This is the subject of the next subsection.

2.2 Iterative substructuring methods

Methods in which the domain Ω is partitioned into non-overlapping subregions are called iterative substructuring methods. Numerous algorithms of this type have been proposed and a number of these are discussed here.

To begin with, it should be noted that there is an important trade-off to be made when choosing the size, H , of the substructures Ω_i . The smaller the choice of H the better the condition number of the preconditioned system (see equation (5)) and the smaller the size of each of the substructure problems (making them both easier to solve and easier to distribute evenly in parallel implementations). On the other hand, small substructures lead to a much larger coarse finite element mesh problem to be solved at each iteration. (Taken to the limiting case where each substructure consists of just one element, this coarse grid problem becomes as hard to solve as the original finite element problem – for that is exactly what it would be.) One widely proposed approach for dealing with this trade-off is to choose H to be quite small but, instead of solving the coarse mesh problem exactly, apply the same additive Schwartz technique to this problem (possibly recursively). Such algorithms are known as multilevel additive Schwartz methods. One of the most well-known multilevel algorithms is that analyzed by Bramble, Pasciak and Xu in [8], with more recent work appearing in [21, 22, 51] for example. For the purposes of this paper however we will only consider two level preconditioners. This allows the use of more general finite element meshes (both structured and unstructured) and still illustrates most of the important features of parallel multilevel domain decomposition preconditioners.

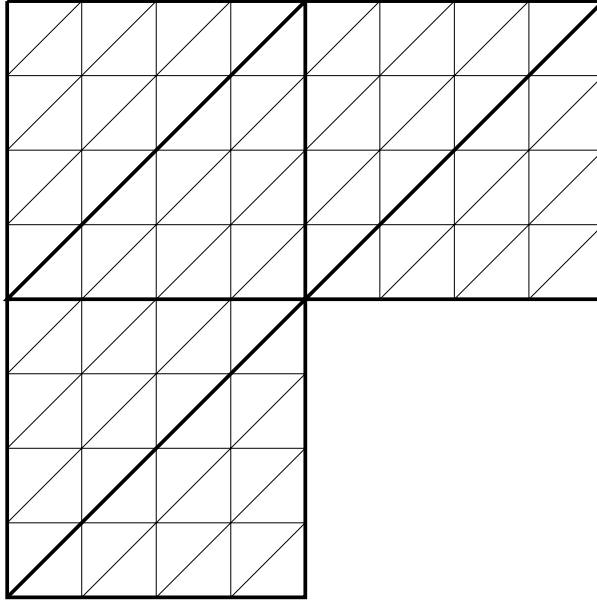


Figure 1: An allowable decomposition of a finite element mesh.

In order to describe some of the common approaches in detail it will be helpful to return to the finite element equations (4). Recall that the domain Ω has been decomposed into non-overlapping subdomains, Ω_i , and that the coarse substructure defined by this decomposition is itself a valid finite element mesh. We further assume that the triangulation of Ω is such that each element is contained within a single subdomain (see figure 1) and we consider the use of a piecewise linear approximation on this triangulation. (For extensions to higher order finite element approximations see [1, 2, 37, 41] for example).

Suppose we number the nodes in our finite element mesh starting with those inside the first subdomain, Ω_1 , followed by those inside Ω_2 , through to those inside the final subdomain, Ω_N say, finishing with those nodes lying on the boundary of the coarse substructure. Then equations (4) will take the following block-arrowhead form:

$$\begin{bmatrix} A_{II(1)} & & & & A_{IB(1)} \\ & A_{II(2)} & & & A_{IB(2)} \\ & & \ddots & & \vdots \\ & & & A_{II(N)} & A_{IB(N)} \\ A_{IB(1)}^T & A_{IB(2)}^T & \cdots & A_{IB(N)}^T & A_{BB} \end{bmatrix} \begin{bmatrix} \underline{u}_{I(1)} \\ \underline{u}_{I(2)} \\ \vdots \\ \underline{u}_{I(N)} \\ \underline{u}_B \end{bmatrix} = \begin{bmatrix} \underline{b}_{I(1)} \\ \underline{b}_{I(2)} \\ \vdots \\ \underline{b}_{I(N)} \\ \underline{b}_B \end{bmatrix}. \quad (6)$$

Here $\underline{u}_{I(i)}$ represents the unknown nodal displacements inside Ω_i whilst \underline{u}_B represents the unknown nodal displacements on the substructure boundary. Now let us number the unknowns on the substructure boundary so that they are ordered one edge at a time, followed by those unknowns which lie at vertices of the substructure. (Figure 2 shows the nodes on substructure edges as circles and the nodes at substructure vertices as squares: solid shapes indicate unknowns whilst unfilled shapes indicate that the nodal values are prescribed, due to Dirichlet boundary conditions for example.)

n in which all of the components not associated with the interior of subdomain i are zero, $R_{E(j)}^T$ returns a vector of length n in which all of the components not associated with edge j are zero, and R_V^T returns a vector of length n in which all of the components not associated with the substructure vertices are zero. We now modify (12) by replacing R_V^T by \tilde{R}_V^T , which is a linear interpolation operator that maps the substructure vertex values onto the edge nodes and into the subdomains. The corresponding matrix \tilde{R}_V therefore acts as a weighted summation operator which accumulates interior and edge values onto the substructure vertices. Hence we have constructed a further domain decomposition preconditioner, given by:

$$\underline{s} = M^{-1}\underline{r} = \sum_{i=1}^N R_{I(i)}^T A_{II(i)}^{-1} R_{I(i)} \underline{r} + \sum_{j=1}^P R_{E(j)}^T A_{EE(j)}^{-1} R_{E(j)} \underline{r} + \tilde{R}_V^T \tilde{A}^{-1} \tilde{R}_V \underline{r}. \quad (13)$$

3 Parallel Implementations of Domain Decomposition Preconditioners

In comparison with the number of publications studying the theory of domain decomposition algorithms there are relatively few published papers which directly address the issues associated with parallel implementations. Many authors content themselves with observing that all of the subproblems appearing in preconditioners such as (13) are independent and may therefore be solved concurrently on a parallel computer. Whilst this is undoubtedly true it is probably fair to say that there is considerably more to implementing a domain decomposition algorithm efficiently in parallel than such authors seem to imply. Some papers which do address these parallel implementation issues however include [13, 17, 27, 28, 32, 34, 39, 40, 46]. Rather than attempt to review each of these papers in this section we instead present a description of how one might implement the preconditioner (13) on a parallel distributed memory computer, based upon ideas appearing in a number of these works.

3.1 A parallel algorithm

We begin by returning to the original system of algebraic equations (4), written in the block-arrowhead form (6). For the time-being we will assume that the number of subdomains, N , is exactly equal to the number of parallel processors available to us and that the mesh for each subdomain is stored on its own processor. It is therefore possible to compute and assemble each of the blocks $A_{II(i)}$, $A_{IB(i)}$ and $\underline{b}_{I(i)}$ independently on the processor storing subdomain i . In addition, each processor can compute and assemble its own contribution to the remaining block of A , A_{BB} , independently – storing it in the block $A_{BB(i)}$ say (hence $A_{BB} = A_{BB(1)} + \dots + A_{BB(N)}$). For the remaining block of \underline{b} , \underline{b}_B , each processor can again compute its own contribution but some communication is required to enable the full assembly of \underline{b}_B to be known on each processor.

Behind the preconditioning step, (13), the next most expensive step in each iteration of the PCG algorithm is the calculation of a matrix-vector product,

$\underline{q} = A\underline{p}$ say. Assuming that \underline{p} and \underline{q} are partitioned into blocks in the same manner as (6) this may be written as

$$\underline{q}_{I(i)} = A_{II(i)}\underline{p}_{I(i)} + A_{IB(i)}\underline{p}_B \quad (14)$$

for $i = 1, \dots, N$ and

$$\underline{q}_B = \sum_{i=1}^N A_{IB(i)}^T \underline{p}_{I(i)} + A_{BB(i)}\underline{p}_B. \quad (15)$$

Clearly (14) shows that each of the blocks $\underline{q}_{I(i)}$ may be computed in parallel but also, (15) demonstrates that with the use of a single global communication (to perform the sum) the final block, \underline{q}_B , of the matrix-vector product may also be computed in parallel.

The preconditioning step, (13), is computed in parallel in much the same way. The simplest component is the calculation of $\sum_{i=1}^N R_{I(i)}^T A_{II(i)}^{-1} R_{I(i)} \underline{r}$ since each term of the sum may be evaluated independently and concurrently on each processor for $i = 1, \dots, N$. This evaluation involves a local subdomain solve on each processor in order to calculate $A_{II(i)}^{-1} (R_{I(i)} \underline{r})$. Such a solve may be completed either through the use of an iterative solver or, perhaps more efficiently, through a sparse Cholesky factorization (see [19] for example). The final summation then requires a single global communication.

The term $\sum_{j=1}^P R_{E(j)}^T A_{EE(j)}^{-1} R_{E(j)} \underline{r}$ in (13) is only a little more complicated to evaluate in parallel. This should first be expressed in the equivalent form

$$\sum_{i=1}^N \sum_{J=1}^3 R_{E(j)}^T [A_{EE(j)}^{-1}] R_{E(j)} \underline{r}, \quad (16)$$

where the J^{th} edge of subdomain i is the substructure edge whose global number is j , and the notation $[A_{EE(j)}^{-1}]$ signifies that when j is an edge in the interior of the substructure the contribution to the sum should only be taken from one of the subdomains on either side of it. In this case it should be observed that some local communication is required in order to assemble $A_{EE(j)}$ on the processor responsible for computing this contribution to the sum. All of these contributions may be completed concurrently provided they are evenly spread across the processors, and they may be calculated with the use of sequential Cholesky factorizations of the matrices $A_{EE(j)}$. (Note that here we have assumed for simplicity that none of the edges j form part of a Dirichlet boundary.)

The final term in (13), $\tilde{R}_V^T \tilde{A}^{-1} \tilde{R}_V \underline{r}$, involves assembling a weighted sum of the components of \underline{r} onto the substructure vertices, followed by a global substructure solve. The first of these operations, computing $\tilde{R}_V \underline{r}$, may be completed in parallel by allowing each subdomain to accumulate the contribution from each of its interior nodes to the weighted sum for each of its three substructure vertices and also the contribution from each of its substructure edge nodes to the weighted sum for the substructure vertex at the start of that edge (in a counter-clockwise sense). It is far less straightforward to perform the global substructure solve in parallel however. Due to the small size of this problem it is generally accepted that this operation is best performed sequentially on a single processor.

Hence a global communication is required to allow a single processor to assemble every contribution to the vector $\tilde{R}_V \underline{r}$; this processor then evaluates $\tilde{A}^{-1} \tilde{R}_V \underline{r}$ before broadcasting the result to all of the other processors so that it may be interpolated into each subdomain and onto each edge locally in parallel.

This completes the present subsection on the implementation of a parallel domain decomposition preconditioner for an iterative method such as the preconditioned conjugate gradient (PCG) algorithm. The next subsection discusses issues relating to the properties that the substructure should possess in order to allow good parallel performance.

3.2 Load balancing and parallel efficiency

So far we have assumed for simplicity that the number of subdomains is equal to the number of parallel processors that are to be used. If we continue with this assumption for the time-being then it follows that, in order to ensure that each processor has the same computational load, each subdomain should contain the same number of elements and unknowns. It is also desirable that the total number of unknowns on the substructure boundary is as small as possible so that the amount of additional computation (and communication) required by the above parallel algorithm is as low as possible. If this is the case then the only significant inefficiency in this algorithm is that the global substructure solve occurs on a single processor and so the other processors are likely to be idle while this takes place. When the number of substructures is equal to the number of processors however this global problem is very small, so the idle time will not be very substantial in practice. Moreover, in reality it is very unlikely that each subdomain will contain an absolutely identical number of unknowns, so by giving the global solve to the least loaded of all of the processors this inefficiency can be made even less significant.

We now consider the more realistic case where the number of processors is not equal to the number of subdomains in our decomposed domain. We will assume however that the number of subdomains is significantly larger than the number of processors. In order to achieve an efficient parallel solution it is necessary to partition the subdomains across the processors so that each processor gets the same total amount of work to do. In the simplest model of computation that we might use this could mean that the total number of unknowns on each processor should be the same. Hence if the subdomains are of equal size then the number of subdomains per processor should be the same where possible, however if the subdomains are of unequal sizes then we would expect different processors to store different numbers of subdomains. An additional constraint is that, to save communication overheads, we would like neighbouring subdomains to be stored on the same processor whenever possible. This problem of partitioning the subdomains across the processors may be expressed as a well-known graph partitioning problem where each subdomain is represented by a node of a graph (with weight equal to the number of degrees of freedom in the mesh in that subdomain), and two nodes of the graph are connected if the corresponding subdomains have any common substructure edges (with the weight of this

	4 Processor Problem (55172 unknowns)		8 Processor Problem (110446 unknowns)	
Preconditioner	No	Yes	No	Yes
No. Iterations	834	114	1273	148
Exec. Time	136.2	53.4	206.4	71.6
Time per It.	0.16	0.47	0.16	0.48

Table 1: Iteration counts and execution times for the PCG solver.

connecting edge being equal to the number of degrees of freedom shared by the subdomains). Although the graph partitioning problem is NP-hard there exist a large number of heuristic methods for obtaining good solutions in polynomial time (including those described in [16, 25, 31, 42, 47] for example). This means that it is possible to partition the substructure across the processors in a well load-balanced manner.

In fact it is possible to attempt to load-balance the entire preconditioning step of the PCG algorithm by adding an estimate of the cost of the global substructure solve to the weight of one of the nodes of the weighted subdomain graph. When the graph is partitioned the processor which is allocated this node of the graph then becomes the processor which performs the global solve. Moreover, it is possible to overlap much of the communication overhead associated with this substructure solve (and the edge solves for those substructure edges on the partition boundary) by completing the associated message passing at the same time as the solves are being completed on the substructure interiors.

In [26] some computations are reported in which Laplace’s equation is solved using the above parallel domain decomposition preconditioning algorithm. Table 1 shows some of these results obtained when a problem is solved on 4 and 8 Intel iPSC/860 processors. In each case the coarse substructure consists of 1331 elements and Ω is a non-trivial geometry (it is the “Texas” geometry taken from [3]). The convergence criterion used for the PCG solver was a reduction in the preconditioned residual of 10^{-8} .

Inspection of these results shows that, although the cost per iteration is increased by a factor of three when the domain decomposition preconditioner is used, the reduction in the number of iterations more than makes up for this; leading to significant savings overall. Moreover, the parallel implementation, as described in this section, can be seen to be scalable as we move from 4 to 8 processors (keeping the number of unknowns per subdomain about the same). Further examples in [26] confirm this scalability and also demonstrate that the growth in the number of iterations as $h \rightarrow 0$ is much slower when the preconditioner is used.

4 Discussion

In this paper we have introduced the concept of domain decomposition preconditioning for the system of linear algebraic equations (4) which arises from

the finite element solution of (1). The ideas behind multiplicative and additive Schwartz methods have been outlined and some typical analytic bounds on the condition number of preconditioned systems have been presented. The particular approach known as iterative substructuring has been described in some detail, as has a typical efficient and scalable parallel implementation.

The final topic that we briefly discuss is the solution of (4) via the Schur Complement Method. This is a very common domain decomposition technique in which (4) is converted into a related problem which only involves those nodes on the substructure edges and vertices. From (6) we see that

$$A_{II(i)}\underline{u}_{I(i)} + A_{IB(i)}\underline{u}_B = \underline{b}_{I(i)} \quad \text{for } i = 1, \dots, N \quad (17)$$

and

$$\sum_{i=1}^N A_{IB(i)}^T \underline{u}_{I(i)} + A_{BB}\underline{u}_B = \underline{b}_B, \quad (18)$$

and combining these two expressions gives

$$(A_{BB} - \sum_{i=1}^N A_{IB(i)}^T A_{II(i)}^{-1} A_{IB(i)})\underline{u}_B = \underline{b}_B - \sum_{i=1}^N A_{IB(i)}^T A_{II(i)}^{-1} \underline{b}_{I(i)}. \quad (19)$$

This means that we need only solve the problem

$$S\underline{u}_B = \underline{g} \quad (20)$$

for the unknowns on the substructure boundaries – where the matrix S , known as the Schur Complement matrix or the Capacitance matrix, is given by

$$S = A_{BB} - \sum_{i=1}^N A_{IB(i)}^T A_{II(i)}^{-1} A_{IB(i)} \quad (21)$$

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

$$\underline{g} = \underline{b}_B - \sum_{i=1}^N A_{IB(i)}^T A_{II(i)}^{-1} \underline{b}_{I(i)}. \quad (22)$$

Once \underline{u}_B is known then (17) allows each of the vectors of interior unknowns, $\underline{u}_{I(i)}$, to be found concurrently.

As with the full system (4) it is most appropriate to solve (20) via an iterative technique such as the preconditioned conjugate gradient (PCG) method, and again it is possible to use domain decomposition preconditioners.

To perform the matrix-vector multiplication step in the PCG algorithm efficiently in parallel is no more difficult than for the full problem since

$$S\underline{p}_B = \sum_{i=1}^N A_{BB(i)}\underline{p}_B - \sum_{i=1}^N A_{IB(i)}^T \left(A_{II(i)}^{-1} \left(A_{IB(i)}\underline{p}_B \right) \right). \quad (23)$$

Hence each product may be obtained using only matrix-vector products and interior solves independently on each subdomain (followed by a single global

communication). This is very similar to the amount of computation and communication as required by (14) and (15) above.

To precondition the Schur Complement matrix it is also possible to directly generalize the approaches described in the previous section. For example [1, 2, 15, 18, 26, 27, 32, 45, 46] all describe algorithms for preconditioning (20) in parallel. In [27] it is demonstrated that the local edge solves require more work for the Schur complement approach than for the full problem because the edge blocks of the matrix S have to be explicitly assembled. This additional work per iteration is shown to be worthwhile by Hodgson in [26] however, where the number of iterations required by the Schur complement solver is significantly less than for the full system. In [45] the domain decomposition preconditioner for (20) is improved further still, by introducing an additional set of local problems to be solved. Each such problem involves a single substructure vertex node and a number of nodes along each edge of the substructure which meet at that vertex. Again it is possible to solve these local problems concurrently but now a bound on the condition number of $M^{-1}A$ is proved which is independent of both h and H .

We close this short review with the remark that the techniques that have been introduced here can be applied to a much wider variety of problems than the simple linear self-adjoint model problem (1). Generalizations have been analyzed and implemented for a host of more complex problems including plate bending [9], the biharmonic equation [10], simple CFD problems [23], nonlinear PDEs [33], convection-diffusion systems [35] and 3-d linear elasticity equations [45]. Domain Decomposition techniques clearly form an important class of method for the solution of a wide variety of differential equations, with both a strong theoretical foundation and a simplicity that allows efficient parallel implementations to be achieved in practice.

Acknowledgements

The parallel computations reported here made use of the Intel iPSC/860 computing facility at the EPSRC Daresbury Laboratory under SERC grant GR/J27066.

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] R.E. Bank, “PLTMG: A Software Package for Differential Equations”, SIAM publications, Philadelphia, 1990.

- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] J. Bramble, J. Pasciak and J. Xu, “*Parallel Multilevel Preconditioners*”, *Mathematics of Computation*, 55, 1–21, 1990.
- [9] S.C. Brenner, “*A Two-level Additive Schwartz Preconditioner for Nonconforming Plate Elements*”, *Numerische Mathematik*, 72, 419–447, 1996.
- [10] S.C. Brenner, “*Two-level Additive Schwartz Preconditioners for Nonconforming Finite Element Methods*”, *Mathematics of Computation*, 65, 897–921, 1996.
- [11] X.-C. Cai, “*An Additive Schwartz Algorithm for Nonselfadjoint Elliptic Equations*”, in *Third International Symposium on Domain Decomposition Methods* (T.F. Chan *et al.*, eds.), SIAM publications, Philadelphia, 1990.
- [12] X.-C. Cai, W.D. Gropp and D.E. Keyes, “*A Comparison of Some Domain Decomposition Algorithms for Nonselfadjoint Elliptic Problems*”, in *Fifth International Symposium on Domain Decomposition Methods* (D.E. Keyes *et al.*, eds.), SIAM publications, Philadelphia, 1992.
- [13] T.F. Chan and J.P. Shao, “*Parallel Complexity of Domain Decomposition Methods and Optimal Coarse Grid Size*”, *Parallel Computing*, 21, 1033–1049, 1995.
- [14] 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.
- [15] M. Dryja and O.B. Widlund, “*Some Domain Decomposition Algorithms for Elliptic Problems*”, in *Iterative Methods for Large Linear Systems*, Academic Press, 1990.
- [16] C. Farhat, “*A Simple and Efficient Automatic FEM Domain Decomposer*”, *Computers and Structures*, 28, 579–602, 1988.

- [17] 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.
- [18] 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.
- [19] A. George and J.W. Liu, “*Computer Solution of Large Sparse Positive Definite Systems*”, Prentice Hall, 1981.
- [20] G.H. Golub and C.F. Van Loan, “*Matrix Computations*”, John Hopkins Press, 2nd edition, 1989.
- [21] M. Griebel, “*Multilevel Algorithms Considered as Iterative Methods on Semidefinite Systems*”, SIAM J. on Scientific Computing, 15, 547–565, 1994.
- [22] M. Griebel and P. Oswald, “*On Additive Schwartz Preconditioners for Sparse Grid Discretizations*”, Numerische Mathematik, 66, 449–463, 1994.
- [23] W.D. Gropp and D.E. Keyes, “*Domain Decomposition Methods in Computational Fluid Dynamics*”, International J. for Numerical Methods in Fluids, 14, 147–165, 1992.
- [24] W.D. Gropp and D.E. Keyes, “*Parallel Performance of Domain-Decomposed Preconditioned Krylov Methods for PDEs with Locally Uniform Refinement*”, SIAM J. on Scientific Computing, 13, 128–145, 1992.
- [25] B. Hendrickson and R. Leland, “*An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*”, SIAM J. on Scientific Computing, 16, 452–469, 1995.
- [26] D.C. Hodgson, “*Efficient Mesh Partitioning and Domain Decomposition Methods on Parallel Distributed Memory Machines*”, Ph.D. Thesis, School of Computer Studies, University of Leeds, 1995.
- [27] D.C. Hodgson and P.K. Jimack, “*A Domain Decomposition Preconditioner for a Parallel Finite Element Solver on Distributed Unstructured Grids*”, To appear in Parallel Computing, 1997.
- [28] K.H. Hoffman and J. Zou, “*Parallel Efficiency of Domain Decomposition Methods*”, Parallel Computing, 19, 1375–1391, 1993.
- [29] 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.
- [30] C. Johnson “*Numerical Solution of Partial Differential Equations by the Finite Element Method*”, Cambridge University Press, 1987.

- [31] G. Karypis and V. Kumar, “*A Fast High Quality Multilevel Scheme for Partitioning Irregular Graphs*”, Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.
- [32] 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.
- [33] D.E. Keyes and V. Venkatakrishnan, “*Newton-Krylov-Schwartz Methods – Interfacing Sparse Linear Solvers with Nonlinear Applications*”, Zeitschrift fur Angewandte Mathematik und Mechanik, 76, 147–150, 1996.
- [34] O. Klaas, R. Niekamp and E. Stein, “*Parallel Adaptive Finite Element Computations with Hierarchical Preconditioning*”, Computational Mechanics, 16, 45–52, 1995.
- [35] D.A. Knoll, P.R. McHugh and D.E. Keyes, “*Domain Decomposition Methods Applied to a System of Convection-Diffusion-Reaction Equations*”, Zeitschrift fur Angewandte Mathematik und Mechanik, 76, 235–238, 1996.
- [36] P.L. Lions, “*Interprétation Stochastique de la Méthode Alternée de Schwartz*”, C. R. Acad. Sci. Paris, 268, 325–328, 1978.
- [37] J. Mandel, “*Iterative Solvers by Substructuring for the p -Version of the Finite Element Method*”, Computer Methods for Applied Mechanics and Engineering, 80, 117–128, 1990.
- [38] T.A. Mantueffel, “*Shifted Incomplete Cholesky Factorization*”, in Sparse Matrix Proceedings (I.S. Duff and G.W. Stewart eds.), SIAM Publications, Philadelphia, 1978.
- [39] G. Meurant, “*Domain Decomposition Methods for PDEs on Parallel Computers*”, Int. J. Supercomputer Applications, 2, 5–12, 1988.
- [40] L.F. Pavarino and M. Ramé, “*Numerical Experiments with an Overlapping Additive Schwartz Solver for 3-D Parallel Reservoir Simulation*”, Int. J. Supercomputer Applications, 9, 3–17, 1995.
- [41] L.F. Pavarino and O.B. Widlund “*A Polylogarithmic Bound for an Iterative Substructuring Method for Spectral Elements in Three Dimensions*”, SIAM J. on Numerical Analysis, 33, 1303–1335, 1996.
- [42] R. Preis and R. Diekmann, “*The PARTY Partitioning Library User Guide*”, Technical Report rsfb-96-024, Department of Computer Science, University of Paderborn, 1996.
- [43] W. Rachowicz, “*An Overlapping Domain Decomposition Preconditioner for an Anisotropic h -Adaptive Finite Element Method*”, Computer Methods for Applied Mechanics and Engineering, 127, 269–292, 1995.

- [44] H.A. Schwartz, “*Über Einige Abbildungsaufgaben*”, J. für Die Reine und Angewandte Mathematik, 70, 105–120, 1869.
- [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] C. Walshaw, M. Cross, M.G. Everett, S. Johnson and K. McManus, “*Partitioning and Mapping of Unstructured Meshes to Parallel Machine Topologies*”, in Irregular ’95: Parallel Algorithms for Irregularly Structured Problems (A. Ferreira and J. Rolim, eds.), Springer, 1995.
- [48] A.J. Wathen, “*An Analysis of some Element-by-Element Techniques*”, Computer Methods for Applied Mechanics and Engineering, 74, 271–287, 1989.
- [49] O.B. Widlund, “*Some Schwartz Methods for Symmetric and Nonsymmetric Elliptic Problems*”, in Fifth International Symposium on Domain Decomposition Methods (D.E. Keyes *et al*, eds.), SIAM publications, Philadelphia, 1992.
- [50] J. Xu, “*Iterative Methods by Space Decomposition and Subspace Correction*”, SIAM Review, 34, 581–613, 1992.
- [51] X. Zhang, “*Multilevel Schwartz Methods*”, Numerische Mathematik, 63, 521–539, 1992.