Parallel implementation of a Schwarz Domain Decomposition Algorithm

Petter E. Bjørstad[1], Maksimillian Dryja[2] and Eero Vainikko[1]

[1] Para//ab, Institutt for Informatikk, University of Bergen, N-5020 Bergen, Norway. Email: petter@ii.uib.no

[2] Department of Mathematics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland. Email: dryja@mimuw.edu.pl

Abstract. We describe and compare some recent domain decomposition algorithms of Schwarz type with respect to parallel performance. A new, robust domain decomposition algorithm – Additive Average Schwarz is compared with a classical overlapping Schwarz code. Complexity estimates are given in both two and three dimensions and actual implementations are compared on a Paragon machine as well as on a cluster of modern workstations.

1 Introduction

This paper is concerned with parallel algorithms for the numerical solution of equations of the form:

Find $u^* \in V(\Omega)$ such that

$$\sum_{i=1}^{N} \int_{\Omega_i} \rho_i \nabla u^* \cdot \nabla v \; dx = \int_{\Omega} f v \; dx \quad \forall v \in V(\Omega), \tag{1}$$

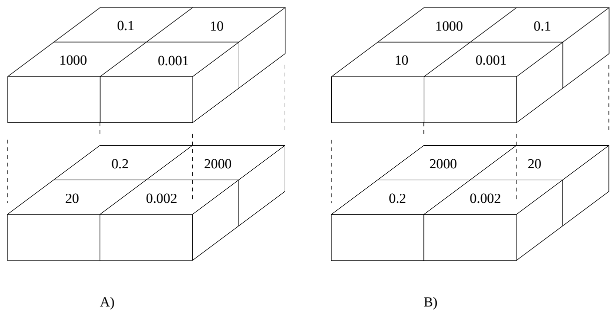
in a space $V(\Omega) \subset H^1(\Omega)$. Here ρ_i are positive constants and $\bar{\Omega} = \cup_{i=1}^{N} \bar{\Omega}_i$.

This problem is important in many fields of application, an example being the study of flow in porous media [2]. Domain decomposition algorithms have received much attention in the last ten years due to their potential for parallel algorithms as well as overall flexibility within the design of more comprehensive software packages.

A prototype algorithm with good parallel properties is the Additive Schwarz method [12]. Its parallel implementation has been reported on in several papers, see for example [3], [8] and [10]. However, when the parameter ρ_i in (1) has significant jumps across internal boundaries (for example across material layers) the rate of convergence suffers. It turns out that the rate of convergence depends on the distribution of the numerical values of ρ_i (see [5] and [6]) and that the estimates for this behavior are sharp [11], [13]. In Figure 1 we show two different distributions called quasi-monotone and not quasi-monotone respectively. The Additive Schwarz method behaves better in the quasi-monotone case. This motivated an alternative algorithm, the Additive Average method, proposed in [1] which has a rate of convergence independent of any specific distribution of the

ρ_i. Additionally, the new method has an advantage when the discretization is unstructured with a possibly very non-uniform mesh.

Fig. 1. The distribution in A) is not quasi-monotone as there exist no monotone path (through faces) from 1000 to 2000. The distribution in B) is quasi-monotone.



In this paper we compare parallel implementations of the two methods. Originally motivated by robustness, it turns out that the new method also has good parallel characteristics. The rest of this paper is organized as follows. In Section 2 we briefly describe the algorithms that will be considered. The reader is referred to [1] and [12] for a more detailed description. In Section 3 we discuss certain aspects of our parallel implementation of the new method. Section 4 discusses the parallel complexity in 2 and 3 dimensions, while Section 5 reports on experiments on current parallel machines where we can compare actual performance. We end with some conclusions in Section 6.

2 The Methods

Let Ω be a polyhedral region in R^d, $d = 2, 3$. We triangulate Ω and partition this triangulation into a set of possibly overlapping subdomains, Ω_i^h, where $\Omega_i \subseteq \Omega_i^h$. We look for a solution $u \in V^h$ where $V^h \subset V$ is a finite element space on our fine triangulation. To keep the presentation simple we only discuss the case $V = H_0^1$.

Find $u \in V^h(\Omega)$ such that

$$a(u, v) = f(v) \quad \forall v \in V^h(\Omega), \tag{2}$$

where

$$a(u, v) = \sum_{i=1}^{N} \int_{\Omega_i} \rho_i \nabla u \cdot \nabla v \, dx, \qquad f(v) = \int_{\Omega} f v \, dx. \tag{3}$$

We split the space V^h into $N + 1$ subspaces

$$V^h = V_0 + V_1 + \cdots + V_N,$$

where for $i = 1, \dots N$ $V_i = H_0^1(\Omega_i^h) \bigcap V^h$ and zero outside of Ω_i^h. The space $V_0(\Omega)$ is a coarse space to be defined below.

For each $V_i, i = 0, ..., N$ we introduce bilinear forms $b_i(u, v)$ on $V_i \times V_i$ defining subproblems that will be used to form our preconditioner.

Following [12] we can describe the methods in a linear algebra framework where the matrix A denotes our discrete problem to be solved. For $i = 1, \dots N$ let R_i be the restriction matrix that selects the nodes from subdomain Ω_i^h. Similarly, we define R_0^T to be an interpolation operator from the coarse space V_0 to V^h. An Additive Schwarz preconditioner is then given by

$$B = \sum_{i=0}^{N} B_i \tag{4}$$

where $B_i = R_i^T A_i^{-1} R_i$. Here $A_i = R_i A R_i^T$ for $i = 1, \dots N$, while A_0 is the matrix representation of a coarse approximation of our problem. We note that the preconditioner (4) can be applied to a vector by applying each term in the sum independently, thus this step can easily use $N+1$ processors in parallel.

Our methods differ in the choice of subspaces V_i and, in particular, on how the coarse space V_0 and the corresponding bilinear form $b_0(u, v)$ are defined and constructed.

2.1 The Classical Additive Schwarz Method

This method uses overlapping subdomains Ω_i^h. Since previous work [4] has shown that a minimal overlap is preferable, we limit our discussion to the case where Ω_i^h extends precisely one grid block (or one element) outside of Ω_i. The coarse space is defined from a discretization of Ω by using the subdomains Ω_i as discretization elements. We note that this (severely) restricts the shape of Ω_i. A linear interpolation from the coarse triangulation back to our original fine mesh defines R_0^T.

2.2 The Additive Average Method

This method is characterized by subdomains that do not overlap, that is, Ω_i^h is a triangulation of Ω_i. Here we do not have any restrictions on the shape of Ω_i. To simplify the presentation we assume that our fine grid discretization parameter h is constant across all subdomains, our implementation allows a variable h, that is, an unstructured grid across all of Ω. In order to define the coarse space let n_i^b denote the number of nodes on $\partial\Omega_i^h$ and let x be a *nodal point* in our discretization. (Note that we do include external boundary nodes of $\partial\Omega$ with a Dirichlet condition in this count.) We define an interpolation operator I_A as follows:

$$I_A = \begin{cases} u(x) & x \in \partial\Omega_i^h \\ \overline{u}_i & x \in \Omega_i^h \end{cases} \tag{5}$$

where

$$\overline{u}_i = \frac{1}{n_i^b} \sum_{x \in \partial \Omega_i^h} u(x), \tag{6}$$

the average of the nodal values of u on the boundary of Ω_i^h.

Our coarse space V_0 is now simply taken as the range of I_A. The bilinear form on V_0 is defined by

$$b_0(u,v) = c_0 h^{d-2} \sum_i \rho_i \sum_{x \in \partial \Omega_i^h} (u(x) - \overline{u}_i)(v(x) - \overline{v}_i). \tag{7}$$

Note that with this choice we have $A_0 \neq A_C = I_A^T A I_A$. A_C is often called the Galerkin coarse grid approximation. Our A_0 defined by $b_0(u,v)$ approximates A_C and it has more structure making it less expensive to use as part of our preconditioner.

2.3 A Multiplicative Average Method

Just as one can define a multiplicative version of the classical Additive Schwarz method, we may define a new multiplicative variant of the Additive Average method. Since we have no subdomain overlap the individual subdomain solutions can still be performed in parallel, while the coarse solve must proceed sequentially. We begin with the coarse solver followed by the subdomain solvers with an additional residual calculation between the two steps. With conjugate gradient as our preferred Krylov subspace accelerator we must end the iteration step by another residual calculation and a final coarse solve in order to preserve symmetry.

2.4 An Additive Diagonal Scaling Method

This method is similar to the Average Additive method on the fine level, the domain is divided into non-overlapping subdomains and the matrices A_i are defined in the same way as in Section 2.2. The coarse space V_0 is the same as in the Additive Schwarz method from Section 2.1. In addition, we introduce an extra space V_{-1} in order to include the nodes from $\partial \Omega_i^h$ in the preconditioner. Each of these boundary nodes are considered to be the single node of a correspondingly small subdomain surrounding it. In the preconditioning the operation of solving such a single node subdomain problem corresponds to a single division (scaling) by the diagonal element of the stiffness matrix A. Unfortunately, this method is sensitive to the distribution of the coefficients ρ_i [1].

3 Parallel Implementation

Parallel implementations of the Additive Schwarz method are well documented [4], [10], [12]. The Additive Average method lends itself to a parallel implementation and we need only discuss the parallel solution of the coarse problem in more detail.

This problem takes the following form:

$$\begin{pmatrix} A_a & A_{ab} \\ A_{ab}^T & A_b \end{pmatrix} \begin{pmatrix} u_a \\ u_b \end{pmatrix} = \begin{pmatrix} 0 \\ v_b \end{pmatrix} \tag{8}$$

where we have introduced one extra unknown per subdomain in the vector u_a (of length N) corresponding to the average values $\overline{u}_i$ from (6). We use the subscripts a and b to denote average (values) and boundary (nodes) respectively. The vector u_b has n_b components one for each node on all internal subdomain boundaries.

The $N \times N$ matrix A_a is diagonal with entries

$$(A_a)_{ii} = c_0 h^{d-2} \rho_i n_i^b, \; i = 1, ..., N.$$

We use $c_0 = 1$ in all cases except when having triangular elements in 2 dimensions where $c_0 = 1.7$ was found to give slightly better convergence.

Denote by M_j $(j = 1, \ldots, n_b)$ the number of subdomains that share node number j on a substructure boundary. Let l_k^j, $k = 1, ..., M_j$ be the indices of the corresponding substructures. The $n_b \times n_b$ matrix A_b is also diagonal with the entries

$$(A_b)_{jj} = c_0 h^{d-2} \sum_{k=1}^{M_j} \rho_{l_k^j}.$$

The matrix A_{ab} has the nonzero entries

$$(A_{ab})_{ij} = -c_0 h^{d-2} \rho_i, \;\; i = 1, ..., N, \, j = 1, ..., n_b$$

in such a way that $A_a u_a + A_{ab} u_b = 0$ is nothing but the relation (6) between the average and the boundary values in each subdomain.

We note that we can store the matrices A_{ab} and A_b in a compact form. The entries corresponding to the boundary nodes that belong to the same set of substructures have the same value. They can be stored as one floating point number for each combination of boundary node owners and a pointer from each boundary node to the corresponding set.

We next describe how to restrict our residual vector r_{ij}, $(i = 1, ..., N, j = 1, ..., n_i)$. Here n_i is the total number of nodes belonging to substructure Ω_i (excluding exterior boundary value nodes of $\partial\Omega$ with a Dirichlet condition). Let I_i be the index set of nodes interior to substructure Ω_i and let v_b denote the vector of boundary nodes extracted from r_{ij}.

We first compute the components of v_a

$$(v_a)_i = \sum_{j \in I_i} r_{ij}, \; i = 1, ..., N. \tag{9}$$

Next, we update the vector v_b according to

$$v_b := v_b - A_{ab}^T A_a^{-1} v_a.$$

In order to solve the system (8) we first compute and store its Schur complement

$$S = A_a - A_{ab} A_b^{-1} A_{ab}^T.$$

This computation is easy due to the structure of the blocks involved. The matrix S is $N \times N$, symmetric and sparse with a nonzero diagonal and additional nonzero entries at positions reflecting the connectivity of substructures in Ω. The contribution from each substructure to S can be computed independently and in parallel. The results are communicated to the process(or) assigned to the coarse solver and stored.

For each coarse problem solution step in the conjugate gradient iteration we must solve

$$Su_a = -A_{ab}A_b^{-1}v_b. \tag{10}$$

Note that the right hand side of (10) can be computed in parallel. Each substructure sends only one floating point number to the coarse solver. After the solution of (10) the values of the vector u_a can be assigned to all interior nodes, that is, there is no further interpolation. The substructure boundary nodes get their values from the formula

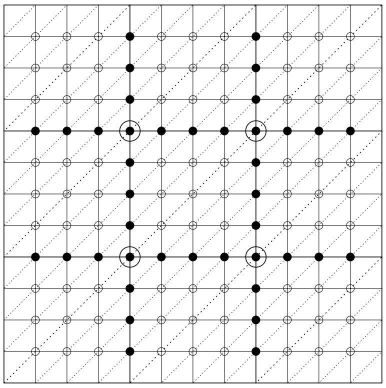
$$u_b = A_b^{-1}(v_b - A_{ab}^T u_a),$$

which again can be computed in parallel.

4 Complexity

We analyze the complexity of the algorithms for a model problem. Consider the case of a regular domain split into m^d subdomains ($d = 2, 3$), each consisting of n^d blocks (see Figure 2 where $d = 2$, $m = 3$, $n = 4$).

Fig. 2. An example of a regular domain divided into 9 subdomains. The dark bullets denote the boundary nodes while the small circles show the inner nodes of the subdomains that take part in the subdomain solves. The four large circles are the nodes that belong to the coarse space in a classical two-level method.



We compare the complexity of one complete conjugate gradient iteration for the Additive and Multiplicative Average method and the classical Additive

Schwarz method with minimal overlap (*i.e.* neighboring subdomains share exactly one node across each common boundary). In Table 1, we give the number

Table 1. Computational complexity for one preconditioning step: classical Additive Schwarz method with minimal overlap compared with the Additive and Multiplicative Average methods.

Task	Classical Additive Schwarz	Additive Average	Multiplicative Average
Subdomain (size)	$(n+1)^d$	$(n-1)^d$	$(n-1)^d$
Coarse prbl. (size)	$(m-1)^d$	m^d	m^d (2 times)
Transfers (flops)	$(18d-3)n^d$	$(2d+15)n^d$	$(6d+19)n^d$
Comm. (words)	$4d\,n^{d-1}$	$2d\,n^{d-1}$	$6d\,n^{d-1}$
Comm. (startups)	$4d+4$	$2d+4$	$6d+5$

of operations for one interior subdomain assuming that all the subdomains are assigned to different processors. We do not include the flops in the coarse problem and the subdomain solution, only the problem sizes are outlined. In this way different solver complexities can be taken into account in any specific implementation. We count the number of words communicated in one send/receive pair and similarly charge such a pair with one communication startup unit. We further only include the highest order terms (in n) in our complexity estimates.

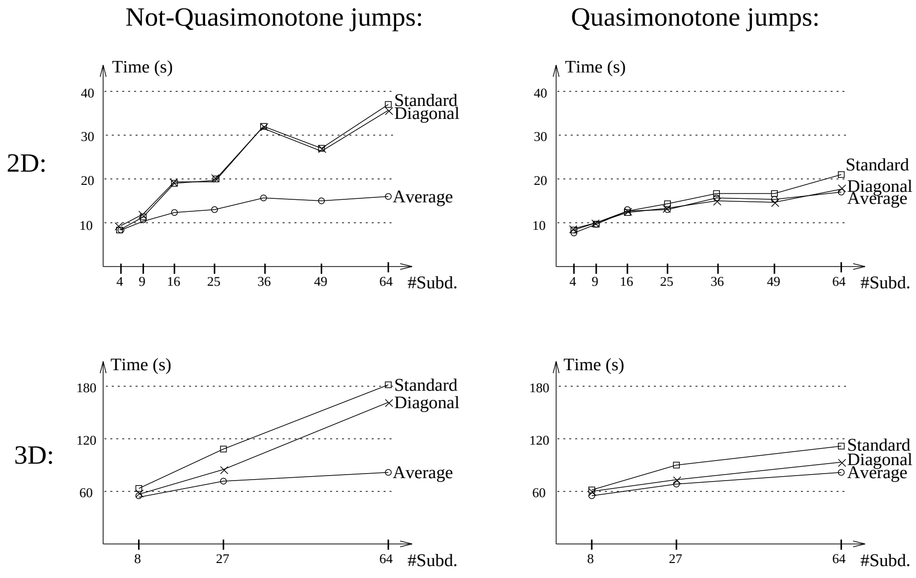
The work in a complete conjugate iteration step can be divided into three parts, the dot products and the vector updates needed in the algorithm itself, the matrix vector product, and the application of the preconditioner. The three vector updates contribute $6n^d$ flops and the three dot products are also $6n^d$ flops and further 3 communication startups. The matrix vector product cost $(2d+1)n^d$ flops. The matrix vector product needs to communicate $2dn^{d-1}$ words to its $2d$ neighbors causing $2d$ startups. These costs are the same for all three methods. The differences in the table are only due to the different cost of the preconditioning step. With no subdomain overlap and no neighbor communication we have only $2n^d$ flops (from equations (4) and (9)) and a single communication startup for the Additive Average method. The standard Additive Schwarz is both more compute intensive in the interpolation/restriction phase and communication intensive with nearest neighbor interaction. The Multiplicative Average method has expensive residual updates that also forces more communication.

The total flop count excluding the solves are listed in the third row of Table 1 called *Transfers*. We have optimized floating point operations over storage, that is, we assume that elementary functions needed in the restriction and the interpolation phase of the classical Additive Schwarz method and some intermediate values in the Average methods are precomputed and stored. Our implementation differs only slightly from this model, and this assumption results in a consistent presentation of the computational complexity of the three methods.

5 Computer Experiments

We have carried out experiments on an Intel Paragon computer and on a cluster of workstations to explore the behavior of the methods from a practical point of view. The standard MPI implementation from Argonne [7] was used for message passing in our algorithms. In practice, it pays to use an inexact solver both for the subdomains as well as for the coarse problem. This can be analyzed by replacing our bilinear forms $b_i(u, v)$ with another form corresponding to the inexact solver. For simplicity we have used two symmetric Gauss-Seidel iterations as our inexact solver throughout this paper. More advanced techniques including a variable number of inner iterations as well as the use of multigrid as an inner solver are possible. Such techniques would reduce the overall computer time since the methods would require fewer iterations to converge. A more accurate subdomain solver tends to be most beneficial for the standard Additive Schwarz algorithm. We note, however, that an accurate solution in this case involves a subdomain problem with a discontinuous coefficient due to the overlap. Such a solution may therefore be more expensive than for the Additive Average method.

Fig. 3. Solution time in seconds for problems with discontinuous coefficients. There are 32×32 gridblocks per subdomain in the 2D and $16 \times 16 \times 16$ gridblocks in the 3D case. The number of subdomains (together with the number of processors) is plotted on the $x-$axis.



In Figure 3 we plot the time in seconds for the Additive Average, the Additive Diagonal Scaling and the standard Additive Schwarz method with minimal overlap. The figure shows algorithm scalability since we keep the size of a subdomain fixed and increase the number of subdomains along with the number of processors. The first row in the figure shows 2-dimensional results while the second row shows a 3-dimensional computation. Similarly, the first column has non quasi-monotone jumps in the coefficients as in Figure 1A, while the second column has a quasi-monotone distribution of ρ_i as given in Figure 1B. We map only one subdomain to each processor. In the 2-dimensional case a subdomain is a 32×32 grid, while we used a $16 \times 16 \times 16$ grid in 3 dimensions.

One should note that as more and more subdomains are added we increase the complexity of the overall (global) problem since we get a similar growth in the number of discontinuous jumps in the coefficients. The parallel scalability of the algorithms should therefore be interpreted with care. A more proper scalability experiment would keep the continuous problem fixed and then refine the discretization and the number of processors such that each processor solves a constant size subproblem.

We observe that the Additive Average method is considerably faster in the case of non quasi-monotone jumps in the coefficients both in 2 and 3 dimensions. In the quasi-monotone case the differences are smaller, but in 3 dimensions the Additive Average method is still the fastest while in 2 dimensions the time is quite close to that of the Additive Diagonal Scaling method. We conclude from the above that both the Additive Diagonal Scaling method and the standard Additive Schwarz method are sensitive to the quasi-monotonicity property while the Additive Average method, as expected (see [1]), show no such dependency. Due to its lower complexity, the Additive Average method seems to have an advantage over the other methods when implemented on a parallel computer.

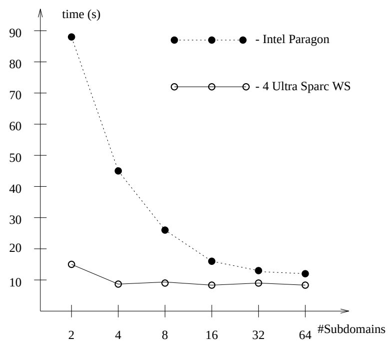
The next tests were carried out on four 200 MHz Ultra Sparc workstations connected with a 10Mbit Ethernet. This interconnect has a rather high latency and relatively low communication bandwidth. The Additive and the Multiplicative Average method are compared. Test results are given for a Poisson problem. The discrete problem size is now fixed at 128×128 quadrilateral elements while the number of subdomains is changed. The processors will compute one subdomain at a time when there are more subdomains than processors. Table 2 shows

Table 2. Time, iteration count and condition number for the Additive Average and the Multiplicative Average method. Ω is discretized using 128×128 quadrilateral elements.

	Additive			Multiplicative		
#Subd.	Time (s)	#it.	cnd.#	Time (s)	#it.	cnd.#
2x2	9.7	81	309.2	10.5	58	137.2
4x4	9.0	62	117.4	9.6	40	44.5
8x8	8.7	44	59.7	9.1	26	17.8
16x16	10.7	43	59.7	10.8	22	16.3
32x32	19.1	50	104.8	19.4	25	28.6

the elapsed computing time, the number of iteration steps and the condition number of the preconditioned system for different number of subdomains. The multiplicative variant uses about half the number of iterations, this is in general agreement with theory. Despite this, due to its higher cost per iteration, it is slightly slower than the additive algorithm in all cases. We also note that the lowest condition number and the best overall times are obtained when the size of the coarse problem is roughly the same as each of the subdomain problems.

Fig. 4. Elapsed time when Ω is discretized in a 128×128 mesh partitioned into a variable number of irregular subdomains.



Finally, in Figure 4 we present results from a case where the subdomains are irregular. Again Ω is discretized using a fixed 128×128 mesh of quadrilateral elements, but the subdomains are created using the Chaco version 2.0 software package [9]. The resulting slightly irregular subdomains are processed by our Intel Paragon, a single subdomain to each node and also by a cluster of four Ultra Sparc workstations with multiple subdomains per processor. The total problem elapsed time is plotted, excluding the partitioning time used by Chaco.

A nice speedup can be observed on the Intel Paragon and also on the cluster of workstations when going from 2 to 4 subdomains. We note that the problem solution time on the workstation cluster is fairly robust with respect to subdomain partitioning as long as there is work to be done on all nodes.

6 Conclusions

New domain decomposition algorithms of both additive and multiplicative type have been implemented as parallel algorithms and tested on both a Paragon computer as well as on a cluster of modern workstations. Based on the experiments performed, the so-called Additive Average method seems to have the best performance, in particular, it is faster than the classical Additive Schwarz method combined with inexact solvers. The new method is also more robust with respect to discontinuous coefficients in the partial differential equation and can easily handle both unstructured meshes and irregular subdomains. We also note that by 1996 a cluster of four workstations typically outperforms a 64 node Paragon, a state of art parallel supercomputer only 3 years earlier.

References

1. P. E. Bjørstad, M. Dryja, and E. Vainikko, *Additive Schwarz methods without subdomain overlap and with new coarse spaces*, in Domain Decomposition Methods in Sciences and Engineering, R. Glowinski, J. Périaux, Z. Shi, and O. B. Widlund, eds., John Wiley & Sons, 1996. Proceedings from the Eight International Conference on Domain Dec omposition Metods, May 1995, Beijing.
2. P. E. Bjørstad and T. Kårstad, *Domain decomposition, parallel computing and petroleum engineering*, in Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering, D. E. Keyes, Y. Saad, and D. G. Truhlar, eds., SIAM, 1995, ch. 3, pp. 39–56.
3. P. E. Bjørstad, R. Moe, and M. Skogen, *Parallel domain decomposition and iterative refinement algorithms*, in Parallel Algorithms for Partial Differential Equations, W. Hackbusch, ed., Braunschweig, Germany, 1991, VIEWEG, pp. 28–46. Notes on Numerical Fluid Mechanics, Vol 31, Proceedings of the sixth GAMM-Seminar, Kiel, January 19-21 1990.
4. P. E. Bjørstad and M. D. Skogen, *Domain decomposition algorithms of Schwarz type, designed for massively parallel computers*, in Domain Decomposition Methods for Partial Differential Equations, D. E. Keyes, T. F. Chan, G. Meurant, J. S. Scroggs, and R. G. Voigt, eds., SIAM, 1992, pp. 362–375. In the proceedings from the Fifth International Symposium on Domain Decomposition Methods, Norfolk, Virginia 1991.
5. J. H. Bramble and J. Xu, *Some estimates for a weighted L^2 projection*, Math. Comp., 56 (1991), pp. 463–476.
6. M. Dryja, M. V. Sarkis, and O. B. Widlund, *Multilevel Schwarz methods for elliptic problems with discontinuous coefficients in three dimensions*, Numer. Math., 72 (1996), pp. 313–348.
7. W. D. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1994.
8. W. D. Gropp and B. F. Smith, *Experiences with domain decomposition in three dimensions: Overlapping Schwarz methods*, Contemporary Mathematics, (1991).
9. B. Hendrickson and R. Leland, *The Chaco user's guide, version 2.0*, Tech. Report SAND 94-2692, Sandia National Laboratories, July 1995.
10. D. E. Keyes and W. D. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 166 – 202.
11. P. Oswald, *On the robustness of the BPX-preconditioner with respect to jumps in the coefficients*, tech. report, Dept. of Math. Texas A&M University, College Station, TX 77843-3368, 1995.
12. B. F. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.
13. J. Xu, *Counter examples concerning a weighted L^2 projection*, Math. Comp., 57 (1991), pp. 563–568.