Contemporary Mathematics
Volume 180, 1994

A comparison of three iterative algorithms based on domain decomposition methods

PATRICK CIARLET JR

ABSTRACT. In this paper we compare three domain decomposition preconditioners for the capacitance matrix and the conjugate gradient methods to solve linear systems arising from the discretization of elliptic partial differential equations. The methods have been implemented on parallel and superscalar architectures. Numerical experiments show that using these preconditioners leads to competitive algorithms, both in terms of number of iterations and parallelization rates.

1. Introduction

In this work, we compare the parallel and superscalar implementations of three different iterative methods designed to solve the finite element discretization of elliptic problems such as

$$-\text{div}\,(\lambda\,\nabla u) = \alpha \text{ in } \Omega, \quad \text{with } \lambda(x,y) = \begin{pmatrix} a(x,y) & 0 \\ 0 & b(x,y) \end{pmatrix}$$
$$u = 0 \text{ on } \partial\Omega.$$

Here $\Omega =]0,1[\times]0,1[$ and a, b and α are given functions, a and b being non-negative over the domain. We approximate these problems by using a P_1 finite element method with right triangles, leading to a five-point centered scheme. Note that by doing this, we are able to handle problems with discontinuous coefficients without any difficulty, as long as the jumps occur on the sides of the triangles. This gives us a linear system $Ax = f$, with a n by n symmetric positive definite matrix A.

We propose to solve this linear system by using either the Capacitance Matrix and Conjugate Gradient methods or the Preconditioned Conjugate Gradient (**PCG**) method. Our goal is to obtain a good trade-off between the convergence

1991 *Mathematics Subject Classification.* 65F10, 65F50, 65N55.
This paper is in final form and no version of it will be submitted for publication elsewhere.

0271-4132/94 $1.00 + $.25 per page

rate for the iterative methods and the Mflops rate on the computers. In order to define the preconditioners, we use domain decomposition methods based on a partition of the domain into boxes or strips. Moreover, to be able to use the iterative methods for solving large problems, we focus our interest on preconditioners requiring reasonable storage, i.e. "sparse" preconditioners.

In the next section, we briefly recall the definition of the Capacitance Matrix method. In section 3, we define the preconditioners. Finally, we compare our iterative methods in the last section.

2.The Capacitance Matrix Method

This method was first investigated by Buzbee and al in [**1**]. Let B be a nonsingular n by n matrix and S an extension matrix such that, using block notation, we have

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \; B = \begin{pmatrix} A_{11} & A_{12} \\ B_{21} & B_{22} \end{pmatrix} \text{ and } S = \begin{pmatrix} 0 \\ I_q \end{pmatrix}.$$

Here only the last q rows of A and B differ. The q by q matrix $C = S^T A B^{-1} S$ is called the capacitance matrix. One can easily prove that the capacitance matrix is nonsingular. Then the original problem $Ax = f$ can be replaced by

$$\begin{cases} Bv = h, \text{ with } h_1 = f_1, \;\; h_2 = 0, \\ C\omega = g, \text{ with } g = S^T(f - Av), \\ Bx = h + S\omega. \end{cases}$$

If C is small enough, then a direct solver can be used. Otherwise the linear system with C can be solved iteratively. In the following, we use either the Diagonally Preconditioned Conjugate Gradient (**DPCG**) method or a Conjugate Gradient-like method with matrix B as a "preconditioner", see [**11**] and [**2**]. Particularly, this means that one linear system in B is solved at each iteration for the second method.

3.The DD preconditioners

The construction of the preconditioners is based on partitioning the domain as indicated on Fig. 1, that is either into strips or boxes. We propose three preconditioners in the following. The first one is based on the partition into strips and the other two on the partition into boxes. Note that here we consider only nonoverlapping subdomains (strips or boxes).

The stripwise partition was introduced by Dryja and Proskurowski in [**6**]. Briefly, they defined a preconditioner of the original problem by keeping the same operator inside the strips and adding boundary conditions on the interfaces: Dirichlet boundary conditions for the white strips and Neumann boundary conditions for the black strips. Here, we replace the Neumann boundary conditions

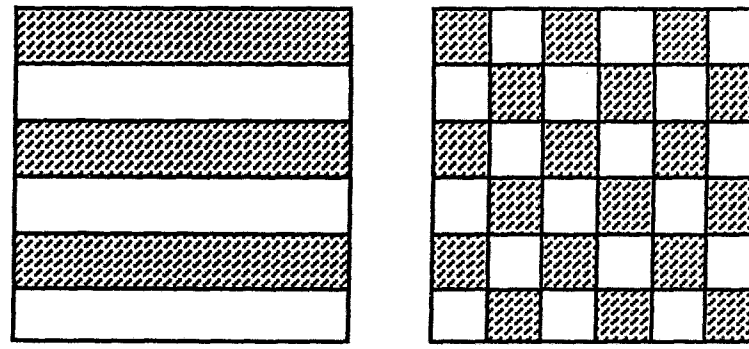

FIGURE 1 DECOMPOSITIONS OF THE DOMAIN

by mixed Neumann-Dirichlet boundary conditions for the black strips, i.e.

$$\mu u + \frac{\partial u}{\partial n} = 0, \quad \mu > 0.$$

Indeed, taking a prescribed nonnegative value for μ improves greatly the convergence rate of the method[1]. See [3] for numerical examples and details of implementation. Let B_1 be the discretized preconditioner. We have three types of unknowns: the nodes in the white strips (W), the nodes in the black strips (B) and the nodes on the interfaces called separators (S).

If we rewrite A and B_1 by block, it can easily be seen that A and B_1 differ only by their SS diagonal block. We therefore use the Capacitance Matrix and Conjugate Gradient methods to solve the problem in this case. Also, solving one linear problem with B_1 can be done in two steps:

(1) one parallel solve on the extended black strips,

(2) one parallel solve on the white strips.

Extended black strips include their surrounding separators. The resulting method is called **Cap1**.

The boxwise partitioning was introduced by Proskurowski and al in [7], [8] and [10]. The second preconditioner B_2 is similar to B_1, except that purely Neumann boundary conditions are considered for the black boxes. Indeed, for a variety of numerical examples ([3] or [4]), one can see that when the domain is divided into boxes, then the purely Neumann boundary condition is "optimal". Adding a fourth type of unknowns: the crosspoints (C), we use the same strategy as before to derive an iterative method also based on the Capacitance Matrix and Conjugate Gradient methods. Now, solving a linear problem with B_2 is done in two steps:

(1) one solve on the extended black boxes and the crosspoints,

(2) one parallel solve on the white boxes.

[1]This is also true for overlapping strips, as proved by Tang [12].

In short, to uncouple the extended black boxes (which include their surrounding separators), we introduce a problem defined only on the crosspoints. The corresponding matrix is sparse with at most seven nonzero entries per row (see [**2**]). The problem (1) which was originally defined on the extended black boxes and the crosspoints is replaced by two problems on the extended black boxes and one problem on the crosspoints:

(1.1) two parallel solves on the extended black boxes,

(1.2) one (parallel) solve on the crosspoints.

The problem defined on the crosspoints is solved by the DPCG method to get a parallel solver. The resulting method is called **Cap2**.

The third preconditioner, called M, is equal to one of the preconditioners studied by Ciarlet and Meurant. M corresponds to the second preconditioner in their terminology; see [**5**] for details. Solving a linear problem with M requires:

(1) two parallel solves on the (black and white) boxes,

(2) two parallel solves on the separators,

(3) one (parallel) solve on the crosspoints.

Step (2) is parallel because it can be shown that M_{SS} is block diagonal, each block corresponding to the nodes around a black box. For step (3), the DPCG method is used. As a matter of fact, the problem defined on the crosspoints for B_2 and M are identical. This last method, based on the PCG method, is called **MPCG**.

4. A few numerical examples

We solve the linear system $Ax = f$ on two computers. The first one is a Sequent Symmetry S81 with 20 Intel 80386 processors: a shared memory parallel architecture. The second computer is an IBM RS6000/560 with a RisC6000 processor: a superscalar architecture. The programming language is Fortran 77. We run the same code on both machines, with a preprocessing step using KAP [**9**], an automatic parallelizer, on the Sequent.

We will study the iterative methods in terms of

(1) the number of floating operations to solve the linear problem,

(2) the number of floating operations per second,

(3) the speed-up on the parallel architecture, from 1 to 16 processors,

(4) the CPU times.

Moreover, we will compare these results with those obtained for a well known iterative parallel method, the **DPCG** method.

The set of problems to be solved is

Problem #1

$a = 1$ in Ω

$b = 1$ in Ω

Problem #2

$$a = \begin{cases} 10 & \text{if } 0 \le y < \frac{1}{2} \\ 1 & \text{elsewhere} \end{cases}$$

$$b = \begin{cases} 10 & \text{if } \frac{1}{2} < x \le 1 \\ 1 & \text{elsewhere} \end{cases}$$

Problem #3

$$a = \begin{cases} 10^2 & \text{if } \frac{1}{4} \le x \le \frac{3}{4} \\ 1 & \text{elsewhere} \end{cases}$$

$b = 1$ in Ω

We set the number of unknowns, n, to 65025 ($= 255^2$). As we iteratively solve the problems, a stopping criterion is prescribed: it is reached when the norm of the residual has been reduced by a factor of 10^{-6}.

Then, we have to choose the number of strips (called n_0) and the number of boxes (called n_0^2). The bigger this number is, the greater the parallelism of the method, as $\frac{1}{2}n_0$ (resp., $\frac{1}{2}n_0^2$, n_0^2) subproblems are solved in parallel for **Cap1** (resp., **Cap2**, **MPCG**). However, note that by the way the domain is partitioned (Fig. 1), n_0 can, at most, be as large as $\frac{1}{4}\sqrt{n}$. Finally, recall that we want "sparse" preconditioners: here, this means that the number of nonzero entries of the preconditioners is proportional to n, as the storage of the original matrix and the vectors for the DPCG method is approximately $11n$. In [**3**], it is shown that this also leads to a value of n_0 proportional to $\sqrt{n}$. Therefore, potentially parallel preconditioners are "sparse" and *vice versa*. In the following, we choose $n_0 = 64$ for the strips and $n_0^2 = 32^2$ for the boxes, leading to the following storage for the original matrix, the preconditioner and the vectors for each iterative method: $20n$ for **Cap1**, $37n$ for **Cap2** and $43n$ for **MPCG**.

The parameter μ, arising in the mixed Neumann-Dirichlet boundary condition for **Cap1**, is set to n_0.

Table 1 compares the number of floating operations for Problem #1.

Table 1

Number of floating operations

Method	**DPCG**	**Cap1**	**Cap2**	**MPCG**
FLOP	928.10^6	343.10^6	244.10^6	312.10^6

The results are quite similar across all methods. The reference method gives by far the worst results. The numbers of iterations required to solve Problem #1 are the following: 645 for **DPCG**, 87 for **Cap1**, 12 for **Cap2** and 17 for **MPCG**. Unfortunately for the box-based methods, the average number of subiterations to solve the problem defined on the crosspoints is equal to 61 for the first method and 63 for the other.

Tables 2 and 3 give the average number (over the three problems) of floating point operations per second for each method. The results are in MFLOPs (millions of FLOPs). Table 2 gathers the results obtained on the RS6000. In Table 3, results are given for one and sixteen processors of the Sequent, and the corresponding speed-up is derived.

Table 2

Number of floating operations per second on the RS6000

Method	**DPCG**	**Cap1**	**Cap2**	**MPCG**
MFLOPs	25	24	20	16

Table 3
Number of floating operations per second and speed-up [bracketed] on the Sequent

Method	**DPCG**	**Cap1**	**Cap2**	**MPCG**
$\text{MFLOPs}_{\{1\}}$	0.31	0.41	0.32	0.38
$\text{MFLOPs}_{\{16\}}$	4.8 [15.7]	5.5 [13.6]	3.6 [11.3]	3.9 [10.5]

Surprisingly, the reference method is not the fastest one on the parallel architecture, though it is for this method that the highest speed-up is reached. In terms of parallel versus sequential parts of the execution of the code, using Amdahl's law, one finds that well over 99% of the reference method is executed in parallel, in comparison to a little less than 99% for the strip-based method and around 97% for the box-based methods. Note that these box-based implementations are actually less parallelized than the strip-based one, although they are potentially more parallelizable.

Finally, we show the CPU times needed to solve Problem #2 on the RS6000 in Table 4 and the CPU times on the 16-processor Sequent to solve Problem #3 in Table 5.

Table 4
CPU times on the RS6000

Method	**DPCG**	**Cap1**	**Cap2**	**MPCG**
time (s)	87	17	16	27

Table 5
CPU times on the 16-processor Sequent

Method	**DPCG**	**Cap1**	**Cap2**	**MPCG**
time (s)	990	47	161	177

The best methods are **Cap1** and **Cap2**, though both box-based methods are competitive versus the strip-based method. Moreover, a closer look at the CPU time shows that one third of the time is spent in the subroutines computing the matrix defined on the crosspoints for the box-based methods. Therefore, one way to reduce the CPU time for these methods is to find a better way to compute this matrix. Note that the strip-based method is particularly efficient for Problem #3, although the first coefficient of the operator λ is only piecewise constant inside each strip.

4. Conclusion

In this paper, we have presented three preconditioners for CG-like methods that seem well suited for parallel architectures. We have shown that the three of them compare very favorably to a fully parallel method. Nevertheless, the box-based methods have to be implemented on a massively parallel architecture

to further enhance their merit, as they are potentially more parallelizable than a strip-based method. For the box-based methods, a solution has to be found to the very costly computation of the matrix defined on the crosspoints. Also, an efficient solver has to be derived for this matrix.

REFERENCES

1. B. Buzbee, F. Dorr, A. George and G. Golub, *The direct solution of the discrete Poisson equation in irregular regions*, SIAM J. Numer. Anal. **8** (1971), 722–736.
2. P. Ciarlet, Jr, *Méthodes itératives de résolution de problèmes elliptiques en 2D adaptées à des architectures (massivement) parallèles*, Technical Report, Note CEA 2688 (1992).
3. P. Ciarlet, Jr, *Etude de préconditionnements parallèles pour la résolution d'équations aux dérivées partielles elliptiques. Une décomposition de l'espace* $L^2(\Omega)^3$, Ph.D. thesis, Univ. Paris VI, France, 1992.
4. P. Ciarlet, Jr, *Implementation of a domain decomposition method well-suited for parallel architectures*, Int. J. of High Speed Computing (to appear).
5. P. Ciarlet, Jr and G. A. Meurant, *A class of domain decomposition preconditioners for massively parallel computers*, Domain Decomposition Methods in Science and Engineering (A. Quarteroni, J. Périaux, Y. A. Kuznetzov and O. B. Widlund, eds.), AMS, 1994, pp. 353–359.
6. M. Dryja and W. Proskurowski, *Capacitance matrix method using strips with alternating Neumann and Dirichlet boundary conditions*, Applied Numer. Math. **1** (1985), 285–298.
7. M. Dryja, W. Proskurowski and O. Widlund, *Numerical experiments and implementation of a domain decomposition method with cross points for the model problem*, Advances in Computer Methods for Partial Differential Equations VI (R. Vichnevetsky and R.S. Stepleman, eds.), IMACS, 1987, pp. 23–27.
8. M. Haghoo and W. Proskurowski, *Parallel implementation of a domain decomposition method*, Technical Report, CRI 88–06 (1988).
9. Kuck and Associates, Inc., *KAP/Sequent user's guide version 6*, 1988.
10. W. Proskurowski and S. Sha, *Performance of the Neumann-Dirichlet preconditioner for substructures with intersecting interfaces*, Domain Decomposition Methods for Partial Differential Equations (T.F. Chan, R. Glowinski, J. Périaux and O. Widlund, eds.), SIAM, 1990, pp. 322–337.
11. W. Proskurowski and O. Widlund, *A finite element-capacitance matrix method for the Neumann problem for Laplace's equation*, SIAM J. Sci. Stat. Comput. **1** (1980), 410–425.
12. W. P. Tang, *Generalized Schwarz splittings*, SIAM J. Sci. Stat. Comput. **13** (1992), 573–595.

CEA, CEL-V/ D.MA, 94195 VILLENEUVE ST GEORGES CEDEX, FRANCE

E-mail address: ciarlet@limeil.cea.fr