Domain Decomposition Versus Block Preconditioning

GÉRARD MEURANT*

Abstract

In this paper, we propose a new class of domain decomposition preconditioners for the conjugate gradient algorithm.

These methods use the block structure of the matrix given by finite difference schemes for 2D elliptic partial differential equations.

The techniques developed in [5] for the approximation of inverses of tridiagonal matrices are used to generate the matrix of the problem to be solved on the interfaces between subdomains.

These methods are very well suited for parallel computation, hence, we compare them with the parallel preconditioners that we proposed in [9].

Preliminary numerical results indicate that it can be beneficial to use domain decomposition when there is a large number of processors on the given parallel computer.

---

*Centre d'Etudes de Limeil-Valenton, BP 27, 94190 Villeneuve St Georges, France.

1. Introduction

In this paper, we are concerned with finding preconditioners for the conjugate gradient method for solving symmetric positive definite linear systems arising from a finite difference discretization of elliptic partial differential equations in two dimensional domains.

For such problems with natural (row-wise) ordering of the unknowns, matrices are block tridiagonal and recently, efficient block preconditioners have been devised ([5],[1]). Variants of these methods have been given for vector and parallel computers ([8],[9],[1]).

In the last several years, there has been a great interest in domain decomposition methods ([2],[3]). Some of these methods were defined using the block structure of the matrix, each block being related to one of the subdomains.

Here we show how to devise preconditioners, combining domain decomposition and some techniques of [5],[8]. These preconditioners will be well suited for parallel computation and we will compare them with the block preconditioners given in [9].

In Section 2, we give briefly formulas for an exact block decomposition when the rectangular domain is partitioned into strips.

In Section 3 we show how this allows us to re-derive results by T. Chan [4] leading to parallel fast solvers for separable equations. The method we used to obtain these results is different from the one given in [4]. They can be extended to general problems and hence we believe it is useful to give this derivation.

Section 4 introduces the domain decomposition preconditioners; Section 5 recalls some fact about the block preconditioners given in [9] and shows they can be considered as extensions and simplifications of domain decomposition preconditioners.

In Section 6, we present results of some preliminary numerical experiments. More details and the study of the case of subdomains with cross points will be given in another paper [10].

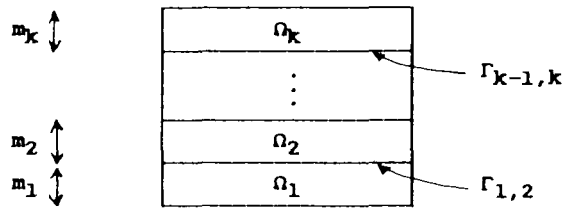
2. An Exact Domain Decomposition Solver

For elliptic p.d.e.'s on a rectangular domain Ω, we consider a five point finite difference scheme and this leads to a matrix A which is block tridiagonal :

$$A = \begin{bmatrix} D_1 & A_2^T & & & & & \\ A_2 & D_2 & A_3^T & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & A_{m-1} & D_{m-1} & A_m^T \\ & & & & & A_m & D_m \end{bmatrix} .$$

Each matrix D_i is point tridiagonal and the matrices A_i are diagonal. For the Poisson model problem $D_i = 2\,I + D$, where D is the tridiagonal matrix of the one dimensional Laplacian : tridiag(-1,2,-1), $A_i = -I$, I being the identity matrix.

Consider the domain Ω to be partitioned into k strips like as given below :



There are m_i lines in each domain Ω_i, excluding the interfaces, the total number of lines is $m=m_1+m_2+\ldots+m_k+k-1$.

In order to balance the workload on a parallel computer, it is advisable that all the number of lines in each domain be approximately equal.

We denote the unknowns associated with the domain Ω_i as x_i and those with the interface $\Gamma_{i,i+1}$ as $x_{i,i+1}$. We re-order the equations in the following form

$$\begin{bmatrix} B_1 & & & C_1 & & & \\ & B_2 & & E_2^T & C_2 & & \\ & & \ddots & & \ddots & \ddots & \\ & & B_k & & & & C_{k-1} \\ C_1^T & E_2^T & & B_{12} & & & E_k \\ & C_2^T & E_3^T & & B_{23} & & \\ & & \ddots & & & \ddots & \\ & & C_{k-1}^T & E_k^T & & & B_{k-1,k} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ x_k \\ x_{12} \\ \cdot \\ x_{k-1,k} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ b_k \\ b_{12} \\ \cdot \\ b_{k-1,k} \end{bmatrix} \quad (2.1)$$

Each B_i is block tridiagonal

$$B_i = \begin{bmatrix} D_i^2 & A_i^{2\,T} & & & \\ A_i^2 & D_i^2 & A_i^{3\,T} & & \\ & \ddots & \ddots & \ddots & \\ & & A_i^{m-1} & D_i^{m-1} & A_i^{m\,T} \\ & & & A_i^m & D_i^m \end{bmatrix} \quad (2.2)$$

Each block being of order n.

Matrices C_i and E_i have a very simple structure

$$C_i^T=(0,\ldots.0,C_i^{m_i}) \qquad E_i^T=(E_i^1,0,\ldots,0) \tag{2.3}$$

with $C_i^{m_i}$ and E_i^1 being diagonal.

For the domain decomposition method the x_i's are eliminated to get a reduced system for the x_{ij}'s. This yields

$$\begin{bmatrix} B'_{12} & F_2^T & & & \\ F_2 & B'_{23} & F_3^T & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & F_{k-1}^T \\ & & & F_{k-1} & B'_{k-1,k} \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{23} \\ \vdots \\ \vdots \\ x_{k-1,k} \end{bmatrix} = \begin{bmatrix} b'_{12} \\ b'_{23} \\ \vdots \\ \vdots \\ b'_{k-1,k} \end{bmatrix} \tag{2.4}$$

where

$$\begin{aligned} B'_{ij} &= B_{ij} - C_i^T B_i^{-1} C_i - E_j^T B_j^{-1} E_j, \\ F_i &= - C_i^T B_i^{-1} E_i, \\ b'_{ij} &= b_{ij} - C_i^T B_i^{-1} b_i - E_j^T B_j^{-1} b_j. \end{aligned} \tag{2.5}$$

To obtain a nicer expression for B'_{ij} we need to compute two Choleski decompositions for each B_i.

Let

$$\Delta_i=\begin{bmatrix} \Delta_i^1 & & \\ & \ddots & \\ & & \Delta_i^{m_i} \end{bmatrix} \qquad L_i=\begin{bmatrix} 0 & & \\ A_i^2 & \ddots & \\ & A_i^{m_i} & 0 \end{bmatrix} \qquad \Gamma_i=\begin{bmatrix} \Gamma_i^1 & & \\ & \ddots & \\ & & \Gamma_i^{m_i} \end{bmatrix}.$$

Consider the equation

$$B_i = (\Delta_i + L_i) \Delta_i^{-1} (\Delta_i + L_i^T) \tag{2.6}$$

By identification

$$\begin{aligned} \Delta_i^1 &= D_i^1 \\ \Delta_i^j &= D_i^j - A_i^j (\Delta_i^{j-1})^{-1} (A_i^j)^T , \quad j=2,\ldots.,m_i. \end{aligned} \tag{2.7}$$

This is called an LU or top-down decomposition. We also consider the following decomposition

$$B_i = (\Sigma_i + L_i^T) \Sigma_i^{-1} (\Sigma_i + L_i). \tag{2.8}$$

Then

$$\begin{aligned} \Sigma_i^{m_i} &= D_i^{m_i} \\ \Sigma_i^{j} &= D_i^{j} - (A_i^{j+1})^T (\Sigma_i^{j+1})^{-1} A_i^{j+1}, \quad j=m_i-1,\ldots,1. \end{aligned} \tag{2.9}$$

Equation (2.8) yields a UL or bottom-up decomposition. Then, we have the following theorem

Theorem 2.1

$B_{ij}^{'}$ is given by formula (2.10) :

$$B_{ij}^{'} = B_{ij} - (C_i^{m_i})^T (\Delta_i^{m_i})^{-1} C_i^{m_i} - (E_j^1)^T (\Sigma_j^1)^{-1} E_j^1$$

This theorem is a consequence of (2.5) and from the solution of the systems

$$B_i X = C_i \quad \text{and} \quad B_j X = E_j.$$

To get an expression for F_i, let G_i^j to be defined as

$$\begin{aligned} G_i^1 &= (\Sigma_i^1)^{-1} E_i^1 \\ G_i^j &= - (\Sigma_i^j)^{-1} A_i^j G_i^{j-1}, \quad j=2,\ldots,m_i \end{aligned} \tag{2.11}$$

then

Theorem 2.2

$$F_i = - C_i^{m_i} G_i^{m_i} \tag{2.12}$$

This is a result derived from solving $B_i X = E_i$.

The problem we must now consider is that of solving the reduced system, since the matrices $B_{ij}^{'}$ and F_i are no longer sparse. However, in the next section, we are going to show that for some separable equations, this method reduces to a Fast Solver.

3. Parallel Fast Solvers

It was noted in [7] that for separable equations, the block Cholesky decomposition can be reduced to a fast solver (using FFT). This technique can also be used with the domain decomposition method described in Section 2. We derive the results of [4] for the model problem and an easy way to generalise them to more complex situations.

In this section, we consider the Poisson problem. Now the spectral decomposition of the diagonal blocks is given by

$$D_i = Q \Lambda Q^T$$

where Λ is diagonal and Q orthogonal ($Q Q^T = I$). Note,

$$(\Lambda)_{jj} = 4 - 2 \cos (j \pi / (n + 1)) = 2 + \sigma_j \quad (3.1)$$

where $\sigma_j = 4 \sin^2 (j \pi / 2(n + 1))$. The σ_j's are the eigenvalues of the one dimensional Laplacian D.

Lemma 3.1

Consider decompositions (2.6) and (2.8), then

$$\Delta_i^j = Q \Lambda_i^j Q^T,$$

$$\Sigma_i^j = Q \Omega_i^j Q^T.$$

The matrices Λ_i^j and Ω_i^j being diagonal matrices whose elements are given by

$$(\Lambda_i^1)_{pp} = \Lambda_{pp}$$

$$(\Lambda_i^j)_{pp} = \Lambda_{pp} - 1 / (\Lambda_i^{j-1})_{pp} \qquad j=2,\ldots,m_i \quad (3.2)$$

$$(\Omega_i^{m_i})_{pp} = \Lambda_{pp}$$

$$(\Omega_i^j)_{pp} = \Lambda_{pp} - 1 / (\Omega_i^{j+1})_{pp} \qquad j=m_i-1,\ldots,1. \quad (3.3)$$

The proof of this result is given in [7]. Lemma 3.1 implies

Theorem 3.2

$$B_{ij}^{-1} = Q (\Lambda - (\Lambda_i^{m_i})^{-1} - (\Omega_j^1)^{-1}) Q^T \quad (3.4)$$

Lemma 3.3

$$G_i^j = Q\, \Theta_i^j\, Q^T$$

Θ_i^j being diagonal matrices with

$$\Theta_i^1 = -(\Omega_i^1)^{-1}$$

$$\Theta_i^j = (\Omega_i^j)^{-1}\, \Theta_i^{j-1}$$

Therefore $\Theta_i^{m_i} = -(\Omega_i^1)^{-1} \ldots (\Omega_i^{m_i})^{-1}$ and

Theorem 3.4

$$F_i = Q\, \Theta_i^{m_i}\, Q^T. \qquad (3.5)$$

To compute elements of Λ's and Ω's, we consider sequences similar to that given by (3.2). This can be written as

$$\lambda_1 = \sigma$$
$$\lambda_i = \sigma - 1 / \lambda_{i-1}, \quad i=1,\ldots. \qquad (3.6)$$

Solving the difference equation (3.6), we have the following

Proposition 3.5

For $\sigma \neq 2$, the solution λ_i to (3.6) is

$$\lambda_i = (r_+^{i+1} - r_-^{i+1}) / (r_+^i - r_-^i)$$

where r_+ and r_- are the solutions of $r^2 - \sigma r + 1 = 0$.

Theorem 3.6

The eigenvalues μ_{ij}^p of B_{ij} are given by (3.7)

$$\mu_{ij}^p = 2 + \sigma_p - (r_+^{m_i} - r_-^{m_i}) / (r_+^{m_i+1} - r_-^{m_i+1}) - (r_+^{m_j} - r_-^{m_j}) / (r_+^{m_j+1} - r_-^{m_j+1}),$$

and the eigenvalues δ_i^p of F_i are

$$\delta_i^p = (r_+ - r_-) / (r_+^{m_i+1} - r_-^{m_i+1}) \qquad (3.8)$$

where r_+ and r_-, which depend on index p, are given by

$$r_\pm = 1 + \sigma_p/2 \pm (\sigma_p + \sigma_p^2/4)^{1/2}.$$

For more complex problems, recurrences for computing the eigenvalues can be solved numerically using analogs of (3.2) and (3.3).

We now, describe a method for solving the reduced system (2.4). We can factor the system (2.4) as

$$\begin{bmatrix} K_{12} & & & \\ F_2 & K_{23} & & \\ & \ddots & \ddots & \\ & & F_{k-1} & K_{k-1,k} \end{bmatrix} \begin{bmatrix} K_{12}^{-1} & & & \\ & K_{23}^{-1} & & \\ & & \ddots & \\ & & & K_{k-1,k}^{-1} \end{bmatrix} \begin{bmatrix} K_{12} & F_2^T & & \\ & K_{23} & F_3^T & \\ & & \ddots & F_{k-1}^T \\ & & & K_{k-1,k} \end{bmatrix}$$

Then

$$K_{12} = B'_{12}$$

$$K_{ij} = B'_{ij} - F_i \left(B'_{i-1,j-1}\right)^{-1} F_i^T.$$

From the spectral decompositions of B'_{ij} and F_i, and noting that K_{ij} has the same eigenvectors as B'_{ij} and F_i, we are able to compute the eigenvalues ω_{ij}^p of K_{ij}

Theorem 3.7

$$\omega_{12}^p = \mu_{12}^p$$

$$\omega_{ij}^p = \mu_{ij}^p - (\delta_i^p)^2 / \omega_{i-1,j-1}^p$$

The reduced system can be efficiently solved using the FFT and solving factored (independent) tridiagonal systems.

Notice that, when k=2, the matrix of the reduced system is B'_{12} and when the number of lines tends to ∞, as we have

$$\lim_{m_i \to \infty} (r_+^{m_i} - r_-^{m_i}) / (r_+^{m_i+1} - r_-^{m_i+1}) = 1 / r_+ = r_-.$$

Then,

$$\mu_{12}^p \to 2 + \sigma_p - 2\, r_- = 2\,(\sigma_p + \sigma_p^2/4)^{1/2}$$

which is similar to the result derived in [6].

Note that the method described above can be extended to more general problems than the Poisson problem, when the matrices D_i^j and A_i^j have the same eigenvectors.

4. Domain Decomposition Preconditioners

In this section we are interested in developing preconditioners for the matrix A. The method for constructing preconditioners will use the formulas of Section 1, approximating the inverses of tridiagonal matrices with sparse matrices.

Now, consider the situation with two strips. The system to be solved is as follows :

$$\begin{bmatrix} B_1 & 0 & C \\ 0 & B_2 & E \\ C^T & E^T & B_{12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_{12} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_{12} \end{bmatrix} \qquad (4.1)$$

Suppose we have symmetric approximations M_i to B_i and M_{12} to $B'_{12} = B_{12} - C^T B_1^{-1} C - E^T B_2^{-1} E$. Then, we take as a preconditioner

$$M = \begin{bmatrix} M_1 & 0 & 0 \\ 0 & M_2 & 0 \\ C^T & E^T & M_{12} \end{bmatrix} \begin{bmatrix} M_1^{-1} & 0 & 0 \\ 0 & M_2^{-1} & 0 \\ 0 & 0 & M_{12}^{-1} \end{bmatrix} \begin{bmatrix} M_1 & 0 & C \\ 0 & M_2 & E \\ 0 & 0 & M_{12} \end{bmatrix} \qquad (4.2)$$

Obviously,

$$M = \begin{bmatrix} M_1 & 0 & C \\ 0 & M_2 & E \\ C^T & E^T & M_{12}^* \end{bmatrix}$$

with $M_{12}^* = M_{12} + C^T M_1^{-1} C + E^T M_2^{-1} E$, so it is likely that M_{12}^* approximates B_{12}.

For the general case with k strips, we write

$$M = L \begin{bmatrix} M_1^{-1} & & & & & & \\ & \ddots & & & & & \\ & & M_k^{-1} & & & & \\ & & & M_{12}^{-1} & & & \\ & & & & \ddots & & \\ & & & & & M_{k-1,k}^{-1} \end{bmatrix} L^T \quad (4.3)$$

with

$$L = \begin{bmatrix} M_1 & & & & & & \\ & M_2 & & & & & \\ & & \ddots & & & & \\ C_1^T & E_2^T & & M_k & & & \\ & C_2^T & E_3^T & & M_{12} & & \\ & & \ddots & & H_2 & M_{23} & \\ & & & C_{k-1}^T & E_k^T & & H_{k-1} \quad M_{k-1,k} \end{bmatrix} . \quad (4.4)$$

To construct the M_i's and M_{ij}'s, we need two approximate decompositions

i) approximate L U or top-down

$$M_i = (\Delta_i + L_i)\, \Delta_i^{-1}\, (\Delta_i + L_i^T) \qquad (4.5)$$

$$\Delta_i^1 = D_i^1$$

$$\Delta_i^j = D_i^j - A_i^j\, \Omega(1)_i^{j-1}\, (A_i^j)^T \quad j=2,\ldots,m_i$$

where $\Omega(p)_i^j$ is a 2p+1 banded matrix whose non zero entries are the same as those of $(\Delta_i^j)^{-1}$

ii) approximate U L or bottom-up

$$M_i = (\Sigma_i + L_i^T)\, \Sigma_i^{-1}\, (\Sigma_i + L_i) \qquad (4.6)$$

$$\Sigma_i^{m_i} = D_i^{m_i}$$

$$\Sigma_i^j = D_i^j - (A_i^{j+1})^T\, \Pi(1)_i^{j+1}\, A_i^{j+1}, \quad j=m_i-1,\ldots,1$$

where $\Pi(p)_i^j$ is a 2p+1 banded matrix whose non zero entries are the same as those of $(\Sigma_i^j)^{-1}$.

The matrices H_i are defined as diagonal matrices given by

$$\begin{aligned} G_i^1 &= \text{diag}(\Pi(1)_i^1 E_i^1) \\ G_i^j &= - \text{diag}(\Pi(1)_i^j A_i^j G_i^{j-1}) \end{aligned} \qquad (4.7)$$

$$H_i = - C_i^{m_i} G_i^{m_i}$$

Note that if we do not make an approximation at each stage, the matrices G_i^j "fill in" as j increases.

H_i is an approximation to the matrix $- C_i^T M_i^{-1} E_i$. This explains why we did not use the reduced system for M.

Then, M_{ij} is defined as follows :

$$\begin{aligned} M_{12} &= B_{12} - (C_1^{m_1})^T \Omega(1)_1^{m_1} C_1^{m_1} - (E_2^1)^T \Pi(1)_2^1 E_2^1 \\ M_{ij} &= B_{ij} - (C_i^{m_i})^T \Omega(1)_i^{m_i} C_i^{m_i} - (E_j^1)^T \Pi(1)_j^1 E_j^1 \\ &\quad - B_i \text{ tridiag } (M_{i-1,j-1}^{-1}) H_i^T. \end{aligned} \qquad (4.8)$$

Tridiag(B) is a tridiagonal matrix whose non zero entries are the same as the corresponding entries of B.

Then the lower right block of L is an approximate factor of the reduced system for M.

We will call this preconditioner DD1. A variant, DD2, will be obtained by taking $H_i = 0$, for all i.

Using the techniques of [5], it is possible to show that these factorizations yield positive definite matrices if A is an M-matrix.

Note that, for DD1 (or DD2), we must solve two linear problems for each M_i at every iteration. For k=2 this can be improved upon, because we can use the twisted factorization given below. This is the INV2P method defined in [9]. Then

$$M = S \; T$$

where

$$S = \begin{bmatrix} \Delta_1^1 & 0 & & & & & & & \\ A_1^2 & \Delta_1^2 & 0 & & & & & & \\ & \ddots & \ddots & \ddots & & & & & \\ & & \ddots & \ddots & 0 & & & & \\ & & & A_1^{m+1} & \Delta_1^{m+1} & A_2^{1\,T} & & & \\ & & & & 0 & \Gamma_2^1 & A_2^{2\,T} & & \\ & & & & & \ddots & \ddots & & \\ & & & & & & \Gamma_2^{m-1} & A_2^{m\,T} \\ & & & & & & 0 & \Gamma_2^m \end{bmatrix} \quad (4.9)$$

and T is the inverse of the diagonal of S multiplied by S^T.

M is block tridiagonal, and the number of operations is approximately half that given in (4.3) with k=2. Then (4.9) can be used in place of (4.5) and (4.6) and this will give a parallel method for four processors.

The question now is "can we find a more economical method than DD1, using approximate Choleski factors ?"

To begin with, we take k=2 and it is tempting to use the decomposition

$$M = \begin{bmatrix} \Delta_1 + L_1 & 0 & 0 \\ 0 & \Delta_2 + L_2 & 0 \\ C^T & E^T & M_{12} \end{bmatrix} \begin{bmatrix} \Delta_1^{-1} & 0 & 0 \\ 0 & \Delta_2^{-1} & 0 \\ 0 & 0 & M_{12}^{-1} \end{bmatrix} \begin{bmatrix} \Delta_1 + L_1^T & 0 & C \\ 0 & \Delta_2 + L_2^T & E \\ 0 & 0 & M_{12} \end{bmatrix}$$

Now in the computation of M, we note that in position (3,1) we have $C^T (I + \Delta_1^{-1} L_1^T)$, and it is easy to see that $C^T \Delta_1^{-1} L_1^T = 0$. In position (3,2), $E^T (I + \Delta_2^{-1} L_2^T)$ is different from E^T. This can be corrected however, by letting M be defined as follows :

$$M = \begin{bmatrix} \Delta_1 + L_1 & 0 & 0 \\ 0 & \Gamma_2 + L_2^T & 0 \\ C^T & E^T & M_{12} \end{bmatrix} \begin{bmatrix} \Delta_1^{-1} & 0 & 0 \\ 0 & \Gamma_2^{-1} & 0 \\ 0 & 0 & M_{12}^{-1} \end{bmatrix} \begin{bmatrix} \Delta_1 + L_1^T & 0 & C \\ 0 & \Gamma_2 + L_2 & E \\ 0 & 0 & M_{12} \end{bmatrix}$$

Then, $E^T (I + \Gamma_2^{-1} L_2) = E^T$.

After renumbering the blocks, it is seen (4.11) is the twisted factorization.

Now, we can extend this method to k processors by letting

$$M = L \begin{bmatrix} \Delta_1^{-1} & & & & & \\ & \Sigma_2^{-1} & & & & \\ & & \Delta_{k-1}^{-1} & & & \\ & & & \Sigma_k^{-1} & & \\ & & & & M_{12}^{-1} & \\ & & & & & \ddots \\ & & & & & & M_{k-1,k}^{-1} \end{bmatrix} L^T.$$

Note that we alternate between the Δ's and Σ's, with

$$L = \begin{bmatrix} \Delta_1 + L_1 & & & & & \\ & \Sigma_2 + L_2^T & & & & \\ & & \ddots & & & \\ & & & \Sigma_k + L_k^T & & \\ C_1^T E_2^T & & & & M_{12} & \\ & C_2^T E_3^T & & & & M_{23} \\ & & \ddots & & & \ddots \\ & & & C_{k-1}^T E_k^T & & & M_{k-1,k} \end{bmatrix}. \quad (4.12)$$

In position (k,1) we have $C_1^T + C_1^T \Delta_1^{-1} L_1^T = C_1^T$ and in position (k,2), $E_2^T + E_2^T \Sigma_2^{-1} L_2 = E_2^T$, but unfortunately in position (k+1,2), $C_2^T + C_2^T \Sigma_2^{-1} L_2$ is different from C_2^T

5. Parallel Block Preconditioners

In [9] we suggested using as a preconditioner for parallel computers with k processors

$$M = (\Delta + L)\, \Delta^{-1}\, (\Delta + L^T)$$

and

The Δ_i are computed as follows :

$$\Delta_1 = D_1$$

$$\Delta_{2im/k} = D_{2im/k} \qquad i=1,\ldots$$

with obvious modifications for i=1, for all i=1,...,k/2

$$\Delta_j = D_j - A_j\, \Omega(1)^{j-1}\, A_j^T \qquad j=(i-1)m/k+1,\ldots,im/k-1$$

$$\Delta_j = D_j - A_{j+1}^T\, \Pi(1)^{j+1}\, A_{j+1} \qquad j=2im/k-1,\ldots,im/k$$

and,

$$L = \begin{bmatrix} 0 & & & & & & & & & & & & \\ A_2 & 0 & & & & & & & & & & & \\ & \ddots & \ddots & & & & & & & & & & \\ & & A_{\frac{m}{k}} & 0 & A_{\frac{m}{k}+1} & & & & & & & & \\ & & & \ddots & \ddots & A^T_{\frac{2m}{k}} & & & & & & & \\ & & & & 0 & 0 & & & & & & & \\ & & & & & A_{\frac{2m}{k}+1} & 0 & & & & & & \\ & & & & & & \ddots & \ddots & & & & & \\ & & & & & & & 0 & & & & & \\ & & & & & & & A_{\frac{3m}{k}} & 0 & A^T_{\frac{3m}{k}+1} & & & \\ & & & & & & & & \ddots & 0 & & & \\ & & & & & & & & & \ddots & 0 & & \\ & & & & & & & & & & A_m^T & 0 & \end{bmatrix}$$

This method, INVkP is very cheap in terms of operations and gives good numerical results, see [9]. It is interesting to investigate the relationship with the methods of the previous section.

After a permutation of M, INVkP requires the solution on half the number of subdomains as in (4.12), taking $M_{ij}=B_{ij}$, and using for each subdomain a twisted approximate factorization. Globally INVkP uses a UL block factorization instead of a LU one.

Therefore the block methods of [9] can be considered as simple variations of the domain decomposition LU preconditioners.

Note that this method can also be considered as a one way dissection block preconditioner.

There is more "fill in" in INVkP methods than in DD methods but the INV methods are simpler to implement and require fewer operations per iteration. The open question is to understand if there is a trade off between the increase in the number of iterations and the decrease of the cost of one iteration.

6. Numerical results

To illustrate the algorithms developed in the preceding sections, we solve the following problem :

$$- \text{div } (\lambda(x,y) \text{ grad } u) = f$$

in $\Omega =]0, 1[\times]0, 1[$, with homogeneous Dirichlet boundary conditions. λ is defined as follows :

$$\lambda(x,y) = 1000 \text{ in } \Omega_1 =]0.25, 0.75[\times]0.25, 0.75[,$$

$$\lambda(x,y) = 1 \text{ elsewhere.}$$

We use the 5 point finite difference scheme with $h = 1 / 101$.

The conjugate gadient iterations are stopped as soon as $||r^k|| \leq 10^{-6} ||r^0||$, $||\cdot||$ being the l_2 norm. The components of r^0 are chosen as random numbers equally distributed between -1 and 1.

Computer times are given in seconds on a CRAY 1S (CFT 1.13). They do not include the initialization times. Results are given in the form, n/t, where n is the number of iterations and t the computing time.

The results for INV, the standard block preconditioner, see [5], are

INV : 22 / 0.221

Following are the results for INVkP, DD1 and DD2 with respect to the number of processors, i.e. subdomains (of course, on a one processor CRAY machine).

nb of proc	INVkP	DD1	DD2
2	22 / 0.236	22 / 0.364	22 / 0.365
4	26 / 0.278	24 / 0.449	24 / 0.451
8	30 / 0.321	26 / 0.504	26 / 0.503
16	34 / 0.361	26 / 0.489	26 / 0.490
24	37 / 0.391	30 / 0.535	30 / 0.537
32	40 / 0.420	31 / 0.521	35 / 0.590
40	43 / 0.454	32 / 0.494	45 / 0.698
50	46 / 0.473	32 / 0.433	54 / 0.735

From these results, we see that for a small number of processors, up to 32, it is beneficial to use INVkP, because although the number of iterations is larger than in DD methods, the computing time is lower. Then, for INVkP, the number of iterations begin to increase faster and it becomes more interesting to use DD1.

The increase in the number of iterations is very slight for DD1.

The number of iterations for DD2 stays the same as for DD1, as long as the number of processors is small. When the number of processors increases, the subdomains become narrow and then it is important to take into account the influence between the interfaces, which is contained in the matrices H_i. Taking $H_i = 0$ as in DD2 leads to a rapid increase in the number of iterations.

This example shows that it can be beneficial to use DD methods, although, INVkP is better for a small number of processors.

More experiments and computations on a parallel computer will be given in [10].

Acknowledgment

The first part of this work has been done while visiting the University of Nijmegen (The Netherlands). The author has greatly beneficiated from conversations with Prof. Owe Axelsson.
Conversations with Prof. G. H. Golub during writing a first draft of this paper have been very helpful.

References

[1] O. AXELSSON
On inverse free factorization methods suitable for vector and parallel processors.
Report 84-31, Dept. of Mathematics, Univ. of Nijmegen, (1984).

[2] P. BJORSTAD & O.B. WIDLUND
Iterative methods for the solution of elliptic problems on regions partitioned into substructures.
SIAM J. on Numer. Anal. v 23, n 6, (1986) pp 1097-1120

[3] J.H. BRAMBLE, J.E. PASCIAK & A.H. SCHATZ
The construction of preconditioners for elliptic problems by substructuring. I.
Math. of Comp. v 47, n 175, (1986) pp 103-134

[4] T. CHAN & D. RESASCO
A domain decomposition fast Poisson solver on a rectangle.
Yale Univ. report, YaleU /DCS/RR 409 (1985)

[5] P. CONCUS, G.H. GOLUB & G. MEURANT

Block preconditioning for the conjugate gradient method.

SIAM J. Sci. Stat. Comput. v 6, (1985) pp 220-252

[6] G.H. GOLUB & D. MAYERS

The use of preconditioning over irregular regions.

pp 3-14 in "Computing methods in applied sciences and

engineering VI" R.Glowinski & J.L. Lions Eds, North-Holland (1984)

[7] G. MEURANT

The Fourier/tridiagonal method for the Poisson equation

from the point of view of block Choleski factorization.

Lawrence Berkeley Laboratory report, LBID-764, (1983)

[8] G. MEURANT

The block preconditioned conjugate gradient method

on vector computers.

BIT v 24 (1984) pp 623-633

[9] G. MEURANT

Multitasking the conjugate gradient on the CRAY X-MP/48.

Stanford Univ., Computer Science Dept. report NA-85-33 (1985)

to appear in Parallel Computing.

[10] G. MEURANT

Domain decomposition preconditioners for parallel computers.

to appear.