Deflation accelerated parallel preconditioned Conjugate Gradient method in Finite Element problems

F.J. Vermolen, C. Vuik and A. Segal[1]

Delft University of Technology, Department of Applied Mathematical Analysis, Mekelweg 4, 2628 CD, Delft, The Netherlands, *F.J.Vermolen@math.tudelft.nl*

Summary. We describe the algorithm to implement a deflation acceleration in a preconditioned Conjugate Gradient method to solve the system of linear equations from a Finite Element discretization. We focus on a parallel implementation in this paper. Subsequently we describe the data-structure. This is followed by some numerical experiments. The experiments indicate that our method is scalable.

1 Introduction

Large linear systems occur in many scientific and engineering applications. Often these systems result from a discretization of model equations. The systems tend to become very large for three-dimensional problems. Some models involve time and space as independent parameters and therefore it is necessary to solve such a linear system efficiently at each time-step.

In this paper we only consider symmetric positive definite (SPD) discretization matrices. Since the matrices are sparse in our applications, we use an iterative method to solve the linear system. In order to get a fast convergence method we use a preconditioned Conjugate Gradient method, where incomplete Choleski factorization is used as a preconditioner. This method is very suitable for parallellization.

The present study involves a parallellization of the Conjugate Gradient method in which the inner products, the matrix-vector multiplication and preconditioning are parallellized. This parallellization is done by the use of domain decomposition, where the domain of computation is divided into subdomains and the overall discretization matrix is divided over the subdomains. To each subdomain we allocate a processor. A well-known problem is that the parallellized method is not scalable: the number of CG-iterations and wall-clock time increase as the number of subdomains increases. To make the method scalable one uses a coarse grid correction (see for an overview and introduction Smith et al [6]) or the deflation method. In [3] it is shown that deflation gives a larger acceleration to the parallel preconditioned CG-method. The idea to use deflation for large linear systems of equations is not new. Among others, Nicholaides [4] and Vuik et al [8, 1, 10, 9] apply this method to solve large ill-conditioned linear systems. The result of deflation is that the components of the solution in the direction of the eigenvectors corresponding to the extremely small eigenvalues are projected to zero. The effective condition number of the resulting singular system becomes more favourable. In the present paper we deal with "algebraic" deflation vectors. For more details on the various types of deflation vectors we refer to [8] and [7].

We assume that the domain of computation Ω consists of a number of disjoint subdomains Ω_j, $i \in \{1, \ldots, m\}$, such that $\cup_{j=1}^{m} \overline{\Omega}_j = \overline{\Omega}$. To each subdomain we allocate a

processor and a deflation vector, z_j, for which we define

$$z_j = \begin{cases} 1, & \text{for } (x,y) \in \Omega_j, \\ 0, \text{ for } (x,y) \in \Omega \setminus \Omega_j. \end{cases} \tag{1}$$

In case of Finite Volume methods we have to distinguish between cell-centered and vertex-centered discretization. In the cell-centered the deflation vector is not defined on the interfaces between consecutive subdomains. In the vertex-centered case, however, we have an overlap at the interface points. In this paper we use a Finite Element discretization, which is always vertex-centered by its construction. The subdomains may be considered as "super"-elements consisting of a set of finite elements. The global stiffness matrix is never constructed, only the "super"-element matrix is constructed. Matrix-vector multiplication is carried out per "super"-element and only after adding of the contributions of each "super"-element the global vector is obtained. In this way parallellization of the Finite Element method can be done in a natural way. For the interface points we use the concept of "average overlap", which is explained as follows: Given a deflation vector z_j on an interfacial node that is shared by Ω_j and p neighbours of Ω_j, then we set at this point:

$$z_j = \frac{1}{p+1}. \tag{2}$$

The deflation method is applied successfully to problems from transport in porous media where coefficients abruptly change several orders of magnitude [9]. In the present paper we consider a Galerkin Finite Element discretization of the Laplace equation with a Dirichlet and a Neumann boundary condition at Γ_D and Γ_N respectively (note that $\Gamma_N \cup \Gamma_D = \partial\Omega$):

$$\begin{cases} -\Delta u = f, & (x,y) \in \Omega \\ u = \tilde{u}(x,y), \text{ for } (x,y) \in \Gamma_D, \ \dfrac{\partial u}{\partial n} = 0, \text{ for } (x,y) \in \Gamma_N \end{cases} \tag{3}$$

where u denotes the solution and $\tilde{u}$ represents a given function. The resulting discretization matrix is symmetric positive definite. The domain is divided into subdomains and the resulting system of linear equations is solved by the use of a parallellized Deflated ICCG. In the text the algorithm is given and the issues of data-structure for the parallellization of the solution method are described. Subsequently, we describe some numerical experiments. For more mathematical background we refer to [1, 10, 7].

2 Deflated Incomplete Choleski preconditioned Conjugate Gradient Method

In this section we describe the deflated method for the symmetric positive definite discretization matrix A. Let $Z = (z_1 \dots z_m)$ represent the matrix whose columns consist of the deflation vectors z_j, as defined in equations (1) and (2). The matrix Z is chosen such that its column space approximates the eigenspace of these eigenvectors that correspond to the smallest eigenvalues. We, then, define the projection $P := I - AZ(Z^TAZ)^{-1}Z^T := I - AZE^{-1}Z^T$. It is shown in [7] that the matrix PA is positive semi-definite (and hence singular). Kaasschieter [2] showed convergence of

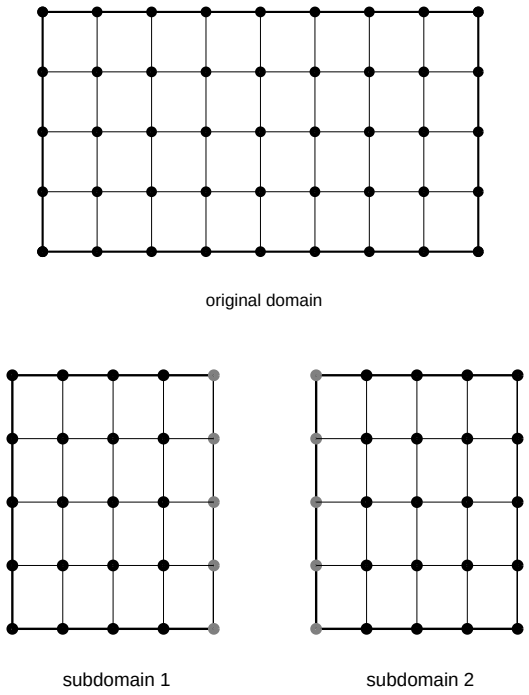

Fig. 1. Domain decomposition for a vertex centered discretization.

the Conjugate Gradient method for cases in which the matrix is singular. Let b be the right-hand side vector and x be the solution vector, then we solve

$$Ax = b. \tag{4}$$

After application of deflation by left multiplication of the above equation by P, we obtain

$$PAx = Pb. \tag{5}$$

Since PA is singular the solution is not unique. We denote the solution that is obtained by use of the ICCG method on equation (5) by $\tilde{x}$. To get the solution x we use

$$x = (I - P^T)x + P^T x. \tag{6}$$

It is shown in [7] that $P^T\tilde{x} = P^T x$, hence the solution of equation (5) can be used. The second part $(I-P^T)x = Z(Z^TAZ)^{-1}Z^Tb$ is relatively cheap to compute. Hence the solution x is obtained by addition of the two contributions, i.e. $x = P^T\tilde{x} + Z(Z^TAZ)^{-1}Z^Tb$. For completeness we give the algorithm of the Deflated ICCG:

Algorithm 1 (DICCG [9]):
$k = 0, \tilde{\underline{r}}_0 = P\underline{r}_0, \underline{p}_1 = \underline{z}_0 = L^{-T}L^{-1}\tilde{\underline{r}}_0$
while $||\tilde{\underline{r}}_k||_2 > \varepsilon$
$k = k + 1, \quad \alpha_k = \dfrac{\tilde{\underline{r}}_{k-1}^T \ \underline{z}_{k-1}}{\underline{p}_k^T PA\underline{p}_k}$
$\underline{x}_k = \underline{x}_{k-1} + \alpha\underline{p}_k, \quad \tilde{\underline{r}}_k = \tilde{\underline{r}}_{k-1} - \alpha_k PA\underline{p}_k$
$\underline{z}_k = L^{-T}L^{-1}\tilde{\underline{r}}_k, \quad \beta_k = \dfrac{\tilde{\underline{r}}_k^T \ \underline{z}_k}{\tilde{\underline{r}}_{k-1}^T \ \underline{z}_{k-1}}$
$\underline{p}_{k-1} = \underline{z}_k + \beta_k\underline{p}_k$
end while

The inner products, matrix vector multiplication and vector updates in the above algorithm are easy to parallellize. Parallellization of the incomplete Choleski preconditioned Conjugate Gradient method has been done before by Perchat et al [5]. We use a restriction and a prolongation operator and block preconditioners for the preconditioning step in the above algorithm. Note that it is necessary to have a symmetric preconditioner. This is obtained by choosing the restriction and prolongation matrices as transposes of each other. Let $\underline{r}_k$ be the residual after k CG-iterations and N be the total number of subdomains, then overall preconditioning is expressed in matrix form by:

$$\underline{z}_k = \left(\sum_{i=1}^{N} R_i^T M_i^{-1} R_i \right) \underline{r}_k. \tag{7}$$

Here $\underline{z}_k$ represents the updated residual after preconditioning. Further, R_i and M_i respectively denote the restriction operator and block preconditioner. We will limit ourselves to the issues of the implementation of the data-structure needed for the parallel implementation of deflation. The above algorithm is a standard ICCG except for the lines that contain the matrix P.

3 Data-structure of the deflation vectors for parallellization

To create P and Pv we need to make z_j and to compute Az_j, $z_i^T Az_j$ and Pv. To do this efficiently we make use of the sparsity pattern of the deflation vectors z_j. We create the vectors z_j in the subdomain Ω_j only and send essential parts to its direct neighbours. We explain the data-structure and communication issues for a rectangular example. In the explanation we use the global numbering from the left part of Figure 2. Note that in the implementation the local numbering is used in the communication and calculation part. The global numbering is used for post-processing purposes only. The example can be generalized easily to other configurations. The situation is displayed in Figure 2.

In Figure 2 on each subdomain Ω_i a deflation vector z_i is created. For all the interface nodes (say numbers 3, 8, 13, 12, 11 of Ω_1) the number of neighbours is determined, then equation (2) is applied to determine the value of the corresponding entry of z_1. This implies that it is necessary to have a list of interface nodes for each subdomain and to have a list with the number of neighbouring subdomains on which a particular interface node is located. However, this information is not sufficient. For example the vector Az_1 must be computed and also be multiplied with vectors z_j. The vector Az_1 has nonzero entries not only inside the domain Ω_1 and on its interfaces, but also in its direct

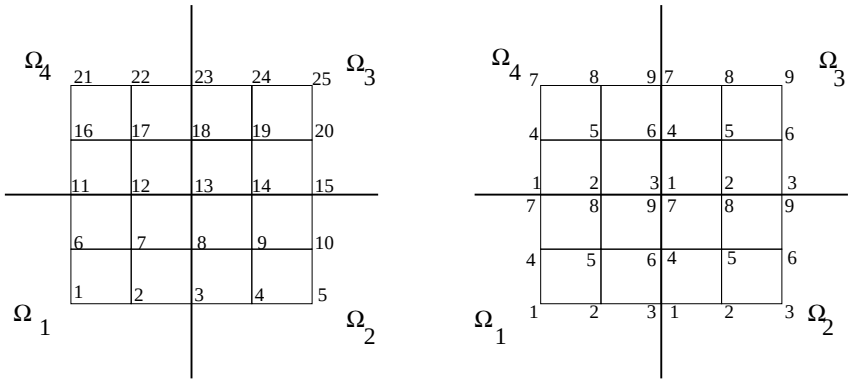

Fig. 2. A sketch of division of Ω into subdomains $\Omega_1, \ldots, \Omega_4$. Left figure represents the global numbering of the unknowns, right figure represents the local numbering.

neighbours, i.e. the points 4, 9, 14, 16, 17, 18, 19. Therefore we also need to extend the vectors z_1 with these points to have a well defined matrix-vector multiplication. In the same way the vectors Az_j, $j \in \{2, 3, 4\}$ have non-zero entries in Ω_1. For the other deflation vectors we proceed analogously.

We further explain the computational part which is relevant to processor 1, i.e. subdomain ω_1 only, the other subdomains are dealt with similarly. For example Az_2 will have a non-zero contribution in all interface points of Ω_1 and Ω_2 but also in the points 2, 7 and 12. All vectors in common points of any subdomain and Ω_1 are given the global value, i.e. the value that is the result of addition. This requires communication between Ω_1 and this particular subdomain. This is in contrast to the matrix A, which is stored only locally per subdomain without the addition at common interfaces. So to compute Az_2 on Ω_1 we need an extra list of neighbouring points of Ω_2 in domain Ω_1 that are not on the common interface. This, however, is not sufficient. For example the value of Az_2 in nodal point 12 also has a contribution of Ω_4. So in Ω_1 we need an extra list of points on common sides of Ω_1 and Ω_j that are direct neighbours of Ω_k $(j \neq k)$ but are not on the interface of Ω_1 and Ω_k. For example node 12 is a common point of Ω_1 and Ω_4 and a neighbour of Ω_2, further Ω_4 is a neighbour of Ω_2. After communication and addition of the values of Az at these particular nodes, the matrix E, consisting of the inner products, is calculated and sent to processor 1.

Then, for a given vector v we compute at each processor its inner product with z_i $(z_i^T v)$. Then all these inner products are sent to processor 1 (Ω_1) where $y = E^{-1}\left(z_1^T v\ z_2^T v\ z_3^T v\ z_4^T v\right)^T$ is computed by Choleski and subsequently the results are sent to all the other neighbouring processors. Then, $Pv = v - ZAy$ is computed locally. All the steps are displayed schematically in algorithm 2, where we explain the situation for a case with two processors. Mark that E has a profile structure where the profile is defined by the numbering pattern of the subdomains. Hence for a block structure in two dimensions we obtain a similar sparsity pattern for E as for a two-dimensional discretization. If, however, a layered structure is used, then E gets the same pattern as a one-dimensional discretization matrix.

Algorithm 2 (Parallellization of Deflation) $P = I - AZ(Z^TAZ)^{-1}Z^T$.

Processor 1		Processor 2
Make z_1		Make z_2
	Communication	
Make Az_1, Az_2^Γ		Make Az_2, Az_1^Γ
	Communication	
$Az_1 = Az_1 + Az_1^\Gamma$		$Az_2 = Az_2 + Az_2^\Gamma$
$E_{11} = z_1^T Az_1$		$E_{22} = z_2^T Az_2$
$E_{12} = z_1^T Az_2^\Gamma$		$E_{22} = z_2^T Az_1^\Gamma$
	Send E to proc 1	
Choleski decomp E		
	$Pv =$	
	$v - AZE^{-1}Z^Tv$	
Compute z_1^Tv		Compute z_2^Tv
	Send z_2^Tv	
$y = E^{-1}\begin{pmatrix} z_1^Tv \\ z_2v \end{pmatrix}$		
	Send y to proc 2	
$v - y_1Az_1 - y_2Az_2^\Gamma$		$v - y_2Az_2 - y_1Az_1^\Gamma$

4 Numerical experiments

To illustrate the advantage of the deflation method we present the number of CG-iterations and wall-clock time as a function of the number of layers (left and right graphs respectively in Figure 3). We start with one layer and extend the domain of computation with one horizontal layer, which is placed on top. This is done consecutively up to 7 layers. In the examples we choose the number of elements the same in each layer. The problem size increases as the number of layers increases. It can be seen that if deflation is not used then the convergence will take more time since the number of CG iterations increases. The use of deflation yields that the number of iterations and wall-clock time for the parallel case does not depend on the number of layers. This is also observed for the number of CG-iterations for the sequential computations. This makes the method scalable.

Further, we present the number of iterations as a function of the number of layers as in the preceeding example for three methods: no projection, coarse grid correction and deflation. The results are shown in Figure 4 (left graph). It can be seen that both the coarse grid correction and the deflation methods are scalable, however deflation gives the best results. This is in agreement with the analysis as presented in [3] where it is proven that the deflated method converges faster than the coarse grid correction. The same behaviour is observed if the domain is extended in a blockwise distribution of added subdomains (see the right graph in Figure 4).

5 Conclusions

The deflation technique has been implemented successfully in a parallellized and sequential ICCG method to solve an elliptic problem by the use of finite elements. The domain

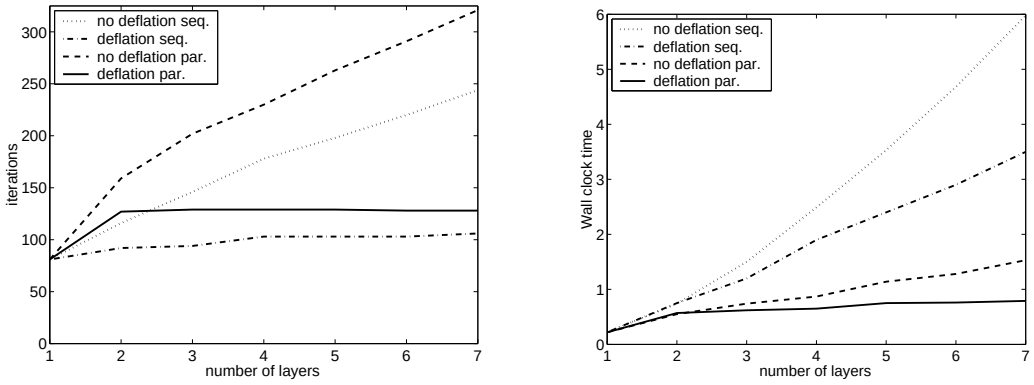

Fig. 3. Left figure: The number of iterations as a function of the number of layers for deflated and non-deflated parallellized and sequential ICCG method. Right figure: The wall clock-time as a function of the number of layers for deflated and non-deflated parallellized and sequential ICCG method.

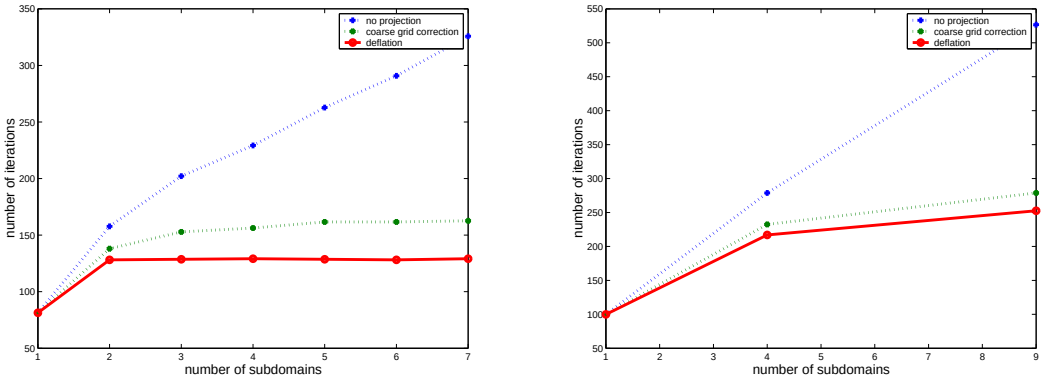

Fig. 4. The number of iterations as a function of the number of layers for the parallellized ICCG method for three methods: no projection, coarse grid correction and deflation. Left graph: layered extension of the domain of computation, Right graph: blockwise extension of the domain of computation.

decomposition can be chosen blockwise and layerwise. Some numerical experiments are shown in the present paper. Further, the number of iterations and wall-clock time become independent of the number of added layers if deflation is applied in a parallel ICCG method. Hence deflation is favourable in both sequential and parallel computing environments.

References

1. J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM J. Sci. Comput.*, pages 442–462, 2001.
2. E. F. Kaasschieter. Preconditioned Conjugate Gradients for solving singular systems. *Journal of Computational and Applied Mathematics*, 24:265–275, 1988.

3. R. Nabben and C. Vuik. A comparison of deflation and coarse grid correction applied to porous media flow. Technical report 03-10, Delft University of Technology, Delft University of Technology, Delft, The Netherlands, 2003.
4. R.A. Nicholaides. Deflation of Conjugate Gradients with applications to boundary value problems. *SIAM J. Numer. Anal.*, 24:355–365, 1987.
5. E. Perchat, L. Fourment, and T. Coupez. Parallel incomplete factorisations for generalised Stokes problems: application to hot metal forging simulation. Report, EPFL, Lausanne, 2001.
6. B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition*. Cambridge University Press, Cambridge, 1996.
7. F.J. Vermolen, C. Vuik, and A. Segal. Deflation in preconditioned Conjugate Gradient methods for finite element problems. *J. Comput. Meth. in Sc. and Engng.*, to appear, 2003.
8. C. Vuik, A. Segal, L. el Yaakoubli, and E. Dufour. A comparison of various deflation vectors applied to elliptic problems with discontinuous coefficients. *Appl. Numer. Math.*, 41:219–233, 2002.
9. C. Vuik, A. Segal, and J. A. Meijerink. An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients. *J. Comput. Phys.*, 152:385–403, 1999.
10. C. Vuik, A. Segal, J. A. Meijerink, and G. T. Wijma. The construction of projection vectors for a Deflated ICCG method applied to problems with extreme contrasts in the coefficients. *J. Comput. Phys.*, 172:426–450, 2001.