NUMERICAL METHODS FOR SCIENTIFIC COMPUTING
VARIATIONAL PROBLEMS AND APPLICATIONS
Y. Kuznetsov, P. Neittanmaki and O. Pironneau (Eds.)

PARALLEL ALGEBRAIC MULTILEVEL PRECONDITIONERS

Gérard Meurant
CEA/DIF
BP 12, 91680 Bruyères le Chatel
France
Email: gerard.meurant@cea.fr
Web page: http://perso.wanadoo.fr/gerard.meurant

Abstract. We study the parallelization of some aspects of algebraic multilevel preconditioners for solving symmetric positive definite linear systems with the conjugate gradient algorithm. We use partitioning of the graph of the matrix. We are particularly concerned with parallelization of the construction of the smoothers and of the coarsening algorithms. Finally, we give some numerical examples showing that when completely parallelized these algorithms may not be fully scalable.

This paper is dedicated to Jacques Périaux on the occasion of his sixtieth birthday

Key words: algebraic multigrid, conjugate gradient, preconditioner.

1 INTRODUCTION

In this paper we study the parallelization of algebraic multilevel preconditioners for solving sparse symmetric positive definite linear systems $Ax = b$ with the conjugate gradient (CG) algorithm. The algebraic multilevel techniques use smoothing operators and coarsening algorithms which are difficult to parallelize. To introduce more parallelism we will study the use of partitioning of the graph of the matrix. In section 2, we briefly review the algebraic multilevel preconditioners see[1,2]. Section 3 describes the parallelization of the coarsening algorithms. Section 4 is concerned with the parallel construction of the smoothers. Finally, section 5 gives numerical results showing that the number of CG iterations depends, even though only slightly, on the number of subdomains or the dimension of the problem.

2 ALGEBRAIC MULTILEVEL PRECONDITIONERS

All these preconditioners use the same design principles as the Algebraic Multi-Grid algorithm (AMG). The standard AMG is a multigrid–like method that has been firstly defined for M–matrices, see Ruge and Stuben[3,4]. Instead of using the mesh of the discretization as in geometric multigrid, AMG uses the graph of A. After some smoothing steps, the equation for the error $Ae = r$ with the residual as the right hand side is solved recursively on a coarser graph, corresponding to a subset of

the unknowns and this method is used recursively until the number of unknowns is small enough to allow for a fast direct solve. In AMG the coarse graphs are defined by looking at the entries of the matrix. The set of dependencies of an unknown (a node in the graph) is defined by (a part of) the neighbours of the given node. An influence set is defined for each unknown as the "transpose" of the set of dependencies. The fine and coarse nodes for each level are found on this basis. Then, knowing the fine and coarse nodes, interpolation weights are computed using the entries of the matrix and the equations of the linear system. The restriction R (going from a grid to the next coarser grid) is the transpose of the interpolation (prolongation) matrix P and the next coarse matrix is generally defined as $A_c = RAP$. As we said before, the method also uses a smoothing operator. An iteration (denoted as a V–cycle) of the recursive AMG algorithm is the following:

1. Do ν iterations of smoothing.
2. Restrict the residual r to $r_c = Rr$.
3. Recursively solve $A_c e_c = r_c$.
4. Interpolate e_c to $e = Pe_c$.
5. Add the correction e to the current iterate.
6. Do ν iterations of smoothing.

If everything is symmetric including the smoothing operator a preconditioner for PCG is obtained by running one iteration of the previous algorithm starting from $x^0 = 0$ going down levels to the coarsest one where the linear system is solved using Gaussian elimination, and then going back up to the finest level.

2.1 The smoother

One way to extend what is done in classical geometric multigrid is to use a symmetric Gauss–Seidel iteration as a smoother which unfortunately is rather sequential. A way to parallelize it is to color the unknowns or to use the previous smoothing iteration when unknowns from other processors are needed. But we will not consider this here.

A very efficient smoother is the Incomplete Cholesky (IC) decomposition $LD^{-1}L^T$ (where L is lower triangular and D diagonal) of the matrix. There are many different variants of this algorithm. The most popular one is to use a decomposition for which the non zero structure of L is the same as the structure of the lower triangular part of A. This is usually denoted as IC(0). This incomplete decomposition is used as a smoother in a Richardson iteration

$$LD^{-1}L^T(x^{k+1} - x^k) = b - Ax^k$$

when solving $Ax = b$. We will denote this smoother by 'ic'. Another idea is to use an approximate inverse M from AINV as a smoother in a Richardson iteration defined as

$$x^{k+1} = x^k + M(b - Ax^k),$$

when solving $Ax = b$ [2,5]. This preconditioner introduced by Benzi and al. [5] computes an approximate factorization $M = ZD^{-1}Z^T$ of the inverse of A where Z is upper triangular and D diagonal and involves a parameter τ used to define which elements are dropped during the factorization. The use of the smoother (denoted by 'ai') is parallel because it involves only matrix vector multiplies.

2.2 The influence matrix

An important part of the algorithm is to decide at a given level which unknowns correspond to the fine "nodes" (or points or unknowns) and which to the coarse nodes which are going to be the unknowns on the next coarser level. Hence, the set $\mathcal{N} = \{1, \ldots, n\}$ of the unknown indices is split into two disjoint sets $\mathcal{N} = F \cup C$.

First of all for each unknown i we define the set of dependencies S_i and an influence matrix S whose rows are the S_i's padded with zeros. This can be done in many ways. The standard AMG definition[1,3,4,6] can be generalized to any matrix by using

$$S_i = \{j \mid |a_{i,j}| > \tau \max_{k \neq i} |a_{i,k}|, \quad \tau < 1\}.$$

However, we will always keep at least one non diagonal element in every row of S. We choose to keep the index with the largest element modulus in A. This is denoted as algorithm 'b' ('a' being the standard one).

2.3 The coarsening algorithm

Once S_i and S are fixed, there are different ways we can follow to decide which are the F and C points. What we are going to denote as the "standard" ('st') coarsening algorithm is the following:

1. Choose the first point i with maximal weight as a C point.
2. Assign the points that i influences as F points.
3. Increment by 1 the weights of the points influencing these new F points.
4. Decrease by 1 the weights of points that depend on i.

This guarantees that each F point has at least one connection to a C point. This is needed for the standard interpolation. It is clear that this algorithm is purely sequential.

3 PARALLEL COARSENING ALGORITHMS

To introduce more parallelism in the construction of the smoothers and the coarsening algorithm we use a partitioning of the graph of A. We are interested in applications on distributed memory parallel computers. Therefore we assume that one subdomain will be allocated to one processor. The graph of A is partitioned with subdomain overlapping or with interfaces and the variables corresponding to one subdomain (subgraph) are allocated to one processor as well possibly as the surrounding interface nodes.

A way to parallelize the standard algorithm is to start by coarsening the overlapping (or interface) and then to coarsen each subdomain in parallel. However, after coarsening the overlapping and before proceeding for each subdomain we flag as F

points the (not yet handled) neighbours of C points in the overlapping region being cautious not to introduce F-F connections. This is important to make this algorithm working properly. An example for the Poisson problem with four subdomains is given in figure 1 which shows the global coarsening, the colors corresponding to the coarse points in the subdomains, the white points being the fine nodes. This is a perfectly regular coarsening. It turns out that this algorithm is a special case of a coloring algorithm described by Jones and Plassman[7].

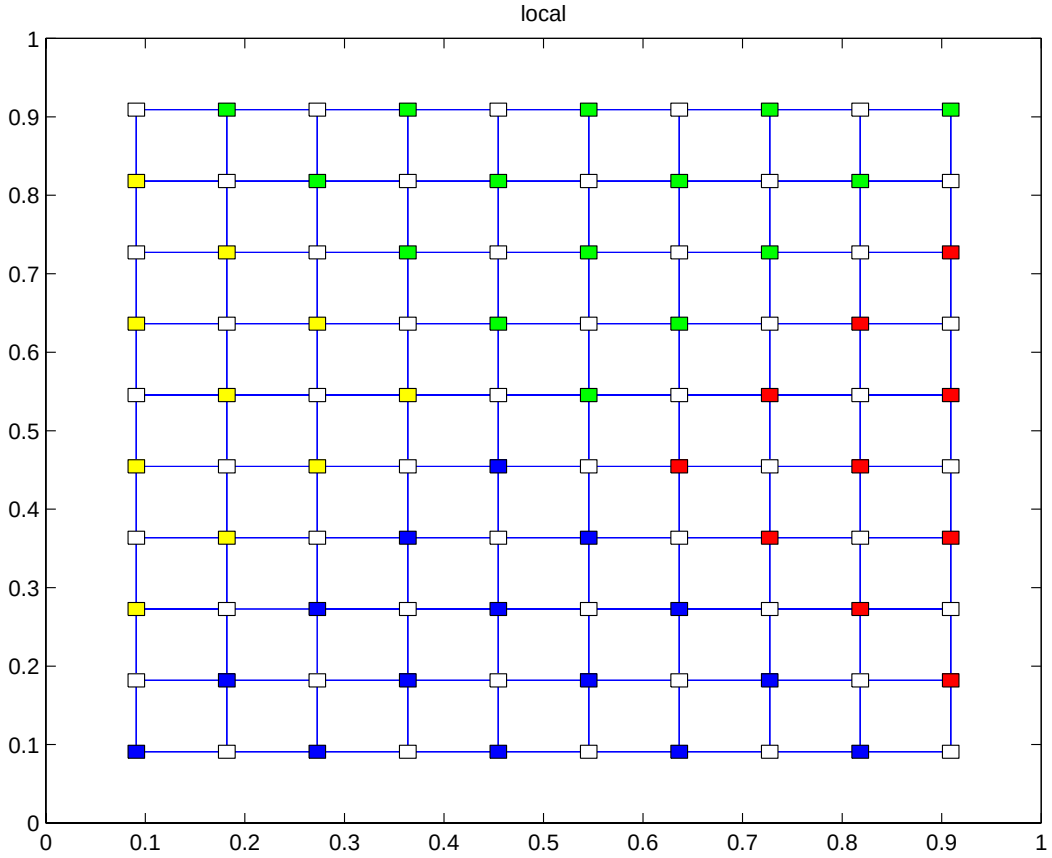

Figure 1: Global result of the coarsening for the Poisson problem, four subdomains.

We have also considered variations of this algorithm where we avoid having F-F connections in the subdomains. However this can lead to a storage which is too large with too many coarse points and refusing F-F connections only on the fine level is most of the time enough. When we use a partitioning with subdomains and interfaces, the problem is more or less the same, the overlapping region being replaced by the interface. There are some others ways to generate in parallel the F and C points. Let us mention an algorithm proposed by Cleary, Falgout, V.A. Henson and Jones[8].

4 PARALLEL SMOOTHERS

To parallelize the construction of the smoothers AINV and IC we again use the partition of the graph.

4.1 AINV

The use of the approximate inverse as a smoother is parallel since it uses a matrix vector product which is easy to parallelize. However, the construction of the preconditioner $M = ZD^{-1}Z^T$ is sequential. The matrix Z is usually computed column by column. To parallelize the construction of Z we drop the entries $z_{i,j}$ if i and j belong to different subdomains. This means that we only eventually keep the fill–ins inside subdomains or within a subdomain and the interface set. We will denote this smoother by 'ad'. The subdomains can now be handled in parallel and then in the end the columns corresponding to the interface are computed sequentially even though this can be partly parallelized.

4.2 Incomplete Cholesky

With domain partitioning and interfaces we apply the same strategy as for AINV. Fill–ins are neglected between subdomains and kept within subdomains or between a subdomain and the interface. Actually at the finest level, there won't be any fill–in between subdomains because of the domain decomposition ordering that is used, but fill–in would eventually happen on coarser levels and this is what is thrown away. We will denote this smoother by 'id'. The problem of the influence of orderings when using IC as a preconditioner was studied experimentally by Duff and Meurant[9] some years ago. After these studies it is now well known that dissection is not a very good ordering with regards to the number of iterations. We will see in the numerical experiments that it is the same when IC is used as a smoother.

5 NUMERICAL EXPERIMENTS

We solve linear systems arising from the finite difference discretization of elliptic PDEs in the unit square with homogeneous Dirichlet boundary conditions. We use a random right hand side and a zero initial vector. The CG iterations are stopped when $\|r^k\| \leq 10^{-10}\|r^0\|$ where r^k is the computed residual.

We will only consider the Poisson equation with an $m \times m$ mesh because this already illustrates the problems that arise with the parallelization . We start by considering the standard sequential smoothers and coarsening algorithms with a partitioning of the graph of A. Therefore, only the ordering of the unknowns is changed. In the tables we give the number of CG iterations, the number of floating point operations and the storage to store the preconditioner. A quadruplet like ('ic', 'b', 'st', 'st') means that the smoother is 'ic', the influence matrix is using algorithm 'b', the coarsening algorithm is the standard one 'st' and we use the standard AMG interpolation formula. We see by comparing tables 1 and 2 that 'ic' is a better smoother than 'ai' when using approximately the same amount of storage. We note that the number of iterations is almost constant when increasing the number of subdomains.

We then turn to the parallel smoothers using the domain decomposition ordering with interfaces but we still use the sequential coarsening algorithm for which results are given in tables 3 to 4. We can see that there is not much increase in the number of iterations and the number of floating point operations when increasing the number of subdomains for a given problem dimension. We notice that for 'id' the number of iterations is larger than for the sequential algorithm.

Then we use the parallel version of the coarsening algorithm. Tables 5 and 6 explore the possibility of refusing the F-F connections only on the fine level. We

nb sd	nb it	flops	storage
1	5	1 598 253	35550
2	8	2 409 136	35548
4	8	2 402 677	35470
8	8	2 371 293	35015
16	8	2 361 605	34290
32	9	2 599 697	34562

Table 1: PCG for Poisson equation, $m = 40$, $\tau = 0.05$, multilevel with interfaces, ('ic', 'b', 'st', 'st').

nb sd	nb it	flops	storage
1	14	3 931 233	35987
2	14	3 918 273	35909
4	13	3 632 561	35717
8	14	3 836 343	35213
16	14	3 802 233	34968
32	16	4 257 199	34540

Table 2: PCG for Poisson equation, $m = 40$, $\tau = 0.05$, multilevel with interfaces, ('ai', 'b', 'st', 'st').

see that the approximate inverse smoother, although being intrinsically less efficient, than the 'ic' smoother is also less sensitive to the parallelization of the coarsening algorithm.

Using overlapping of the subdomains will be considered in another paper. The results are of the same quality than with interfaces. To assess the scalability of the algorithms we can look at the results for 64 subdomains for $m = 150$, 32 subdomains for $m = 110$, 16 subdomains for $m = 80$, 8 subdomains for $m = 60$ and 4 for $m = 40$. Although the number of unknowns per subdomain is not exactly constant (around 400), this gives us an idea about what is going on. Let us first compare the Incomplete Cholesky smoothers with interfaces. Figure 2 shows the results for 'ic' using only one subdomain and the usual ordering, 'ic' with the domain decomposition (dissection) ordering and 'id' throwing away some fill–ins on the coarser levels. We see that the standard non parallel algorithm gives a constant number of iterations when there is an increase when using the parallel version which is clearly due to the ordering that is used since the coarsening algorithm was the sequential standard one for these runs.

Let us now turn to the other smoothers. Figure 3 shows that the AINV smoother is much less sensitive to the ordering than the Incomplete Cholesky decomposition. We normalize the matrix and use 'ai' with the standard ordering, 'ai' with the dissection ordering and 'ad' throwing away the fill–ins between subdomains. We can see that there is not much difference in the number of iterations.

6 Conclusion

In this paper we have studied some issues in parallelizing multilevel algebraic preconditioners. We used graph partitioning to parallelize the construction of the

nb sd	nb it	flops	storage
1	5	1 598 253	35550
2	8	2 409 136	35548
4	8	2 402 677	35470
8	8	2 371 293	35015
16	8	2 361 605	34290
32	9	2 599 697	34562

Table 3: PCG for Poisson equation, $m = 40$, $\tau = 0.05$, multilevel with interfaces, ('id', 'b', 'st', 'st').

nb sd	nb it	flops	storage
1	14	3 929 793	36005
2	14	3 916 833	35897
4	13	3 633 233	35723
8	14	3 833 943	35193
16	14	3 805 233	34993
32	16	4 262 231	34577

Table 4: PCG for Poisson equation, $m = 40$, $\tau = 0.05$, multilevel with interfaces, ('ad', 'b', 'st', 'st').

Incomplete Cholesky and AINV smoothers as well as the coarsening algorithm. Numerical experiments show that for a given problem dimension the number of iterations is not very sensitive to the number of subdomains but when increasing the dimension the scalability of the sequential algorithm is lost. However, the results when using AINV are much less sensitive to the variation of the dimension than with the Incomplete Cholesky smoother.

nb sd	nb it	flops	storage
2	8	2 484 338	36790
4	7	2 186 602	36555
8	8	2 517 864	37529
16	9	2 824 986	37545
32	11	3 586 071	34670

Table 5: PCG for Poisson equation, $m = 40$, $\tau = 0.05$, multilevel with interfaces, refuse F-F connections only on the fine level, ('id', 'b', 'sd', 'st').

nb sd	nb it	flops	storage
2	15	4 277 693	36882
4	14	3 971 013	36611
8	13	3 749 853	37236
16	12	3 502 164	37133
32	12	3 750 659	34536

Table 6: PCG for Poisson equation, $m = 40$, $\tau = 0.05$, multilevel with interfaces, refuse F-F connections only on the fine level, ('ad', 'b', 'sd', 'st').

REFERENCES

[1] V.E. Henson, *An algebraic multigrid tutorial*, MGNET, http://www.mgnet.org, (1999).

[2] G. Meurant, *Numerical experiments with algebraic multilevel preconditioners*, Electronic Transactions on Numerical Analysis, vol 12, (2001).

[3] J.W. Ruge and K. Stuben, *Algebraic multigrid*, in Multigrid methods, S.F. Mc Cormick ed., SIAM, (1987), pp 73-130.

[4] W.L. Briggs, V.E. Henson and S.F. McCormick, *A multigrid tutorial, second edition*, SIAM, (2000).

[5] M. Benzi, C.D. Meyer and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., vol 17, (1996), pp 1135-1149.

[6] C. Wagner, *Introduction to algebraic multigrid*, Course notes version 1.1, University of Heidelberg, (1998).

[7] M.T. Jones and P.E. Plassman, *A parallel graph coloring algorithm*, SIAM J. on Sci. Comp., vol 14 no 3, (1993).

[8] A. Cleary, R. Falgout, V.E. Henson and J. Jones, *Coarse–grid selection for parallel algebraic multigrid*, LLNL report, (1999).

[9] I.S. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT v29, (1989), pp 635-657.

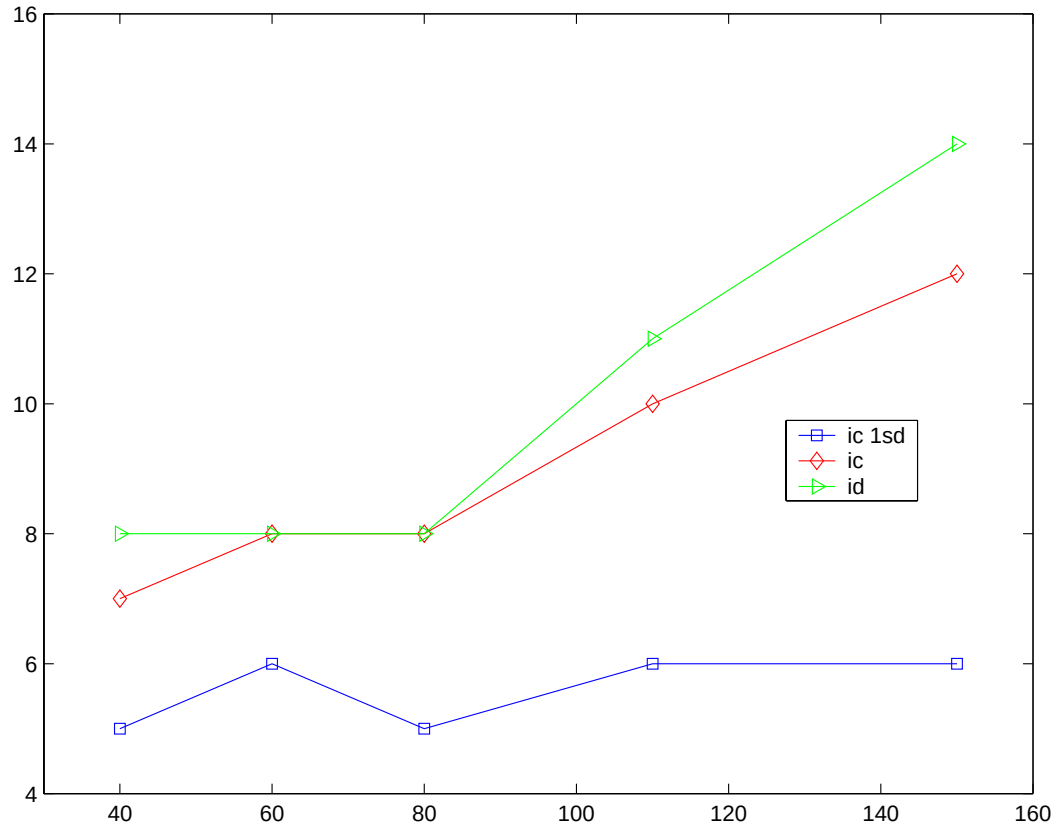

Figure 2: Scalability for the Poisson problem, number of iterations, Incomplete Cholesky smoothers.

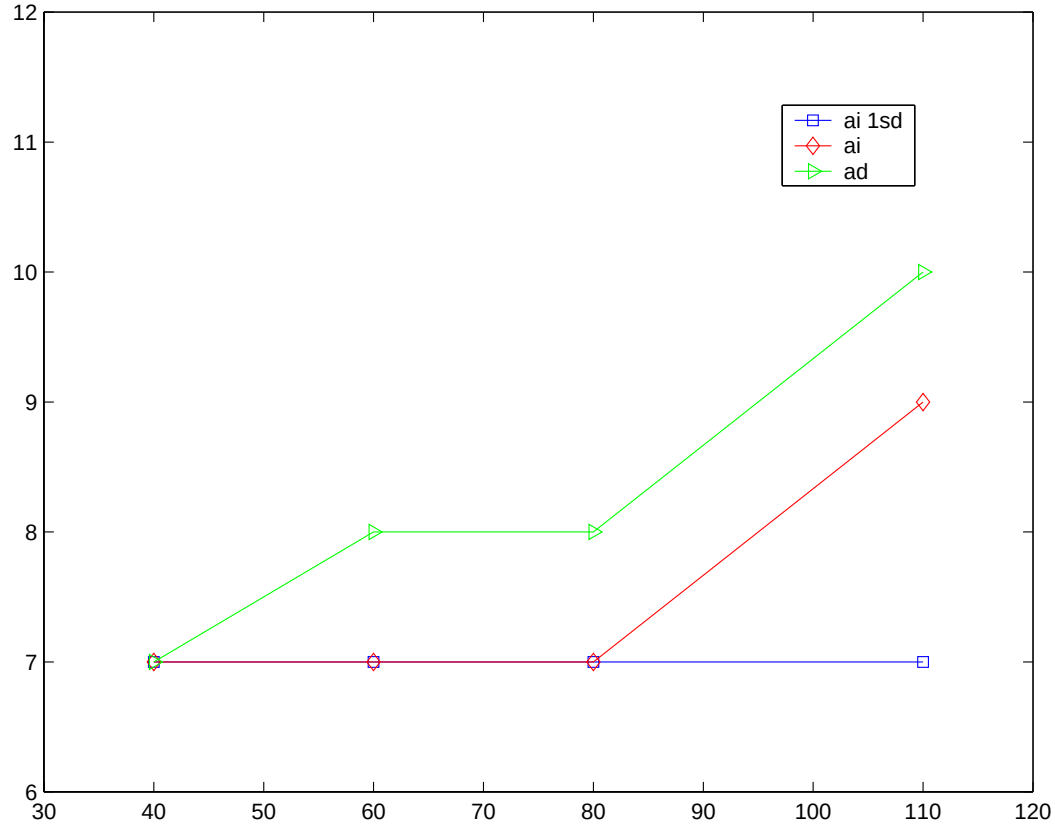

Figure 3: Scalability for the Poisson problem, number of iterations, AINV smoothers.