Electronic Transactions on Numerical Analysis.
Volume 37, pp. 123-146, 2010.

ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

AN AGGREGATION-BASED ALGEBRAIC MULTIGRID METHOD*

YVAN NOTAY†

Abstract. An algebraic multigrid method is presented to solve large systems of linear equations. The coarsening is obtained by aggregation of the unknowns. The aggregation scheme uses two passes of a pairwise matching algorithm applied to the matrix graph, resulting in most cases in a decrease of the number of variables by a factor slightly less than four. The matching algorithm favors the strongest negative coupling(s), inducing a problem dependent coarsening. This aggregation is combined with piecewise constant (unsmoothed) prolongation, ensuring low setup cost and memory requirements. Compared with previous aggregation-based multigrid methods, the scalability is enhanced by using a so-called K-cycle multigrid scheme, providing Krylov subspace acceleration at each level. This paper is the logical continuation of [SIAM J. Sci. Comput., 30 (2008), pp. 1082–1103], where the analysis of a anisotropic model problem shows that aggregation-based two-grid methods may have optimal order convergence, and of [Numer. Lin. Alg. Appl., 15 (2008), pp. 473–487], where it is shown that K-cycle multigrid may provide optimal or near optimal convergence under mild assumptions on the two-grid scheme. Whereas in these papers only model problems with geometric aggregation were considered, here a truly algebraic method is presented and tested on a wide range of discrete second order scalar elliptic PDEs, including nonsymmetric and unstructured problems. Numerical results indicate that the proposed method may be significantly more robust as black box solver than the classical AMG method as implemented in the code AMG1R5 by K. Stüben. The parallel implementation of the method is also discussed. Satisfactory speedups are obtained on a medium size multi-processor cluster that is typical of today computer market. A code implementing the method is freely available for download both as a FORTRAN program and a MATLAB function.

Key words. Multigrid, linear systems, iterative methods, AMG, preconditioning, parallel computing.

AMS subject classifications. 65F10, 65N55.

1. Introduction.

We consider the iterative solution of large sparse $n \times n$ linear systems

$$A\mathbf{u} = \mathbf{b}$$

arising from the discretization of second order elliptic PDEs. In this context, multigrid methods [39] are among the most efficient solution techniques. Whereas geometric multigrid methods require a predetermined hierarchy of grids and discretizations, algebraic multigrid (AMG) methods are set up using only the information present in the system matrix [3, 37].

These algorithms combine the effect of a *smoother* and a *coarse grid correction*. In AMG schemes, the smoother is fixed and generally based on a simple iterative method such as the (symmetric) Gauss-Seidel method. The coarse grid correction consists of computing an approximate solution to the residual equation on a coarser grid, that is, solving a linear system of smaller size. This solution is then transferred back to the actual grid by means of an appropriate prolongation. In AMG methods, this coarse grid correction is entirely defined once the prolongation is known, that is, once an appropriate prolongation matrix has been set up by applying a so-called *coarsening* algorithm to the system matrix.

The improvement of AMG schemes is a hot research topic. The current trend (e.g., [4, 5, 7, 15, 18]) leads to more involved algorithms with denser prolongation matrices, which increases setup costs and memory requirements. In this paper, we take the opposite viewpoint. We consider coarsening by aggregation of the unknowns, which leads to prolongation matrices with at most one nonzero entry per row, which are much sparser than the ones obtained by the classical AMG approach, as developed in [3, 34, 36, 37].

*Received February 14, 2008. Accepted December 4, 2009. Published online April 25, 2010. Recommended by Thomas Manteuffel.

†Service de Métrologie Nucléaire (C.P. 165/84), Université Libre de Bruxelles, 50, Av. F.D. Roosevelt, B-1050 Brussels, Belgium. Supported by the Belgian FNRS (“Directeur de recherches”) (ynotay@ulb.ac.be).

ETNA
Kent State University
http://etna.math.kent.edu

Aggregation schemes are not new and trace back to [2, 6][1]. They are not much popular because it is difficult to obtain grid independent convergence on this basis [37, pp. 522–524]; see also [43, p. 663], where a three-grid analysis is presented for the Poisson model problem. This may be connected to the fact that aggregation-based prolongations do not correspond to an interpolation which is exact for all polynomials of degree 1, as required by the theory of geometric multigrid [14, Sections 3.5 and 6.3.2].

That is why aggregation is often associated with *smoothed* aggregation, as introduced in [41, 42]. There, it is proposed to overcome intrinsic difficulties associated with aggregation, by *smoothing* the interpolation matrix, that is, an effective prolongation is obtained from a "tentative" aggregation-based prolongation matrix P_0 (with one nonzero entry per row), letting $P = MP_0$, where M is a matrix that smooths the interpolation; e.g., $M = I - \omega A$, where ω is a relaxation parameter. In this work, we take an opposite viewpoint and stay with the "pure" (unsmoothed) prolongation matrix.

The approach we follow is the logical continuation of two previous works [26, 31]. In [26], analyzing a two-dimensional anisotropic model problem, it is shown that aggregation-based *two*-grid methods may have optimal order convergence properties (with respect to the number of unknowns) provided that, in the presence of anisotropy, aggregates are formed following the strong coupling direction. It does not mean that the above mentioned difficulties are not real, but that they appear when considering *multi*-grid V- or W-cycles, which scale poorly with the number of levels. Here the results in [31] enter the scene. This paper introduces K-cycle multigrid, with which Krylov subspace acceleration is applied at every level. It is shown that this provides enhanced robustness and scalability, compared with standard V- or W-cycles, giving support to earlier observations in, e.g., [32]. Furthermore, preliminary numerical results in [26] indicate that aggregation-based multigrid methods may then indeed exhibit convergence that is independent or near-independent of the number of levels.

These results were obtained for symmetric positive definite two-dimensional model problems, with regular (geometric-based) aggregation schemes. In this work, we address more general problems, including nonsymmetric ones, and consider automatic aggregation with an appropriate coarsening algorithm, resulting in a fully algebraic method. As in most previous aggregation-based multilevel algorithms [2, 21, 30], we start with a pairwise aggregation scheme, forming pairs or "matchings" in the matrix graph. However, this results in a relatively slow coarsening, which does not permit the efficient use of K-cycle schemes. Hence we repeat the process, forming in a second pass pairs of pairs from the first pass. We call this strategy "double pairwise aggregation". Note that we essentially reuse the algorithms proposed in [30] for another type of multilevel method.

We also discuss the parallelization of the algorithms used. With the classical AMG approach, the parallelization of the coarsening raises some nontrivial issues [10, 16, 22]. Here, assuming that each task running in parallel receives a portion of the matrix rows, we show that the coarsening algorithm may be applied independently to these subsets of rows, forming aggregates with the corresponding (local) variables. The associated restriction and prolongation do not require any communication, and good scalability is achieved with a purely local smoother of block Jacobi type.

It is worth mentioning the method in [21], which presents some similarity with ours: it is also based on a pairwise aggregation algorithm and stays with "pure" aggregation-based prolongation. However, besides the technical details in the aggregation algorithm and the smoother, there are two major differences. In [21], only simple pairwise aggregation is considered, whereas we use double pairwise aggregation, which makes the coarsening just twice

[1]The aggregation concept was introduced earlier in other fields; e.g., its use in economics dates back as far as [24].

ETNA
Kent State University
http://etna.math.kent.edu

as fast. Next, we use K-cycle multigrid to circumvent the relatively bad scalability of the standard V-cycle, whereas the method in [21] resorts to V-cycle improved by doubling the number of smoothing steps from one level to the next.

This paper is organized as follows. The coarsening is presented and discussed in Section 2. The solution method is described in detail in Section 3. The results of numerical experiments are reported in Section 4, and some concluding remarks are given in Section 5.

2. Coarsening. An algebraic coarsening algorithm sets up a prolongation matrix P using only the information available in A. The prolongation is an $n \times n_c$ matrix, where $n_c < n$ is the number of coarse variables. It allows to transfer on the fine grid a vector defined on the coarse variable set $[1, n_c]$. Further, it entirely determines the coarse grid correction. Formally, the latter also depends on a restriction matrix and on a coarse grid matrix, but, as it is usual with AMG methods, one takes the restriction equal to the transpose of the prolongation, and the coarse grid matrix is computed from the Galerkin formula

$$A_c = P^T AP. \tag{2.1}$$

In the classical AMG coarsening [37], one first selects a subset of fine grid variables as coarse variable, by inspecting the graph of A. Next, the matrix entries are used to build interpolation rules, that define P.

Coarsening by aggregation works differently. One needs to define *aggregates* G_i, which are disjoint subsets of the variable set. The number of coarse variables n_c is then the number of such subsets, and P is given by

$$P_{ij} = \begin{cases} 1, & \text{if } i \in G_j, \\ 0, & \text{otherwise,} \end{cases} \qquad (1 \le i \le n, 1 \le j \le n_c). \tag{2.2}$$

If $\cup_i G_i = [1, n]$ (i.e., if the aggregates form a partitioning of $[1, n]$), P is a Boolean matrix with exactly one nonzero entry per row. As seen below, it is however sometimes advantageous to leave some variables outside the set of aggregates, in which case the rows of P corresponding to these variables are zero. Note that there is no need to explicitly form P, and the coarse grid matrix (2.1) is in practice computed by

$$(A_c)_{ij} = \sum_{k \in G_i} \sum_{\ell \in G_j} a_{k\ell} \qquad (1 \le i, j \le n_c). \tag{2.3}$$

Many aggregation algorithms proposed in the literature (e.g., [2, 21, 30]) starts by forming pairs, or “matchings”, in the matrix graph. Here we reuse the algorithm from [30], because it discriminates between different neighbors of a node, giving preference to the strongest negative coupling(s). For two-dimensional anisotropic model problems, this produces aggregates aligned with the strong coupling direction, as desired according to the analysis in [26]. Some connection may also be made with the classical AMG coarsening, which is also based on strong negative couplings.

Details are given in Algorithm 2.1, which works as follows. Essentially like in the classical AMG coarsening [37, p. 473], one first defines the set of nodes S_i to which i is strongly negatively coupled, using the Strong/Weak coupling threshold β:

$$S_i = \left\{ j \neq i \,|\, a_{ij} < -\beta \max_{a_{ik}<0} |a_{ik}| \right\}.$$

Then, one picks up an unmarked node i at a time, giving priority to node(s) with minimal m_i, where m_i is the number of unmarked nodes that are strongly negatively coupled to i (that is,

ETNA
Kent State University
http://etna.math.kent.edu

ALGORITHM 2.1 (Pairwise aggregation).

Input: Matrix $A = (a_{ij})$ with n rows;
Strong/Weak coupling threshold β (default: $\beta = \mathbf{0.25}$);
Logical parameter *CheckDD*.
Output: Number of coarse variables n_c and subset (aggregates)
G_i, $i = 1 \ldots, n_c$ (such that $G_i \cap G_j = \emptyset$ for $i \neq j$).

Initialization: If (*CheckDD*): $U = [1, n] \backslash \{ i \,|\, a_{ii} > 5 \sum_{j \neq i} |a_{ij}| \}$,
otherwise: $U = [1, n]$,
For all i: $S_i = \{ j \in U \backslash \{i\} \,|\, a_{ij} < -\beta \max_{a_{ik}<0} |a_{ik}| \}$;
For all i: $m_i = |\{ j \,|\, i \in S_j \}|$;
$n_c = 0$.

Algorithm: While $U \neq \emptyset$ do
1. Select $i \in U$ with minimal m_i; $n_c = n_c + 1$.
2. Select $j \in U$ such that $a_{ij} = \min_{k \in U} a_{ik}$.
3. If $j \in S_i$: $G_{n_c} = \{i, j\}$,
   otherwise: $G_{n_c} = \{i\}$.
4. $U = U \backslash G_{n_c}$.
5. For all $k \in G_{n_c}$, update: $m_\ell = m_\ell - 1$ for $\ell \in S_k$.

---

m_i is the number of sets S_j to which i belongs and that correspond to an unmarked node j). This rule is designed to favor a regular covering of the matrix graph. Probably other rules could work as well. We consider this one because we observed that in some nonsymmetric examples, at any stage of the process, there is at least one node j for which $m_j = 0$. Then, if this node is not selected, it has many chances to become a singleton, since no unmarked node is strongly negatively coupled to it.

A tentative aggregate is next formed by grouping the picked up node i with the unmarked node it is most strongly negatively coupled to, that is, (one of) the unmarked node(s) j for which a_{ij} is minimal. This tentative pair is then accepted if and only if $j \in S_i$. If not, the node i initially picked up stays alone in the aggregate. This may occur only if all nodes k for which a_{ik} is minimal have already been marked (or do not correspond to a negative coupling). For the kind of applications targeted here, this occurs only incidentally, for instance near boundaries. In general, the algorithm progresses smoothly along the direction of strongest negative coupling, and most picked up nodes are effectively associated with (one of) their neighbor(s) they are most strongly negatively coupled with. Note that, as long as this happens, one has, at step 3 of Algorithm 2.1, $j \in S_i$ whatever the chosen Strong/Weak coupling threshold β, and it is also unimportant whether the other couplings in row i are labeled "strong" or "weak". Hence β has only a slight influence on the coarsening process, and its role is much less critical than in the classical AMG coarsening.

In Algorithm 2.1, we also include an optional check for rows strongly dominated by their diagonal element. The nodes corresponding to these rows are marked, but they are not associated to any aggregate. We apply this only to the top level matrix, based on the heuristic argument that, when a row is strongly diagonally dominant enough, a fast reduction of the error at the corresponding node can be obtained without multilevel enhancement. This process also guarantees a proper treatment of finite element matrices in which Dirichlet boundary conditions have been imposed by adding a large number to the corresponding diagonal elements.

Coarsening by simple pairwise aggregation is relatively slow, which does not favor op-

ETNA
Kent State University
http://etna.math.kent.edu

timal performance of multilevel methods. A faster coarsening can be obtained by repeating the process, defining aggregates by forming pairs of pairs—more precisely, by forming pairs of aggregates from the first pass, some of which are singletons. Forming pairs of pairs is considered in [2], but the procedure tries to maximize the number of edges internal to the aggregates. For two-dimensional discrete PDEs, this favors a coarsening that mimics well the geometric coarsening with "boxwise" aggregates. However, the analysis in [26] shows that in the presence of anisotropy, it is better to use "linewise" aggregates, aligned with the direction of strong negative coupling. This motivates us to follow the approach in [30], where the second pass just exploits the same pairwise aggregation algorithm used in the first pass. That is, we compute the coarse grid matrix (2.3) resulting from a first application of Algorithm 2.1, and apply again this algorithm to the latter matrix, forming pairs of aggregates from the first pass. Details are given in Algorithm 2.2.

ALGORITHM 2.2 (Double pairwise aggregation).

Input: Matrix $A = (a_{ij})$;
Strong/Weak coupling threshold β (default: $\beta = \mathbf{0.25}$);
Logical parameter *CkDD*.
Output: Number of coarse variables n_c and subset (aggregates)
G_i, $i = 1 \ldots, n_c$ (such that $G_i \cap G_j = \emptyset$ for $i \neq j$).
Algorithm:

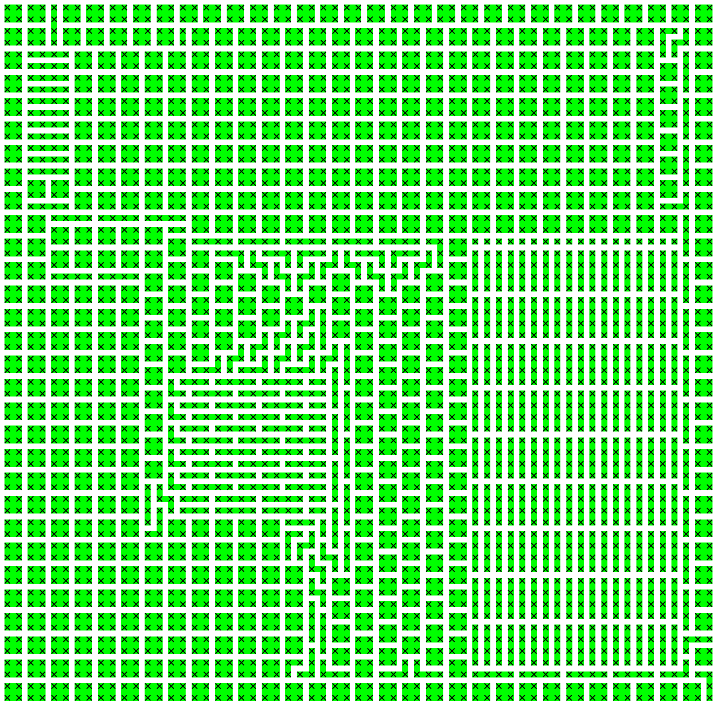
1. Apply Algorithm 2.1 to A with threshold β
   and *CheckDD=CkDD*.
   Output: n_{c_1}, and $G_i^{(1)}$, $i = 1, \ldots, n_{c_1}$.
2. Compute the $n_{c_1} \times n_{c_1}$ auxiliary matrix $A_1 = \left(a_{ij}^{(1)}\right)$ with
$$a_{ij}^{(1)} = \sum_{k \in G_i^{(1)}} \sum_{\ell \in G_j^{(1)}} a_{k\ell}.$$
3. Apply Algorithm 2.1 to A_1 with threshold β
   and *CheckDD=False*.
   Output: n_c and $G_i^{(2)}$, $i = 1, \ldots, n_c$.
4. For $i = 1, \ldots, n_c$: $G_i = \cup_{j \in G_i^{(2)}} G_j^{(1)}$.

For the class of applications considered in this paper, the resulting aggregates are mostly quadruplets, with some triplets, pairs and singletons left. Consequently, the coarsening ratio n/n_c is slightly less than 4, regardless of the problem at hand. Applied to two-dimensional model problems, the procedure produces essentially "boxwise" aggregates if the coefficients are isotropic, whereas strong anisotropy induces "linewise" aggregates aligned with the direction of strong negative coupling. It is more difficult to figure out what comes out for three-dimensional problems, especially when the coefficients are isotropic, but the numerical results show that the approach is robust.

Of course, to generate a multilevel structure, Algorithm 2.2 is applied recursively to the successive coarse grid matrices. This is illustrated in Figure 2.1 for the problem JUMP2D and in Figure 2.2 for the problem CD1. These problems correspond to the five-point finite difference discretization of elliptic PDEs, and are fully described in Section 4. Problem JUMP2D is the standard diffusion equation with jumps in the coefficients, which are isotropic everywhere but in two rectangular regions. One indeed sees in the pictures that the coarsening

ETNA
Kent State University
http://etna.math.kent.edu

First coarse grid



Second coarse grid

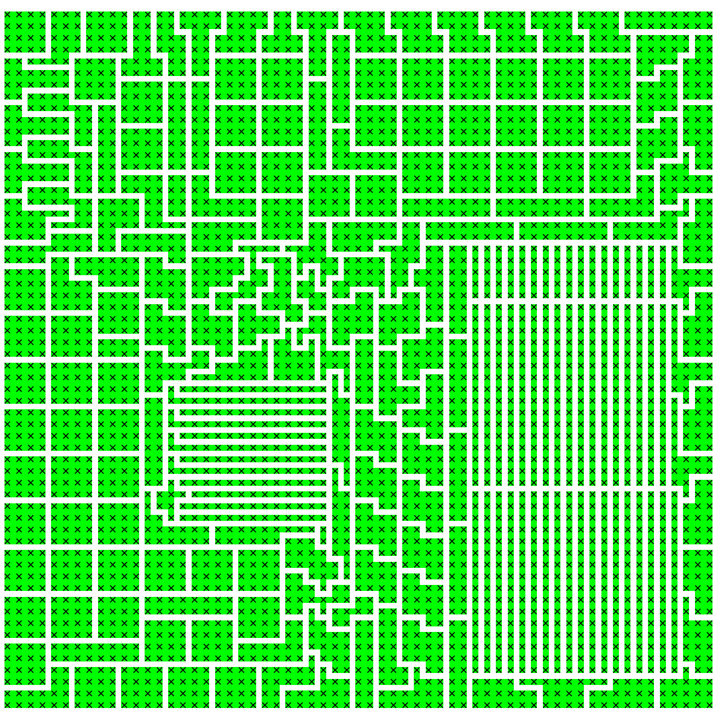


FIGURE 2.1. *Coarsening for problem* JUMP2D *(*$h = 1/60$*).*

ETNA
Kent State University
http://etna.math.kent.edu

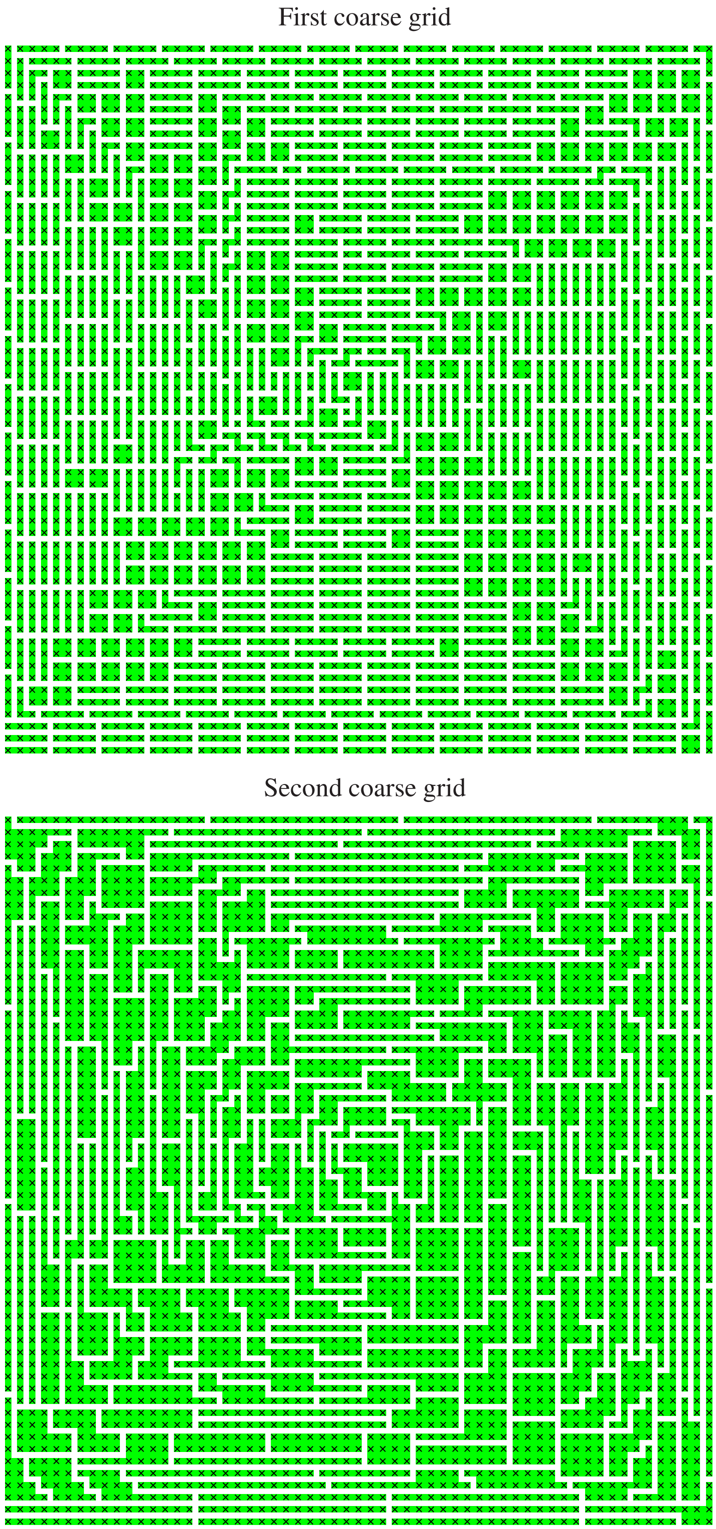


FIGURE 2.2. *Coarsening for problem* CD1 *(*$h = 1/60$ *and* $\nu = 10^{-4}$*).*

ETNA
Kent State University
http://etna.math.kent.edu

proceeds differently in these regions, producing aggregates aligned with the strong coupling direction. Problem CD1 is a convection-diffusion equation with a divergence-free convective flow rotating around the middle of the domain. Here, one sees that the coarsening "follows" the flow, producing aggregates aligned with it in the regions where it is largest in magnitude.

2.1. Parallelization. Coarsening the matrix graph by aggregation is one of the key ingredients of the multilevel partitioning strategy proposed in [20]. From there, a possible way to parallelize our method consists of applying the coarsening sequentially, and then partition the matrix at the coarsest level, inducing the partitioning at all levels, as in [20]. However, this raises several issues that lie outside the scope of this paper, such as the sequential bottleneck represented by this coarsening, and the selection of the partitioning strategy at the coarsest level. Note, nevertheless, that the resulting method would have exactly the same grid hierarchy for sequential and parallel executions.

We discuss in more detail another context, in which several instances of the program run in parallel, each having only a portion of the rows of the matrix. It means that some partitioning has been applied before calling the solution module, for instance at the level of the discretization or with a standard tool like METIS [19]. It is then important to work with local information only, as assembling the global matrix on one processor may be infeasible.

In such cases, our coarsening algorithm may be parallelized in the following natural way: each task has a portion of the rows, that is, the local matrix is a rectangular matrix with as many rows as local variables, but more columns, the extra columns corresponding to non-local variables. Observing that Algorithm 2.1 accesses the matrix only row by row, it may be applied as is to this local rectangular matrix, with the convention that the local variables have indices $1, \ldots, n$, corresponding to local rows. Hence, non-local variables are treated as if they were already marked (the set of unmarked nodes is initialized to $[1, n]$), and they are therefore excluded from the aggregation process. Non-local variables are however taken into account when checking if the diagonal dominance is strong enough, and in the definition of S_i, to decide if a local coupling is strong or not. Algorithm 2.2 may also be applied as is, with a slight modification of step 2, to take into account that the first application of Algorithm 2.1 produces aggregates for local nodes only. To avoid any extra communication at this stage, we use

$$a_{ij}^{(1)} = \begin{cases} \displaystyle\sum_{k \in G_i^{(1)}} \sum_{\ell \in G_j^{(1)}} a_{k\ell}, & \text{if } 1 \le i, j \le n_{c_1}, \\ \displaystyle\sum_{k \in G_i^{(1)}} a_{kj}, & \text{if } 1 \le i \le n_{c_1} \text{ and } j \notin [1, n], \end{cases}$$

that is, A_1 is built as if all non-local variables were associated to distinct aggregates, and the corresponding column indices were kept untouched.

Hence, each task computes locally the coarsening scheme for its local variables. Communication is only needed to compute the coarse grid matrix, which requires knowledge of the coarsening of non-local variables. Further, because aggregates are all formed with variables assigned to the same task, applying the associated prolongation (2.2) or its transpose to some vector does not require any communication.

This parallel coarsening is illustrated in Figure 2.3 for the problem JUMP2D, parallelized according to a partitioning of the domain in 3×2 rectangular subdomains. One sees that the aggregates do not cross over subdomain boundaries, and have on the whole a similar shape, as in the sequential coarsening.

3. The solution algorithm. We first describe the sequential implementation. The few adaptations needed by the parallel version are discussed later.

ETNA
Kent State University
http://etna.math.kent.edu

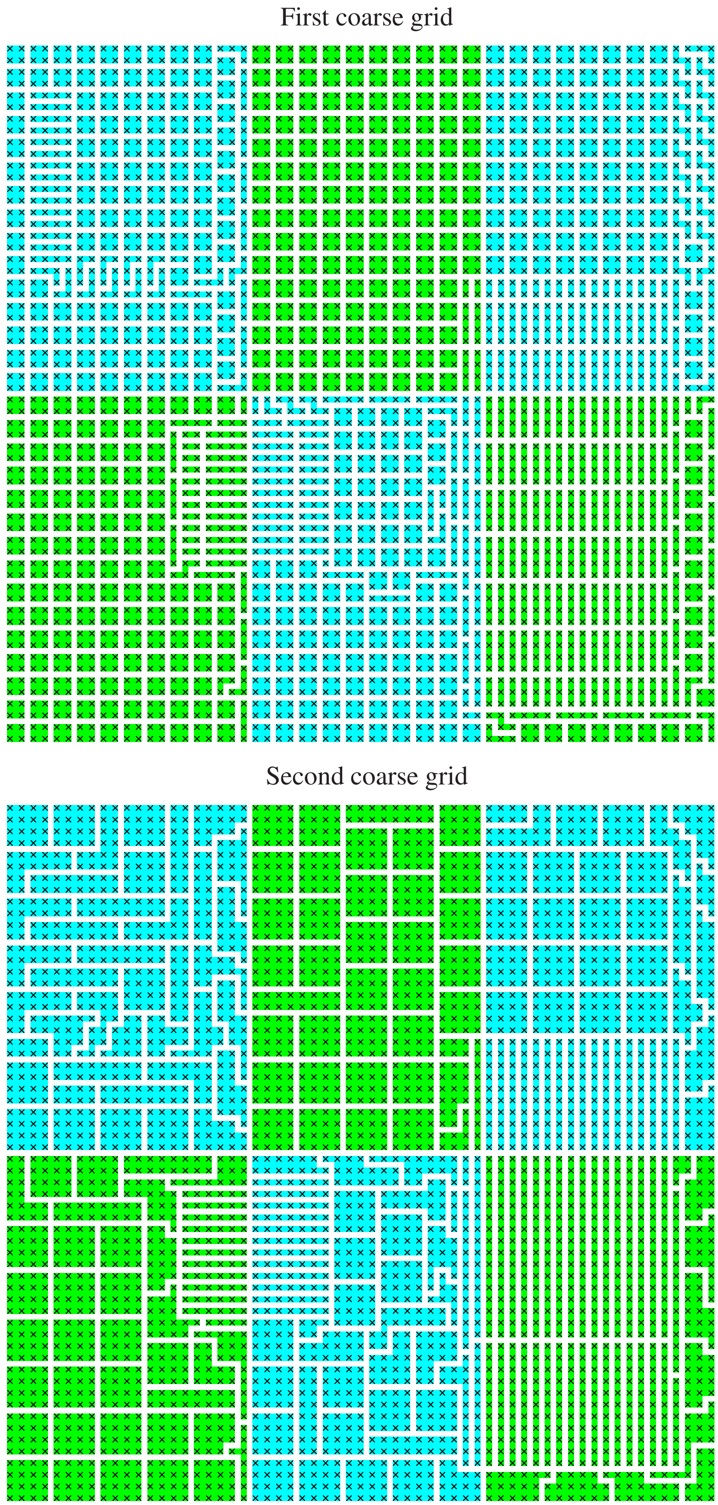


FIGURE 2.3. *Parallel coarsening for problem* JUMP2D *(*$h = 1/60$*).*

We use multigrid as a preconditioner in a main iteration routine, which is based on the flexible conjugate gradient method (FCG), if the matrix is symmetric positive definite. More precisely, we use FCG(1) from [29] or, equivalently, IPCG from [13]. This method is similar to the standard conjugate gradient method, except that scalar coefficients are computed in a slightly different way, at the expense of one more inner product computation per iteration. This modification enhances the stability of the method in the presence of variable preconditioning. It is needed here, because use of the K-cycle induces slight variations in the multigrid preconditioning.

In nonsymmetric cases, we use a preconditioned variant of GCR [11], referred to as GMRESR in [40]. This method provides the minimal residual norm solution and allows for variable preconditioning. We use an improved implementation, given in Algorithm 3.1 for the sake of completeness. In the standard implementation, at step 3(b) one applies to $\mathbf{z}_j$ a recursion similar to the one applied to $\mathbf{c}_j^{(\cdot)}$. This allows us to obtain the approximate solution at each step with a simple recursion, but doubles the cost of step 3(b), which is the most expensive part of the algorithm. In Algorithm 3.1, instead, the computation of the approximate solution is performed only upon completion of the main loop. Note that one has effectively $\mathbf{r}_m = \mathbf{b} - A\mathbf{u}_m$, because $\mathbf{r}_m = \mathbf{r}_0 - (\mathbf{c}_1 \ \cdots \ \mathbf{c}_m)\mathbf{a}$ whereas $A(\mathbf{z}_1 \ \cdots \ \mathbf{z}_m) = (\mathbf{c}_1 \ \cdots \ \mathbf{c}_m)\Gamma$, entailing $\mathbf{r}_m = \mathbf{r}_0 - A(\mathbf{z}_1 \ \cdots \ \mathbf{z}_m)(\Gamma^{-1}\mathbf{a})$. This further shows that Algorithm 3.1 is mathematically equivalent to the original GCR/GMRESR algorithm, since the residual is computed as in the latter. This improved implementation is discussed in detail in [17] for the unpreconditioned case, where it is shown to also have superior stability properties.

ALGORITHM 3.1 (Preconditioned GCR – economical version).

Data: Matrix A; right-hand-side $\mathbf{b}$; initial approximation $\mathbf{u}_0$;
Maximal number of iterations m; tolerance ε.
Output: Approximate solution $\mathbf{u}_m$; residual $\mathbf{r}_m = \mathbf{b} - A\mathbf{u}_m$.
Initialization: $\mathbf{r}_0 = \mathbf{b} - A\mathbf{u}_0$.
Algorithm:
For $j = 1, \dots, m$ do
1. Apply preconditioner: $\mathbf{z}_j$ =Prec$(\mathbf{r}_{j-1})$.
2. $\mathbf{c}_j^{(1)} = A\mathbf{z}_j$.
3. For $i = 1, \dots, j-1$ do
   (a) $\gamma_{ij} = \mathbf{c}_i^T \mathbf{c}_j^{(i)}$,
   (b) $\mathbf{c}_j^{(i+1)} = \mathbf{c}_j^{(i)} - \gamma_{ij}\mathbf{c}_i$,
4. $\gamma_{jj} = \|\mathbf{c}_j^{(j)}\|$; $\quad \mathbf{c}_j = \gamma_{jj}^{-1}\mathbf{c}_j^{(j)}$.
5. $\alpha_j = \mathbf{c}_j^T \mathbf{r}_{j-1}$; $\quad \mathbf{r}_j = \mathbf{r}_{j-1} - \alpha_j \mathbf{c}_j$.
6. **If** $\|\mathbf{r}_j\| < \varepsilon\|\mathbf{b}\|$, **exit** do loop and reset $m = j$.

$\mathbf{u}_m = \begin{bmatrix}\mathbf{z}_1 & \cdots & \mathbf{z}_m\end{bmatrix} \left(\Gamma^{-1}\mathbf{a}\right)$
where

$$\mathbf{a} = \begin{bmatrix}\alpha_1 \\ \vdots \\ \alpha_m\end{bmatrix} \qquad \text{and} \qquad \Gamma_{ij} = \begin{cases}\gamma_{ij}, & \text{if } i \le j, \\ 0, & \text{otherwise.}\end{cases}$$

In our experiments, to save on computational cost and memory requirements, we use GCR(10), that is, the maximal number of iteration is set to 10, and Algorithm 3.1 is restarted

ETNA
Kent State University
http://etna.math.kent.edu

if needed. With the improved implementation, preconditioned GCR has about the same cost as flexible GMRES (FGMRES [35]). GCR is further slightly cheaper upon restart, because FGMRES requires an additional matrix-vector product to compute the residual. In fact, we also tested FGMRES, and found that it was slightly slower in most cases, in agreement with the conclusions in [23].

ALGORITHM 3.2 (Multigrid as a preconditioner at level k ($k \geq 1$)).

Input: $\mathbf{r}_k$.

Output: $\mathbf{z}_k = \text{MGprec}(\mathbf{r}_k, k)$.

Data: matrix A_k, smoother M_k, prolongation P_k; matrix A_{k-1};
if $k > 1$: cycle type (V or K), iteration type (FCG or GCR), threshold t.

Algorithm:

1. Relax using smoother M_k: $\mathbf{z}_k^{(1)} = M_k^{-1}\mathbf{r}_k$.
2. Compute new residual: $\widetilde{\mathbf{r}}_k = \mathbf{r}_k - A_k\mathbf{z}_k^{(1)}$.
3. Restrict residual: $\mathbf{r}_{k-1} = P_k^T\widetilde{\mathbf{r}}_k$.
4. Compute an (approximate) solution $\widetilde{\mathbf{x}}_{k-1}$ to $A_{k-1}\mathbf{x}_{k-1} = \mathbf{r}_{k-1}$:
   if $k = 1$, **then** $\widetilde{\mathbf{x}}_{k-1} = A_{k-1}^{-1}\mathbf{r}_{k-1}$;
   else if V-cycle, **then** $\widetilde{\mathbf{x}}_{k-1} = \text{MGprec}(\mathbf{r}_{k-1}, k-1)$;
   else if K-cycle, **then** Perform 1 or 2 iterations with multigrid prec.:
   $\mathbf{c}_{k-1} = \text{MGprec}(\mathbf{r}_{k-1}, k-1)$; $\quad \mathbf{v}_{k-1} = A_{k-1}\mathbf{c}_{k-1}$;
   $$\begin{cases} \rho_1 = \mathbf{c}_{k-1}^T\mathbf{v}_{k-1}; \quad \alpha_1 = \mathbf{c}_{k-1}^T\mathbf{r}_{k-1}; & \text{if FCG} \\ \rho_1 = \|\mathbf{v}_{k-1}\|^2; \quad \alpha_1 = \mathbf{v}_{k-1}^T\mathbf{r}_{k-1}; & \text{if GCR} \end{cases}$$
   $\widetilde{\mathbf{r}}_{k-1} = \mathbf{r}_{k-1} - \frac{\alpha_1}{\rho_1}\mathbf{v}_{k-1}$;
   if $\|\widetilde{\mathbf{r}}_{k-1}\| \leq t\|\mathbf{r}_{k-1}\|$, **then** $\widetilde{\mathbf{x}}_{k-1} = \frac{\alpha_1}{\rho_1}\mathbf{c}_{k-1}$;
   else
   $\mathbf{d}_{k-1} = \text{MGprec}(\widetilde{\mathbf{r}}_{k-1}, k-1)$; $\quad \mathbf{w}_{k-1} = A_{k-1}\mathbf{d}_{k-1}$;
   $$\begin{cases} \gamma = \mathbf{d}_{k-1}^T\mathbf{v}_{k-1}; \beta = \mathbf{d}_{k-1}^T\mathbf{w}_{k-1}; \alpha_2 = \mathbf{d}_{k-1}^T\widetilde{\mathbf{r}}_{k-1}; & \text{if FCG} \\ \gamma = \mathbf{w}_{k-1}^T\mathbf{v}_{k-1}; \beta = \|\mathbf{w}_{k-1}\|^2; \alpha_2 = \mathbf{w}_{k-1}^T\widetilde{\mathbf{r}}_{k-1}; & \text{if GCR} \end{cases}$$
   $\rho_2 = \beta - \frac{\gamma^2}{\rho_1}$;
   $\widetilde{\mathbf{x}}_{k-1} = \left(\frac{\alpha_1}{\rho_1} - \frac{\gamma\alpha_2}{\rho_1\rho_2}\right)\mathbf{c}_{k-1} + \frac{\alpha_2}{\rho_2}\mathbf{d}_{k-1}$;
   end if
   end if
5. Prolongate coarse-grid correction: $\mathbf{z}_k^{(2)} = P_k\widetilde{\mathbf{x}}_{k-1}$.
6. Compute new residual: $\overline{\mathbf{r}}_k = \widetilde{\mathbf{r}}_k - A_k\mathbf{z}_k^{(2)}$.
7. Relax using smoother M_k: $\mathbf{z}_k^{(3)} = M_k^{-1}\overline{\mathbf{r}}_k$.
8. $\mathbf{z}_k = \mathbf{z}_k^{(1)} + \mathbf{z}_k^{(2)} + \mathbf{z}_k^{(3)}$.

Details of the multigrid preconditioning are given in Algorithm 3.2. It is called by the main iteration routine at the top level $k = \ell$ with the matrix $A_\ell = A$, and it recursively calls itself with a smaller index until $k = 1$. We thus follow the usual convention that coarser levels correspond to smaller indexes, although the number of levels is not known in advance: in fact, we stop the coarsening when the coarse grid matrix has 200 rows, or less, allowing fast direct inversion with LAPACK routines [1].

For the sake of clarity, Algorithm 3.2 foresees only one pre- and one post-smoothing

ETNA
Kent State University
http://etna.math.kent.edu

step. In our experiments, we use symmetric Gauss-Seidel smoothing, that is,

$$M_k = \mathrm{low}(A_k)\,\mathrm{diag}(A_k)^{-1}\mathrm{upp}(A_k), \tag{3.1}$$

where low$(\cdot)$, diag$(\cdot)$ and upp$(\cdot)$ stand for the lower, the diagonal, and the upper triangular part of a matrix, respectively. Numerical results indicate that, with this smoother, the scheme is indeed most efficient with only one pre- and one post-smoothing step. Note that the word "symmetric" refers to the fact that this smoother brings the same effect as one forward Gauss-Seidel sweep, followed by one backward Gauss-Seidel sweep, that is, the Gauss-Seidel scheme is symmetrized. However, M_k itself is symmetric if and only if A_k is symmetric.

At step 4, the coarse grid system is solved with either a V-cycle or a K-cycle formulation. The V-cycle corresponds to one application of the preconditioner at the next coarser level, whereas with the K-cycle a few steps of a Krylov subspace iterative method are performed. According to the conclusions from [31], at most 2 iterations are allowed, and the second one is skipped if the relative residual error is below the threshold t after the first step. In practice, as discussed at the beginning of Section 4, we use $t = 0.25$.

The Krylov subspace iterative method used here is also either FCG (for symmetric positive definite matrices) or GCR. The implementation inside step 4 of Algorithm 3.2 is nonstandard: it takes advantage at the fact that the number of iterations is at most 2, to minimize the work and the number of synchronization points. This implementation is, however, mathematically equivalent to the standard one. This is straightforward to check if only one iteration is performed. Otherwise, this may be seen by checking that the residual corresponding to the computed solution $\widetilde{\mathbf{x}}_{k-1}$ is orthogonal to $\mathbf{c}_{k-1}(\mathbf{d}_{k-1}$ in case of FCG), and orthogonal to $\mathbf{v}_{k-1}(\mathbf{w}_{k-1}$ in case of GCR). This indeed ensures that the computed solution is the linear combination of $\mathbf{c}_{k-1}$ and $\mathbf{d}_{k-1}$ which minimizes the A_{k-1}-norm of the error in case of FCG, and the residual norm in case of GCR.

Now, this residual is

$$\overline{\mathbf{r}}_{k-1} = \mathbf{r}_{k-1} - \left(\frac{\alpha_1}{\rho_1} - \frac{\gamma\alpha_2}{\rho_1\rho_2}\right)\mathbf{v}_{k-1} - \frac{\alpha_2}{\rho_2}\mathbf{w}_{k-1}.$$

Since $\mathbf{c}_{k-1}^T\mathbf{w}_{k-1} = \mathbf{d}_{k-1}^T\mathbf{v}_{k-1}$, it holds that

$$\left\{\begin{array}{l} \mathbf{c}_{k-1}^T\overline{\mathbf{r}}_{k-1}\ \text{(for FCG)} \\ \mathbf{v}_{k-1}^T\overline{\mathbf{r}}_{k-1}\ \text{(for GCR)} \end{array}\right\} = \alpha_1 - \left(\frac{\alpha_1}{\rho_1} - \frac{\gamma\alpha_2}{\rho_1\rho_2}\right)\rho_1 - \frac{\alpha_2}{\rho_2}\gamma = 0,$$

whereas, using

$$\overline{\mathbf{r}}_{k-1} = \widetilde{\mathbf{r}}_{k-1} + \frac{\gamma\alpha_2}{\rho_1\rho_2}\mathbf{v}_{k-1} - \frac{\alpha_2}{\rho_2}\mathbf{w}_{k-1},$$

one obtains

$$\left\{\begin{array}{l} \mathbf{d}_{k-1}^T\overline{\mathbf{r}}_{k-1}\ \text{(for FCG)} \\ \mathbf{w}_{k-1}^T\overline{\mathbf{r}}_{k-1}\ \text{(for GCR)} \end{array}\right\} = \alpha_2 + \frac{\gamma\alpha_2}{\rho_1\rho_2}\gamma - \frac{\alpha_2}{\rho_2}\beta = \frac{\alpha_2}{\rho_2}\left(\rho_2 + \frac{\gamma^2}{\rho_1} - \beta\right) = 0.$$

Hence, in both cases the required orthogonality conditions are satisfied.

The coarsening algorithm has been designed to ensure a reduction by a factor of about 4 of the number of variables from one level to the next, while avoiding a significant increase of the mean number of nonzero entries per row. If this is achieved, the K-cycle formulation may be used at every level while keeping the overall cost bounded. Indeed, let C be a constant such that the cost of one iteration with the multigrid preconditioner at level k, except step 4, is bounded by $C\,nnz(A_k)$, where $nnz(\cdot)$ stands for the number of nonzero entries. Further, let σ be an upper bound for $nnz(A_{k-1})/nnz(A_k)$, valid for $k = 2, \ldots, \ell$; note that σ defined in this way depends on the coarsening ratio n_k/n_{k-1}, but takes also into account a possible increase of the mean number of nonzero entries per row. Assuming the cost of the inversion of A_0 to be negligible, the global cost of each main (top level) iteration is then bounded by summing the contribution of the top level, and that of each level k ($1 \le k < \ell$), taking into account the number of allowed iterations (2), and the maximal number of times this level is visited in the recursion ($2^{\ell-k-1}$):

$$\text{Cost} \le C\left(nnz(A_\ell) + \sum_{k=1}^{\ell-1} 2^{\ell-k} nnz(A_k)\right) \tag{3.2}$$

$$\le C\,nnz(A_\ell)\sum_{k=1}^{\ell}(2\sigma)^{\ell-k}$$

$$\le \frac{2\sigma}{1-2\sigma}\,C\,nnz(A). \tag{3.3}$$

This is indeed nicely bounded if σ is close to $1/4$.

However, there is no a priori guarantee on the coarsening speed. We therefore set the parameters η_k, $k = \ell-1, \ell-2, \ldots, 2$, as follows

$$\eta_k = \begin{cases} 2, & \text{if } \dfrac{nnz(A_\ell)}{nnz(A_k)}\xi^{\ell-k}\left(\displaystyle\prod_{j=k+1}^{\ell-1}\eta_j\right)^{-1} \ge \dfrac{3}{2}, \\ 1, & \text{otherwise,} \end{cases} \tag{3.4}$$

where $\xi = \frac{3}{5}$. Then, at level k, we use Algorithm 3.2 with V-cycle if $\eta_{k-1} = 1$, and with K-cycle if $\eta_{k-1} = 2$. Note that, since $\prod_{j=k+1}^{\ell-1}\eta_j \le 2^{\ell-k-1}$, one has

$$\frac{nnz(A_\ell)}{nnz(A_k)}\xi^{\ell-k}\left(\prod_{j=k+1}^{\ell-1}\eta_j\right)^{-1} \ge 2\left(\frac{\xi}{2\sigma}\right)^{\ell-k}.$$

Hence, the K-cycle is, in particular, allowed at every level if $\sigma \le \frac{\xi}{2} = \frac{3}{10}$. Then, the relation (3.3) shows that the cost of each main iteration is at most $\frac{3}{2}C\,nnz(A)$.

More generally, the above rule allows us to keep the cost of the application of the preconditioner bounded in any case. We have to correct (3.2) to take into account that, at level k ($1 \le k < \ell$), the number of allowed iterations is now η_k, and the maximal number of times this level is visited is now $\prod_{j=k+1}^{\ell-1}\eta_j$:

$$\text{Cost} \le C\left(nnz(A_\ell) + \sum_{k=1}^{\ell-1}\left(\prod_{j=k}^{\ell-1}\eta_j\right)nnz(A_k)\right).$$

If $\eta_{\ell-1} = \cdots = \eta_k = 1$, then there holds

$$nnz(A_k)\prod_{j=k}^{\ell-1}\eta_j \le \sigma^{\ell-k} nnz(A_\ell). \tag{3.5}$$

ETNA
Kent State University
http://etna.math.kent.edu

Otherwise, let m_k be the smallest index such that $m_k \geq k$ and $\eta_{m_k} = 2$; thus $m_k = k$ if $\eta_k = 2$. Using the condition in (3.4) for having $\eta_{m_k} = 2$, one has

$$\begin{aligned} nnz(A_k) \prod_{j=k}^{\ell-1} \eta_j &\leq \eta_{m_k} \sigma^{m_k - k} nnz(A_{m_k}) \prod_{j=m_k+1}^{\ell-1} \eta_j \\ &\leq \frac{4}{3} \sigma^{m_k-k} \xi^{\ell-m_k} nnz(A_\ell). \end{aligned} \tag{3.6}$$

Hence, with $\zeta = \max(\sigma, \xi)$, combining (3.5) and (3.6), one has

$$\text{Cost} \leq C\, nnz(A_\ell) \left(1 + \frac{4}{3} \sum_{k=1}^{\ell-1} \zeta^{\ell-k} \right) \leq C\, nnz(A_\ell) \left(1 + \frac{4\zeta}{3(1-\zeta)} \right).$$

In practice, we expect $\sigma \leq \xi = \frac{3}{5}$; $\sigma = \frac{3}{5}$ would already indicate a very slow coarsening. Then, the above relation means that the cost of each top level iteration is bounded by $3C\, nnz(A_\ell)$.

3.1. Parallel implementation. We use the same algorithms in parallel, with the coarsening adapted as indicated in Section 2. We still use symmetric Gauss-Seidel smoothing, but locally, that is, the smoother is given by (3.1), but ignoring offdiagonal entries connecting nodes assigned to different tasks, so that M_k is block diagonal with respect to the partitioning of the unknowns. On the other hand, the coarse grid matrix is factorized exactly using a parallel sparse direct solver (namely MUMPS [25]) as soon as the global number of unknowns is below $400N_p$, where N_p is the number of concurrent tasks. This makes it possible to avoid working with excessively small grids, for which the communication/computation ratio is large. As discussed later, although this is not essential, we also use a slightly different value for the threshold t in Algorithm 3.2, namely $t = 0.35$ instead of $t = 0.25$.

4. Numerical results. We consider the following test problems. In all cases we use a uniform mesh with constant mesh size h in all directions.

Problem MODEL2D: linear system resulting from the five-point finite difference approximation of $-\Delta u = 1$ in $\Omega = (0,1) \times (0,1)$, with boundary conditions $u = 0$ everywhere on $\partial\Omega$.

Problem ANI2D: linear system resulting from the five-point finite difference approximation of $-\frac{\partial^2 u}{\partial x^2} - b\frac{\partial^2 u}{\partial y^2} = 1$ in $\Omega = (0,1) \times (0,1)$, with constant coefficient b and boundary conditions

$$\begin{cases} u = 0, & \text{for } x = 1,\ 0 \leq y \leq 1, \\ \frac{\partial u}{\partial n} = 0, & \text{elsewhere on } \partial\Omega. \end{cases}$$

Problem ANIBFE: linear system resulting from the *bilinear finite element* approximation of $-\frac{\partial^2 u}{\partial x^2} - b\frac{\partial^2 u}{\partial y^2} = 1$ in $\Omega = (0,1) \times (0,1)$, with constant coefficient b and boundary conditions $u = 0$ everywhere on $\partial\Omega$.

Problem JUMP2D: linear system resulting from the five-point finite difference approximation of $-\frac{\partial}{\partial x}\left(a\frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(b\frac{\partial u}{\partial y}\right) = f$ in $\Omega = (0,1) \times (0,1)$, with boundary conditions

$$\begin{cases} u = 0, & \text{for } y = 1,\ 0 \leq x \leq 1 \\ \frac{\partial u}{\partial n} = 0, & \text{elsewhere on } \partial\Omega, \end{cases}$$

ETNA
Kent State University
http://etna.math.kent.edu

and coefficients given by

$$\begin{cases} a=1, & b=100, & f=0, & \text{in } (0.65,0.95)\times(0.05,0.65), \\ a=100, & b=1, & f=0, & \text{in } (0.25,0.45)\times(0.25,0.45), \\ a=100, & b=100, & f=1, & \text{in } (0.05,0.25)\times(0.65,0.95), \\ a=1, & b=1, & f=0, & \text{elsewhere.} \end{cases}$$

Problem CD1 [12]: linear system resulting from the five-point finite difference approximation (upwind scheme) of $-\nu\Delta u + \overline{v}\overline{\nabla} u = 0$ in $\Omega = (0,1)\times(0,1)$, with boundary conditions

$$\begin{cases} u=1, & \text{for } y=1,\ 0\le x\le 1, \\ u=0, & \text{elsewhere on } \partial\Omega, \end{cases}$$

and convective flow given by

$$\overline{v}(x,y) = \begin{bmatrix} x(1-x)(2y-1) \\ -(2x-1)y(1-y) \end{bmatrix}.$$

Problem CD2 [33]: linear system resulting from the five-point finite difference approximation (upwind scheme) of $-\nu\Delta u + \overline{v}\overline{\nabla} u = 0$ in $\Omega = (0,1)\times(0,1)$, with boundary conditions

$$\begin{cases} u=1, & \text{for } y=1,\ 0\le x\le 1, \\ u=0, & \text{elsewhere on } \partial\Omega, \end{cases}$$

and convective flow given by

$$\overline{v}(x,y) = \begin{bmatrix} \cos\left(\pi\left(x-\frac{1}{3}\right)\right)\sin\left(\pi\left(y-\frac{1}{3}\right)\right) \\ -\cos\left(\pi\left(y-\frac{1}{3}\right)\right)\sin\left(\pi\left(x-\frac{1}{3}\right)\right) \end{bmatrix}$$

inside the circle of center $(\frac{1}{3},\frac{1}{3})$ and radius $\frac{1}{4}$, and $\overline{v}(x,y)=0$ outside.

Problem MODEL3D: linear system resulting from the seven-point finite difference approximation of $-\Delta u = 1$ in $\Omega = (0,1)\times(0,1)\times(0,1)$, with boundary conditions $u=0$ everywhere on $\partial\Omega$.

Problem ANI3D: linear system resulting from the seven-point finite difference approximation of $-\frac{\partial^2 u}{\partial x^2} - b\frac{\partial^2 u}{\partial y^2} - c\frac{\partial^2 u}{\partial z^2} = 1$ in $\Omega = (0,1)\times(0,1)\times(0,1)$, with constant coefficients b, c, and boundary conditions

$$\begin{cases} u=0, & \text{for } x=1,\ 0\le y,z\le 1, \\ \frac{\partial u}{\partial n}=0, & \text{elsewhere on } \partial\Omega. \end{cases}$$

Problem JUMP3D: linear system resulting from the seven-point finite difference approximation of $-\frac{\partial}{\partial x}\left(a\frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(b\frac{\partial u}{\partial y}\right) - \frac{\partial}{\partial z}\left(c\frac{\partial u}{\partial z}\right) = f$ in $\Omega = (0,1)\times(0,1)\times(0,1)$, with boundary conditions

$$\begin{cases} u=0, & \text{for } z=1,\ 0\le x,y\le 1, \\ \frac{\partial u}{\partial n}=0, & \text{elsewhere on } \partial\Omega, \end{cases}$$

and coefficients given by

$$\begin{cases} a=b=c=d, & f=1, & \text{in } \left(\frac{1}{4},\frac{3}{4}\right)\times\left(\frac{1}{4},\frac{3}{4}\right)\times\left(\frac{1}{4},\frac{3}{4}\right), \\ a=b=c=1, & f=0, & \text{elsewhere,} \end{cases}$$

ETNA
Kent State University
http://etna.math.kent.edu

where d is a parameter.

Problem CD3D: linear system resulting from the seven-point finite difference approximation (upwind scheme) of $-\nu\Delta u+\overline{v}\overline{\nabla}u=0$ in $\Omega=(0,1)\times(0,1)\times(0,1)$, with boundary conditions

$$\begin{cases} u=1, & \text{for } z=1,\ 0\le x,y\le 1, \\ u=0, & \text{elsewhere on } \partial\Omega, \end{cases}$$

and convective flow given by

$$\overline{v}(x,y,z)=\begin{bmatrix} 2x(1-x)(2y-1)z \\ -(2x-1)y(1-y) \\ -(2x-1)(2y-1)z(1-z) \end{bmatrix}.$$

Besides these problems, we also consider 3D unstructured problems arising in the simulation of complex electrochemical processes [27]. For these problems, statistics on matrix data are given in Table 4.1.

TABLE 4.1
Matrix statistics for electrochemical problems; "%PosOf" is the percentage of positive offdiagonal entries, and "%RwNeg" is the percentage of rows with negative row-sum; more precisely, with computed row-sum below $(nnz(A)/n)\varepsilon_{\mathrm{mach}}$.

Problem	n	$nnz(A)/n$	%PosOf	%RwNeg
P1	826719	14.2	23	36
P2	2190	12.6	21	50
P3	59771	14.1	23	40
P4	826408	13.8	23	34

In all experiments, the systems where solved using the zero vector as initial approximation, and tolerance $\varepsilon=10^{-6}$ on the relative residual norm. All times reported are elapsed (wall clock) time. Those reported in Figures 4.1 and 4.2 were obtained on an Intel XEON 32bit processor at 3.05GHz with 2GB of RAM memory. Times reported in Tables 4.2, 4.5 and 4.6 were obtained on a multi-processor cluster with two Intel XEON L5420 processors at 2.50GHz and 16Gb RAM memory per computing node, with Infiniband (half bandwidth) interconnect; note that the Intel XEON at 3.05GHz is from an older generation and about three times slower than the Intel XEON L5420.

Before presenting the results obtained with default settings, we first show the importance of using K-cycle multigrid. In Table 4.2, we consider two of the test problems, using both sequential and parallel computation, and compare V- and W-cycle with K-cycle for several values of the threshold parameter t in Algorithm 3.2; note that $t=0.00$ means that two inner iterations are enforced at each level. The lack of scalability of V- and W-cycles appears clearly. The K-cycle works fine with $t=0.00$, but increasing t to 0.25 does not modify the number of (outer) iterations, while the cost of the multigrid preconditioner slightly decreases. Sometimes, increasing t to 0.35 further reduces the computing time, but there is sometimes a penalty because the number of outer iterations slightly increases. These observations are supported by many other experiments. In the following, the threshold parameter t was set to 0.25 for all sequential runs and to 0.35 for the parallel ones. This difference of treatment is motivated by the fact that communications are more penalizing on small grids, hence using a larger t is slightly more beneficial in parallel, as this reduces the number of times these grids are visited.

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.2
Comparison of cycling strategies; for the parallel runs, one task per computing node was used and the problem size was scaled in such a way that the load per processor is approximately the same as that for the corresponding sequential run; times reported are total elapsed time in seconds.

	#it	Time	#it	Time
Problem JUMP2D, sequential				
	$n = 1.00e6$		$n = 9.00e6$	
V-cycle	49	17.6	111	372
W-cycle	35	14.6	44	179
K-cycle, $t = 0.00$	21	9.3	22	95
K-cycle, $t = 0.25$	21	9.3	22	95
K-cycle, $t = 0.35$	25	10.5	23	98
Problem JUMP2D, 48 processors				
	$n = 47.9e6$		$n = 432.0e6$	
V-cycle	148	61.7	382	1533
W-cycle	67	44.0	85	440
K-cycle, $t = 0.00$	25	18.0	25	137
K-cycle, $t = 0.25$	25	17.9	25	138
K-cycle, $t = 0.35$	25	17.8	25	138
Problem JUMP3D, sequential				
	$n = 1.02e6$		$n = 64.3e6$	
V-cycle	18	9.22	40	1367
W-cycle	14	8.57	20	922
K-cycle, $t = 0.00$	12	7.72	12	603
K-cycle, $t = 0.25$	12	7.44	12	601
K-cycle, $t = 0.35$	12	7.08	12	557
Problem JUMP3D, 48 processors				
	$n = 48.5e6$		$n = 3065.0e6$	
V-cycle	27	16.7	66	3182
W-cycle	17	17.0	23	1597
K-cycle, $t = 0.00$	12	12.8	16	1158
K-cycle, $t = 0.25$	12	12.8	16	1156
K-cycle, $t = 0.35$	12	12.5	16	1118

Numerical results for all test problems in sequential are reported in Tables 4.3 and 4.4; there, "#lev" is the number of levels,

$$\mathcal{C} = \frac{1}{nnz(A_\ell)} \sum_{k=1}^{\ell} nnz(A_k)$$

is the (operator) complexity, "#it" the number of iterations, and "SolC" the relative solution cost, that is, the number of floating point operations needed to solve the system (excluding setup) divided by the number of floating point operations required by an *unpreconditioned* conjugate gradient iteration, or, in other words, the number of unpreconditioned conjugate gradient iterations one could perform with the amount of work used by the multigrid method to solve the system.

One sees that the complexity is small in all cases, indicating that setup costs and memory requirements are low. The solution cost is independent of the mesh size in the best cases, and nearly independent in the most difficult ones. Note that, except for Problem P4, the

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.3
Numerical results for 2D problems.

Problem	$h^{-1}=300$ #lev	$\mathcal{C}$	#it	SolC	$h^{-1}=1200$ #lev	$\mathcal{C}$	#it	SolC
MODEL2D	6	1.33	11	56	8	1.33	11	57
ANI2D ($b=100$)	6	1.33	15	75	8	1.33	20	103
ANI2D ($b=10^4$)	6	1.33	16	81	8	1.33	17	86
JUMP2D	6	1.35	18	93	8	1.38	22	120
ANIBFE ($b=1$)	6	1.26	10	53	8	1.26	11	59
ANIBFE ($b=10$)	6	1.33	19	106	8	1.33	21	123
ANIBFE ($b=100$)	6	1.33	20	109	8	1.33	23	131
ANIBFE ($b=10^3$)	6	1.33	20	110	8	1.33	23	131
CD1 ($\nu=1$)	6	1.37	9	50	8	1.41	10	60
CD1 ($\nu=10^{-2}$)	6	1.42	15	89	8	1.40	12	70
CD1 ($\nu=10^{-4}$)	7	1.45	17	108	8	1.40	23	137
CD1 ($\nu=10^{-6}$)	6	1.41	13	81	8	1.39	16	97
CD2 ($\nu=1$)	6	1.35	9	47	8	1.35	10	54
CD2 ($\nu=10^{-2}$)	6	1.35	13	69	8	1.35	14	75
CD2 ($\nu=10^{-4}$)	6	1.39	14	81	9	1.41	14	85
CD2 ($\nu=10^{-6}$)	6	1.39	20	126	9	1.40	23	144

TABLE 4.4
Numerical results for 3D problems.

Problem	$h^{-1}=60$ #lev	$\mathcal{C}$	#it	SolC	$h^{-1}=120$ #lev	$\mathcal{C}$	#it	SolC
MODEL3D	7	1.36	9	46	8	1.34	10	54
ANI3D ($b=1, c=100$)	7	1.34	13	62	8	1.34	15	75
ANI3D ($b=10, c=100$)	7	1.34	15	70	8	1.34	13	73
ANI3D ($b=100, c=100$)	7	1.34	9	47	8	1.34	10	52
ANI3D ($b=100, c=10^4$)	7	1.34	15	70	8	1.34	15	74
JUMP3D ($d=100$)	7	1.40	11	64	8	1.39	11	69
JUMP3D ($d=10^4$)	7	1.40	11	63	8	1.39	11	69
JUMP3D ($d=10^6$)	7	1.40	11	65	8	1.39	11	68
CD3D ($\nu=1$)	7	1.59	12	59	8	1.58	11	68
CD3D ($\nu=10^{-2}$)	7	1.58	12	72	8	1.56	13	80
CD3D ($\nu=10^{-4}$)	7	1.58	12	74	9	1.59	16	104
CD3D ($\nu=10^{-6}$)	7	1.57	12	74	8	1.55	16	101
P1	8	1.35	9	52				
P2	3	1.31	5	28				
P3	6	1.35	9	52				
P4	8	1.34	37	203				

solution cost, as measured by "SolC", never exceeds three times the solution cost for the model problem.

Timing results are shown in Figures 4.1 and 4.2, where a comparison is made with the old but classical code AMG1R5 by K. Stüben, based on the "standard" AMG method in [34, 36]. This code was used with the same initial approximation and the same stopping criterion as our method. Other parameters were set to default, except that we tested the program both

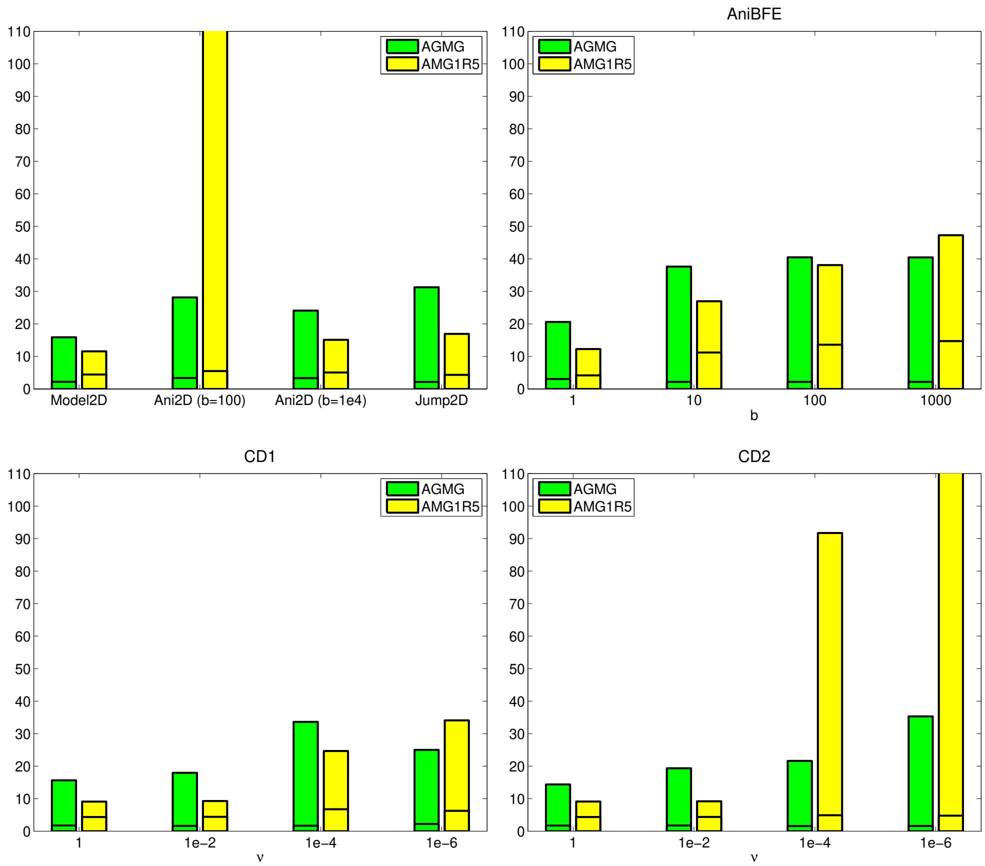


FIGURE 4.1. *Timing results* ($\frac{\textit{Elapsed Time}}{n \cdot 10^{-6}}$) *for 2D problems with* $h^{-1} = 120$*; AMG1R5 did not converge in the allowed 100 iterations for Problem* ANI2D *with* $b = 100$ *and Problem* CD2 *with* $\nu = 10^{-6}$.

with and without conjugate gradient acceleration, recording, for each problem, only the best of the two timings; by default, conjugate gradient is not used in AMG1R5, but the comparison could then be seen as unfair since our method always uses Krylov subspace acceleration. In the figures, “AGMG” refers to the method described in this paper. The bottom part of the bars represents the setup time, and the upper part the solution time (the total height gives thus the total time). Reported times are elapsed times in seconds per million of unknowns.

With the exception of problem P4, our method solves all test cases in a time between 15 and 50 seconds per million of unknowns. In addition, setup times are fairly small. AGMG is not always the winner, especially in 2D cases, but appears more robust. When AMG1R5 is faster, gains are relatively marginal, whereas losses may be much more significant, with failure in some cases. Note that AGMG has *always* lower setup time.

Results for parallel runs are reported in Tables 4.5 and 4.6. In all cases, the partitioning was imposed at the discretization level, based on the partitioning of the domain in rectangular or parallelepipedal subdomains. We give both the global complexity $\mathcal{C}_{\text{glob}}$, and the maximum of the local ones $\mathcal{C}_{\max}$; their comparison gives an idea on how the initial load balancing is preserved throughout all levels. The solution cost “SolC” reported is also the maximum of the local ones, that is, each task counts the numbers of floating point operations it performed to solve the system, and divides it by the numbers of floating point operations executed on that

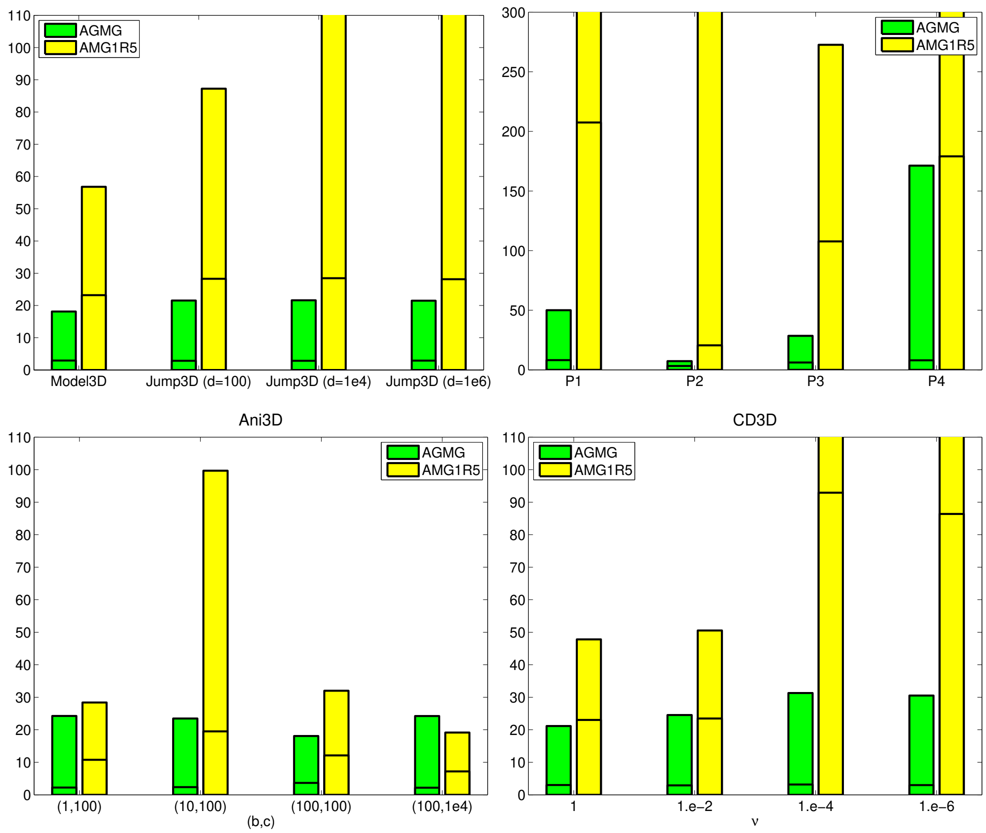


FIGURE 4.2. *Timing results* ($\frac{\textit{Elapsed time}}{n \cdot 10^{-6}}$) *for 3D problems with* $h^{-1} = 120$ *(in* MODEL3D, ANI3D, JUMP3D, CD3D*); AMG1R5 did not converge in the allowed 100 iterations for Problem* JUMP3D *with* $d = 10^6$ *and for Problem P4, whereas it broke down for Problem P2; for Problem* JUMP3D *with* $d = 10^4$*, Problem P1 and Problem* CD3D *with* $\nu = 10^{-4}$ *or* $\nu = 10^{-6}$*, the total time for AMG1R5 just goes over the upper limit needed to properly display results with AGMG: these times are, respectively, 171, 522, 582 and 433 seconds per million of unknowns.*

task when performing one unpreconditioned conjugate gradient iteration; what is reported in Tables 4.5 and 4.6 is then the maximum over all tasks of this quantity.

Two sequences of runs were performed: one with one task per computing node, and a second one with two tasks per computing node. These latter runs allow us to benefit from the fact that there are two processors per computing node in the cluster. This level of parallelism is, however, not that efficient because the two processors share the same memory. Hence, firstly, one cannot in this way really increase the size of the problems that can be solved. This is why, in the tables, we report results for problem sizes scaled in such a way that the number of unknown per *computing node* is approximately constant. Secondly, as for any computation with large sparse matrices, a significant amount of time is spent fetching and loading data from and to main memory, implying that the two processes running on the same node slow down each other.

Now, the results indicate that the solution cost, as reported in “SolC”, is nearly independent of both the problem size and the number of processors or tasks. On the other hand, timing results show that, with 1 task per computing node, using 48 nodes allows us to solve a problem 48 times larger than sequentially in twice the time or less. Using 2 tasks per computing node has mitigated effects on smallest problems with many processors, but, for largest

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.5

Results in parallel for Problem JUMP2D*; #p is the number of concurrent tasks and #cn the number of computing nodes effectively used.*

n	#p	#cn	n/#cn	$\mathcal{C}_{\text{glob}}$	$\mathcal{C}_{\text{max}}$	#it	SolC	Time
$\approx 10^6$ **unknowns per computing node**								
1 task per computing node								
1.00e6	1	1	1.00e6	1.36	1.36	25	127	10.81
7.96e6	8	8	0.99e6	1.34	1.35	23	120	10.98
24.01e6	24	24	1.00e6	1.34	1.41	24	135	15.00
47.89e6	48	48	1.00e6	1.34	1.41	25	140	17.96
2 tasks per computing node								
1.00e6	2	1	1.00e6	1.35	1.37	23	119	5.86
7.96e6	16	8	0.99e6	1.34	1.41	23	128	8.79
24.01e6	48	24	1.00e6	1.34	1.35	25	135	14.36
47.89e6	96	48	1.00e6	1.34	1.41	26	146	20.29
$\approx 9 \cdot 10^6$ **unknowns per computing node**								
1 task per computing node								
9.0e6	1	1	9.0e6	1.35	1.35	23	122	99
71.9e6	8	8	9.0e6	1.35	1.39	24	132	112
216.1e6	24	24	9.0e6	1.34	1.37	30	162	153
431.8e6	48	48	9.0e6	1.34	1.41	25	142	137
2 tasks per computing node								
9.0e6	2	1	9.0e6	1.35	1.36	24	126	64
71.9e6	16	8	9.0e6	1.34	1.38	23	127	87
216.1e6	48	24	9.0e6	1.34	1.39	25	138	103
431.8e6	96	48	9.0e6	1.34	1.41	25	141	115

problems, this allows, using 48 nodes, to solve a problem 48 times larger than sequentially in a time that is only 15–30% larger than the (purely) sequential time.

5. Conclusions. We have presented a multigrid method based on the aggregation of the unknowns. The procedure is fully algebraic, that is, it works with the information present in the system matrix only. Numerical experiments have been performed on a wide set of discrete second order scalar elliptic PDEs, including two- and three-dimensional problems with jumps and/or anisotropy in the PDE coefficients, convection-diffusion problems with high Reynolds number and circulating convective flow, as well as some problems from industrial chemistry with many positive offdiagonal entries. The results are promising and indicate that the approach is robust as a black box solver. On the average, the method is also faster than the classical AMG method as implemented in the AMG1R5 code when using the latter with default settings, that is, also as a black box solver. Of course, classical AMG performances are significantly improved in some cases by tuning parameters or implementing variants as developed in, e.g., [8–10]. In fact, AMG1R5 is "just the first realization of an AMG method, and there are many improvements introduced during the last years" [38]. The code developed to produce the numerical results with our method is, however, also the first realization of a multigrid method based on the proposed double pairwise aggregation algorithm and the use of K-cycle multigrid to enhance scalability. Moreover, it seems that, so far, no AMG variant can be used as widely as black box solver. Note that the code implementing AGMG is available for download, both as a FORTRAN program and a MATLAB function [28], the latter

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.6
Results in parallel for Problem JUMP3D *with* $d = 10^6$*; #p is the number of concurrent tasks and #cn the number of computing nodes effectively used.*

n	#p	#cn	n/#cn	$\mathcal{C}_{\text{glob}}$	$\mathcal{C}_{\text{max}}$	#it	SolC	Time
$\approx 10^6$ **unknowns per computing node**								
1 task per computing node								
1.02e6	1	1	1.02e6	1.38	1.38	12	65	7.90
8.08e6	8	8	1.01e6	1.37	1.40	11	67	8.10
24.05e6	24	24	1.00e6	1.37	1.40	13	81	10.65
48.49e6	48	48	1.01e6	1.37	1.40	12	78	12.49
2 tasks per computing node								
1.02e6	2	1	1.02e6	1.44	1.46	12	72	4.90
8.08e6	16	8	1.01e6	1.37	1.39	11	66	5.98
24.05e6	48	24	1.00e6	1.36	1.43	12	77	7.58
48.49e6	96	48	1.01e6	1.37	1.40	12	75	11.03
$\approx 64 \cdot 10^6$ **unknowns per computing node**								
1 task per computing node								
64.0e6	1	1	64.0e6	1.38	1.38	12	70	575
513.0e6	8	8	64.0e6	1.36	1.38	13	80	826
1531.0e6	24	24	64.0e6	1.35	1.40	13	86	906
3065.0e6	48	48	64.0e6	1.36	1.45	16	110	1112
2 tasks per computing node								
64.0e6	2	1	64.0e6	1.37	1.38	12	67	444
513.0e6	16	8	64.0e6	1.36	1.40	13	84	557
1531.0e6	48	24	64.0e6	1.35	1.38	13	84	634
3065.0e6	96	48	64.0e6	1.35	1.40	15	97	728

illustrating well the black box capabilities of the method.

The parallelization has also been addressed. A strategy has been proposed for which the total work is nearly independent of the number of processors, that is, the parallel implementation incurs almost no penalty from the algorithmic point of view. Timing results are also promising, and satisfactory speedups have been obtained on a 48-node processors cluster with Infiniband interconnect, which may be seen as representative of today market of medium size multi-processor computers.

Finally, note that all results were obtained for matrices arising from scalar elliptic PDEs, that is, with only one vector in the near-kernel that is well approximated by the constant vector. One subject of future research is the extension of the method to more general problems with different type of near-kernel, such as those arising from systems of PDEs.

REFERENCES

[1] E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. D. CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK Users' Guide, 3rd ed.*, SIAM, 1999.

[2] D. BRAESS, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55 (1995), pp. 379–393.

[3] A. BRANDT, S. F. MCCORMICK, AND J. W. RUGE, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and its Application, D. J. Evans, ed., Cambridge University Press, Cambridge, 1984, pp. 257–284.

[4] M. BREZINA, A. J. CLEARY, R. D. FALGOUT, V. E. HENSON, J. E. JONES, T. A. MANTEUFFEL, S. F. MCCORMICK, AND J. W. RUGE, *Algebraic multigrid based on element interpolation (AMGe)*, SIAM J. Sci. Comput., 22 (2000), pp. 1570–1592.

[5] M. BREZINA, R. FALGOUT, S. MACLACHLAN, T. MANTEUFFEL, S. MCCORMICK, AND J. RUGE, *Adaptive smoothed aggregation (αSA)*, SIAM Rev., 47 (2005), pp. 317–346.

[6] V. E. BULGAKOV, *Multi-level iterative technique and aggregation concept with semi-analytical preconditioning for solving boundary-value problems*, Comm. Numer. Methods Engrng., 9 (1993), pp. 649–657.

[7] T. CHARTIER, R. D. FALGOUT, V. E. HENSON, J. JONES, T. MANTEUFFEL, S. MCCORMICK, J. RUGE, AND P. S. VASSILEVSKI, *Spectral AMGe (ρAMGe)*, SIAM J. Sci. Comput., 25 (2004), pp. 1–26.

[8] A. J. CLEARY, R. D. FALGOUT, V. E. HENSON, J. E. JONES, T. A. MANTEUFFEL, S. F. MCCORMICK, G. N. MIRANDA, AND J. W. RUGE, *Robustness and scalability of algebraic multigrid*, SIAM J. Sci. Comput., 21 (2000), pp. 1886–1908.

[9] H. DE STERCK, R. D. FALGOUT, J. NOLTING, AND U. M. YANG, *Distance two interpolation for parallel algebraic multigrid*, Numer. Linear Algebra Appl., 15 (2008), pp. 115–139.

[10] H. DE STERCK, U. M. YANG, AND J. J. HEYS, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1019–1039.

[11] S. C. EISENSTAT, H. C. ELMAN, AND M. H. SCHULTZ, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[12] H. ELMAN AND D. SILVESTER, *Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations*, SIAM J. Sci. Comput., 17 (1996), pp. 33–46.

[13] G. H. GOLUB AND Q. YE, *Inexact preconditioned conjugate gradient method with inner-outer iterations*, SIAM J. Sci. Comput., 21 (1999), pp. 1305–1320.

[14] W. HACKBUSCH, *Multi-grid Methods and Applications*, Springer, Berlin, 1985.

[15] V. E. HENSON AND P. S. VASSILEVSKI, *Element-free AMGe: General algorithms for computing interpolation weights in AMG*, SIAM J. Sci. Comput., 23 (2002), pp. 629–650.

[16] V. E. HENSON AND U. M. YANG, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177.

[17] P. JIRÁNEK, M. ROZLOZNÍK, AND M. H. GUTKNECHT, *How to make Simpler GMRES and GCR more stable*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1483–1499.

[18] J. E. JONES AND P. S. VASSILEVSKI, AMGE *based on element agglomeration*, SIAM J. Sci. Comput., 23 (2001), pp. 109–133.

[19] G. KARYPIS, *METIS software and documentation*. Available at http://glaros.dtc.umn.edu/gkhome/views/metis.

[20] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[21] H. KIM, J. XU, AND L. ZIKATANOV, *A multigrid method based on graph matching for convection-diffusion equations*, Numer. Linear Algebra Appl., 10 (2003), pp. 181–195.

[22] A. KRECHEL AND K. STÜBEN, *Parallel algebraic multigrid based on subdomain blocking*, Parallel Comput., 27 (2001), pp. 1009–1031.

[23] J. LANGOU, *GCR vs Flexible GMRES*. Talk presented at "Conference on Applied Linear Algebra", Düsseldorf, 2006.

[24] W. LEONTIEF, *The Structure of the American Economy 1919-1939*, Oxford University Press, New York, 1951.

[25] *MUMPS software and documentation*. Available at http://graal.ens-lyon.fr/MUMPS/.

[26] A. C. MURESAN AND Y. NOTAY, *Analysis of aggregation-based multigrid*, SIAM J. Sci. Comput., 30 (2008), pp. 1082–1103.

[27] G. NELISSEN AND P. F. VANKEIRSBILCK, *Electrochemical modelling and software genericity*, in Modern Software Tools for Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Boston, Cambridge, 1997, pp. 81–104.

[28] Y. NOTAY, *AGMG software and documentation*. Available at http://homepages.ulb.ac.be/~ynotay/AGMG.

[29] ———, *Flexible conjugate gradients*, SIAM J. Sci. Comput., 22 (2000), pp. 1444–1460.

[30] ———, *Aggregation-based algebraic multilevel preconditioning*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 998–1018.

[31] Y. NOTAY AND P. S. VASSILEVSKI, *Recursive Krylov-based multigrid cycles*, Numer. Linear Algebra Appl., 15 (2008), pp. 473–487.

[32] C. W. OOSTERLEE AND T. WASHIO, *Krylov subspace acceleration of nonlinear multigrid with application to recirculating flows*, SIAM J. Sci. Comput., 21 (2000), pp. 1670–1690.

[33] A. REUSKEN, *A multigrid method based on incomplete Gaussian elimination*, Numer. Linear Algebra Appl., 3 (1996), pp. 369–390.

[34] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, 1987, pp. 73–130.

[35] Y. SAAD, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.

[36] K. STÜBEN, *Algebraic multigrid (AMG): experiences and comparisons*, Appl. Math. Comput., 13 (1983), pp. 419–452.

[37] ———, *An introduction to algebraic multigrid*, in Trottenberg et al. [39], 2001, pp. 413–532. Appendix A.

[38] ———, Private communication granting the right to publish results obtained with AMG1R5, 2007.

[39] U. TROTTENBERG, C. W. OOSTERLEE, AND A. SCHÜLLER, EDS., *Multigrid*, Academic Press, San Diego, 2001.

[40] H. A. VAN DER VORST AND C. VUIK, *GMRESR: a family of nested GMRES methods*, Numer. Linear Algebra Appl., 1 (1994), pp. 369–386.

[41] P. VANĚK, M. BREZINA, AND R. TEZAUR, *Two-grid method for linear elasticity on unstructured meshes*, SIAM J. Sci. Comput., 21 (1999), pp. 900–923.

[42] P. VANĚK, J. MANDEL, AND M. BREZINA, *Algebraic multigrid based on smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[43] R. WIENANDS AND C. W. OOSTERLEE, *On three-grid Fourier analysis for multigrid*, SIAM J. Sci. Comput., 23 (2001), pp. 651–671.