Coarse Grid Classification: A Parallel Coarsening Scheme For Algebraic Multigrid Methods

Michael Griebel[1], Bram Metsch[1], Daniel Oeltz[1], and Marc Alexander Schweitzer[1]

[1] *Institut für Numerische Simulation, Universität Bonn, Wegelerstraße 6, D-53115 Bonn*

SUMMARY

In this paper we present a new approach to the parallelization of algebraic multigrid (AMG), i.e., to the parallel coarse grid selection in AMG. Our approach does not involve any special treatment of processor subdomain boundaries and hence avoids a number of drawbacks of other AMG parallelization techniques. The key idea is to select an appropriate (local) coarse grid on each processor from *all* admissible grids such that the composed coarse grid forms a suitable coarse grid for the whole domain, i.e. there is no need for any boundary treatment. To this end, we first construct multiple equivalent coarse grids on each processor subdomain. In a second step we then select exactly one grid per processor by a graph clustering technique. The results of our numerical experiments clearly indicate that this approach results in coarse grids of high quality which are very close to those obtained with sequential AMG. Furthermore, the operator and grid complexities of our parallel AMG are mostly smaller than those obtained by other parallel AMG methods, whereas the scale-up behavior of the proposed algorithm is similar to that of other parallel AMG techniques. However a significant improvement with respect to the speed-up performance is achieved.

KEY WORDS: algebraic multigrid; parallel computing

AMS SUBJECT CLASSIFICATION: 65N55; 65Y05; 65F10

1. Introduction

The solution of large sparse linear systems $Au = f$ is an essential ingredient in most scientific computations. This solution step must involve only a similar amount of operations and computational storage as the discretization process itself to allow for an efficient and scalable numerical simulation. Such a sparse linear solver, which requires only $O(N)$ operations and memory for N degrees of freedom, is called optimal. However, there exists no general (direct) algebraic solution technique yielding this complexity.

For linear systems arising from a finite difference or finite element discretization of an elliptic partial differential equation (PDE) on a structured grid, geometric multigrid methods [4, 11]

Contract/grant sponsor: Sonderforschungsbereich 611, *Singuläre Phänomene und Skalierung in mathematischen Modellen*, sponsored by the *Deutsche Forschungsgemeinschaft*.

give such an optimal (iterative) solver. But for many applications it is difficult to construct a sequence of (nested) discretizations or meshes needed for geometric multigrid. Furthermore, geometric multigrid methods are in general not robust with respect to a deterioration or a singular perturbation of the coefficients of the operator. In the 1980s algebraic multigrid (AMG) methods were developed [5, 6, 7, 15] to cope with these problems by extending the main ideas of geometric multigrid methods to a purely algebraic setting. This approach is based only on information available from the linear system to be solved. However, this flexibility has a price: A *setup phase*, in which the multigrid hierarchy, i.e., the sequence of grids, transfer operators and coarse grid operators, is constructed, has to be carried out before the well-known multigrid cycle (the *solution phase*) can be started. While the parallelization of the solution phase is straightforward, see e.g. [16], the techniques for constructing the coarse grids in AMG are inherently sequential which makes the parallelization of the setup phase a challenging task.

Many different approaches to the parallelization of this step have been proposed over the years, e.g. [12, 14]. Most of them follow a data decomposition approach and need to employ a special treatment of the processor subdomain boundaries to cope with non-matching coarse grids. Hence, the interior of each processor subdomain is still coarsened in the classical way, but the coarse grid structure on the subdomain boundaries does not respect the classical coarsening principles. Since the distribution of a fine grid discretization of fixed size among an increasing number of processors leads to smaller subdomain interiors, increasingly smaller parts are actually coarsened by classical AMG. This leads to a significant deterioration of the quality of the coarse grid and the speed-up of these parallel AMG strategies.

In this paper we present a method for the construction of coarse grids of high quality in parallel without an explicit boundary treatment. However, we still use the classical coarsening scheme [15]. We first construct multiple coarse grids on each processor domain individually. From these coarse grids, we then choose exactly one per processor domain by graph clustering techniques such that the composed coarse grid shows no inconsistencies near the processor subdomain boundaries. Hence, no explicit boundary treatment is required such that we obtain operator and grid complexities similar to those of sequential AMG. Moreover, as no additional coarse grid points are added on or near the processor subdomain boundary, a growing number of processors —and thus a growing number of subdomain boundary points— has only little impact on the coarse grid. Hence, the speed-up of the proposed method is much better than the speed-up of the algorithms employing an explicit boundary treatment and the scale-up of our algorithm is optimal.

The remainder of this paper is organized as follows: In Section 2 we give a short review of the (sequential) AMG algorithm and the respective heuristics, which led to its development. Then, we present the main challenges which classical AMG can encounter in the parallel case. We then summarize briefly, in Section 3, the previous parallel coarsening methods and their shortcomings. In Section 4 we present our new coarse grid classification (CGC) algorithm. We discuss the results of our numerical experiments in Section 5 where we compare the quality of the CGC algorithm with that of the other methods. To this end we consider discretizations of PDE's in two and three dimensions with several millions of unknowns on a massively parallel computer with up to 256 processors. The results of these experiments clearly show the smaller memory and compute time requirements of our algorithm with a multigrid convergence rate similar to that of sequential AMG. As expected, we achieve a significantly better speed-up and an optimal scale-up for our CGC algorithm compared with other parallelization techniques. Finally, we conclude with some remarks in Section 6.

2. Algebraic multigrid

In this section we give a short review of the algebraic multigrid (AMG) method. We consider a linear system $Au = f$, which comes from a discretization of a PDE on a grid $\Omega = \{1, \dots, N\}^*$, with $A = (a_{ij})_{i,j=1}^N$ being a large sparse real matrix, $u = (u_i)_{i=1}^N$ and $f = (f_i)_{i=1}^N$ vectors of length N. We assume that A is a symmetric, positive definite M-matrix or an essentially positive matrix. To be able to solve this equation using the multigrid scheme (Program 1†), we

Program 1 multigrid cycle (solution phase of AMG) $MG(A^l, f^l, u^l)$

```
begin
  for ν ← 1 to ν1 do u^l ← S^l u^l; od;                    pre-smoothing
  r^l ← f^l − A^l u^l;                                     residual
  f^{l+1} ← R^l r^l;                                       restriction
                                                           coarse grid correction
  if l + 1 = L
    then
         u^{l+1} ← (A^{l+1})^{-1} f^{l+1};                 apply directly
     else
         for μ ← 1 to μ do
             MG(A^{l+1}, f^{l+1}, u^{l+1}));               solve recursively
         od;
  fi;
  u^l ← u^l + P^l u^{l+1};                                 update solution
  for ν ← 1 to ν2 do u^l ← S^l u^l;                        post-smoothing
  end
```

need to specify the sequence of coarse grid operators A^l, transfer operators P^l and $R^l = (P^l)^T$, the smoothing operators S^l and the sequence of coarse grids Ω^l. In geometric multigrid, the grids Ω^l are constructed by coarsening the original grid Ω^1, for example by doubling the mesh size $h \to 2h$ in all or a subset of the spatial dimensions. The operator A^l is constructed by discretizing the considered PDE on the grid Ω^l and the interpolation operator P^l is defined as the natural injection between the associated spaces $V^{l+1} \to V^l$ defined on Ω^{l+1} and Ω^l respectively. With these components all fixed, the only way to improve the speed of convergence is to change the smoothing process S^l, e.g. by using incomplete LU-factorization or alternate line projection instead of the usually employed Gauss–Seidel or Jacobi relaxation methods. However, in three spatial dimensions and for complex geometries these smoothers become difficult —if not impossible or ineffective— to carry out.

Algebraic multigrid methods were first introduced by Brandt in the early 1980s [5, 6, 7]. Instead of fixing the sequence of grids and operators, a simple smoothing scheme S^l is chosen and the grids $\{\Omega^l\}_{l=1}^L$, the transfer operators $\{P^l\}_{l=1}^{L-1}$ and the coarse grid operators $\{A^l\}_{l=1}^L$ are constructed depending on the fine grid operator $A = A^1$ automatically.

This construction is carried out during the *setup phase*, as displayed in Program 2. In this paper, we focus on the coarsening step. This step consists of splitting the grid Ω^l into two

*We denote grid points with their respective counting index in any dimension.
†In classical geometric MG, Ω^L denotes the finest grid and Ω^1 the coarsest grid. Here, as usual in AMG, the levels are numbered starting with Ω^1 as the finest grid.

Program 2 *AmgSetup*$(\Omega, A, N_{min}, L_{max}, L, \{A^l\}_{l=1}^{L}, \{P^l\}_{l=1}^{L-1}\{R^l\}_{l=1}^{L-1})$

begin
  $\Omega^1 \leftarrow \Omega$;
  $A^1 \leftarrow A$;
  for $l \leftarrow 1$ **to** $L_{max} - 1$ **do**
    partition Ω^l *into* C^l *and* F^l; coarsening step
    $\Omega^{l+1} \leftarrow C^l$;
    compute interpolation P^l; interpolation step
    $R^l \leftarrow (P^l)^T$;
    $A^{l+1} \leftarrow R^l A^l P^l$; Galerkin step
    if $|\Omega^{l+1}| \leq N_{min}$ **then** *break*; **fi**;
  od;
  $L \leftarrow l + 1$;
end.

disjoint sets C^l and F^l, i.e.,

$$\Omega^l = C^l \dot\cup F^l.$$

The set of *coarse grid points* C^l is taken as the grid on the next level,

$$\Omega^{l+1} := C^l,$$

while the remaining points $j \in F^l$ are denoted as *fine grid points*. For the sake of simplicity, we will omit the level index l in the following. We denote by u the exact solution of the equation $Au = f$ and by u^{it} the computed approximation after it steps of a relaxation method S, e.g. Jacobi-relaxation or Gauss–Seidel-relaxation. The conditions for the coarse grid selection are imposed by the requirement of a stable interpolation formula. Recall from multigrid theory [11, 15] that the low-frequency error components (also called smooth error components) have to be eliminated by the coarse grid correction step. Hence, the interpolation scheme has to approximate the smooth error components accurately. With such an interpolation P^l, we then define the restriction $R^l = (P^l)^T$ and the coarse grid operator $A^{l+1} = R^l A^l P^l$ by the Galerkin identity. The setup phase is then restarted recursively with A^{l+1} as input.

For M-matrices and essentially positive matrices $A = (a_{ij})_{i,j=1}^N$, a smooth error varies slowly in the direction of large negative couplings [15], e.g. $a_{ij} \ll 0$. This observation leads to the definition of the *strong couplings* S_i for a grid point i, which are the set of grid points

$$S_i := \{j \neq i : \ |a_{ij}| \geq \alpha \max_{k \neq i} |a_{ik}|\}, \text{ and } S_i^T := \{j \neq i : \ i \in S_j\}$$

with a typical value of $\alpha = 0.25$. A first approach to the construction of an interpolation formula for a point $i \in F$ is given by interpolating from the coarse grid points $j \in S_i \cap C$ which are strongly connected to i. This so-called *direct interpolation* [15] is however not stable since strong connections to other fine grid points $j \in S_i \cap F$ are ignored. This can be cured by the *standard interpolation* technique [15], where, for each $j \in S_i \cap F$, the strong dependency on j is replaced by a dependency on the coarse grid points which are strongly connected to j, i.e. all $k \in S_j \cap C$. This extends the set of interpolatory variables $\mathcal{P}_i$, i.e. the set of coarse grid points whose values influence the value at i. However, the value at j should also be influenced by the coarse grid points $m \in S_i \cap C$, which influence i directly, to a reasonable amount, i.e. the inequality

$$\frac{\sum_{k \in S_i \cap C} a_{jk}}{\max_k |a_{jk}|} > \beta \cdot \frac{a_{ij}}{\max_k |a_{ik}|}, \tag{1}$$

with $\beta = 0.35$ typically, should hold. The validity of this inequality ensures that for each fine grid point i, a sufficient amount of its strongly coupled neighbors S_i are coarse grid points $j \in S_i \cap C$. It furthermore guarantees that every fine grid point has at least one strong connection to a coarse grid point. Note that if a fine grid point does not have any strong connections at all, the error at this point can be reduced efficiently by smoothing only, so that such a point does not require interpolation.

We now formulate the resulting three conditions on the coarse grid.

C1 Let $i \in F$. Then each $j \in S_i$ is either in C or relation (1) holds.
C2 If $i \in C$ and $j \in S_i \cup S_i^T$, then $j \notin C$. ($C \subset \Omega$ is an *independent* or *stable* set.)
C3 C is a maximal set with these properties.

Generally, it is not possible to satisfy all these conditions simultaneously. For the stability of interpolation, condition C1 is most important and will be enforced strictly, while the second condition is intended to limit the size of the grid and thus to reduce the memory and compute time requirements. On the other hand, a large amount of coarse grid variables around a fine grid point yields a high quality of smooth error interpolation, which motivates C3.

Program 3 *AmgPhaseI*(Ω, S, S^T, C, F)

```
begin
  U ← Ω;
  C ← ∅;
  F ← ∅;
  for i ∈ Ω do λ_i ← |S_i^T|; od;
  while max_{i∈U} λ_i ≠ 0 do
        i ← arg max_{j∈U} λ_j;
        C ← C ∪ {i};
        for j ∈ S_i^T ∩ U do
            F ← F ∪ {j};
            for k ∈ S_j ∩ U do
                λ_k ← λ_k + 1;
            od;
        od;
        for j ∈ S_i ∩ U do
            λ_j ← λ_j − 1;
        od;
  od;
  F ← F ∪ U;
end
```

Program 3 shows the first phase of the classical Ruge–Stüben coarsening algorithm [15]. This procedure determines an independent set of C-variables among the grid Ω and assures that each F-variable has at least one strong connection to a C-variable. Note that there is some freedom in choosing the first point of the coarse grid, a fact we will utilize in the development of our parallel coarsening algorithm.

After applying this algorithm, the inequality (1) may not be fulfilled for all pairs of strongly connected fine grid points $i, j \in F$ (the so-called $F-F$-couplings). This is fixed by the "second pass", which checks all such pairs and inserts i or j into the coarse grid if necessary.

Note that each point i chosen for the coarse grid C results in a change of the weights λ_j of all points j within two layers around the grid point i. This states the sequential character of the coarsening algorithm, as the weight updates propagate throughout the whole domain during the iteration.

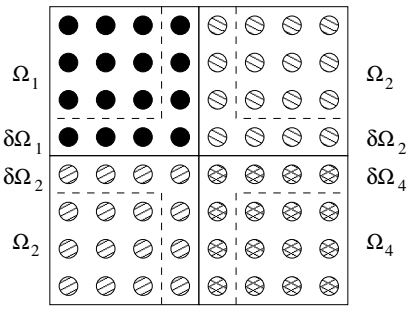


Figure 1. Disjoint partitioning of a discretized domain.

3. Parallel coarsening

In this section we shortly review some previously developed approaches to the construction of coarse grids in parallel. Most of them are based on the coarsening scheme presented in Section 2 and employ a special boundary treatment. Note that all discussed parallel AMG schemes are based on an initially given static partitioning of the data onto the processors. Such a partitioning is usually obtained from geometric information during the discretization process.

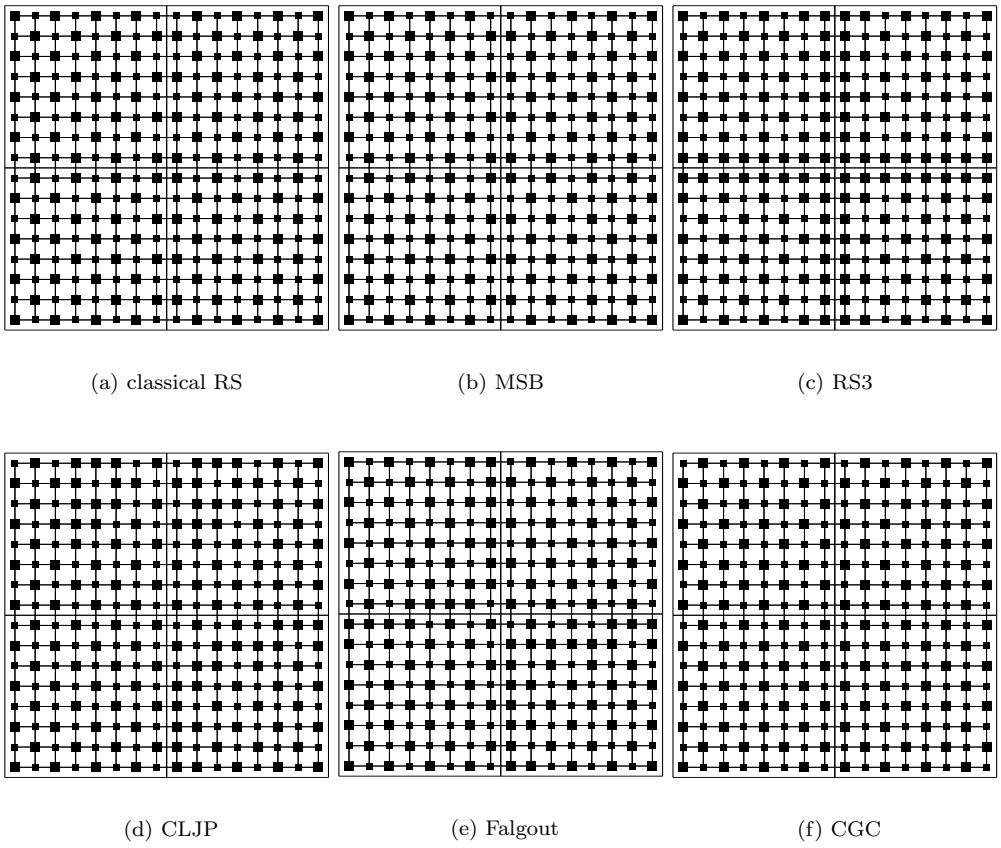
We denote by Ω_p the part of Ω that is being handled by processor $p \in \{1, \ldots, P\}$. The boundary $\partial\Omega_p$ is defined as the points $i \in \Omega_p$ which are coupled to points $j \notin \Omega_p$ on other processors, i.e.

$$\partial\Omega_p := \{i \in \Omega_p : \ a_{ij} \neq 0 \text{ for some } j \notin \Omega_p\},$$

see Figure 1.

Note that if we simply apply the sequential algorithm given in Program 3 to each processor domain separately (i.e. without regarding points on neighboring domains), inconsistencies along the processor boundaries can occur as shown in Figure 2(a) (we refer to this scheme in the following as NONE). We see that F-variables are coupled strongly across the subdomain boundaries, furthermore both involved F-points are not strongly coupled to a common C-point which is a massive violation of condition (1). In the following, we describe which techniques were proposed to overcome this problem.

Minimum Subdomain Blocking. A first approach to parallel AMG was given by Krechel and Stüben in [14]. This so-called *minimum subdomain blocking (MSB)* decouples the coarsening on the individual processor domains by first coarsening the processor boundaries $\partial\Omega_p$ by means of the classical coarsening scheme considering only couplings *inside* of $\partial\Omega_p$. This ensures that each fine grid point $i \in \partial\Omega_p$ has a strong connection to a coarse grid point $j \in \partial\Omega_p$. After the boundaries are coarsened, the interior of each subdomain is coarsened using the classical Ruge–Stüben algorithm. However, strong couplings across the processor boundaries are ignored in this approach (compare Figure 2(b)), i.e. neither relation (1) nor condition C2 is checked, and the grid layout inside the boundary may not respect anisotropies of the underlying operator.



(a) classical RS (b) MSB (c) RS3

(d) CLJP (e) Falgout (f) CGC

Figure 2. Application of the various parallelization techniques to the 5-point discretization of the Laplace operator, distributed among 4 processors. Depicted are the C/F-splittings on the finest level, where the small squares indicate the fine grid points $i \in F$ and the large squares the coarse grid points $j \in C$.

Third pass (RS3) coarsening. Another method for processor boundary treatment is the third pass coarsening method [12]. This method fixes strong $F-F$-connections across processor boundaries by applying the second pass of the classical Ruge–Stüben algorithm to all pairs (i, j), with $i \in \partial\Omega_p \cap F$, $j \in (S_i \cap F) \setminus \Omega_p$, of strongly coupled fine grid points on different processors in an additional *third pass*, after the first and second pass of the classical coarsening algorithm have been carried out on each processor subdomain individually. This third pass requires the matrix row A_j belonging to point j to be transferred from processor $q \neq p$ to processor p, and vice versa if also $i \in S_j$. Note that a processor p can select additional coarse grid points not only for its own domain Ω_p, but also for adjacent domains Ω_q. Hence a special strategy has to be applied to determine which of the involved processors enforces the final C/F-splitting. A practical approach is that every processor p keeps all the coarse grid points inside its domain it has selected and also inserts all coarse grid points that are suggested by

processors with lower rank $q < p$. This, however, can lead to a slight load imbalance. Figure 2(c) shows the coarse grids obtained by the RS3 algorithm for a discretization of the Laplacian with the 5-point stencil. In this case, every boundary point $i \in \partial\Omega_p$ needed to be added to the coarse grid for a stable interpolation.

CLJP coarsening. A completely different way of coarsening is given by the CLJP algorithm [8, 12]. It is based on parallel graph coloring methods developed by Luby, Jones and Plassman. The coarsening is done iteratively, where each iteration involves a communication phase. In each step, an independent set is chosen from the vertices of the directed graph defined by the strong couplings. The points inside this set are added to the set of coarse grid points C and all incident edges are removed from the graph. A point is inserted into F if all its incident edges have been removed and it is not already member of C. The iteration stops, if all edges of the graph have been removed. As strongly coupled neighbors $j \in S_i$ of coarse grid points i are not immediately determined to be fine grid points, this algorithm constructs a coarse grid of significantly larger magnitude than the classical coarsening scheme, compare Figure 2(d). This in turn increases the memory requirements as well as the number of floating-point operations during the solution phase. Note that the construction of the interpolation and coarse grid operators is still done in the classical way.

Falgout coarsening. The so-called Falgout coarsening [12] is a combination of the classical Ruge–Stüben coarsening and the CLJP coarsening. First, each processor domain is being coarsened using the classical algorithm. Then, the coarse grid points in the *interior* of each processor domain, $i \in C \cap (\Omega_p \setminus \partial\Omega_p)$, are taken as the first independent set for the CLJP algorithm which proceeds until all points have been assigned. This way, the interior of a domain is coarsened as in the classical method, while the boundaries are coarsened "CLJP-like", see Figure 2(e). This also means that many coarse grid points are chosen near (and not only on) the subdomain boundary, which in turn increases the operator and grid complexity.

Table I. Advantages (+) and drawbacks (–) of the coarsening algorithms.

	MSB	RS3	CLJP	Falgout
+	No communication during coarsening.	Enforces inequality (1) exactly.	Parallel coarsening independent of domain decomposition.	Acceleration of the CLJP method.
–	No treatment of strong $F-F$-couplings.	Requires communication of matrix parts. Can add many additional points.	Too many coarse grid points chosen. Multiple communication steps.	Inserts many coarse grid points along the boundaries. Requires multiple communication steps.

In Table I we summarize advantages and disadvantages of these parallelization techniques. These properties can also be observed from a comparison of the constructed coarse grids depicted in Figure 2. Our goal is to construct a coarsening algorithm that avoids these disadvantages yet can also provide some of the advantages.

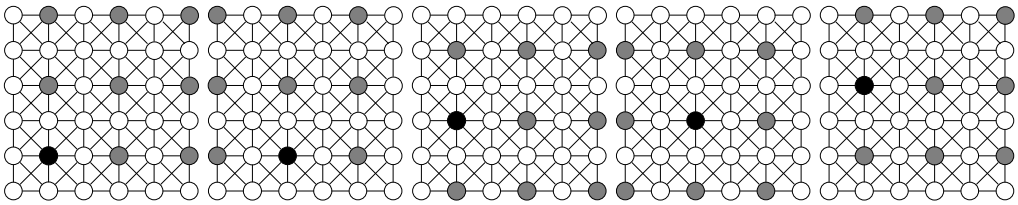


Figure 3. Resulting coarse grids for a 9-point discretization of the Laplace operator constructed by five different initial choices. The gray points indicate the respective coarse grid points, the black point indicates the first coarse grid point chosen.

4. Coarse Grid Classification

In the previous section we discussed several methods for generating coarse grids in parallel. Most of these algorithms include some kind of boundary treatment and thus construct a different coarse grid structure for the interior and the boundary of a processor subdomain. However, such a special boundary treatment can lead to a large number of additional coarse grid points near the boundary (or even throughout the whole domain when using the CLJP method). Thus, the overall solver requires a larger amount of memory and compute time.

In our approach, we will not change the classical point selection scheme but keep the original Ruge–Stüben scheme on each processor, which has been shown to produce good results in the sequential case. At first sight this might seem contradictionally to the demands for a true parallel algorithm since the MSB, RS3 and Falgout algorithms all employ a special boundary treatment. But in fact another property of the classical Ruge–Stüben scheme will allow us to avoid an explicit boundary treatment despite its sequential nature.

Since the points are chosen sequentially, it is clear that by changing the initial choice for the first coarse grid points the Ruge–Stüben algorithm gives a different coarse grid, see Figure 3. Note that the quality of the resulting coarse grids with respect to multigrid convergence and memory requirements is very similar. Hence, there is no special advantage of using either one of these coarse grids in a sequential computation. In parallel computations, however, this gives us a degree of freedom to consistently *match* the coarse grids obtained on each processor individually by the Ruge–Stüben scheme at processor subdomain boundaries. This observation is the starting point for our coarse grid construction algorithm.

More precisely our approach is as follows. First, we construct multiple coarse grids on each processor domain independently by running the classical algorithm multiple times with different initial coarse grid points. Note that this procedure is computationally efficient, since the classical Ruge–Stüben algorithm requires only a very small amount of computational time compared with the construction of the transfer and coarse grid operators while a well-constructed grid can save a large amount of time during the operator construction and in the multigrid cycle.

After the construction of these coarse grids on all processors, we need to select exactly one grid for each processor domain such that the union of these coarse grids forms a suitable coarse grid for the whole domain. We achieve this by defining a weighted graph whose *vertices* represent the *grids* constructed by the multiple coarsening runs. *Edges* are defined between vertices which represent grids on neighboring processor domains. Each edge weight measures the quality of the boundary constellation if these two grids are chosen to be part

of the composed grid. Finally, we use this graph to choose one coarse grid for each processor subdomain which automatically matches with most of its neighbors.

Program 4 CGC algorithm $CGC(S, S^T, ng, \{C_i\}_{i=1}^{ng}, \{F_i\}_{i=1}^{ng})$

```
for j ← 1 to |Ω| do λ_j ← |S_j^T|; od;
C_0 ← ∅;  λ_max ← arg max_{k∈Ω} λ_k;
do
    U ← Ω \ ⋃_{i≤it} C_i;
    if max_{k∈U} λ_k < λ_max then break; fi;
    it ← it + 1;  F_it ← ∅;  C_it ← ∅;
    do
        j ← arg max_{k∈U} λ_k;
        if λ_j = 0 then break; fi;
        C_it ← C_it ∪ {j};  λ_j ← 0;
        for k ∈ S_j^T ∩ U do
            F_it ← F_it ∪ {k};  λ_k ← 0;
            for l ∈ S_k ∩ U do λ_l ← λ_l + 1; od;
        od;
        for k ∈ S_j ∩ U do λ_k ← λ_k − 1; od;
    od;
od
ng ← it;
```

In our implementation, see Program 4, each processor p first determines the maximal weight $\lambda_{\max}$ of all points $i \in \Omega_p$. Every point with this weight can be chosen as an initial point for the classical coarsening algorithm. We choose one particular point and construct a coarse grid $C_{(p),1}$. Now we select another point with weight $\lambda_{\max}$ which is not already a member of $C_{(p),1}$ and construct a second coarse grid $C_{(p),2}$ starting with this point. Note that we construct disjoint coarse grids $C_{(p),1} \cap C_{(p),2}$ only. We repeat these steps as long as there is a point with weight $\lambda_{\max}$ that is not already a member of a coarse grid $C_{(p),it}$. Note that the number of iterations is bounded by the maximal number of strong couplings $|S_i|$ over all points $i \in \Omega_p$, which in turn is bounded by the maximal stencil width. Hence, the number of constructed grids ng_p per processor p is independent of the number of unknowns N and the number of processors P.

We now have obtained ng_p valid coarse grids $\{C_{(p),i}\}_{i=1}^{ng_p}$ on each processor p. To determine which grid to choose on each processor, we construct a directed, weighted graph $G = (V, E)$ whose vertices represent the created coarse grids,

$$V_p := \{C_{(p),i}\}_{i=1,\ldots,ng_p}, \quad V := \cup_{p=1}^{P} V_p.$$

The set of edges E consists of all pairs (v, u), $v \in V_p$, $u \in V_q$ such that $q \in \mathcal{S}_p$ is a neighboring processor of p,

$$E_p := \{\cup_{q \in \mathcal{S}_p} \cup_{v \in V_p,\ u \in V_q} (v, u)\}, \quad E := \cup_{p=1}^{P} E_p,$$

where $\mathcal{S}_p$ is defined as the set of processors q with points j which strongly influence points i on processor p, i.e.

$$\mathcal{S}_p := \{q \neq p : \ \exists i \in \Omega_p,\ j \in \Omega_q : \ j \in S_i\}.$$

To determine the weight $\gamma(e)$ of the edge $e = (v, u)$, we consider the nodes $v \in V_p$, $u \in V_q$. Each of these particular nodes represents a local C/F-splitting (C_p, F_p) for Ω_p and (C_q, F_q) for Ω_q, respectively. Together they form a C/F-splitting for the domain $\Omega_p \cup \Omega_q$. At the processor

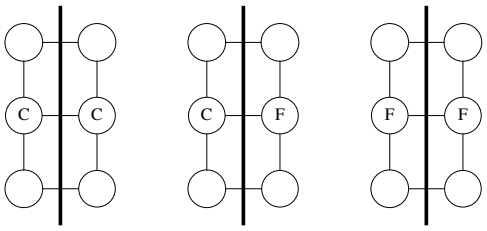


Figure 4. Three possible C/F-constellations at a processor's domain boundary.

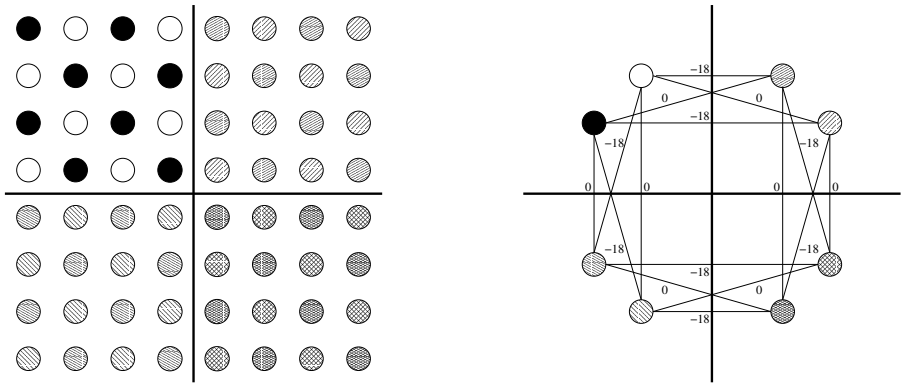


Figure 5. Application of the CGC algorithm to a 5-point stencil, distributed among 4 processors. The figure on the left shows the assignment of the points to the coarse grids, the figure on the right shows the weighted graph.

subdomain boundary, three grid configurations can occur, see Figure 4. We denote by $c_{C,C}$ the number of strong $C-C$-couplings (left), by $c_{C,F}$ the number of strong $C-F$-couplings (center) and by $c_{F,F}$ the number of strong $F-F$-couplings (right). Based on these classes of couplings we define the edge weight

$$\gamma(e) := c_{C,C}\gamma_{C,C} + (c_{C,F} + c_{F,C})\gamma_{C,F} + c_{F,F}\gamma_{F,F}$$

with $\gamma_{C,C},\ \gamma_{C,F},\ \gamma_{F,F} \in \mathbb{R}$ defined as follows. The most important case is the $F-F$-coupling case. Here, two fine grid points $i \in F_p$ and $j \in F_q$ are strongly coupled, which can lead to two problems: These two points may not have a common C-point to interpolate from, which violates relation (1). On the other hand, even if (1) is satisfied, we have to transfer the matrix rows A_i and A_j to construct a stable interpolation operator. Therefore, this situation must be avoided, which motivates us to penalize strong $F-F$-couplings with a large negative weight $\gamma_{F,F} := -8$.

The strong $C-C$-couplings should also be avoided because they can increase the operator and the grid complexity. We therefore set $\gamma_{C,C} := -1$. In the remaining case, which can be considered as the (optimal) sequential coarsening scenario, we do not add an additional weight, i.e. $\gamma_{C,F} := 0$.

Figure 5 shows the graph $G = (V, E)$ obtained by our CGC algorithm for a 5-point discretization of the Laplacian. We can observe that constellations with $C-C$–couplings

and $F-F$–couplings are heavily penalized, while constellations with $C-F$–couplings are weighted by zero only.

Now that we have constructed the graph G of admissible local grids, we can use it to choose a particular coarse grid for each processor such that the union of these local grids automatically matches at subdomain boundaries. Observe that the number of vertices is related to the number of processors P only; i.e., it is much smaller than the number of unknowns N. Furthermore, the cardinality of E is small compared to N since edges are only constructed between neighboring processors. Thus, we can transfer the whole graph onto a single processor without large communication costs.

On this processor, we choose exactly one node v_p from each subset $V_p \subset V$ with the following scheme. We denote by $\mathcal{C}$ the set of the selected local coarse grids.

1. First, we define *heavy edges or couplings* H_v between the nodes v of the graph, where p denotes the processor v belongs to (i.e. $v \in V_p$),
$$H_v := \cup_{q\in\mathcal{S}_p}\{w \mid \gamma(v,w) = \max_{u\in V_q}\gamma(v,u)\}, \text{ and } H_v^T := \{w \mid v \in H_w\}.$$
The heavy edges indicate which coarse grid on processor q can be fitted best to the coarse grid represented by $v \in V_p$. We assign a weight λ_v to each node v, where $\lambda_v = |H_v|+|H_v^T|$. This weight indicates how many coarse grids on other processors can be fitted to the coarse grid represented by v.
2. For some processors p, all nodes $v \in V_p$ might have weight $\lambda_v = 0$. As this means there are no strong couplings across the subdomain boundary, *any* grid constructed on this processor can be chosen. Here, we arbitrarily choose one arbitrary $v \in V_p$ and remove V_p from the graph.
3. We choose the node $v \in V_p$ with maximal weight, put it into $\mathcal{C}$ and remove the subset V_p from the graph, as a coarse grid for domain Ω_p is now determined. We then increase the weight of each node $w \in H_v \cup H_v^T$ to the maximal weight of all remaining nodes in the graph plus one (so that one of these will be chosen in the next step), and repeat this step as long as the graph is not empty.

This procedure takes up to P steps, one for each processor domain, see program 5 for details. After running the algorithm, we transfer the choice $v_p \in \mathcal{C} \cap V_p$ back to processor p. The union of all elements in $\mathcal{C}$ now defines the global consistent grid for the complete domain, see Figure 2(f).

Recall that after applying the first phase of the Ruge–Stüben coarsening, there may exist a few fine grid points with strong connections to other fine grid points only. However, these strong couplings are very rare. In the sequential phase, this is corrected by the second pass of the classical coarsening scheme. To correct these very few couplings across processor boundaries, we employ a more straightforward method: We check whether a fine grid point is strongly coupled to other fine grid points only and insert this point into the coarse grid if that is the case.

5. Numerical Results

We now compare the coarsening algorithms experimentally. We implemented all coarsening techniques discussed in this paper using the library PETSc [1, 2, 3].

Program 5 *AmgCGCChoose*$(V, H, \mathcal{C})$

```
begin
  C ← ∅;
  U ← V;
  for v ∈ U do λ_v ← |H_v| + |H_v^T|; od;
  for p ← {1, ..., P} do
      if λ_v = 0 for all v ∈ V_p
        then
              C ← {v};                                   arbitrary v ∈ V_p
              U ← U \ V_p;
      fi;
  od;
  while U ≠ ∅
        do
        v ← arg max_{w∈U} λ_w;
        C ← C ∪ {v};
        U ← U \ V_p such that v ∈ V_p;
        λ_max ← max_{w∈U} λ_w;
        for w ∈ (H_v ∪ H_v^T) ∩ U do λ_w ← λ_max + 1; od;
  od;
end
```

Two important measures for the quality of the hierarchy built during the setup phase are the *operator complexity* c_A and the *grid complexity* c_G,

$$c_A := \frac{\sum_{l=1}^{L_{\max}} nonzeros(A^l)}{nonzeros(A^1)}, \quad c_G := \frac{\sum_{l=1}^{L_{\max}} |\Omega^l|}{|\Omega^1|}. \tag{2}$$

These values give an indication of the increase in required memory during the coarsening algorithm compared to the size of the original linear system.

In our experiments, we set $\alpha = 0.25$ for the first phase and $\beta = 0.35$ for the second phase of the algorithm, see Section 2. We use the truncated standard interpolation [15] scheme with a truncation parameter $\epsilon_{tr} = 0.2$. The setup phase is stopped if no strong couplings are present.

As an initial guess for the solution phase we use a random-valued vector u_0 with $\|u_0\|_{l_2} = 1$. We use a (subdomain) block-Jacobi smoother with one inner Gauss–Seidel relaxation step on all levels and employ a $V(1,1)$-cycle, i.e. one pre- and one post-smoothing step per level. The iteration is stopped if the l^2-norm of the residual $r_{it} = f - Au_{it}$ drops below 10^{-10}. The convergence factor is determined as

$$\rho = \left(\frac{\|r_{it}\|_{l^2}}{\|r_1\|_{l^2}} \right)^{\frac{1}{it-1}}. \tag{3}$$

The examples were run on a cluster consisting of 41 IBM p690 with 32 CPUs and 128GB memory each which are connected through an HPS network.

Example 5.1 (Uniform grids). In our first experiment we consider the Poisson problem,

$$-\Delta u = 0 \text{ in } \Omega = (0,1)^2, \tag{4}$$

with vanishing Dirichlet boundary conditions on the unit square, discretized with a 5-point finite difference stencil on 511×511 points per processor. Table II shows the setup times for this problem, obtained with up to 256 CPUs of the IBM cluster. We see that the CGC method requires the same amount of time as the sequential algorithm (NONE), since no expensive

Table II. Setup time in seconds for Example 5.1.

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	20.8	20.8	20.8	25.1	20.8	20.8
4	24.8	25.3	24.9	30.4	27.7	24.8
16	25.4	26.7	25.9	34.1	30.6	26.5
64	29.7	31.6	31.9	39.2	35.2	29.6
256	41.1	44.3	48.8	55.7	49.5	42.6

boundary treatment is involved and the additional coarsening runs require essentially no time compared to the construction of the coarse grid operator. The most expensive algorithm is, as expected, the CLJP algorithm which does not allow for a fast coarsening. Regarding the

Table III. Operator complexity (2) for Example 5.1.

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	2.59	2.59	2.59	3.84	2.59	2.59
4	2.60	2.67	2.62	3.90	2.65	2.60
16	2.64	2.78	2.67	3.94	2.69	2.61
64	2.64	2.83	2.72	3.95	2.70	2.61
256	2.66	2.87	2.77	3.57	2.71	2.61

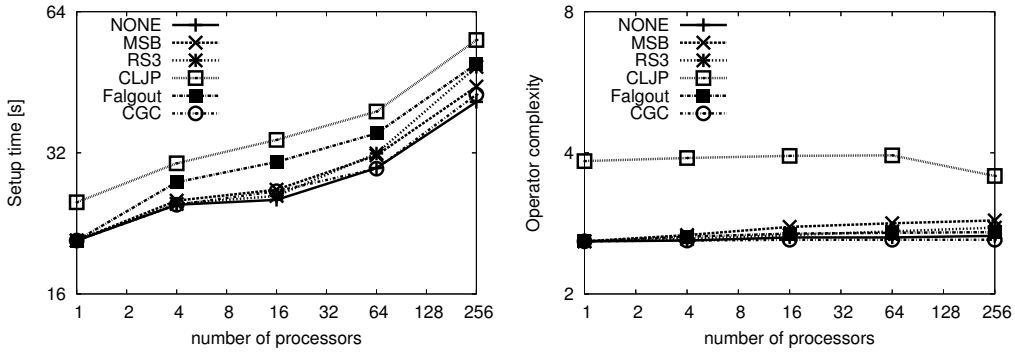


Figure 6. Setup time in seconds (left) and operator complexity (2) (right) for Example 5.1.

operator complexities, see Table III, we observe that the CLJP algorithm leads to large c_A values due to the amply selection of coarse grid points. The CGC algorithm achieves even slightly better values than the sequential algorithm (NONE), as the grids are matched and $C - C$ couplings across the boundaries are avoided. The same holds for the grid complexity c_G, which equals 1.97 for the CLJP algorithm, while the other methods give values up to 1.69 only. From the measured solution times given in Table IV we see a large increase of computational work especially for the NONE, MSB and RS3 algorithms. These methods do

Table IV. Solution time in seconds for Example 5.1.

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	6.49	6.49	6.49	6.86	6.49	6.49
4	17.3	16.6	16.1	11.9	7.71	8.18
16	72.3	57.0	53.1	27.1	17.6	19.6
64	207	178	205	136	98.5	59.3
256	743	1071	797	754	360	370

Table V. Convergence factors for Example 5.1.

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	0.13	0.13	0.13	0.16	0.13	0.12
4	0.44	0.48	0.39	0.25	0.19	0.26
16	0.81	0.72	0.72	0.56	0.39	0.41
64	0.91	0.89	0.91	0.82	0.82	0.71
256	0.97	0.96	0.96	0.96	0.94	0.94

Table VI. Overall time in seconds for Example 5.1.

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	27.3	27.3	27.3	31.4	27.3	27.3
4	42.1	41.9	41.0	42.4	35.5	33.0
16	97.7	83.7	79.0	61.3	48.2	46.1
64	237	209	237	175	134	88.9
256	785	1116	847	810	410	413

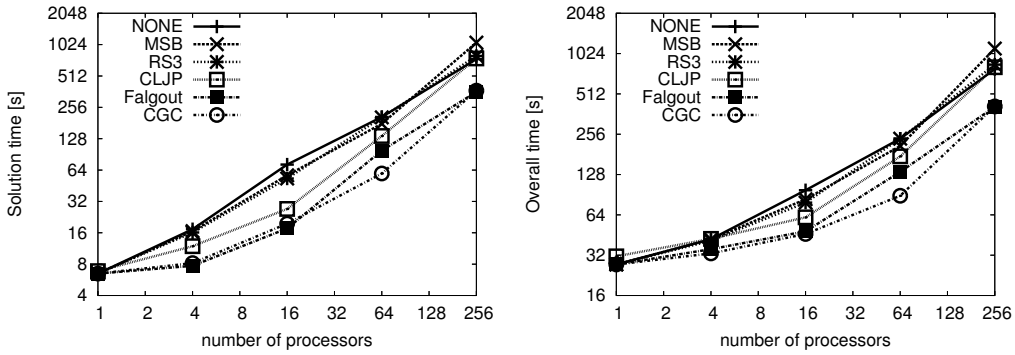


Figure 7. Solution time (left) and overall time (right) in seconds for Example 5.1.

not allow coarsening up to a single point which in turn increases the number of V-cycles. The CLJP algorithm produces a smaller number of unknowns on the coarsest level, but the coarsening process is less efficient than that of the Ruge–Stüben method and more operations are performed during each V-cycle. This drawback is cured by the Falgout method, which allows both for an efficient coarsening as well as few points on the coarsest level. The CGC algorithm, on the other hand, allows for a smaller number of points on the coarsest level than the NONE, MSB and RS3 algorithms due to the well-matched grids and the absence of additional inserted coarse grid points near the boundary. However, the coarsest CGC grid is

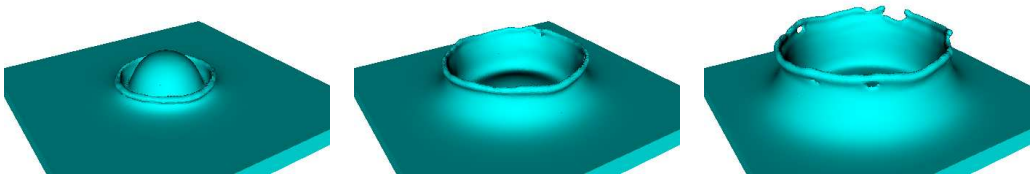


Figure 8. Solution of two-phase flow problem at several time steps.

slightly larger than that obtained with the Falgout scheme.

In summary, this experiment shows that the proposed CGC scheme gives a parallel AMG with similar scale-up behavior as the Falgout scheme. Yet, the CGC scheme leads to operator and grid hierarchies with better complexities. To fully assess the potential of the CGC method, however, we must be concerned also with its speed-up properties which we consider in the following.

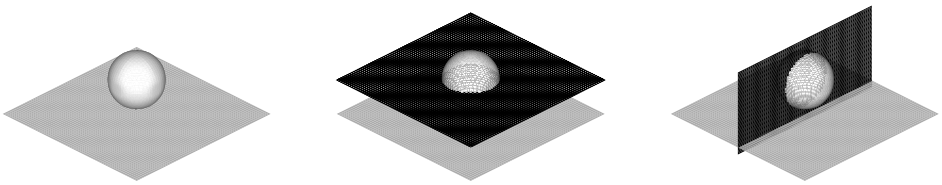
Example 5.2 (Navier–Stokes Equations). Here, we use our parallel AMG to solve the Pressure-Poisson equation in a two-phase flow simulation based on a Chorin projection method. Our finite difference CFD solver [9] determines the pressure p^{n+1} of the time step $n+1$ by solving the equation

$$\nabla \cdot \left(\frac{1}{\rho(\phi^{n+1})} \nabla p^{n+1} \right) = \nabla \cdot \frac{\vec{u}^*}{\delta t}, \tag{5}$$

where $\vec{u}^*$ is the current velocity field, δt the time step, ϕ^{n+1} a (time-depending) level set function which describes the location of the free surface and ρ denotes the density, which can be expressed depending on the level-set function. Near the surface, a large jump of the density can occur (in our case, a factor of ≈ 773). This leads to very large condition numbers for the resulting linear system.

In our simulation, we consider a box of size $2 \times 2 \times 1$ cm. Inside this box, a drop of water is falling into a water basin, see Figure 9. We first present the sequence of grids produced by the various parallelization techniques using a grid of $64 \times 64 \times 64$ points (for reasons of visualization only) partitioned onto four processors. Here, we visualize the results obtained for the initial time step only, however, similar results are obtained for all time steps, see Figure 8.

From the plots depicted in Figures 10 and 11 we see that the coarse grid points are clustered near the phase boundaries for all parallel coarsening techniques, as it is expected for an AMG method. However, we can observe some unphysical clustering of coarse grid nodes due to the parallel data decomposition. The RS3 algorithm inserts a band of additional coarse grid points near the processor subdomain boundary, the Falgout algorithm even adds coarse grid points away from the boundaries due to the employed CLJP algorithm for the boundary treatment. The grids produced by the CGC algorithm, however, resemble those produced in the sequential case very well. In contrast to all other methods, we find no artifacts due to the domain decomposition in the grids constructed by our CGC method. Hence, we expect that the CGC method is least affected by a worsening surface to volume ratio and has superior speed-up properties over the other parallelization techniques.



(a) Simulation setup (b) $x - y$ slice (c) $x - z$ slice

Figure 9. Free surfaces of the water drop and basin. Depicted is the constellation leading to the coarse grids in Figure 10 and 11

.

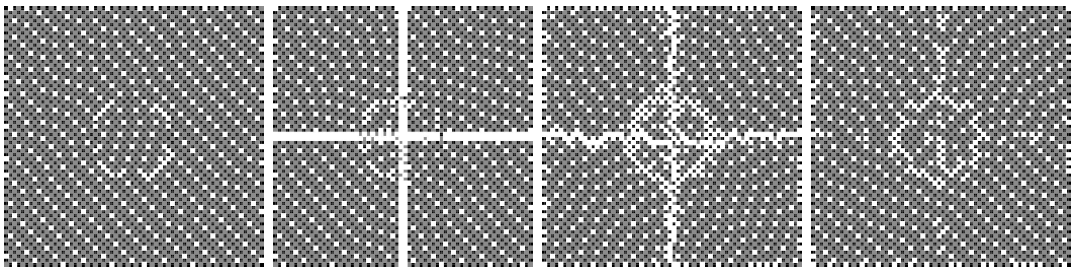


Figure 10. Results of various coarsening schemes. Depicted are the grid points of the $x - y$ slice at $z = 31h_z$ where h_z is the mesh width in z-direction. The black points belong to the fine grid Ω^1 only, the gray points also belong to grid Ω^2 and the white points belong to grid Ω^3. From left to right: sequential coarsening (one processor), RS3 algorithm, Falgout algorithm, CGC algorithm (on 4 processors each).

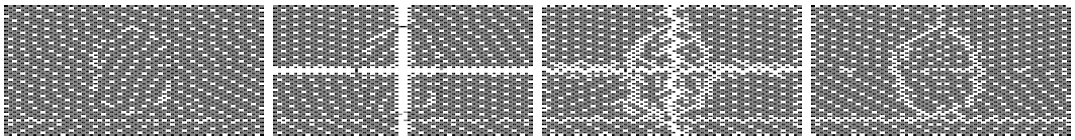


Figure 11. Results of various coarsening schemes. Depicted are the grid points of the $x - z$ slice at $y = 31h_y$ where h_y is the mesh width in y-direction. The black points belong to the fine grid Ω^1 only, the gray points also belong to grid Ω^2 and the white points belong to grid Ω^3. From left to right: sequential coarsening (one processor), RS3 algorithm, Falgout algorithm, CGC algorithm (on 4 processors each).

To compare the speed-up properties of the various coarsening schemes we consider a problem

Table VII. Setup time (left) in seconds and operator complexity (2) (right) for Example 5.2.

P	NONE	RS3	Falgout	CGC
1	473	473	473	473
2	280	280	280	280
4	143	151	160	142
8	68.7	80.2	77.8	74.6
16	50.0	56.4	52.5	42.5
32	32.9	36.1	33.6	24.4
64	22.2	22.3	28.7	13.0
128	20.1	15.7	19.3	8.47
256	14.5	10.5	13.1	5.51

P	NONE	RS3	Falgout	CGC
1	3.89	3.89	3.89	3.89
2	4.98	4.54	6.43	4.09
4	6.10	5.14	6.99	4.19
8	5.73	5.26	7.77	3.80
16	6.14	5.50	8.37	4.01
32	5.91	5.57	8.77	4.21
64	5.90	5.66	9.33	4.53
128	5.86	5.90	9.79	4.60
256	6.26	6.48	10.5	5.07

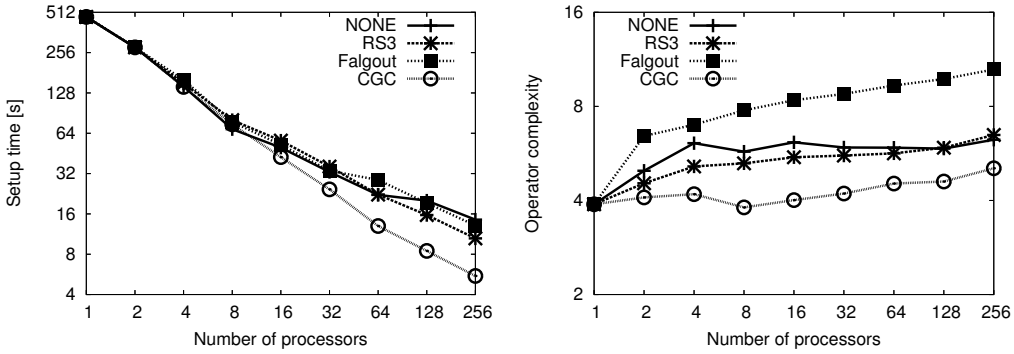


Figure 12. Setup time (left) and operator complexity (2) (right) for Example 5.2.

of size $128 \times 128 \times 128$. For this problem we employed a subdomain agglomeration scheme.[‡] For large processor numbers, the CGC algorithm only needs about half the setup time (Table VII) than the other methods. The well-fitted grids produced by the CGC algorithm allow the computation of the coarse grid operator to be carried out in half the time the classical coarsening (NONE) needs, a feature that compensates the time needed for the additional coarsening iterations easily. The well-matched grids also yield a low operator complexity (Figure VII). The same holds for the grid complexity, which only increases from 1.67 to 1.74 as the number of processors increases from 8 to 256, while the Falgout algorithm produces a grid hierarchy of complexity 1.97 using 256 processors. We summarize the total compute time in Figure 13. Here, we used a block-Jacobi smoother with inner Gauss–Seidel relaxation on all levels, i.e. no direct coarse solver. The convergence factors for this problem all stay below 0.1 independent of the number of processors and the employed parallelization technique. Hence, the setup phase is the most expensive part of the AMG code and this phase also dominates the overall costs. From the graphs depicted in Figure 13 we can clearly observe that the CGC

[‡]If more than 70% of the points are boundary points, e.g. $i \in \partial\Omega_p$ for some p, two neighboring subdomains are merged.

Table VIII. Overall time in seconds (left) and parallel efficiency (right) for example 5.2.

P	NONE	RS3	Falgout	CGC
1	553	553	553	553
2	322	324	363	328
4	168	172	184	164
8	85.1	93.4	95.4	86.2
16	63.7	66.0	62.5	47.4
32	41.9	41.8	43.0	27.7
64	28.5	26.3	36.2	15.1
128	25.8	18.5	23.3	9.97
256	19.5	12.7	15.3	6.75

P	NONE	RS3	Falgout	CGC
1	1	1	1	1
2	0.86	0.86	0.77	0.85
4	0.82	0.80	0.75	0.84
8	0.81	0.74	0.73	0.80
16	0.54	0.52	0.55	0.73
32	0.41	0.41	0.40	0.62
64	0.30	0.33	0.24	0.57
128	0.17	0.23	0.19	0.43
256	0.11	0.17	0.14	0.32

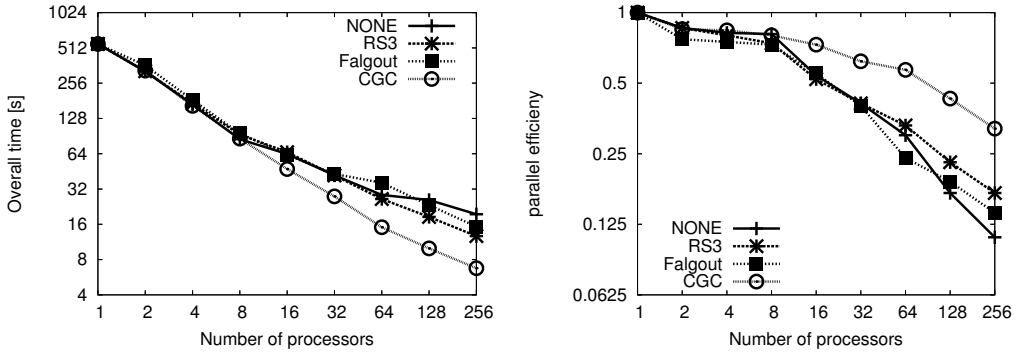


Figure 13. Overall time in seconds (left) and parallel efficiency (right) for Example 5.2.

algorithm achieves a parallel efficiency over 50% up to 64 processors (i.e. $32 \times 32 \times 32$ points per processor), while the efficiency of all other methods drops below this level for more than 16 processors already. Hence, the CGC method allows us to utilize a larger part of a parallel computer with acceptable parallel efficiency than the other parallel AMG techniques. Note that the speed-up properties of a linear solver are especially important for time-dependent problems like the considered flow problem. Since the spatial resolution is coupled to the time discretization, engineers are very interested in solving a problem of given size faster simply by increasing the number of CPUs.

6. Concluding remarks

In this paper we presented a new approach to the construction of coarse grids for parallel AMG solvers. As the classical Ruge–Stüben coarsening scheme [14] performs excellently in the sequential case, our aim was to keep this scheme in the parallel case without coarsening the processor subdomain boundaries by another method. To this end, we construct multiple coarse grids by the classical coarsening scheme. Then, we define a weighted graph which describes the relationship between the constructed coarse grids and choose one coarse grid per processor using graph clustering techniques. This allows us to match the coarse grids at the processor

boundaries automatically such that a further boundary treatment is not required.

The results of our numerical experiments showed that the proposed CGC coarsening algorithm leads to coarse grids which stay very close to those obtained by sequential AMG, i.e. CGC-AMG shows virtually no artifacts due to parallelization. Furthermore, the proposed method leads to a faster setup phase while maintaining the convergence behavior of sequential AMG. Finally, we have compared the RS3 method, the Falgout method and the CGC method for solving the Pressure-Poisson equation arising in the discretization of a two-phase flow simulation. In this case, the CGC method allowed a considerably faster setup phase, led to lower memory requirements compared to the other approaches and showed a much better parallel efficiency than the other parallel AMG techniques.

REFERENCES

1. S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
2. S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc web page*, 2001. http://www.mcs.anl.gov/petsc.
3. S. Balay, V. Eijkhout, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.
4. A. Brandt, *Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems*, Proc. of the Third Int. Conf. on Numerical Methods in Fluid Mechanics, Univ. Paris 1972 (New York, Berlin, Heidelberg) (H. Cabannes and R. Teman, eds.), Springer, 1973.
5. A. Brandt, *Algebraic multigrid theory: The symmetric case*, Appl. Math. Comput., 19 (1986), pp. 23–56.
6. A. Brandt, S. F. McCormick, and J. Ruge, *Algebraic Multigrid (AMG) for automatic multigrid solution with application to geodetic computations*, 1982.
7. A. Brandt, S. F. McCormick, and J. Ruge, *Algebraic Multigrid (AMG) for sparse matrix equations*, in Sparsity and its Applications, D. Evans, ed., Cambridge University Press, Cambridge, 1984, pp. 257–284.
8. A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones, *Coarse-grid selection for parallel algebraic multigrid*, in Proc. of the Fifth International Symposium on Solving Irregularly Structured Problems in Parallel, vol. 1457 of Lecture Notes in Computer Science, Springer Verlag, 1998, pp. 104–115. Also available as Lawrence Livermore National Laboratory technical report UCRL-JC-130893.
9. R. Croce, M. Griebel, and M. A. Schweitzer, *A parallel level-set approach for two-phase flow problems with surface tension in three space dimensions*, Preprint 157, Sonderforschungsbereich 611, Universität Bonn, 2004.
10. E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. Meier Yang, *A Survey of parallelization techniques for Multigrid Solvers*, Tech. Rep. UCRL-BOOK-205864, Lawrence Livermore National Laboratory, Aug. 2004.
11. W. Hackbusch, *Multi-Grid Methods and Applications*, Springer series in Computational Mathematics, Springer-Verlag, Berlin, Heidelberg, 1985.
12. V. E. Henson and U. Meier Yang, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Tech. Rep. UCRL-JC-141495, Lawrence Livermore National Laboratory, Mar. 2001.
13. B. Metsch, *Ein paralleles graphenbasiertes algebraisches Mehrgitterverfahren*, Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, Bonn, Germany, 2004.
14. A. Krechel and K. Stüben, *Parallel algebraic multigrid based on subdomain blocking*, Tech. Rep. REP-SCAI-1999-71, GMD, Dec. 1999.
15. K. Stüben, *Algebraic Multigrid (AMG): an introduction with applications*, tech. rep., GMD - Forschungszentrum Informationstechnik GmbH, Mar. 1999.
16. G. W. Zumbusch, *Adaptive parallel multilevel methods for partial differential equations*, Habilitation, Universität Bonn, 2001.