Coarse Grid Classification: AMG on Parallel Computers

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

Institut für Numerische Simulation, Universität Bonn,
Wegelerstraße 6, 53115 Bonn, Germany
E-mail: {griebel, metsch, schweitzer}@ins.uni-bonn.de

1 Introduction

Many phenomena in science and engineering can be described by a mathematical model. Such a model describes the relations between the variables and constants that are important for this problem, like velocities, forces or material properties. In many cases, these relationships are expressed by partial differential equations (PDEs).
Often it is not possible to obtain an analytical solution for these PDEs. Hence, one has to compute an approximate numerical solution. To this end, the underlying domain is discretized using a finite grid Ω. On this grid, the PDE is discretized using finite differences, finite elements of finite volumes. This results in a sparse (i.e. only few entries per row are different from zero) linear system of equations $Au = f$.
Multigrid Methods (MG)[1,6] are known to be optimal methods for such kind of linear systems. Algebraic Multigrid Methods (AMG)[2,10] extend the multigrid idea to a purely algebraic setting, i.e. the only input they need is the matrix A. They can be used if a geometric multigrid hierarchy is not available. This flexibility has a price: A *setup phase*, has to be carried out in which the multigrid hierarchy, i.e., the sequence of grids, transfer operators and coarse grid operators, is constructed automatically from A, before the well-known multigrid cycle (the *solution phase*) can be started.
For linear systems with millions of unknowns, parallel computation is necessary to speed up the computation and to be able to store the system in the memory. While the parallelization of geometric multigrid and hence the solution phase of algebraic multigrid is straightforward, the algorithms for creating the coarse grids in AMG are inherently sequential.
Various approaches for the parallelization of this step have been proposed over the years[7,8,3]. In this paper, we briefly describe the Coarse Grid Classification algorithm[4,5], as well as its extension CGC-ML to very large numbers of processors.

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 $\Omega = \{1, \ldots, N\}$[a], with $A = (a_{ij})_{i,j=1}^{N}$ being a large sparse real matrix, $u = (u_i)_{i=1}^{N}$

[a] We denote grid points with their respective counting index in any dimension.

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[6], we 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. To this end, in an algebraic multigrid method, we choose a simple smoothing scheme S^l, e.g. Gauss–Seidel or Jacobi relaxation. Then, we construct the grids $\{\Omega^l\}_{l=1}^L$, the transfer operators $\{P^l\}_{l=1}^{L-1}$ and the coarse grid operators $\{A^l\}_{l=1}^L$ in a recursive fashion depending on the fine grid operator $A = A^1$ only. This construction is carried out in the so-called setup phase which consists of three steps: the selection of an appropriate coarse grid

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

the construction of a stable prolongation operator P^l, and the computation of the Galerkin coarse grid operator $A^{l+1} := (P^l)^T A^l P^l$. In the following, we omit the level index l where possible to simplify the notation.

Essential to many AMG schemes is the notion of a strong coupling between the grid point i and the grid point j. A grid point i is called *strongly coupled* to a grid point j if the corresponding matrix entry a_{ij} is relatively large. In the Ruge–Stüben coarsening (RSC) algorithm[9,10] for instance the coarsening (see Program 1) is based on the sets

$$S_i := \{j \neq i : \ -a_{ij} \geq \alpha \max_{k \neq i} |a_{ik}|\}, \ S_i^T := \{j \neq i : \ i \in S_j\}, \tag{1}$$

(typically $\alpha = 0.25$) which describe the strong connectivity graph of a matrix A. Along these strong couplings, the smooth parts of the error vary slowly. For an efficient coarse-grid correction of these parts, interpolation must also follow these couplings. This leads to three criteria for the choice of the coarse grid,

1. Let $i \in F$. Then for each $j \in S_i$ holds either $i \in C$ or $S_j \cap (C \cap S_i) \neq \emptyset$.
2. For all $i \in C$ and $j \in S_i \cup S_i^T$ holds $j \notin C$
3. C is a maximal set satisfying these properties.

In general, it is not possible to fulfill all these criteria. We choose to enforce the first one, as the stability of the interpolation depends on it. The coarsening process is carried out in

Program 1 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
```

two phases. In the first phase, see Program 1, an independent set C of coarse grid points

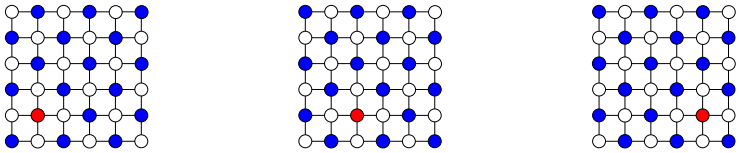


Figure 1. Resulting coarse grids for a 5-point discretization of the Laplace operator constructed by three different initial choices. The blue points indicate the respective coarse grid points, the red point indicates the first coarse grid point chosen.

is determined. Observe that we have a degree of freedom while choosing the first coarse grid point i (see the line indicated by $(*)$). After the first phase is carried out, a second coarsening phase checks that Criterion 1 is satisfied for all pairs $i, j \in F$ of fine grid points. If the condition $S_j \cap (C \cap S_i) \neq \emptyset$ is not fulfilled for a certain pair, then one of these points is added to the set of coarse grid points C.

Note that in Program 1 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 shows the sequential character of this coarsening algorithm, as the weight updates propagate throughout the whole domain during the coarsening loop. Hence, the parallelization of Ruge–Stüben-based AMG schemes is not straightforward.

3 Coarse Grid Classification

As can be seen from Figure 1, different choices for the first coarse grid point yield different coarse grids. 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 RSC scheme at processor subdomain boundaries. This observation is the starting point for our coarse grid construction algorithm.

The CGC coarsening scheme employs a *two-stage* coarsening algorithm. First, we construct multiple coarse grids on each processor domain Ω_p independently by running the RSC algorithm multiple times with different initial coarse grid points. 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.

For details of the first stage, we refer to Program 2. 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. The later constructed coarse grids may be of inferior quality but the selection mechanism described in the following will avoid them.
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 candidate coarse grids,

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

Program 2 CGC $(S, S^T, ng, \{C_i\}_{i=1}^{ng}, \{F_i\}_{i=1}^{ng})$

for $j \leftarrow 1$ **to** $|\Omega|$ **do** $\lambda_j \leftarrow |S_j^T|$; **od**;
$C_0 \leftarrow \emptyset$; $\lambda_{\max} \leftarrow \arg\max_{k\in\Omega} \lambda_k$;
do
  $U \leftarrow \Omega \setminus \bigcup_{i\le it} C_i$;
  if $\max_{k\in U} \lambda_k < \lambda_{\max}$ **then** $break$; **fi**;
  $it \leftarrow it + 1$; $F_{it} \leftarrow \emptyset$; $C_{it} \leftarrow \emptyset$;
  do
    $j \leftarrow \arg\max_{k\in U} \lambda_k$;
    if $\lambda_j = 0$ **then** $break$; **fi**;
    $C_{it} \leftarrow C_{it} \cup \{j\}$; $\lambda_j \leftarrow 0$;
    for $k \in S_j^T \cap U$ **do**
      $F_{it} \leftarrow F_{it} \cup \{k\}$; $\lambda_k \leftarrow 0$;
      for $l \in S_k \cap U$ **do** $\lambda_l \leftarrow \lambda_l + 1$; **od**;
    od;
    for $k \in S_j \cap U$ **do** $\lambda_k \leftarrow \lambda_k - 1$; **od**;
  od;
od
$ng \leftarrow it$;

(a) candidate coarse grids (b) consistent coarse grid

Figure 2. Discretization of the Laplace operator using finite differences. The left figure shows the candidate coarse grids indicated by different colors, the right figure shows the CGC graph.

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 each edge $e = (v, u)$ we assign a weight $\gamma(e)$ which measures the quality of the composed coarse grid if v and u are chosen to be part of it[4]. Figure 2 shows an example of the candidate coarse grids and the respective graph. After 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.

Program 3 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}; 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
```

To this end, we transfer the whole graph onto a single processor. On this processor, we choose exactly one node v_p from each subset $V_p \subset V$ with the following scheme: We first define heavy edges,

$$H_v := \cup_{q \in \mathcal{S}_p} \{w \,|\gamma(v,w) = \max_{u \in V_q} \gamma(v,u)\}, \text{ and } H_v^T := \{w \mid v \in H_w\}.$$

that indicate which candidate coarse grids can be fitted best to the coarse grid represented by $v \in V_p$. We then employ Program 3 to create the global coarse grid. This procedure takes up to P steps, one for each processor domain, see Program 3 for details. After running the algorithm, we transfer the choice $v_p \in \mathcal{C} \cap V_p$ back to processor p.

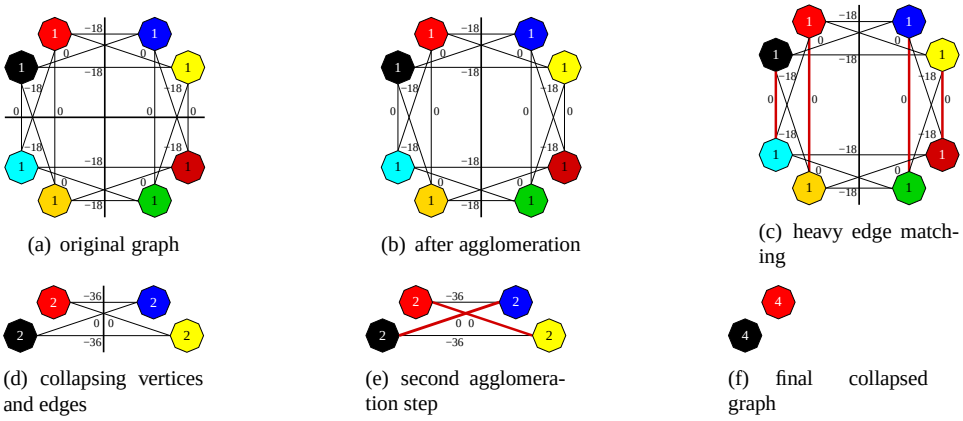
4 Multilevel Coarse Grid Classification

The main advantage of CGC over other parallel AMG is that the constructed coarse grids are very close to those a sequential AMG would produce. However, the original CGC has one major drawback: The graph representing the candidate coarse grids needs to be transferred to a single processor. For large numbers of processors ($P \gtrapprox 1000$) this leads to large communication costs as well as a significant run-time for the coarse grid selection algorithm AmgCGCChoose.

To overcome this issue, we have developed the CGC-ML algorithm which carries out the coarse grid selection procedure in a recursive, multilevel manner. We construct the graph $G = (V, E)$ as described in the last section. In addition, we assign a weight to each vertex which denotes the number of processor subdomains covered by the coarse grid represented by this vertex. Naturally, this weight is initialized with 1.

We do *not* transfer the whole graph onto a single processor. Instead, we proceed as follows:

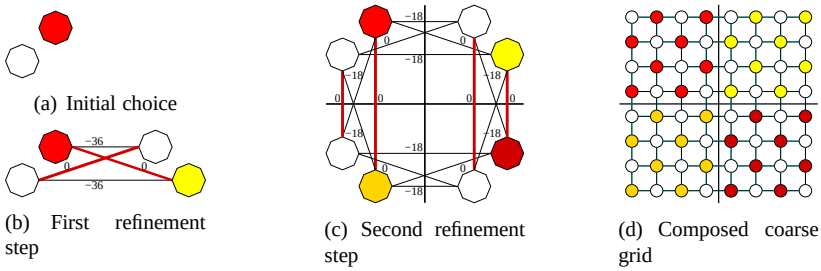
1. We agglomerate the graph on a subset of the processors, see Figure 3(b). Hence, a part of the edges (in this case, the vertical edges) do not cross the processor boundaries any more.
2. We can now employ a *heavy matching* on the inner edges of each processor subdomain, see Figure 3(c). Note that all edge weights are negative, so in fact the selected edge is the one with the smallest absolute weight.



(a) original graph (b) after agglomeration (c) heavy edge matching

(d) collapsing vertices and edges (e) second agglomeration step (f) final collapsed graph

Figure 3. Graph clustering process. The graph is constructed by the CGC algorithm to a 5-point finite-difference discretization of the Laplace operator, distributed among four processors (cp. Figure 2(a)). The numbers in the vertices denote the number of subdomains covered by the coarse grid which is represented by the respective vertex. the number at each edge denotes the edge weight.



(a) Initial choice

(b) First refinement step (c) Second refinement step (d) Composed coarse grid

Figure 4. CGC-ML refinement process.

3. We collapse the matched vertices and merge the edge sets, see Figure 3(d). Each vertex now represents a candidate coarse grid on an union of processor subdomains. Accordingly, we update the vertex weights. The edge set of each vertex u is the union of the edge sets of the vertices v, w that were collapsed into u: $E_u \leftarrow E_v \cup E_w$. However, we never create an edge between two vertices which represent candidate coarse grids on the same processor subdomain. If two edges are collapsed into the same edge, we add their edge weights.
4. We proceed matching and collapsing the graph. If no further matching is possible, we again agglomerate the graph on a smaller subset of processors. If we are already on a single processor, we stop, see Figure 3(e) – 3(f).

We have now obtained a small set of vertices on a single processor. Now, we choose one vertex u such that it covers a maximal number of processor subdomains and mark it, see Figure 4(a). Then, we mark the vertices v and w that were collapsed into u. We recursively proceed refining this choice until we have reached the original graph, see 4(b) – 4(c). Now, on each processor subdomain, the candidate coarse grid represented by the marked vertex

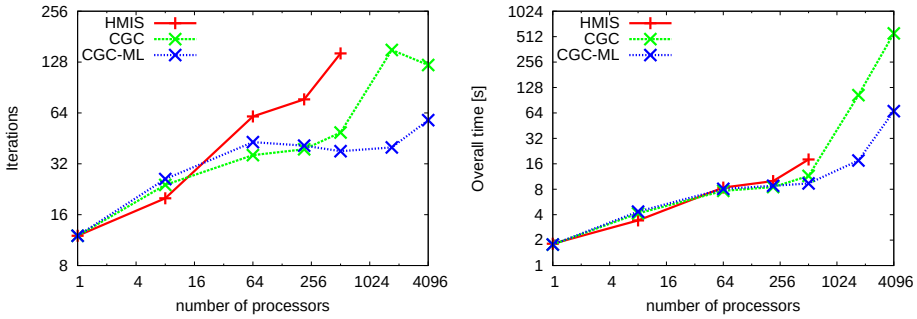


Figure 6. Number of iterations (left) and overall timings (right) for the model problem

is selected as coarse grid for this processor subdomain and we obtain a coarse grid for the global discretization domain as depicted in Figure 4(d).

During this refinement process, we must ensure that one vertex is marked per processor subdomain on each level of the graph hierarchy. In consequence, this will guarantee that after finishing the refinement, we have selected one candidate coarse grid on each processor subdomain. In our implementation, we proceed as follows. At each step in the refinement process where more processors are involved as in the previous step (i.e. a processor agglomeration was performed in the matching phase), we determine if a vertex is marked on *each* processor. If this is not the case, we mark the vertex that is most heavily coupled to the marked vertices on neighboring processors.

Hence, on *each* level of the AMG multilevel hierarchy, we employ a multi-level graph coarsening algorithm.

5 Numerical Results

In this section, we present first numerical results obtained on the JUBL supercomputer. In particular, we employ AMG as a preconditioner for the conjugate gradient method and we compare the CGC-ML algorithm with the original CGC algorithm as well as the HMIS parallel coarsening algorithm[3].

We consider an model problem in three spatial dimensions,

$$-\nabla \cdot (a \nabla u) = f \tag{2}$$

on $[0,1]^3$ with Dirichlet boundary conditions. The diffusion coefficient a depends on (x, y, z) as depicted in Figure 5. We employ a 7-point finite difference scheme to discretize the problem on $31 \times 31 \times 31$ points per processor subdomain.

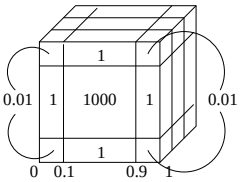


Figure 5. Values of the diffusion coefficient for Eq. 2

As strength threshold in (1), we set $\alpha = 0.25$. We omit the second coarsening pass of the RSC scheme as we use AMG as preconditioner only. Furthermore, we use the modified classical interpolation, see Ref. 7. On each level of the multigrid hierarchy, we employ a hybrid Gauss-Seidel/Jacobi smoother. We start the iterations with a zero initial vector u_0. The iteration is stopped if the residual

$r_{it} = f - Au_{it}$ drops below 10^{-8} measured in the l^2-norm.
In Figure 6 we give the plots of the number of iterations and overall run-time for the considered parallel AMG schemes. From these plots we see that the CGC-ML algorithm achieves robust preconditioning for this problem up to thousands of processors. In contrast, the iteration numbers for HMIS coarsening increase significantly and the algorithm does not converge within 1000 steps for more than 512 processors. We see an increase of the total wall time requirements beyond 1024 processors which is caused by the slower communication between the racks. However, The CGC-ML algorithm shows a significantly improved scale-up behavior compared with the original CGC algorithm.

Acknowledgments

This work was sponsored by the Sonderforschungsbereich 611, *Singuläre Phänomene und Skalierung in mathematischen Modellen*, sponsored by the *Deutsche Forschungsgemeinschaft*. We want to thank the Forschungszentrum Jülich for compute time on the JUMP and JUBL supercomputers.

References

1. A. Brandt, *Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems*, In Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics, H. Cabannes and R. Teman, (eds.), pages 82–89, New York, Berlin, Heidelberg, Springer Verlag, 1973.
2. A. Brandt, S.F. McCormick, and J. Ruge, *Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations*, Institute for Computational Studies, POB 1852, Fort Collins, Colorado, 1982.
3. H. De Sterck, U. M. Yang, and J.J. Heys *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM J. on Matrix Analysis and Applications, **27**, pp. 1019–1039, 2006.
4. M. Griebel, B. Metsch, D. Oeltz, and M. A. Schweitzer, *Coarse grid classification: A parallel coarsening scheme for algebraic multigrid methods*, Numerical Linear Algebra with Applications, **13(2–3)**, pp.193–214, 2006.
5. M. Griebel, B. Metsch, and M. A. Schweitzer, *Coarse grid classification–Part II: Automatic coarse grid agglomeration for parallel AMG*, Preprint 271, Sonderforschungsbereich 611, Universität Bonn, 2006.
6. W. Hackbusch, *Multi-grid methods and applications*, Springer Series in Computational Mathematics. Springer-Verlag, Berlin, Heidelberg, 1985.
7. V. E. Henson and U. M Yang, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics **41**, pp. 155-177, 2002.
8. A. Krechel and K. Stüben, *Parallel algebraic multigrid based on subdomain blocking*, Parallel Computing **27**, pp. 1009-1031, 2001.
9. J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, In Multigrid Methods, S. F. McCormick (Ed.), Frontiers in Applied Mathematics Vol. **5**, SIAM, 1986.
10. K. Stüben, *Algebraic multigrid (AMG): An introduction with applications*, In Multigrid, U. Trottenberg, C. W. Oosterlee, and A. Schüller, pages 413–532, Academic Press, 2001.