Parallel algebraic multigrid based on subdomain blocking

Arnold Krechel, Klaus Stüben

German National Research Center for Information Technology (GMD)
Institute for Algorithms and Scientific Computing (SCAI)
Schloss Birlinghoven, D-53754 St. Augustin, Germany

e-mail: krechel@gmd.de, stueben@gmd.de

GMD-Report 71, November 1999

Abstrakt. Algebraisches Mehrgitter (AMG) bietet Ansätze zur Entwicklung rein algebraischer Verfahren zur effizienten Lösung großer Gleichungssysteme, die auf unstrukturierten, zwei- oder dreidimensionalen Gittern basieren. Während sequentielles AMG zur Lösung von Problemen mit mehreren Millionen Unbekannten benutzt werden kann, setzt eine weitere Erhöhung der Problemgröße eine effiziente Parallelisierung voraus. Weil sich, im Gegensatz zu geometrischen Mehrgitterverfahren, die Vergröberungshierarchie und die zugehörigen Grobgitteroperatoren in einer Aufsatzphase von AMG dynamisch entwickeln, ist eine direkte Parallelisierung sehr kompliziert. Darüberhinaus würde eine "naive" Parallelisierung unvorhersehbare und überaus komplexe Kommunikationsmuster erfordern, mit der Konsequenz einer ernsten Beschränkung der erreichbaren Skalierbarkeit, insbesondere der ohnehin vergleichsweise teuren Aufsatzphase. In diesem Papier betrachten wir eine klassische AMG-Variante, die sich als sehr robust und effizient bei der Lösung großer Gleichungssysteme erwiesen hat, die bei der Diskretisierung elliptischer Differentialgleichungen mit finiten Differenzen oder finiten Volumen entstehen. Basierend auf einer einfachen Partitionierung der Variablen (durch eines der vielen algebraischen Partitionierungstools wie z.b. Metis), schlagen wir einen Parallelisierungsansatz vor, bei dem die Kommunikation minimiert wird ohne in komplexen Anwendungen das Konvergenzverhalten nachhaltig zu verschlechtern. Wir präsentieren Ergebnisse für verschiedene industrielle Anwendungen aus den Bereichen CFD und Ölreservoir-Simulation.

Schlagwörter. AMG, parallel, Algebraisches Mehrgitter, unstrukturierte Gitter, unstrukturierte Matrizen, hierarchische Löser.

Abstract. The algebraic multigrid (AMG) approach provides a purely algebraic means to tackle the efficient solution of systems of equations posed on large unstructured grids, in 2D and 3D. While sequential AMG has been used for increasingly large problems (with several million unknowns), its application to even larger applications requires a parallel version. Since, in contrast to geometric multigrid, the hierarchy of coarser levels and the related operators develop dynamically during the setup phase of AMG, a direct parallelization is very complicated. Moreover, a "naive" parallelisation would, in general, require unpredictable and highly complex communication patterns which seriously limit the achievable scalability, in particular of the costly setup phase. In this paper, we consider a classical AMG variant which has turned out be highly robust and efficient in solving large systems of equations corresponding to elliptic PDEs, discretized by finite differences or finite volumes. Based on a straightforward partitioning of variables (using one of the available algebraic partitioning tools such as Metis), a parallelisation approach is proposed which minimizes the communication without sacrificing convergence in complex situations. Results will be presented for industrial CFD and oil-reservoir simulation applications on distributed memory machines, including PC-clusters.

Keywords. AMG, parallel, algebraic multigrid, unstructured grids, unstructured matrices, hierarchical solvers.

Submitted to: Parallel Computing Journal.

1 Introduction

Algebraic multigrid (AMG) provides an approach to developing hierarchical and purely algebraic multi-level methods for solving certain classes of (sparse) matrix equations. Based on the ideas in [1, 2, 3], the first public domain AMG program, AMG1R5, was described and investigated in [6, 7, 8]. Although it was already clear at that time that there was much potential for generalizations, this program essentially aimed at the algebraic solution of systems of equations involving M-matrices.

Since the early nineties, and even more since the mid nineties, there was a strong increase of interest in algebraically oriented multi-level methods. One reason for this was the increasing geometrical complexity of applications (see Figure 1 for an example) which, technically, limited the immediate use of geometric multigrid. Another reason was the steadily increasing demand for hierarchical "plug-in" solvers. In particular in commercial codes, this demand was driven by increasing problem sizes which made clear the limits of classical one-level solvers used in most packages.

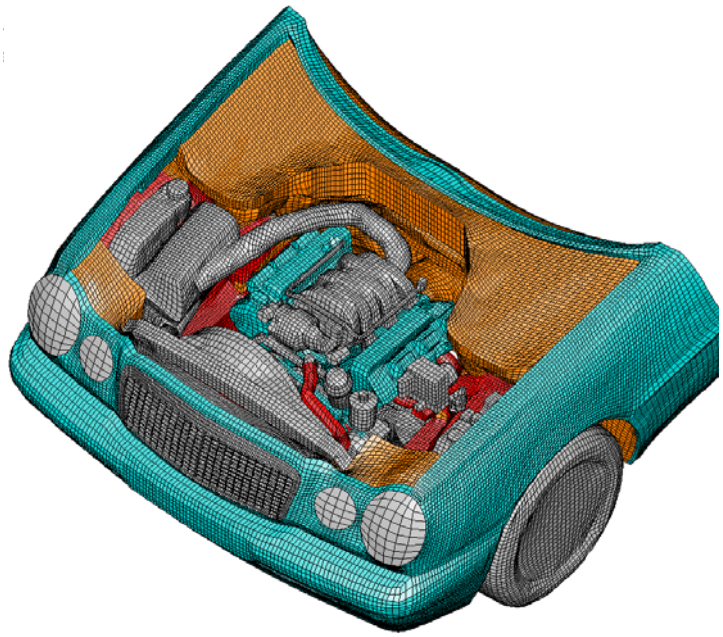

Figure 1: Mesh for computing the underhood flow of a Mercedes-Benz E-Class

As a consequence, the investigation of algebraic multi-level approachs was strongly pushed forward, partly aiming at extending the range of applicability. Several new approaches have been investigated such as, for instance, smoothed aggregation-based AMG, variants based on energy minimization principles to define interpolation, and AMGe. Moreover, a variety of hierarchical algebraic approaches has been (and is still being) developed which are significantly different from AMG in the sense that these approaches are not based on the fundamental multigrid principles, smoothing and coarse-level correction. For a review of the AMG development, see [10].

This paper refers to the (distributed memory) parallelization of the variant of "classi-

cal" AMG introduced in [9]. For important classes of problems posed on large unstructured grids, this approach has proved to combine the asymptotic optimality features of geometric multigrid (h-independent convergence) with the advantages of easy-to-use plug-in solvers in a way which made it highly interesting for an industrial use[1]. Typical target applications are symmetric positive definite (s.p.d.) systems of equations corresponding to scalar second order elliptic PDEs, discretized by means of finite differences or finite volumes. In the industrial context, the major strengths of AMG are its robustness (for instance, w.r.t. anisotropies and jumping coefficients) and its immediate applicability to unstructured grids, both in 2D and 3D. Although the approach is being extended to cover also the solution of certain *systems* of PDEs (e.g. from elasticity), major research is still ongoing and much remains to be done to obtain an efficiency and robustness comparable to the case of scalar applications. Therefore, this paper considers only the scalar case.

In order to achieve best parallel performance, we stay as close as possible to the sequential algorithm. Unfortunately, a "direct" parallelization of AMG is, technically, fairly complicated. This is because the hierarchy of coarser levels and the corresponding operators, constructed in a special setup phase, develop dynamically from one level to the next. Moreover, a "naive" parallelisation would, in general, require unpredictable and highly complex communication patterns which seriously limit the achievable scalability, in particular of the costly setup phase. Algorithmical modifications are required to limit the communication cost without sacrificing convergence significantly. Finally, AMG's coarsening strategy is inherently sequential and has to be replaced by a reasonable parallel analog (in the context of the original AMG, this has been discussed in [4]).

For completeness, we want to point out that some AMG approaches allow a simpler parallelization. Such simplifications, however, are either at the expense of a sub-optimal performance of the respective sequential algorithm (e.g. *non-smoothed* aggregation-based AMG [5]) or at the expense of a restricted generality (e.g. by introducing geometry in order to limit the complexity of the coarser AMG levels).

In Section 2, we first give a summary of the sequential AMG to an extent necessary for the remainder of this paper. Section 3 contains general remarks on the parallelisation (which is analogous to that of geometric multigrid) and points out some critical aspects. In Section 4 we propose a relatively simple way of parallelization, aiming at a minimization of communication, in particular in AMG's setup phase. Although the proposed modifications involve the risk of influencing the behavior of the parallel code compared to its sequential analog (e.g. in terms of convergence speed), for relevant applications of significant size, this influence turned out to be very low. This is demonstrated in Section 5, where some characteristic results for complex 3D industrial test cases are presented (from CFD and oil-reservoir simulation). It is seen that the parallel approach scales reasonably well on distributed memory computers as long as the number of variables per processor is sufficiently large (some 20,000 to 30,000 variables, say).

Throughout this paper, we assume the reader to have some basic understanding of AMG as well as of the parallelization of standard geometric multigrid. For details on the particular AMG approach considered in this paper, we refer to [9, 10].

[1]This development has partly been funded by Computational Dynamics Ltd., London, which also provided most CFD test cases considered in this paper. The largest test case (full Mercedes-Benz E-Class model) has been provided by Daimler-Chrysler.

2 Summary of sequential AMG

Structurally, an AMG cycle to solve a (sparse) s.p.d. system of equations,

$$A^{(1)}u^{(1)} = f^{(1)} \quad \text{or} \quad \sum_{j\in\Omega^1} a_{ij}^{(1)}u_j^{(1)} = f_i^{(1)} \quad (i \in \Omega^1 = \{1,2,...,n\}) \ , \tag{1}$$

is completely analogous to a Galerkin-based geometric multigrid cycle. The only formal difference is that, in the context of AMG, we are not (necessarily) dealing with grids and subgrids but rather with sets of variables and subsets of variables (represented by their respective indices). That is, given some nested sequence

$$\Omega^1 \supset \Omega^2 \supset \cdots \supset \Omega^\ell \supset \cdots \supset \Omega^L$$

and related (full rank) interpolation operators $I_{\ell+1}^\ell$ $(\ell = 1,2,....,L-1)$, we define the coarse-level correction operators by

$$A^{(\ell+1)} = I_\ell^{\ell+1}A^{(\ell)}I_{\ell+1}^\ell \quad \text{where} \quad I_\ell^{\ell+1} = (I_{\ell+1}^\ell)^T \ .$$

In contrast to geometric multigrid, AMG's coarsening process (i.e. the recursive construction of $\Omega^{\ell+1}$ and $I_{\ell+1}^\ell$) is fully automatic, in the simplest case based merely on algebraic information contained in $A^{(\ell)}$ such as size and sign of its entries. Once these components have been constructed in a separate setup phase, normal multigrid cycling can be performed to iteratively solve (1). If all components are chosen properly, smoothing by plain Gauss-Seidel relaxation is sufficient, even if the given problem exhibits strong anisotropies or jumping coefficients.

In the following section, we summarize the main aspects of AMG's setup to an extent which is necessary in the context of this paper. For more details, we refer to [9, 10].

2.1 The setup phase

Assuming Ω^ℓ and $A^{(\ell)}$ to be known, we first construct a *C/F-splitting*, $\Omega^\ell = C^\ell \cup F^\ell$, with C^ℓ representing those variables which are to be contained in the next coarser level (*C-variables*) and F^ℓ being the complementary set (*F-variables*). Setting $\Omega^{\ell+1} = C^\ell$, we then define interpolation $e^{(\ell)} = I_{\ell+1}^\ell e^{(\ell+1)}$ of the form

$$e_i^{(\ell)} = \left(I_{\ell+1}^\ell e^{(\ell+1)}\right)_i = \begin{cases} e_i^{(\ell+1)} & \text{if } i \in C^\ell \\ \sum_{k\in P_i^\ell} w_{ik}^{(\ell)} e_k^{(\ell+1)} & \text{if } i \in F^\ell \ . \end{cases} \tag{2}$$

Here $P_i^\ell \subset C^\ell$ is called the set of *interpolatory variables* of level ℓ. (For simplicity, we identify the index $i \in \Omega^\ell$ with the i-th variable on level ℓ.)

Interpolation (2) can only be reasonable under two conditions. First, P_i^ℓ should correspond to a *small* subset of C-variables "near" i (to ensure sparsity of $A^{(\ell+1)}$). Second, P_i^ℓ should represent (a sufficient number of) those variables on which i *strongly depends* (or, is *strongly connected to*). This, in turn, imposes conditions on the C/F-splitting. In particular, it has to be such that there is a sufficiently strong F-to-C connectivity.

Hence, a reasonable (problem-dependent) definition of strength of connectivity is the basis of AMG's coarsening strategy. In practice, several ways of definition are considered.

For the purpose of this paper, we consider only the simplest one which is based directly on the matrix entries of $A^{(\ell)}$: a variable $i \in \Omega^\ell$ is called *strongly connected* to a variable $j \in \Omega^\ell$ if $a_{ij}^{(\ell)}$ is negative and if its absolute value exceeds a pre-scribed threshold. Setting

$$S_i^\ell = \{j \in \Omega^\ell \ : \ i \text{ strongly connected to } j\} \ ,$$

we then have $S_i^\ell \subset N_i^\ell$ where $N_i^\ell = \{j \in \Omega^\ell : j \neq i, a_{ij}^{(\ell)} \neq 0\}$ denotes the *direct neighborhood* of $i \in \Omega^\ell$.

The following three properties then characterize the C/F-splitting $\Omega^\ell = C^\ell \cup F^\ell$ used by AMG (called *standard coarsening*):

Property 1. The variables in C^ℓ are not strongly connected to each other. That is, for each pair $i, j \in C^\ell$, we have $i \notin S_j^\ell$ and $j \notin S_i^\ell$. (We note that, generally, the relation of being strongly connected is not symmetric.)

Property 2. For each $i \in F^\ell$ we have $i \in S_j^\ell$ for (at least) one $j \in C^\ell$.

Property 3. C^ℓ is a *maximal* set with the previous two properties.

Remark 2.1 For obvious reasons, we here exclude certain degenerated situations such as the occurrence of isolated equations (e.g. due to Dirichlet boundary conditions). Clearly, variables corresponding to isolated equations should always become F-variables. $\ll$

While Properties 1-2 allow rapid coarsening, Property 3 limits this coarsening somewhat, aiming at C/F-distributions which are most suitable for constructing interpolation. Geometrically speaking, F-variables tend to be "surrounded" by strongly connected C-variables (for an example, see Remark 2.2 below). However, while Property 2 is definitely required to be satisfied, it is neither necessary nor easily possible in general to *exactly* satisfy Properties 1 and 3. Fast greedy-like algorithms can be used to provide practically sufficient splittings. For instance, the splitting algorithm proposed in [7, 9] aims at locally maximizing the number of strong F-to-C connections in each greedy step (rather than the number of C-variables).

Once the splitting has been computed, interpolation (2) is defined based on the fact that an algebraically smooth error (i.e. an error which is slow to converge w.r.t. the smoothing process and, hence, has to be interpolated particularly well in order to ensure good AMG convergence) approximately satisfies

$$e_i^{(\ell)} = -\Big(\sum_{j \in N_i^\ell} a_{ij}^{(\ell)} e_j^{(\ell)}\Big)/a_{ii}^{(\ell)} \quad (i \in \Omega^\ell) \ . \tag{3}$$

For any $i \in F^\ell$, the simplest interpolation is obtained by just replacing N_i^ℓ in (3) by $P_i^\ell = S_i^\ell \cap C^\ell$ and using some rescaling to preserve row sums. (Note that Property 2 implies $P_i^\ell \neq \emptyset$.) Since this interpolation is only via "direct" couplings, it is called *direct interpolation*. Generally, more robust AMG convergence is obtained by employing what is called *standard interpolation* in [9]. This is defined similarly as the direct one, except that it is not only based on (3), but also on the corresponding relations in the direct neighborhood of i. Another possibility of enhancing robustness is to apply *Jacobi F-relaxation* to the

direct interpolation. In either case, compared to the direct interpolation, the "radius" of interpolation increases. We recall that, in order to be practical, such interpolation should be truncated (before computing the Galerkin operator), by ignoring small entries and re-scaling the remaining weights so that their total sum remains unchanged.

Remark 2.2 The importance of Property 3 is illustrated in Figure 2 which displays the coarsening obtained for the Laplace 9-point stencil

$$\frac{1}{3h^2}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}_h \tag{4}$$

on a regular square grid. Figure 2a depicts the result obtained by standard coarsening. In particular, the resulting (direct) interpolation corresponds to linear interpolation (which is known to be very efficient in this model case). If, in contrast to Property 3, we *minimise* the number of C-variables, we obtain the coarsening as indicated in Figure 2b. The resulting interpolation is piecewise constant which is known to cause very slow (and problem dependent) convergence. ≪

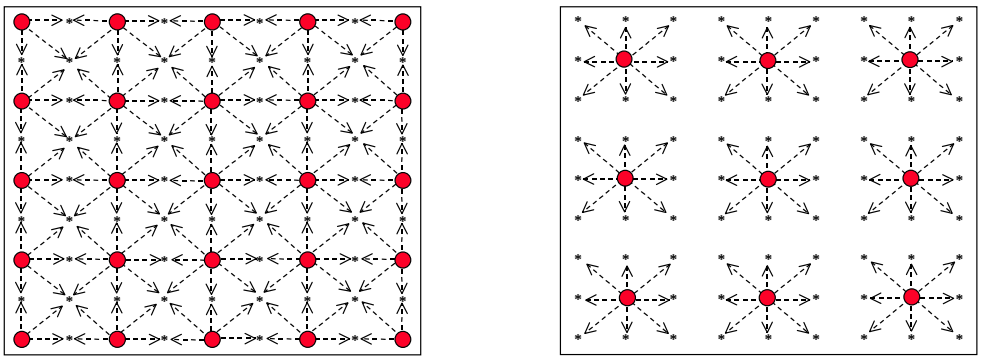

Figure 2: a) Standard coarsening for the 9-point stencil (4). b) Coarsening if the set of C-variables is *minimized* rather than maximized. The circles mark the C-variables. The arrows indicate from which C-variables an F-variable interpolates from.

Remark 2.3 If memory is an issue (as it is often in commercial environments), one may wish to employ a more rapid coarsening at the expense of a reduced convergence speed. A simple way to achieve this is by applying the above splitting strategy a second time, now restricted to the set of C-variables which resulted from the original splitting. This just requires strong connectivity between these C-variables to be reasonably defined, which is most naturally done by considering connections via "paths of length two". Using such *aggressive coarsening*, interpolation can most easily be performed in (at most three) passes: for instance, in each pass either a direct interpolation is used or interpolation formulas from previous passes are exploited, whatever is possible. This type of interpolation is called *multi-pass interpolation*. ≪

2.2 Sequential performance

As an example, we consider the application of AMG to solve the pressure-correction equation which occurs as the most time-consuming part of the *segregated solution approach* to solve the Navier-Stokes equations. The discretization is based on finite-volumes. The concrete application is the industrial computation of the exterior flow over a complete Mercedes-Benz E-Class model (see Figure 3). The underlying mesh consists of 10 million cells and is highly complex; in particular, it includes all the modelling details shown in Figure 1. (Due to memory restrictions, our test runs refer to a reduced mesh consisting of 2.23 million cells.)

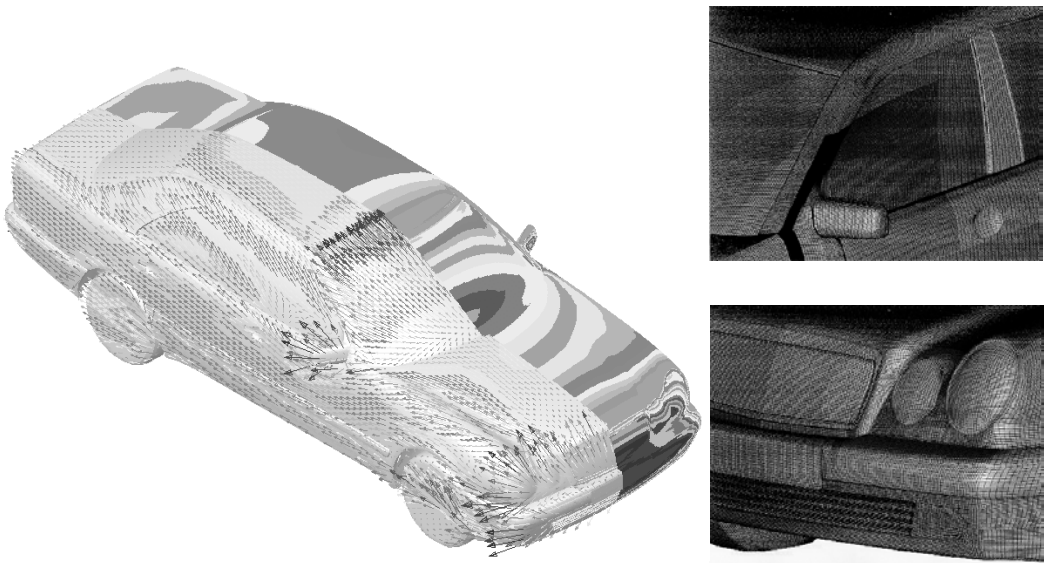

Figure 3: Model of a Mercedes-Benz E-Class

For such large applications, low-memory AMG variants are of primary industrial interest, even if the reduction of memory requirement is at the expense of a (limited) increase of the computational time. Compared to standard one-level solvers, a memory overhead of some tens of percents is acceptable. In any case, however, the *operator complexity*,

$$c_A = \sum_{\ell=1}^{L} |A^{(\ell)}|/|A^{(1)}| \,, \tag{5}$$

must not be larger than 2.0, say. (Here, $|A^{(\ell)}|$ denotes the number of non-zero entries contained in $A^{(\ell)}$. Note that c_A reflects the fill-in produced on the coarser levels and is directly related to the overall memory requirement.) Consequently, we employ a coarsening strategy which has turned out to be industrially most relevant for all our applications: Standard coarsening and interpolation are used on all levels but the first one. On the first level, aggressive coarsening (combined with multi-pass interpolation, cf. Remark 2.3) is employed. AMG's coarsening is terminated when the coarsest level contains some 40 variables.

Remark 2.4 Originally, classical AMG has been designed to be used stand-alone. However, in practice, AMG's efficiency and robustness can substantially be further enhanced by using it as a pre-conditioner rather than stand-alone. This is true, in particular, in connection with aggressive coarsening. In this paper, we therefore always use AMG as a pre-conditioner for conjugate gradient. (For various comparisons of AMG used with and without conjugate gradient, see [9].) $\ll$

AMG should not be regarded as a competitor of geometric multigrid but rather as an efficient alternative to popular one-level methods. Therefore, Figure 4 compares AMG's V-cycle convergence history with that of ILU(0) pre-conditioned conjugate gradient (ILU/cg). One step of plain Gauss-Seidel CF-relaxation (i.e., C-variables are relaxed first), is used for pre- and post-smoothing on each AMG level. The coarsest-level equations are solved directly. The results in the figure refer to the solution of the pressure-correction equation at one particular time step taken from a normal production run. In order to reduce the residual by nine orders of magnitude, AMG and ILU/cg require 25 and 1151 iterations, respectively. In terms of total computational time, AMG is about 19 times faster. (This corresponds to the fact that, roughly, one AMG iteration costs about the same as two ILU/cg iterations and AMG's setup cost amounts to the cost of about 5 AMG iterations.) The operator complexity (5) is very satisfactory too, namely, $c_A = 1.46$.

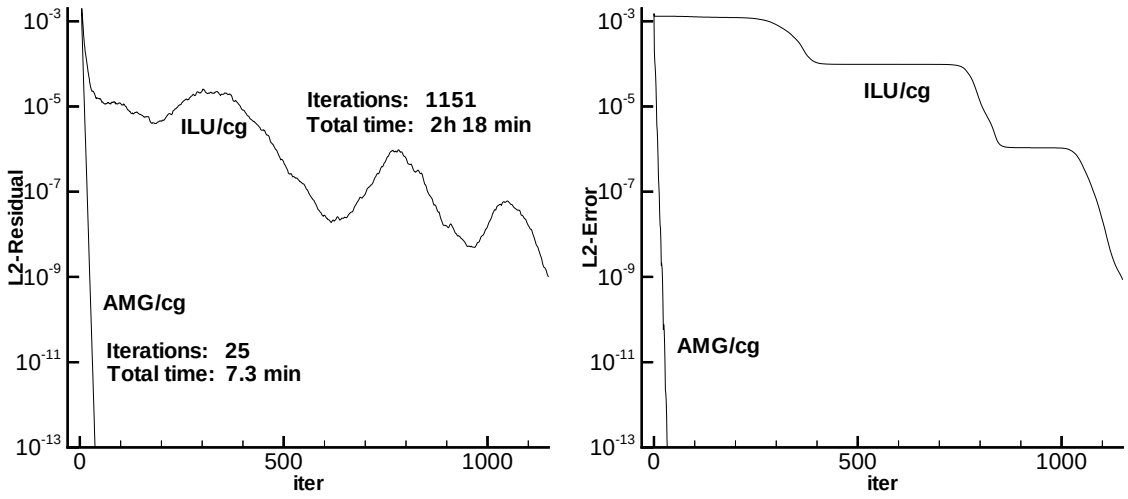

Figure 4: Convergence histories of AMG and ILU/cg in terms of a) the residual, b) the error (both measured in the ℓ_2-norm)

The above results reflect the typical performance of AMG in geometrically complex applications of the type and size considered here. Since AMG's convergence is virtually independent of the problem size, compared to standard one-level methods, the gain by employing AMG significantly grows with increasing problem size. This is confirmed by extensive comparisons as found, for example, in [9, 10]. These comparisons include also applications which exhibit strong anisotropies and jumping coefficients such as those arizing in oil reservoir simulation or in electromagnetics.

Remark 2.5 Particularly in steady-state computations, the pressure-correction equation usually needs to be solved only with a low accuracy of only one digit, say. One might expect that then the use of AMG is an "overkill" (in particular, because of the high setup cost involved) and simple one-level methods would become more efficient. Indeed, this seems to be true if one looks at the convergence histories depicted in Figure 4a: Only very few ILU/cg iterations are required to reduce the residual by one digit.

Generally, however, one has to be very careful in drawing conclusions from small residual reductions to corresponding reductions in the error. Indeed, if one compares AMG with ILU/cg on the basis of *error* rather than residual reductions, the picture looks completely different, see Figure 4b. In fact, while AMG needs only 3 iterations to reduce the error by one digit, ILU/cg requires about 400 iterations. Consequently, according to the above mentioned rough estimate of the computational cost of AMG versus ILU/cg, AMG is approximately 25 times faster than ILU/cg in reducing the error by one digit. That is, the benefit of using AMG is even *higher* than before! This advantageous behavior of AMG in terms of error reduction is related to its property to *globally* reduce errors much more effectively than any one-level method does.

Nevertheless, in computing low-accuracy approximations, AMG's setup cost becomes quite substantial in the sense that most of AMG's total computing time may be spent during setup. Incidentally, however, in situations requiring only low accuray computations, typically chains of (often many hundreds or thousands of) problems have to be solved for which the underlying matrices usually change only slowly from one step to the next. Consequently, for many steps, the complete setup can be "frozen" (or updated by just re-computing the Galerkin operators) which enhances the efficiency of AMG further. $\ll$

3 General parallelization approach

Analogous to geometric multigrid, our parallelization approach is based on a *partitioning of variables*. That is, given P processors, we will (disjointly) distribute the variables Ω^ℓ on each of the recursively constructed AMG levels across these processors (except for the very coarse levels, see Section 3.3),

$$\Omega^\ell = \bigcup_{p=1}^{P} \Omega_p^\ell \, ,$$

where Ω_p^ℓ denotes the subset of variables assigned to processor p. Accordingly, all (complete) rows of $A^{(\ell)}$ and $I_{\ell+1}^\ell$ belonging to $i \in \Omega_p^\ell$ are assigned to processor p. We denote the corresponding sets of matrix rows by $A^{(\ell)}[p]$ and $I_{\ell+1}^\ell[p]$, respectively. For $\ell = 1$, we assume a reasonable load-balanced partitioning to be provided by the user (e.g. based on one of the available partitioning tool such as Metis). For $\ell > 1$, we assign all variables to the same processor to which their finest-level analog had been assigned,

$$\Omega_p^\ell = \Omega_p^1 \cap \Omega^\ell \, . \tag{6}$$

During AMG's *setup phase*, we assume that the C/F-splittings $\Omega^\ell = C^\ell \cup F^\ell$ are recursively performed in parallel, that is, each processor p constructs

$$\Omega_p^\ell = C_p^\ell \cup F_p^\ell \tag{7}$$

independently of the other processors (see Section 3.1). All rows of $I_{\ell+1}^{\ell}[p]$ and $A^{(\ell+1)}[p]$ are then exclusively assembled (and stored) by processor p. During AMG's *solution phase*, we proceed as in standard geometric multigrid assuming that processor p needs access only to such data of a neighboring processor q which corresponds to variables inside the standard *overlap area* (cf. shaded area in Figure 5)

$$\partial\Omega_{q,p}^{\ell} = \{j \in \Omega_q^{\ell} : a_{ij}^{(\ell)} \neq 0 \text{ for some } i \in \Omega_p^{\ell}\} \ . \tag{8}$$

Note that $\partial\Omega_{q,p}^{\ell}$ is the algebraic analog of a geometric overlap of "width 1". Below, we also refer to overlap areas of width 2 or more. The meaning of these terms is obvious.

Remark 3.1 Clearly, the "static" coarse-level partitioning (6) is a source of potential *load imbalance*. First, AMG's coarsening strategy does not ensure that Ω_p^{ℓ} contains approximately the same number of variables for each p. Second, the total arithmetic load on level ℓ, roughly given by $|A^{(\ell)}[p]|$, can vary among the different processors: The Galerkin matrices, though sparse on the average, generally have sparsity patterns which may change substantially between variables. A load balance optimization on level ℓ would require a re-mapping of data. However, for all industrially relevant applications considered so far, very satisfactory parallel efficiencies were obtained even without any particular optimization. In fact, the substantial communication overhead involved in a re-mapping would (most probably) offset the resulting gain. $\ll$

An efficient parallelization based on the above general approach requires certain modifications to the sequential AMG. Critical issues are briefly discussed in the following.

3.1 Critical issues regarding the setup phase

For the purpose of the following discussion, let us assume that each processor p in parallel constructs its local splitting (7) based on Properties 1-3 of Section 2.1, applied to Ω_p^{ℓ} rather than Ω^{ℓ}. Since this essentially means that all strong connections across processor interfaces are ignored, the resulting global splitting will be different from the one constructed by sequential AMG. Most importantly, due to the lack of "synchronization", coarsening near processor interfaces will be somewhat coincidental.

However, *standard* interpolation is fairly insensitive to such modifications of the splitting since, in interpolating to a particular F-variable i, the neighborhood of i is also taken into account. In contrast to this, the simple *direct* interpolation may, in an unpredictable manner, become one-sided near interfaces, resulting in a higher risk of a locally reduced "accuracy" of interpolation. For very regular model problems – for which the sequential AMG often tends to produce a (geometrically) optimal coarsening and interpolation – this may indeed cause a reduced convergence speed. In general, however, for geometrically complex applications as we have in mind here (where interpolation is not optimal anyway), the dependence of convergence on the splitting near interfaces has been seen to be fairly low, in particular, if AMG is used as a pre-conditioner (cf. Remark 2.4).

After having finished its splitting, processor p can, in principle, assemble $I_{\ell+1}^{\ell}[p]$ and $A^{(\ell+1)}[p]$ completely in parallel. However, quite some communication is required (for an illustration of the situation, see Figure 5).

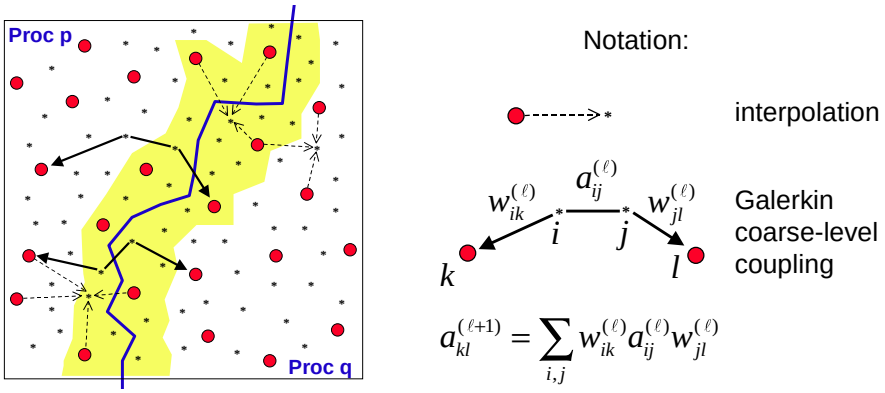

Figure 5: Illustration of (direct) interpolation and coarse-level couplings near processor interfaces. C- and F-variables are marked by circles and asterisks, respectivly. The shaded areas denote the standard overlaps of processor p and q.

The assembling of $I_{\ell+1}^{\ell}[p]$ is easy if only *direct* interpolation is to be used since processor p then just needs to additionally know the C/F-distribution inside all of its overlap areas $\partial\Omega_{q,p}^{\ell}$. However, *standard* interpolation would require the additional knowledge of all rows of $A^{(\ell)}[q]$ belonging to variables in the overlap areas $\partial\Omega_{q,p}^{\ell}$ and of the C/F-distribution of variables inside corresponding overlap areas of width 2. The situation gets considerably more involved if *multi-pass interpolation* or *F-relaxation of interpolation* are to be used. The final interpolation is then defined in stages and, after each stage, intermediate interpolation rows, belonging to variables near the processor interfaces, have to be exchanged between neighboring processors. The final interpolation formulas near interfaces may deeply penetrate into the area of neighboring processors.

Once $I_{\ell+1}^{\ell}[p]$ has been assembled, the computation of $A^{(\ell+1)}[p]$ requires further additional information from the neighboring processors q. The required amount of information may become substantial, even if interpolation is based only on direct connections. In the latter case, one easily confirms that, first, processor p needs to know all rows of $A^{(\ell)}[q]$ which belong to variables in the overlap areas $\partial\Omega_{q,p}^{\ell}$. Second, interpolation rows of $I_{\ell+1}^{\ell}[q]$ need to be known for all $i \in F_q^{\ell}$ in an overlap area of width 2. Finally, the C/F-distribution has to be known even within an overlap area of width 3. The computation of $A^{(\ell+1)}[p]$ gets even much more complicated if the interpolation is based on one of the more robust approaches: rows of $A^{(\ell)}[q]$ and $I_{\ell+1}^{\ell}[q]$ have to be communicated to processor p within several layers of overlaps.

Even without going into more detail, it becomes clear that the communication involved in computing $I_{\ell+1}^{\ell}[p]$ and $A^{(\ell+1)}[p]$, in general, may become unacceptably high, causing a fairly limited scalability of the setup phase. A reasonable scalability of this phase, however, is highly important since its share of the total solution cost is significant: on serial computers, the setup cost typically amounts to the cost of several AMG cycles. The goal of the modifications introduced in Section 4 is to drastically reduce this communication.

3.2 Critical issues regarding the solution phase

As mentioned before, smoothing is done by Gauss-Seidel CF-relaxation. In the parallel context, each processor performs CF-relaxation independently of the other processors, using frozen values in the overlap areas (8). By an exchange of data, the overlap areas are refreshed after each partial step of relaxation. In practice, this change of the smoothing process has no essential effect on the overall convergence.

The remaing parts of an AMG cycle can be performed in parallel just like in geometric multigrid. However, apart from the fact that $\partial\Omega^{\ell}_{q,p}$ typically represents fairly complicated irregular sets of variables, interpolation and restriction generally need *their own overlap areas*. For instance, in order to perform interpolation, processor p needs to know all corrections inside the overlap areas $\partial C^{\ell}_{q,p} \subset C^{\ell}_{q}$, defined as

$$\partial C^{\ell}_{q,p} = \{k \in C^{\ell}_{q} : w^{(\ell)}_{ik} \neq 0 \text{ for some } i \in F^{\ell}_{p}\} \ .$$

Similarly, the fine-to-coarse residual transfer requires the residuals to be known in a third kind of overlap,

$$\partial F^{\ell}_{q,p} = \{i \in F^{\ell}_{q} : w^{(\ell)}_{ik} \neq 0 \text{ for some } k \in C^{\ell}_{p}\} \ .$$

Finally, all non-zero weights $w^{(\ell)}_{ik}$ $(i \in \partial F^{\ell}_{q,p}, k \in C^{\ell}_{p})$ have to be known to processor p.

The most simple way to restrict communications in a cycle to data exchanges via the standard overlaps $\partial\Omega^{\ell}_{q,p}$ only (as requested at the beginning of this Section 3), is by not interpolating across processor interfaces (see Section 4).

3.3 Agglomeration on the coarsest levels

A reasonable treatment of the coarsest AMG levels is of particular importance. In parallel geometric multigrid, applied to relatively simple grids, one usually either stops coarsening whenever the number of variables per processor becomes too small (and uses some iterative method to approximately solve the corresponding correction equations) or one continues coarsening, allowing an increasing number of processors to become idle. Although, in principle, we can proceed similarly in parallel AMG, this is not necessarily the best strategy.

Generally, towards coarser levels, the parallel splitting causes a slower reduction of the number of variables than its sequential analog. This is particularly true for the modifications proposed in our final parallel splitting algorithm (see Section 4) which enforces a slower reduction of variables near processor interfaces. Consequently, at a certain coarse level ℓ_0, it usually becomes inefficient to continue parallel coarsening involving all processors. Instead, it is generally more efficient to perform some *agglomeration process* for $\ell \geq \ell_0$. That is, coarsening continues, but more and more (neighboring) subdomains are joined and treated by fewer and fewer processors. If only one processor is left, we solve the respective equations by either a direct solver or by sequential AMG. Note that the selection of ℓ_0 is crucial. If agglomeration starts too early, the resulting coarse-level load imbalance may substantially influence the overall load balance; if it starts too late, the work on the coarsest levels may become unnecessarily high. A reasonable switching criterion has to be based on a comparison of the interior work load of the processors with that near their interfaces.

To simply stop coarsening at level ℓ_0 (rather than explicitly performing agglomeration) is no real alternative: although the number of variables *per processor* will typically be relatively small (some 30-50, say), the *total* number of variables will still be fairly large. In complex applications, the convergence properties of parallel versions of standard iterative methods (e.g. based on relaxation or ILU/cg) to approximately solve the level ℓ_0 correction equations, are hardly predictable. It well happens that the effort invested on level ℓ_0 (with a particularly bad computation/communication ratio) becomes a substantial fraction of the total work per cycle, substantially reducing the parallel efficiency.

4 Parallelization by subdomain blocking

In order to reduce communication, we have to prevent the recursively defined interpolation operators – and through this the Galerkin operators – from penetrating deeply into neighboring processor areas. Formally, this can easily be achieved by suitably modifying the way of coarsening and interpolation near processor interfaces and leaving the rest of AMG essentially unchanged. Clearly, for a modification to be practical, it should not seriously affect convergence. Equally important, the operator complexity (5) should be comparable in the sequential and the parallel algorithm.

4.1 Full subdomain blocking

In the most radical approach, we recursively force *all* variables near processor interfaces to become C-variables ("full subdomain blocking"). To be more specific, we define the *boundary layer* and the *interior* of processor p on level ℓ by

$$\partial\Omega_p^\ell = \{i \in \Omega_p^\ell : a_{ij}^{(\ell)} \neq 0 \text{ for some } j \in \Omega_q^\ell,\, q \neq p\} = \bigcup_{q \neq p} \partial\Omega_{p,q}^\ell \quad \text{and} \quad \mathring{\Omega}_p^\ell = \Omega_p^\ell \setminus \partial\Omega_p^\ell \,,$$

respectively. Using this notation, processor p defines its local splitting (7) independently of the other processors in two steps (cf. Figure 7a):

Step 1. All boundary layer variables $i \in \partial\Omega_p^\ell$ are forced to become C-variables.

Step 2. A C/F-splitting process, based on properties analogous to Properties 1-3 in Section 2.1, is applied to $\mathring{\Omega}_p^\ell$. More precisely, the final splitting (7) should satisfy

Property 1. The variables in $\mathring{\Omega}_p^\ell \cap C_p^\ell$ are not strongly connected to each other. Moreover, variables $i \in \mathring{\Omega}_p^\ell \cap C_p^\ell$ and $j \in \partial\Omega_p^\ell \cap C_p^\ell$ are not strongly connected to each other.

Property 2. Each $i \in \mathring{\Omega}_p^\ell \cap F_p^\ell$ is strongly connected to (at least) one $j \in C_p^\ell$.

Property 3. $\mathring{\Omega}_p^\ell \cap C_p^\ell$ is a maximal set with the previous two properties.

Remark 4.1 According to Property 1, adjacent to the boundary layer of each processor, there will be a strip consisting only of F-variables (cf. dashed line in Figure 7a). $\ll$

Remark 4.2 If requested, the resulting set of interior C-variables, $\overset{\circ}{\Omega}{}_p^\ell \cap C_p^\ell$, can a posteriori be reduced by means of aggressive coarsening. This is completely analogous to the sequential AMG (see Remark 2.3). $\ll$

Using full subdomain blocking, all interpolation approaches mentioned in Section 2.1 become local to each processor and, trivially, the assembling of $I_{\ell+1}^\ell[p]$ requires no communication. Obviously, the Galerkin operators do not penetrate into the area of neighboring processors and the assembling of $A^{(\ell+1)}[p]$ does not require any communication either. In fact, all boundary layers stay the same on each level, $\partial\Omega_p^{\ell+1} = \partial\Omega_p^\ell$, and all cross-processor couplings remain unchanged: $a_{ij}^{(\ell+1)} = a_{ij}^{(\ell)}$ for all $i \in \partial\Omega_p^\ell$ and $j \in \partial\Omega_q^\ell$ if $p \neq q$.

The main advantage of the full blocking approach is its simplicity. First, as pointed out above, essentially no communication is required in the setup phase (except, of course, for the agglomeration process, see Section 3.3). Second, data exchanges during AMG's solution phase need to be performed only via the standard overlap areas $\partial\Omega_{q,p}^\ell$ as requested at the beginning of Section 3. Moreover, for applications as considered in this paper, the parallel code typically converges faster than its sequential analog. Heuristically, this is because the number of C-variables used is much larger in the parallel code. Nevertheless, there are serious drawbacks which, generally, make the full blocking approach impractical.

First, since the overlap areas are the same on all levels, the amount of data which has to be communicated during the solution phase, does not decrease for increasing ℓ. However, since the total number of levels depends only logarithmically on the number of fine-level variables, this by itself might still be acceptable in practice. Second, and much more importantly, the operator complexity (5) strongly increases compared to the sequential AMG. This is for two reasons: On the one hand, due to the slower coarsening near processor interfaces, the coarse-level Galerkin matrices contain substantially more rows. On the other hand, the number of non-vanishing entries in matrix rows corresponding to variables *inside the boundary layers* grows dramatically with increasing ℓ.

What the latter really means in terms of an increase of the *average* fill-in, to some extent depends on the concrete geometrical situation, in particular the shape and size of the processor interfaces. To illustrate this, Figure 6 depicts the size of $|A^{(\ell)}|$ as a function of ℓ for the sequential AMG and its parallel analog on 8 processors, based on full subdomain blocking. The top figure refers to the cooling jacket of a four-cylinder engine. Compared to the sequential code, $|A^{(\ell)}|$ is reduced much slower for increasing ℓ; for $\ell \geq 4$ the matrix size does hardly decrease further. However, this geometry still represents a relatively good case since the use of a standard mesh partitioning tool (such as Metis) here typically results in relatively simple and small processor interfaces. A much more complex case is shown in the bottom part of Figure 6. Here, for $\ell \geq 2$ the matrix size is even slightly increasing!

Remark 4.3 Figure 6 refers to the case that *no agglomeration* is assumed to be performed. In practice, however, we perform successive agglomeration as soon as the interior work load becomes comparable to the work load in the boundary layer (see Section 3.3). This substantially reduces the negative effects of full domain blocking: After an agglomeration of the area of neighboring processors to a smaller number of processors, many variables which have been boundary layer variables before, are now interior variables and further coarsening becomes effective. In cases such as the cooling jacket, we indeed obtain

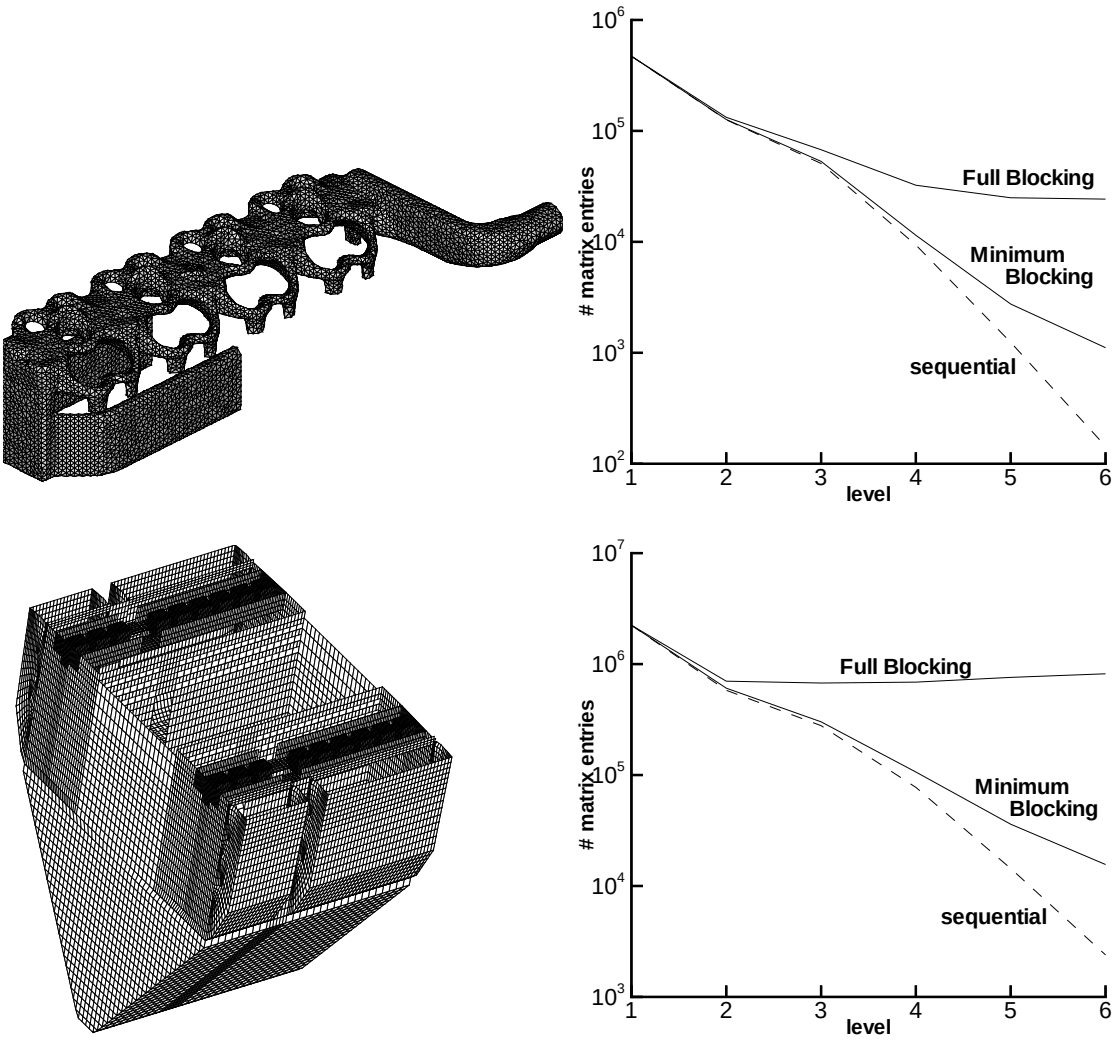

Figure 6: Global matrix size $|A^{(\ell)}|$ as a function of levels (*without agglomeration* on 8 processors). **Top:** Cooling jacket of a four-cylinder engine (100,000 mesh cells). **Bottom:** Coal furnace model (325,000 mesh cells); only some particular part is shown here.

a reasonably efficient method this way. Geometrically more complex cases, however, would generally require agglomeration to start too early (in the coal furnace case already at level 2!), causing an unacceptably high load imbalance. ≪

4.2 Minimum subdomain blocking

We can essentially maintain the simplicity of the previous parallelization approach even without forcing all boundary layer variables to stay in the coarse levels. In fact, we just need to restrict interpolation at boundary layer variables to be performed strictly *from within that layer* ("minimum subdomain blocking"). More specifically, we use the same

parallel C/F-splitting process as before except that Step 1 is replaced by

Step 1′. Split the boundary layer into C- and F-variables based on Properties 1-3 in Section 2.1, applied to $\partial\Omega_p^\ell$ (rather than Ω^ℓ) in a straightforward way.

Remark 4.4 This splitting ensures that each boundary layer F-variable has (at least) one strong connection to a boundary layer C-variable. Consequently, boundary layer variables which do not have any strong connections *inside this layer*, will become C-variables. In particular, if a boundary layer is in the direction of *weak* connectivity in a strongly anisotropic problem, the boundary layer will (and should) not be coarsened. ≪

Remark 4.5 As in the full blocking case, the set of interior C-variables, $\mathring{\Omega}_p^\ell \cap C_p^\ell$, can a posteriori be reduced by means of aggressive coarsening. In principle, aggressive coarsening might also be used inside each boundary layer. In this paper, however, we only consider standard coarsening of boundary layers. ≪

Using minimum subdomain blocking, the parallelization of the remaining parts of AMG becomes similarly straightforward as before if we restrict interpolation near processor interfaces so that it has the following properties (for an illustration, see Figure 7b):

1. Each boundary layer F-variable $i \in \partial\Omega_p^\ell \cap F_p^\ell$ interpolates only from bounday layer C-variables $j \in \partial\Omega_p^\ell \cap C_p^\ell$ (cf. Remark 4.4).

2. Each interior F-variable $i \in \mathring{\Omega}_p^\ell \cap F_p^\ell$ interpolates only from local C-variables, $j \in C_p^\ell$.

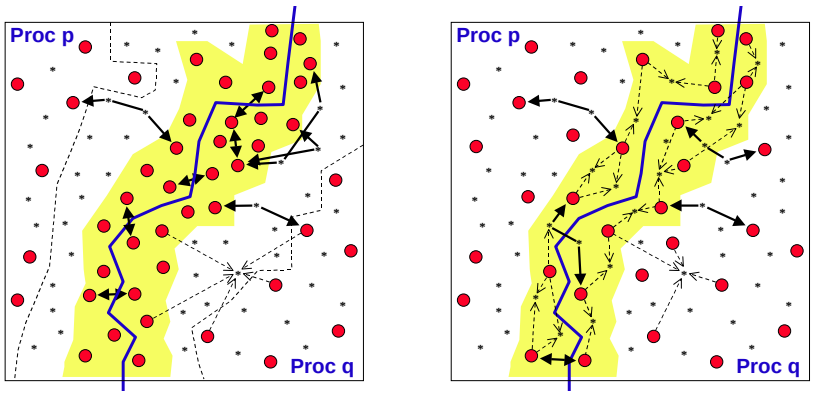

Figure 7: Illustration of parallel coarsening by a) full b) minimum subdomain blocking. The notation is as in Figure 5.

By slightly modifying the general approaches of how to define interpolation (summarized in Section 2.1 and described in detail in [9]), the above properties can easily be satisfied. In the following, we assume this to be realised in the most straightforward way. For instance, in setting up the boundary layer interpolation, we simply take only strong connections inside each boundary layer into account; all "cross-connections" (i.e. all connections to variables outside the boundary layer) are considered as weak. Interpolation

$I_{\ell+1}^{\ell}[p]$ then is not only completely local to processor p but its assembling does also not require any communication. In Section 5, we will see that, for geometrically complex applications, these modifications of interpolation have only a low influence on AMG's convergence.

The main practical consequence of the above two properties of interpolation is that the Galerkin operator $A^{(\ell+1)}$ does not penetrate into the interior area of neighboring processors. In fact, the boundary layers are nested, $\partial\Omega_p^{\ell+1} \subseteq \partial\Omega_p^{\ell}$. However, unlike before, the explicit assembling of $A^{(\ell+1)}[p]$ now requires some communication, namely, processor p needs to know the interpolation formulas of all neighboring processors q belonging to variables in $\Omega_{q,p}^{\ell} \cap F_q^{\ell}$. Apart from this, no communication is required in the parallel setup phase. As in the case of full blocking, AMG's solution phase only requires the typical multigrid communication via the standard overlaps $\partial\Omega_{q,p}^{\ell}$.

Summarizing, the parallelization by minimum blocking is still fairly simple. Compared to the full blocking approach, some limited communication is required in the setup phase. On the other hand, the amount of data to be communicated in the solution phase is substantially lower. Most importantly, however, the (global) operator complexity c_A is dramatically reduced. A comparison between the full and the minimum blocking approach, as typically observed in our applications, is depicted in Figure 6. We recall that the figure refers to the case that no agglomeration is done on coarser levels (see Remark 4.3). In practice, however, we always do perform agglomeration which reduces c_A even further.

5 Results

In this section we apply parallel AMG, based on minimum subdomain blocking, to some industrial test cases (using partitionings based on Metis). Comparisons are made with the most relevant sequential AMG strategy as described in Section 2.2. The parallel strategy is analogous, except that no aggressive coarsening is performed inside boundary layers. Agglomeration is performed by joining areas of pairs of neighboring processors. For interpreting the below results, we briefly summarize the potential reasons for a limitation of the parallel efficiency. (We here do not mention well-known limitations which are common to all multi-level approaches, for instance, caused by bad computation/communication ratios on coarser levels.)

- **Reduced convergence.** Due to the simplified boundary layer interpolation, the number of parallel iterations may depend on the number of processors as well as the concrete partitioning. However, for all test cases and within the processor ranges used (up to 64), the number of iterations increased (at most) by 10-20% only.

- **Increased computational work.** For a fixed problem and increasing P, the (global) operator complexity (5) moderately increases. This is mainly because we do not perform aggressive coarsening in the boundary layers (see above) and the Galerkin matrix rows corresponding to variables near processor interfaces have a tendency to be less sparse than the average. This increase has turned out to be not significant as long as the given application is reasonably large (see below).

- **Load imbalance.** A first source of load imbalance is the simple partitioning we use on the coarser levels (see Remark 3.1). Second, on the coarsest levels, a sub-

stantial load imbalance is introduced by the agglomeration process (see Section 3.3). Whether or not this will cause a significant *overall* load imbalance, essentially depends on the level where agglomeration actually starts in a given application. Clearly, for a fixed mesh, agglomeration will start the earlier, the more processors are employed. In any case, agglomeration will first affect the parallel efficiency of the setup phase. This is because the setup phase requires a re-distribution of complete (relatively dense) matrices on the respective levels. In contrast to this, cycling requires only a re-distribution of vectors (solution and right hand sides).

In practice, load imbalance turns out to be the most crucial aspect in terms of limiting the parallel performance. But even without optimizing load balancing in any way, very satisfactory parallel efficiencies are obtained if the problem size per processor is sufficiently large. Depending on the geometrical complexity and size of the processor interfaces, the number of variables per processor should be about 20,000 to 30,000 in order to achieve parallel efficiencies of approximately 50% or more. In an industrial context, this is a very reasonable requirement.

Figures 8 and 9 display typical speedup curves obtained for three complex 3D meshes of fixed size used in industrial CFD simulation (cf. Section 2.2). Results are given separately for the setup phase, single cycles and total run time (the latter includes a potential increase of the number of iterations compared to the sequential AMG). In all cases, a residual reduction by 7 orders of magnitude is required.

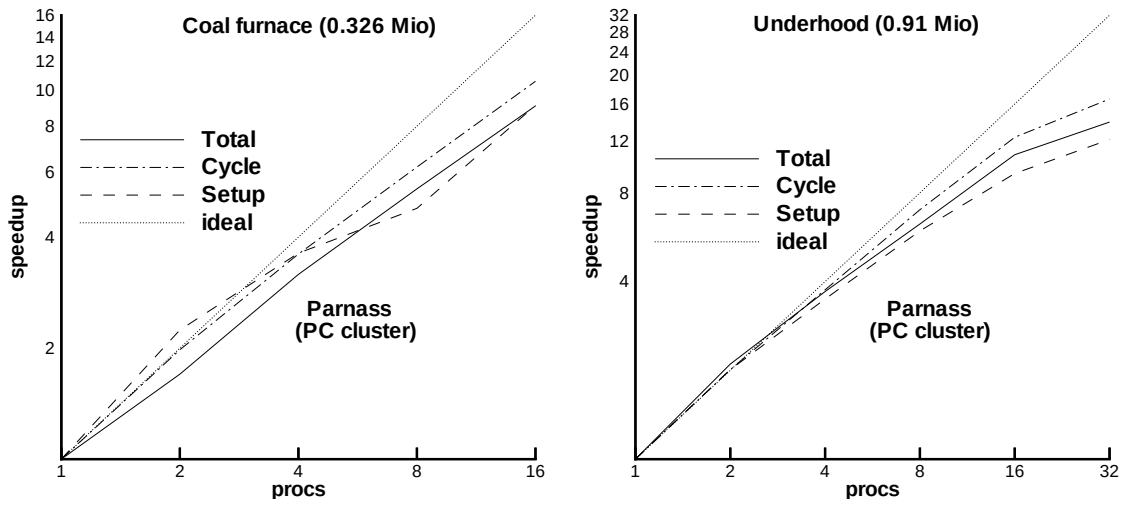

Figure 8: Speedups obtained on PC cluster: a) Coal Furnace, b) Underhood

Figure 8 refers to a PC cluster[2] consisting of Pentium II processors (400 MHz), interconnected via Myrinet. Results are shown for a Coal Furnace and an Underhood model (the geometries are displayed in Figures 6 and 1, respectively), using up to 32 processors. For the relatively small Coal Furnace case (326,000 cells), the speedup per cycle is nearly 11 on 16 processors. The fact that the total speedup is somewhat lower (approximately 9) indicates that the total number of iterations has slightly increased compared to the

[2]We thank M. Griebel and his group for giving access to the PC cluster at the University of Bonn.

sequential AMG. Relative to the cycle speedup, the setup phase also scales satisfactorily. The larger Underhood case (910,000 cells) behaves very similar except that there is a kink in all speedup curves if P is increased from 16 to 32. This is obviously caused by some kind of change in the behavior of the interconnection network: This kink does not occur on the NEC Cenju4, see Figure 9a. We point out that this test case has extremely complex processor interfaces, much more complicated than the Coal Furnace case.

In Section 2.2 we have discussed the performance of sequential AMG in case of a fairly large and very complex test case, a complete E-Class Mercedes-Benz model (2,23 million cells, see Figure 3). The corresponding parallel results are shown in Figure 9b. We see that for this large problem the parallel efficiency still exceeds 50% on 64 processors. (Note that the speedup depicted here is relative to the 4-processor run.) For eight processors the total speedup is even higher than the individual speedups, indicating that a lower number of iterations is required on 8 than on 4 processors.

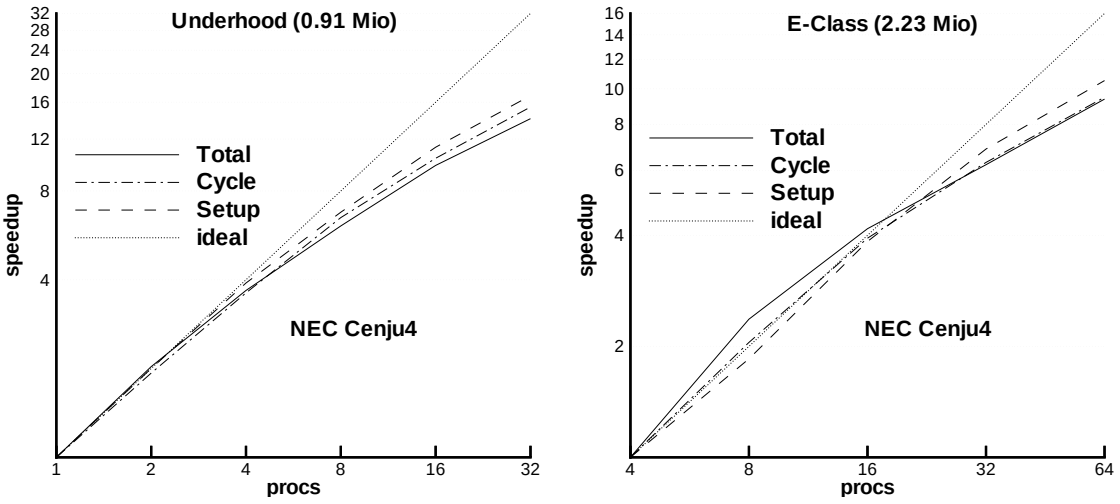

Figure 9: Speedups obtained on NEC Cenju4: a) Underhood, b) E-Class

Practically more interesting than speedups are wall times. Figure 10 shows corresponding times for both the Underhood and the E-Class test cases. In both cases, the PC cluster performs best and is roughly twice as fast as the NEC Cenju4 (using standard compiler options for optimization). Figure 10b shows that the performance of the IBM SP2 (thin nodes) is somewhere in between. Although a single SP2 node is faster than a single node of the Cenju4, there is a cross-over point of the total performance for 32 processors. This indicates that the interconnection network of the Cenju4 is more advantageous for AMG.

As a last test case, we consider a problem of the type typically solved in a commercial *oil reservoir simulation* code. Compared to CFD, geometries in oil reservoir simulation are typically much simpler. However, the underlying problems have strongly anisotropic and discontinuous coefficients. Figure 12a refers to a model with 1.16 million cells (the coefficients discontinuously change by 4 orders of magnitude in a fashion as outlined in Figure 11)[3], using the same AMG approach as before. We first observe that the cycle speedup is very good, namely, approximately 26 on 32 processors. The parallel efficiency

[3]This test case has been provided by StreamSim Technologies, Inc.

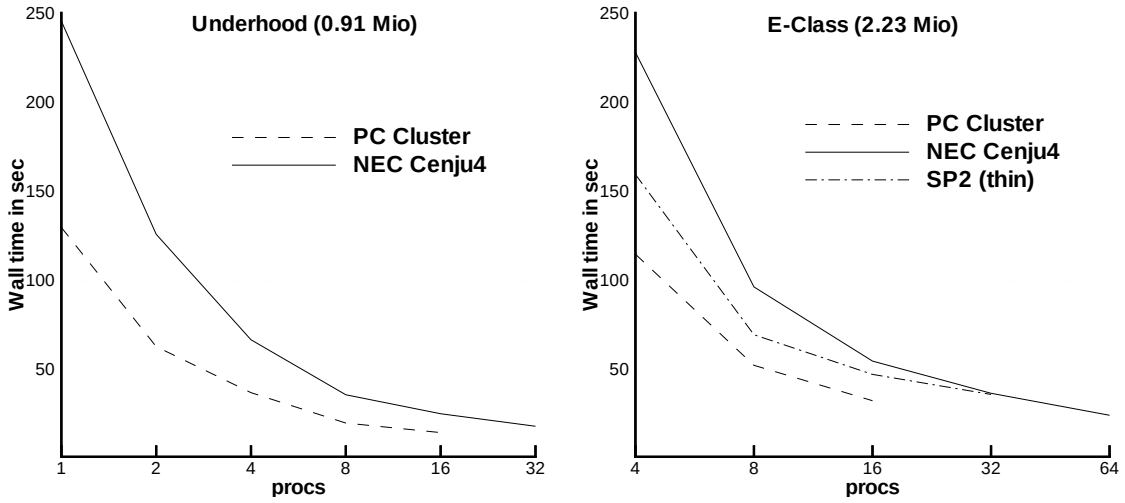

Figure 10: Wall times for a) Underhood, b) E-Class test case

of the setup, however, drops if we increase P from 16 to 32. This is essentially caused by the agglomeration (more precisely, the related communication involved in the re-distribution of data) which, as mentioned above, affects the parallel efficiency of the setup earlier than that of the cycle. The total parallel efficiency, however, is still 62.5%.

One might expect that AMG without agglomeration (using an iterative solver on the coarsest level instead) is more efficient. However, as already pointed out in Section 3.3, this is not necessarily the case. This is demonstrated in Figure 12b where we iteratively solve the coarsest-level equations by conjugate gradient, pre-conditioned by local sequential AMG. As expected, the parallel performance of the setup phase substantially increases. On the other hand, however, the performance of the cycle substantially decreases due to the high computational and communication time spent on the coarsest level. Altogether, the total parallel efficiency decreases.

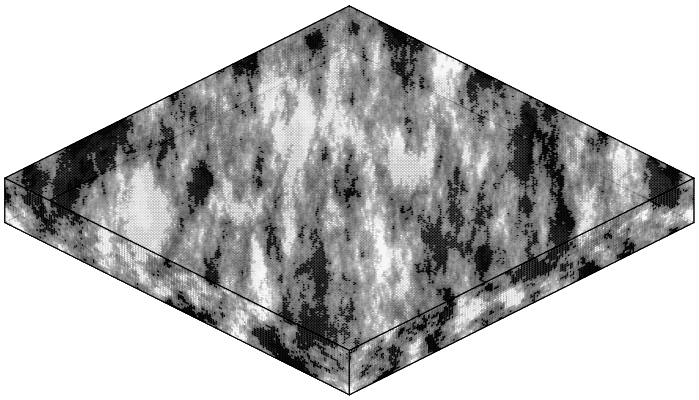

Figure 11: Distribution of permeability (coefficients) as a function of space

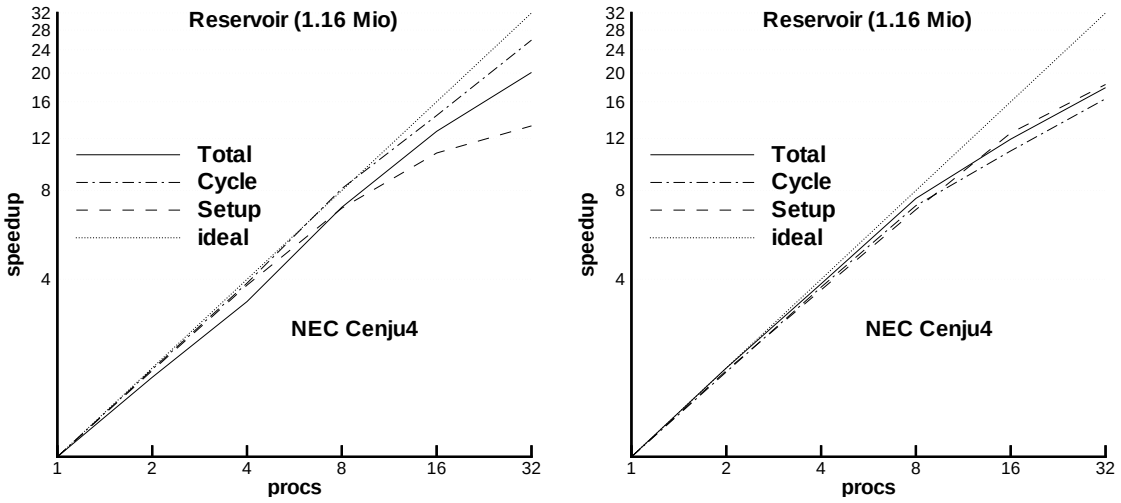

Figure 12: Reservoir test case: a) with agglomeration, b) without agglomeration

6 Conclusions

We have presented a relatively simple parallelization approach for the classical AMG variant described in detail in [9]. One major goal of this approach was to minimize the communication overhead involved in both the setup and the solution phase. This has been achieved by a modification of AMG's coarsening and interpolation strategies near processor interfaces. However, these modifications are at the expense of a potential dependence of AMG's convergence speed and the arithmetical work on the number of processors as well as on the complexity of the processor interfaces.

Practical experience has shown that these dependencies are fairly low if the problem size per processor is reasonably large. In fact, very satisfactory parallel efficiencies have been obtained for various complex industrial applications of relevant size. This is in spite of the fact that the current implementation does not take any optimization of the load balance into account. Indeed, two sources of load imbalance have been discussed which, for a fixed problem size and a relatively large number of processors, limit the parallel efficiency. This will be investigated further.

References

[1] Brandt, A.: *Algebraic multigrid theory: the symmetric case*, Appl. Math. Comp. 19, pp. 23-56, 1986.

[2] Brandt, A.; McCormick, S.F.; Ruge, J.: *Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations*, Institute for Computational Studies, POB 1852, Fort Collins, Colorado, 1982.

[3] Brandt, A.; McCormick, S.F.; Ruge, J.: *Algebraic multigrid (AMG) for sparse matrix equations*, in "Sparsity and its Applications", D.J. Evans (ed.), Cambridge University Press, pp. 257-284, Cambridge, 1984.

[4] Cleary, A.J.; Falgout, R.D.; Henson, V.E.; Jones, J.E.: *Coarse-grid selection for parallel algebraic multigrid*, Proceedings of the "Fifth International Symposium on Solving Irregularly Structured Problems in Parallel", Lecture Notes in Computer Science 1457, Springer-Verlag, New York, pp. 104-115, 1998.

[5] Robinson, G.: *Parallel computational fluid dynamics on unstructured meshes using algebraic multigrid*, Parallel Computational Fluid Dynamics 92 (Pelz, R.B.; Ecer, A.; Häuser, J. eds.), Elsevier Science Publishers B.V., 1993.

[6] Ruge, J.W.; Stüben, K.: *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, Multigrid Methods for Integral and Differential Equations (Paddon, D.J.; Holstein H.; eds.), The Institute of Mathematics and its Applications Conference Series, New Series Number 3, pp. 169-212, Clarenden Press, Oxford, 1985.

[7] Ruge, J.W.; Stüben, K.: *Algebraic Multigrid (AMG)*, In "Multigrid Methods" (McCormick, S.F., ed.), SIAM, Frontiers in Applied Mathematics, Vol 5, Philadelphia, 1986.

[8] Stüben, K.: *Algebraic multigrid (AMG): Experiences and comparisons*, Appl. Math. Comp. 13, pp. 419-452, 1983.

[9] Stüben, K.: *Algebraic Multigrid (AMG): An Introduction with Applications*, GMD Report 70, November 1999. This report will appear as a guest contribution (appendix) in the book "Multigrid" by U. Trottenberg, C.W. Oosterlee, A. Schüller; Academic Press, to appear 1999.

[10] Stüben, K.: *A Review of Algebraic Multigrid*, GMD Report 69, November 1999, to appear in Journal of Computational and Applied Mathematics, 2000.