Electronic Transactions on Numerical Analysis.
Volume 10, pp. 351-366, 2010.
Copyright © 2010, Kent State University.
ISSN 1068-9613.

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

BENCHMARKING AGGREGATION AMG FOR LINEAR SYSTEMS IN CFD SIMULATIONS OF COMPRESSIBLE INTERNAL FLOWS [*]

MAXIMILIAN EMANS[†]

Abstract. The performance of parallel implementations of three fundamentally different aggregation AMG (algebraic multigrid) solvers, including novel k-cycle methods, for systems of linear equations appearing in industrial CFD simulations are examined. The results show that the k-cycle methods are a good choice for cases with less than 20000 unknowns per process if the cost of the setup tends to become critical; for most other applications, however, established methods proved to be equally efficient or superior.

Key words. algebraic multigrid, fluid dynamics, finite volumes, pressure-velocity coupling

AMS subject classifications. 15A06, 65F08, 76G25

1. Introduction.

The three-dimensional simulation of various processes related to fluid dynamics requires the approximate solution of the Navier-Stokes equations and, eventually, an energy equation for steady or unsteady and for compressible or incompressible flows in terms of pressure, temperature, and velocity fields. Contemporary simulation tools provide a considerable amount of freedom with respect to geometry that requires a discretisation on unstructured meshes. Due to the resolution necessary for reasonable modelling, the size of the problems is in the range of one million grid cells or more which makes the use of parallel computers using typically a few CPUs inevitable to keep the computing times at an acceptable level.

A common approach to provide an appropriate approximation of the solution of the Navier-Stokes equations in this context is the discretisation by means of finite volumes. The SIMPLE (“Semi-Implicit Method for Pressure-Linked Equations”; see Patankar [21]) algorithm or a method derived from it can be used to obtain an approximate solution of this coupled non-linear system. To the knowledge of the author, these kind of methods are used in most of the actual commercial CFD-tools since they can be employed to a wide range of problems. The SIMPLE algorithm requires the solution of linear systems of equations. For this task, AMG methods are an appropriate choice since they are fast and sufficiently robust.

However, the term AMG stands for a class of algorithms rather than for a certain method. Meanwhile a number of methods, well suited for the particular requirements of CFD, are known, but recently Notay [20] devised a promising, conceptually new method. Since, as we will explain later, there is some hope that this method can lead to a significant improvement compared to the known algorithms, especially with respect to the particular requirements of CFD application, we will compare it in terms of computational efficiency to other successful techniques in this contribution.

As far as the computing time (in fact the most relevant property for industrial application) is concerned, it is not obvious that the properties of the AMG variants described in the literature can be transferred to the practical application in CFD. It is well possible that a particular algorithm is reported to perform excellently, e.g., in solving a particular diffusion problem, but that the same algorithm exhibits unacceptable runtimes in our applications. On the one hand, this is due to the different requirements to the accuracy of the solution; on the other hand, the practical application is decisive for the properties of the system of linear equa-

[*]Received January 7, 2010. Accepted for publication April 18, 2010. Published online November 9, 2010. Recommended by R. Lehoucq.

[†]AVL List GmbH, Hans-List-Platz 1, 8020 Graz, Austria and IMCC, Altenbergerstr. 69, 4040 Linz, Austria (maximilian.emans@avl.com).

tions: Even if fundamental matrix properties can be verified, the discretisation with different geometrical cell types or varying gradients in flow variables might lead to matrices that are quite different from those of idealised problems. Therefore, a general understanding of the fundamental algorithms for the flow simulation is necessary. Although these algorithms are known, we will spend a certain portion of this article to describe them in detail in a uniform matrix based notation since the original papers often use an implementation oriented notation that is sometimes quite tedious to read for someone not familiar with this practice.

This paper is dedicated to the performance of different aggregation based AMG algorithms in the context of CFD software of the industrial practice. We will examine computational aspects of the behaviour of these algorithms applied as linear solvers for systems that appear in the simulation of subsonic internal compressible flow at the example of a combustion engine. Both, the algorithms and the hardware configuration of our test cases are deliberately chosen to be as similar as possible to the tools that are used, e.g., by CFD engineers in their daily professional practice. Since our test cases are taken from a single practical application of simulation software and since our conclusions are only backed by experimental data, it is not always straightforward to transfer our observations and conclusions to other applications. A second reason to set great value on a detailed description and classification of the test cases is therefore to preserve at least a certain amount of generality.

The remainder of this article is organised as follows: In Section 2 we compile the relevant algorithms for the solution of the Navier-Stokes equations from various papers and present them in a uniform matrix based notation. In Section 3 we recall the main features of the known aggregation based multigrid algorithms and the k-cycle method of Notay [20], work out relevant differences, and give details of our parallel implementations. Section 5 is devoted to the presentation of results of numerical experiments with these algorithms. Finally, Section 6 contains our conclusions.

2. SIMPLE for incompressible flows.

Numerical simulation of fluid flow usually relies on a solution of the Navier-Stokes equations and the energy equation, along with (eventually) a certain number of transport equations. The method discussed in this paper is typical for industrial CFD codes: its spacial discretisation is based on the technique of the finite volumes with a collocated variable arrangement that allows to use unstructured meshes. The employed discretisation practice was described in detail in other publications starting with Demirdžić and Muzaferija [3] and continuing with Ferziger and Perić [10], Marthur and Marthy [17], and Basara [1]. With such a scheme, the discretised momentum and continuity equations for the unknown velocity field $\vec{u}$ and the pressure field $\vec{p}$ may be written as

$$\begin{aligned} A(\hat{\vec{u}})\vec{u} + M\vec{p} &= \vec{b}, \\ C\hat{\vec{u}} &= \vec{c}. \end{aligned}$$

Here, A denotes the discretised and linearised operator that acts on the velocity field in the momentum equations, i.e., it expresses convective, diffusive, and eventually inertia components. M is the discretisation of the gradient operator of the pressure term, $\vec{b}$ the body force term, C represents the discretised continuity equation, and $\vec{c}$ is a mass source. The hat ($\hat{\ }$) on top of the vectors indicates that the vector is discretised on the grid that corresponds to the cell faces. The discretisation of the operators requires that both the velocity in the cell centre $\vec{u}$ and the velocity at the cell faces $\hat{\vec{u}}$ appear in the system of equations. Both variables are linked by a linear interpolation operator S.

The most challenging task within numerical simulations of fluid flows is to compute the solution of this non-linear and coupled system. For this, the SIMPLE algorithm, see Patankar [21], can be used. A detailed deduction of the SIMPLE algorithm in a similar notation can be found in Emans [5]. In the following we restrict ourselves to the steps of the SIMPLE algorithm.

In this iterative procedure, the velocity field that is calculated in the m-th iteration is split into a tentative velocity field $\vec{u}^*$ and a velocity update $\vec{u}'$

$$\vec{u}^{(m)} = \vec{u}^* + \vec{u}'. \tag{2.1}$$

A similar splitting is applied to the pressure:

$$\vec{p}^{(m)} = \vec{p}^{(m-1)} + \vec{p}', \tag{2.2}$$

where $\vec{p}'$ is referred to as the pressure-correction.

In the first step of the SIMPLE algorithm the equation

$$A(\vec{u}^{(m-1)})\vec{u}^* + M\vec{p}^{(m-1)} = \vec{b} \tag{2.3}$$

is solved. The solution is the preliminary velocity $\vec{u}^*$; it is a solution of the momentum equation, but in general it is not a solution of the continuity equation. The dependence of A on $\vec{u}$ reflects the non-linearity of the system. Moreover, the pressure of the previous iteration is used. Both, the pressure and the (final) velocity of the current iteration are determined later in the iteration. The solution can be considered converged with respect to the threshold ϵ if the change in $\vec{u}$ lies below this threshold. This implies $|\vec{u}^{(m)} - \vec{u}^{(m-1)}| < \epsilon$ and $|\vec{u}^* - \vec{u}^{(m)}| < \epsilon$ or $|\vec{u}'| < \epsilon$.

The next step is to compute a pressure-correction $\vec{p}'$ that is used to update the pressure and to drive the velocity field towards being a solution of the continuity equation, too. The SIMPLE algorithm on collocated grids requires the velocity on the cell faces $\hat{\vec{u}}$. A stable method to couple velocity and pressure has been devised by Rhie and Chow [22]. Within this scheme the velocity at the cell faces is calculated from

$$\hat{\vec{u}}^* = S(\vec{u}^* + A_D^{-1}M\vec{p}^{(m-1)}) - \hat{A}_M\hat{M}\vec{p}^{(m-1)}, \tag{2.4}$$

where $A_D := \mathrm{diag}(A)$, $\hat{A}_M := [SA_D^{-1}\vec{1}]$, $\hat{M}$ is the gradient operator on the cell faces based on the difference between the values of the two adjacent cells, and $[\vec{x}]$ stands for a diagonal matrix that has the element of value x_j (j-th component of vector $\vec{x}$) in the j-the row. With this, the coefficients and the right-hand side of the pressure-correction equation

$$-C\hat{A}_M\hat{M}\vec{p}' = \vec{c} - C\hat{\vec{u}}^* \tag{2.5}$$

can be assembled and this linear system of equation can be solved. In the deduction of this equation the term $-SA_D^{-1}(A - A_D)\vec{u}'$ (that is driven towards zero in the course of the iteration) has been neglected. Since in the incompressible case C is the divergence operator (multiplied by the constant density), $\hat{A}_M$ merely scales the system matrix and $\hat{M}$ is the gradient operator (on the cell faces), the system matrix is positive semi-definite (with row sum zero). The solution of this system is one of the computationally most expensive tasks within the SIMPLE algorithm; for realistic simulation it requires typically between 30% and 80% of the total computing time.

From the pressure-correction a velocity correction is calculated with

$$\hat{\vec{u}}' = -\hat{A}_M\hat{M}\vec{p}'. \tag{2.6}$$

The velocity and the pressure for the iteration m are then obtained by updating both variables according to (2.1) and (2.2), respectively. If the solution has not converged yet, another iteration is taken, starting with the solution of the momentum equation (2.3). In short, the SIMPLE algorithm is outlined in Algorithm 1.

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

Algorithm 1 SIMPLE for incompressible flows and collocated variable arrangement

```
1: while <not converged> do
2:     assemble $A(\vec{u}^{(m-1)})$ and solve Eqn. (2.3) for $u^*$
3:     compute $\hat{\vec{u}}^*$ with Eqn. (2.4), assemble right-hand side and matrix of Eqn. (2.5) and
       solve for pressure-correction $\vec{p}'$
4:     compute velocity-correction with Eqn. (2.6)
5:     update velocity and pressure with Eqns. (2.1) and (2.2)
6: end while
```

2.1. SIMPLE for compressible flows. For compressible flows, the density ϱ depends on the pressure field $\vec{p}$ and the temperature field $\vec{T}$ (which may be uniform if the flow is isothermal). This implies that the continuity operator C depends linearly on the density and that the time derivative of the density appears in the continuity equation. The following formulation of this equation reflects this and is valid for first order time discretisation with time step δt:

$$C_c[\hat{\vec{\varrho}}]\hat{\vec{u}} = -[\vec{v}](\vec{\varrho} - \vec{\varrho}_0).$$

The components of $\vec{v}$ are $v_j = V_j/\delta t$, $\vec{\varrho}_0$ denotes the density of the previous time step.

For an ideal gas, the relation between density, pressure, and temperature can be expressed as

$$\vec{\varrho} = [\vec{r}]\vec{p}, \tag{2.7}$$

where $\vec{r}$ denotes the vector with the components $r_j = 1/(R_j \cdot T_j)$. R_j is the gas constant in the cell j that depends on the composition of the gas in this cell, T_j is the component of $\vec{T}$ associated with cell j.

The formal introduction of a density update

$$\vec{\varrho}' = [\vec{r}]\vec{p}' = \vec{\varrho}^{(m)} - \vec{\varrho}^{(m-1)},$$

substitution and neglect of products of updates yields the discrete pressure-correction equation for compressible flows

$$\begin{aligned}\left\{C_c\left([\hat{\vec{\varrho}}^{(m-1)}](-\hat{A}_M\hat{M}) + [\hat{\vec{u}}^*][\hat{\vec{r}}]\right) + [\vec{v}][\vec{r}]\right\}\vec{p}' = \\ -[\vec{v}](\vec{\varrho}^{(m-1)} - \vec{\varrho}_0) - C_c[\hat{\vec{\varrho}}^{(m-1)}]\hat{\vec{u}}^* + \vec{c}.\end{aligned} \tag{2.8}$$

Note that the hat on top of a vector indicates that this vector is discretised on the cell faces, i.e., the values at a face are interpolated linearly between the two adjacent cells. The SIMPLE algorithm for compressible flows on collocated grids that has been introduced by Demirdžić et al. [4] may be written as Algorithm 2; for more details we refer again to Emans [5].

Considering the composition of the left-hand side operator of (2.8), one can see that the operator is certainly positive definite if the velocity field $\hat{\vec{u}}^*$ and the density satisfy the continuity equation. Roughly speaking, this property might be lost if the defect of the continuity equation is no longer balanced by the contribution of $[\hat{\vec{v}}][\hat{\vec{r}}]$ to the operator on the left-hand side of (2.8). However, under the condition that the flow field is initialised consistently, the matrix can be supposed to be definite; e.g., all matrices of the benchmarks described in this article are indeed positive definite.

It is important to note that the solution of the pressure-correction equation takes place within an iterative scheme. Experience has shown that a numerically exact solution of these systems of equations is not needed such that for practical application the reduction of any residual norm by a factor between 10 and 10000 is sufficient.

Algorithm 2 SIMPLE for incompressible flows and collocated variable arrangement

```
1:  while <not converged> do
2:    (re)compute density field $\vec{\varrho}^{(m-1)}$ with Eqn. (2.7) using $\vec{p}^{(m-1)}$ and $\vec{T}^{(m-1)}$
3:    assemble $A(\vec{u}^{(m-1)})$ and solve Eqn. (2.3) for $\vec{u}^*$
4:    compute $\hat{\vec{u}}^*$ with Eqn. (2.4), assemble right-hand side and matrix of Eqn. (2.8) and
      solve for pressure-correction $\vec{p}'$
5:    compute velocity-correction with Eqn. (2.6)
6:    update velocity and pressure with Eqns. (2.1) and (2.2) to obtain $\vec{u}^{(m)}$ and $\vec{p}^{(m)}$
7:    if <flow not isothermal> then
8:      assemble matrix and right-hand side of the energy equation using $\vec{\varrho}^{(m-1)}$, $\vec{u}^{(m)}$,
        and $\vec{p}^{(m)}$ and solve for $\vec{T}^{(m)}$
9:    end if
10: end while
```

2.2. PISO. The convergence of the SIMPLE algorithm is accelerated if the neglected term $SA_D^{-1}(A - A_D)\vec{u}\,'$ is approximated rather than set to zero. A common method is known as PISO algorithm ("Pressure-Implicit with Splitting Operators"); see Issa [13]. By means of an additional iteration it makes sure that after each iteration of PISO the continuity equation is solved at least up to the accuracy of the solution of the pressure-correction equation. For the incompressible case the correlation between $\vec{p}'$ and $\hat{\vec{u}}\,'$ without neglecting any term in the original equations is

$$\hat{\vec{u}}\,'^{(i+1)} = -\hat{A}_M\hat{M}\vec{p}\,'^{(i)} - SA_D^{-1}(A - A_D)\vec{u}\,'^{(i)}. \tag{2.9}$$

Formally, the pressure-correction equation with this is

$$-C\hat{A}_M\hat{M}\vec{p}\,'^{(i+1)} = \vec{c} - C(\hat{\vec{u}}^* - SA_D^{-1}(A - A_D)\vec{u}\,'^{(i+1)}), \tag{2.10}$$

where $\vec{u}\,'^{(i+1)} = S^T\hat{\vec{u}}\,'^{(i+1)}$. Since $\hat{\vec{u}}\,'^{(0)}$ is not known, it is set to zero. Then (2.10) is solved for $\vec{p}\,'^{(1)}$. Now, for the next i, the right hand side of (2.9) can be evaluated and in this way an iteration between (2.10) and (2.9) is done. In the course of this additional iteration, $\hat{\vec{u}}\,'$ approaches a certain vector that reflects the velocity correction needed to satisfy the continuity equation. In the practical application of this algorithm typically up to five iterations are required to reduce the difference between $\hat{\vec{u}}\,'^{(i+1)}$ and $\hat{\vec{u}}\,'^{(i)}$ below a given threshold that leads to a significant reduction of the (outer) PISO iterations.

The resulting algorithm has shown to converge significantly faster than the standard SIMPLE such that the extra effort to evaluate $\hat{\vec{u}}\,'^{(i+1)}$ using (2.10) and to solve the system of equations (2.10) pays off in many situations. Note that the matrix of the pressure-correction equation does not change from one internal iteration within PISO (marked by index i in (2.9) and (2.10)) to the next. PISO can be transferred easily from incompressible to compressible calculations.

3. Parallel AMG algorithms. The focus of this article is on the comparison of aggregation based k-cycle methods to other established aggregation methods. In this section we describe relevant common features and the differences of the parallel AMG algorithms under examination. The fundamental AMG algorithm is not repeated here. For this we refer the reader to the literature; e.g., to the early AMG publication of Ruge and Stüben [23] or to the appendix of Trottenberg et al. [27].

3.1. General aspects of the parallelisation and implementation. We restrict ourselves to the employment of AMG methods for the solution of the pressure-correction equations in one of the schemes described above. The parallelisation of these schemes in AVL FIRE$^{\text{(R)}}$ 2009 is done by a geometric domain decomposition, i.e., the total number of finite volumes (global domain) is divided into disjoint subdomains and distributed to the available processes; whenever necessary, information between two adjacent subdomains is exchanged. For the solver of the systems of linear equations the same distributed memory approach is used.

For the computation of the coarse-grid operator we follow the Galerkin approach; see Trottenberg et al. [27]. Given a system matrix B and a restriction operator R, we compute the coarse-grid operator B^C explicitly as

$$B^C = RBR^T.$$

Note that the interpolation operator is R^T.

Even though our fine-grid matrices are sparse and have typically not more than 6 off-diagonal elements, rows with 60 and more elements are frequently found in coarse-grid operators; see, e.g., Yang [31] for similar observations. As a matter of fact, the amount of data to be transmitted and processed is considerable due to this effect and it increases the cost of the setup significantly. Since, on the other hand, one realises quickly that in our application the setup costs tend to be prohibitive to the application of AMG, we allow only local interpolation in all our algorithms. This is a straightforward method to avoid the necessity to transfer parts of the matrix B and to reduce efficiently the number of elements of R that need to be exchanged between two adjacent domains for the explicit computation of B^C. Possibly a deteriorated convergence of parallel computations is the price that is to be paid for this restriction.

It is widely agreed upon that a full parallelisation of the smoother is not practicable. We follow the usual practice and employ a fully parallel Jacobi scheme on the boundaries between two domains and a Gauß-Seidel lexicographic scheme for the interior points. This technique is referred to as hybrid Gauß-Seidel smoother; see Henson and Yang [12]. Moreover, in parallel computations, an agglomeration strategy is employed to treat the systems assigned to the coarsest grids; see Trottenberg et al. [27]: all the information of grids with less than 200 nodes is passed to one of the neighbours until one grid remains. The system of equations of this coarsest grid is then solved directly by Gaußian elimination by a single process while the other processes idle.

For any performance test, the implementation is an important issue. In this study we use a uniform interface similar to the one described by Falgout and Yang [9]. The data is stored in CRS (compressed row storage) format; for details see Falgout et al. [8]. All linear algebra operations are implemented uniformly without the use of external libraries. The choice of the CRS format for the matrix determines the implementation of all matrix-vector operations in the sequential case. The solver related parts of the program are coded in FORTRAN 90 and compiled by the Intel FORTRAN compiler version 10.1.

For the communication in the parallel case we use the hp-MPI library, version 2.3 (with C-binding). Field data at the domain boundaries is always exchanged through the asynchronous “immediate send” mechanism. In the parallel setup phase, the exchange of data is started and the local operations are performed simultaneously; once the exchange and the internal work has been completed, the boundaries are treated. The restriction operators are also stored in CRS format once they have been computed in the setup phase. The same mechanism applies to the solution phase: the exchange of the information at the inter-domain boundaries is started and the local work is done while the exchange takes place; each operation that requires data exchange at the domain boundaries (matrix-vector product or smoothing sweep) is finished with the computation of the boundaries after the exchange and the internal work have been completed. Apart from this asynchronous communication only collective communication patterns occur: for the synchronisation of the setup phase on the one hand and for the computation of inner products and norms during the solution phase on the other hand.

3.2. Remarks on the selection of algorithms. The so-called “classical” AMG of Ruge and Stüben [23, 26] is based on a C-F-splitting, i.e., it chooses from the set of fine-grid points (F-points) a set of points the coarse-grid consists of (C-points). Aggregation methods, on the other hand, divide the set of fine-grid points into a number of disjoint subsets that are called aggregates. These aggregates then become the abstract coarse-grid points. This difference is illustrated in Figure 3.1. Whereas for “classical” AMG methods the interpolation formulae between C-points and F-points are a very sensitive issue, for aggregation methods in principle a constant interpolation can be used. Even if a C-F-splitting method employs very efficient and stable algorithms such as the ones suggested by De Sterck et al. [25], the computation of the interpolation weights is still rather expensive and tends not to pay off in our kind of cases; see Emans [6]. We therefore do not further follow the concept of C-F-splitting methods in the present article.

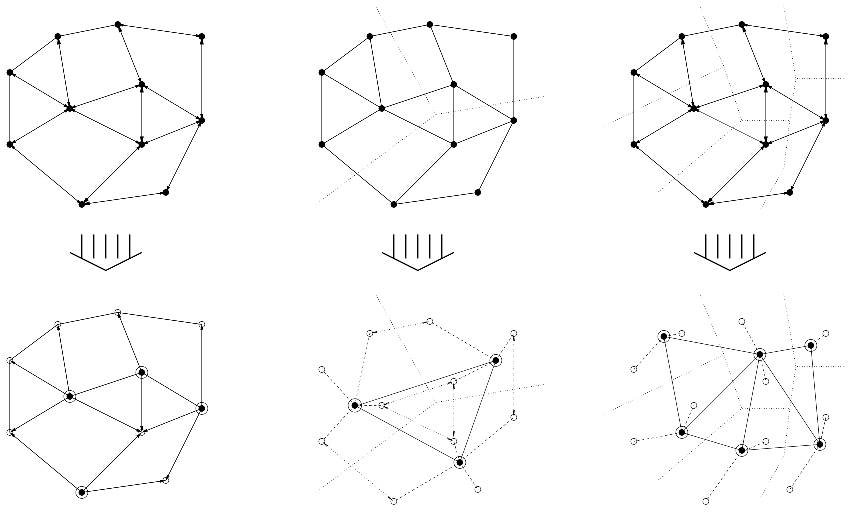
FIG. 3.1. *Coarse-grid selection of “classical” AMG of Stüben [26] (left), Smoothed Aggregation (center), and pairwise aggregation of Notay [19] (right).*

There have been, on the other hand, many variations of aggregation methods described in the literature. In the context of the pressure-correction schemes like SIMPLE the Smoothed Aggregation AMG of Vaněk et al. [29] is an appropriate algorithm; see, e.g., Emans [6]. This algorithm is characterised by large aggregates, comparatively large cost of the setup, but low memory requirements. In order to show satisfying convergence, it has to be used as a preconditioner of a Krylov-subspace method. Another type of methods forms small aggregates from very few (typically two or three, see, e.g., Weiss et al. [30]) nodes of the fine grid, such that the computation of the coarse-grid operators is very simple and cheap, while the memory requirement can be high. From the viewpoint of application, the k-cycle method of Notay [20] is an attempt to combine the advantages of simple operator computation and larger aggregates by employing a special Krylov-backed cycling strategy. It is the cheap setup and the hope for very good convergence properties that makes this method seem attractive for the CFD applications described in this article.

3.3. Smoothed Aggregation AMG. A serial implementation of this method is described in detail in Vaněk et al. [29]. This algorithm considers the mutual influence of i onto j (and vice versa) and tries to group degrees of freedom (dof) linked by matrix elements with

$$|a_{ij}| \geq \varepsilon \cdot \sqrt{a_{ii} \cdot a_{jj}}$$

together. The parameter ε depends on the level in the grid hierarchy l ($l = 1$: finest grid) and it is defined as

$$\varepsilon := 0.08 \cdot 0.25^{(l-1)}.$$

The number of dof per aggregate is rather large leading to a rather rapid coarsening process. In this method, the interpolation operator with constant interpolation is not used directly; instead, it is refined ("smoothed") by an application of one Jacobi step along the paths of the graph of the fine-grid matrix in order to get the final interpolation operator. For parallel computations, we do not permit aggregates that range over more than one subdomain and restrict the smoothing to local dof. Through this, the parallel computation of the coarse-grid operator is kept relatively simple. In fact, more complex approaches that allow to waive these restrictions do not necessarily seem to result in better performance, in particular not if the number of unknowns per node is not too small; see Tuminaro and Tong [28]. However, the way of choosing the aggregates influences the convergence significantly; see Fujii et al. [11]. In the parallel algorithm, we therefore take the dof adjacent to domain boundaries as a first set of root points for the aggregation process.

The Smoothed Aggregation approach produces interpolation schemes and coarse-grid operators that are structurally the same as the corresponding elements of the classical AMG of Stüben [26]; the coarsening process is fairly rapid, the size of the aggregates typically ranges between 10 and more than 50. The interpolation corresponds to a linear interpolation; see the appendix of Trottenberg [27]. We use this method as a preconditioner for the conjugate gradient algorithm; see Saad [24]. We have implemented a v-cycle scheme with two pre- and two post-smoothing hybrid Gauß-Seidel sweeps. For this algorithm we use the abbreviation **ams1cg**.

3.4. Basic AMG. The contribution to the computational cost of the setup of both, C–F-splitting methods as well as Smoothed Aggregation technique to the total computing time of these AMG variants amounts to more than 40%. Within the setup, the most expensive part is the (parallel) computation of the interpolation weights and of the elements of the coarse-grid operators. If constant interpolation is employed, the computation of the interpolation weights is completely avoided and the computation of the coarse-grid operators is dramatically simplified (in fact it reduces to an addition of rows of the fine-grid operator).

Since for our type of application the required accuracy of the solution is low and consequently only a few iterations are necessary, an expensive setup is particularly painful. Therefore in our basic AMG method constant interpolation is employed. The disadvantage of constant interpolation is a comparatively bad representation of the fine-grid problem on the coarse grid for aggregates of the size that is typical for, e.g., the Smoothed Aggregation method. Using smaller aggregates is therefore mandatory to keep the convergence rates at an acceptable level.

An algorithm that constructs aggregates of not more than two degrees of freedom has been described by Notay [20]. In order to form the aggregates, strongly connected degrees of freedom according to the relation

$$a_{ij} < -\beta \max_{(k)}(|a_{ik}|)$$

with $\beta = 0.25$ are preferred. This distinction is similar to that used in the successful early AMG of Stüben [26]. The goal of sufficiently accurate interpolation is reached with this algorithm. The price to pay is the slow coarsening process, i.e., since the number of dof is reduced only by a factor close to two from one grid to the next finer grid, much more grids need to be constructed compared to, e.g., the Smoothed Aggregation algorithm.

Although it would be possible to employ this algorithm as a preconditioner of a CG method (as the Smoothed Aggregation AMG in this paper), we use it as a "stand-alone" solver. Satisfying convergence rates are obtained in this configuration if an f-cycle scheme (recursively a w-cycle is followed by a v-cycle; see Trottenberg et al. [27]) is applied. Only two hybrid Gauß-Seidel sweeps are necessary after the return to the finer grid, i.e., there is no pre-smoothing done. The algorithm is here referred to as **amggs2**

3.5. Aggregation-based AMG with Krylov-acceleration. Here, the coarse-grid operator is computed in two steps. In the first step, after the fine-grid nodes have been grouped into pairs using the same algorithm of Notay [20] as for amggs2, an intermediate coarse-grid operator is computed assuming constant interpolation within the aggregates. The second step is a repetition of this aggregation starting with the aggregates of pairs of the fine-grid nodes, generated by the first step, and the corresponding coarse-grid operator. The result is a coarse-grid hierarchy with a reduction factor of almost four.

The cycling strategy is referred to as a k-cycle: On each level the corresponding set of equations is iterated by a Krylov-subspace method that is in turn recursively preconditioned by a k-cycle using the hierarchy of the coarser grid. The number of iterations on each level is either one or two, depending on an error estimate. The control of the recursive preconditioning follows Notay [20]. Since the preconditioning operation is adaptive, the standard CG method has been replaced by an economic version of the conjugate gradient algorithm with explicit orthogonalisation of the search directions called "flexible conjugate gradients;" see Notay [18]. To avoid exhaustive memory consumption the method is restarted after six iterations. Again two pre- and two post-smoothing hybrid Gauß-Seidel sweeps are done. We refer to our implementation of the k-cycle algorithm (including computation of coarse-grid operators) described in Notay [20] as **amk1fc**.

In the previous publication of Notay [19], a particular modification for parallel computations has been suggested. It is a switch to standard v-cycle for the two coarsest grids if less than eight processes are involved. For higher degree of parallelism, three grids are treated by v-cycles. The modification was motivated by the observation that due to the recursive structure of the k-cycle algorithm the coarse grids are visited frequently which entails a large number of communication events. For test purposes we implemented this modification also for the serial computation. We refer to this algorithm as **amk2fc**. Some properties of the employed algorithms are compared in Table 3.1.

TABLE 3.1
Some properties of the compared algorithms (the maximum and average aggregate size refers to the coarsening of the finest grid of case 0000; see Section 4.1).

	ams1cg	amggs2	amk1fc	amk2fc
employment as	prec. of CG	solver	prec. of flexible CG	
interpolation	linear	constant	constant	
maximum aggregate size	83	2	4	
average aggregate size	23.13	2.0	4.0	
cycling strategy	v-cycle	f-cycle	k-cycle	k/v-cycle

4. Computations of flows in an engine. Our test cases are taken from a typical simulation of a full cycle of a gasoline engine. The three-dimensional computational domain is subject to change in time: It contains the interior of the cylinder and the parts of the ducts

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

through which the air is sucked into the cylinder or expelled from it. A three-dimensional simulation of a full engine cycle comprises the simulation of the (compressible) flow of cold air into the cylinder while the piston is moving downward, the subsequent compression after the valves are closed, the combustion of the explosive mixture, and the discharge of the hot gas while the piston moves upward. In the simulation the fluid dynamics are modelled by the Navier-Stokes equations amended by a standard k-ε turbulence model of Jones and Launder [14]; the thermal and thermodynamic effects are considered through the solution of the energy equation for enthalpy and the computation of the material properties using the coefficients of air. The combustion model is based on the approach of Magnussen and Hjertager [16]. This model relates the rate of combustion to the dissipation of eddies and it expresses the rate of reaction in terms of the concentration of a reacting species, turbulent kinetic energy, and the dissipation. The model fuel that is burnt is octane. Since a single simulation run on a parallel computer will still take a few hours computing time, we pick out four characteristic periods of a few time steps, one from each of the four strokes of the cycle. The data describing the engine cycle is shown in Figure 4.1, the meshes can be seen in Figure 4.2.

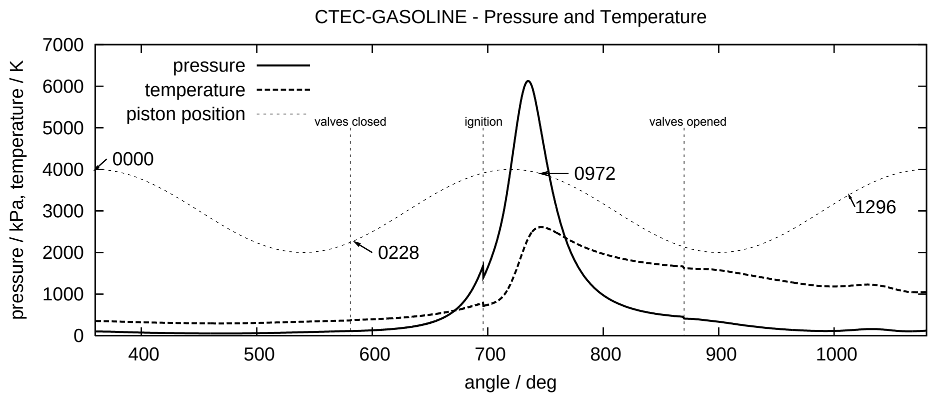

FIG. 4.1. *Scheme of the engine cycle along with notation for the considered partial problems.*

4.1. Description of the numerical experiments. The computationally relevant information about the cases is compiled in Table 4.1 that contains, e.g., the time step dt, the number of time steps n_t, the number of systems to solve n_{sy} for SIMPLE and PISO and the number of setups n_{se} for PISO. When the SIMPLE algorithm is used, in each SIMPLE iteration the system matrix is different and consequently the setup has to be done each time, i.e., $n_{sy} = n_{se}$. All system matrices are positive definite and have no positive side diagonal elements. The row sum of all systems is greater than zero. The maximum ratio of the value of an off-diagonal element to the value of the diagonal element can be read from Figure 4.1.

The performances of the AMG algorithms described in the previous section are compared to each other. We ran each case for each number of processes once using each of these algorithms as solver of the pressure-correction equations of the SIMPLE algorithm. All computations were repeated employing PISO instead of SIMPLE. The number of time steps n_t was the same for SIMPLE and PISO; as expected, the number of SIMPLE and PISO iterations was different. The iteration of the linear solver was terminated as soon as the 1-norm of the residual had been reduced by a factor of 20. From experience it is known that this threshold is sufficient for a stable convergence of SIMPLE and PISO. A stricter criterion would entail additional computational work for the solution of the pressure-correction equation while it

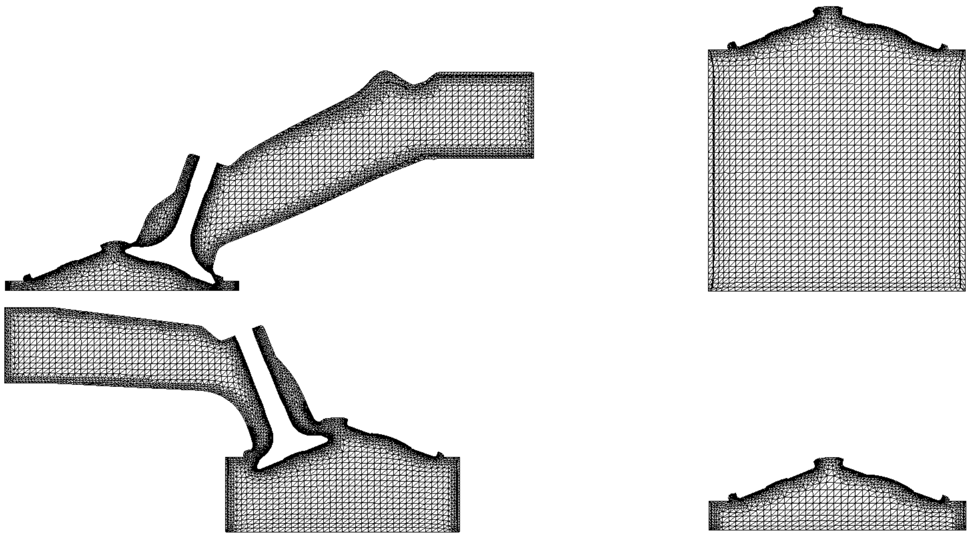

FIG. 4.2. *Slices through the three-dimensional meshes of the partial problems, from left top clockwise: load, compression, combustion, discharge.*

TABLE 4.1
Characterisation of the cases.

	0000	0228	0972	1296
stroke	load	compression	combustion	discharge
crank angle	$360°$	$585°$	$746°$	$1013°$
problem size	111.0 MB	23.0 MB	18.6 MB	52.1 MB
dt	$3.03 \cdot 10^{-5}$	$3.03 \cdot 10^{-5}$	$6.06 \cdot 10^{-6}$	$3.03 \cdot 10^{-5}$
n_t	5	20	20	15
boundaries	mass flow, wall	wall	wall	pressure, wall
n_{sy}, SIMPLE	130	172	378	175
n_{sy}, PISO	120	224	512	130
n_{se}, PISO	52	69	201	58
mesh: hex/tet	80.0 / 1.2 %	76.6 / 1.2 %	71.7 / 1.5 %	74.2 / 1.0 %
matrix: $max_i(\frac{-a_{ij}}{a_{ii}})$	0.87	0.79	0.76	0.85

does usually not speed up the convergence of SIMPLE, whereas a weaker criterion can lead to divergence of the computation, mainly as a consequence of an insufficient conservation of the mass in the physical system.

For the measurements we used up to four nodes à 2 quad-cores (i.e., 8 cores) of a Linux cluster (Intel Xeon CPU X5365, 3.00GHz, main memory 16 GB, L1-cache 2·4·32 kB, L2-cache 2·2·4 MB) connected by a Mellanox Infiniband network with an effective bandwidth of around 750 Gbit/s. The test cases were run within the environment of the software AVL FIRE$^{(R)}$ 2009 on 1, 2, 4, 8, and 16 processes, where the domain decomposition was performed once for each case by the graph partitioning algorithm METIS; see Karypis and Kumar [15]. Computations with 1, 2, and 4 processes were done on a single node, for 8 and 16 processes we used 2 and 4 nodes respectively such that each process had full access to 4 MB L2-cache since in preliminary experiments it has been found that the L2-cache can be the bottleneck for such kind of computations. Although distributing two or four tasks to two or four nodes would increase the performance, we used a single node for these computations since the gain in performance does usually not justify the occupation of the additional cores in the practical applications. Note that the inter-nodal communication is implemented by a shared memory approach in the MPI library that has been used.

5. Results of the numerical experiments. The raw data of our evaluation is the computing time of the setup that is independent of the number of iterations and the computing time of the solution phase for the SIMPLE and PISO computations; see Figures 5.1 and 5.2. Furthermore, we present the operator complexity c, the number of levels n_l of the coarse-grid hierarchy generated by each of the three methods,

$$c = \sum_{(l)} \frac{\text{number of matrix elements of level } l}{\text{number of matrix elements of level 1}},$$

and the cumulative iteration count; see Table 5.1. From the measured times the parallel efficiency E_p is computed as

$$E_p = \frac{t_1}{p \cdot t_p},$$

where t_p denotes the computing time on p processes. The values of E_p for SIMPLE and PISO are very similar; for SIMPLE they may be found in Figures 5.1 and 5.2.

TABLE 5.1
Algorithm related results of numerical experiments.

n_p	1	4	16		1	4	16	
Case		amk1cg		$c/(n_l)$		amk2fc		$c/(n_l)$
0000	776	775	780	1.57	752	743	720	1.57
0000P	724	721	717	(7)	692	681	681	(7)
0228	480	464	464	1.59	477	462	471	1.59
0228P	673	659	659	(6)	670	653	655	(6)
0972	1414	1408	1382	1.59	1363	1351	1382	1.59
0972P	2734	2307	2235	(6)	2109	2166	2436	(6)
1296	570	572	568	1.60	543	543	541	1.60
1296P	454	464	462	(6)	435	441	439	(6)
n_p	1	4	16		1	4	16	
Case		ams1cg		$c/(n_l)$		amggs2		$c/(n_l)$
0000	416	446	484	1.51	569	568	564	2.55
0000P	419	447	477	(3)	538	525	524	(13)
0228	316	348	400	1.52	419	417	424	2.59
0228P	427	498	534	(3)	555	554	574	(11)
0972	788	1125	1134	1.50	1148	1149	1147	2.60
0972P	1534	1664	1817	(3)	1761	1759	1747	(11)
1296	372	430	455	1.53	447	451	499	2.61
1296P	308	373	393	(3)	402	412	447	(12)

5.1. Performance of AMG algorithms within SIMPLE. In all four cases, the basic AMG amggs2 is the fastest algorithm for computations on up to four processes. The algorithms amk1fc and amk2fc come close in some cases, whereas ams1cg is significantly slower (note the logarithmic scale of the y-axis of the diagrams!). For computations on more than

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

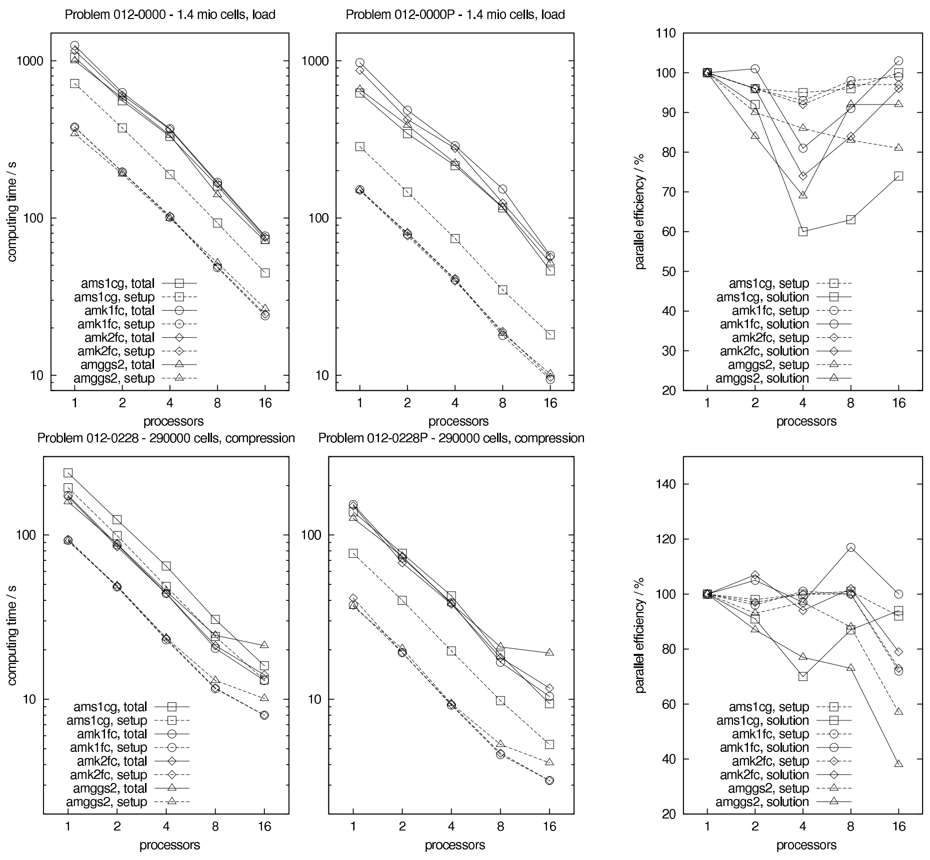

FIG. 5.1. *Computing times for cases 0000 and 0228, using SIMPLE (left) and PISO (center), and parallel efficiency for solution and setup phase (right) for SIMPLE.*

four processes the parallelisation reduces the computing time reasonably for ams1cg, to a significantly lower extend also for the k-cycle methods, but in a certainly not satisfying manner for amggs2. In the smaller cases 0228 and 0972, where the number of unknowns per process is less than 20000, the performance of the k-cycle algorithms is very attractive.

To analyse this, we turn to the parallel efficiency. The curves are influenced by two main opposed effects. On the one hand, E_p becomes worse as the number of processes is increased. This has three reasons: first the well-known parallel overhead that depends on communication requirements of the algorithm: without further analysis, as for example done by Čiegis et al. [2] for a simpler point-to-point communication pattern, this effect is difficult to quantify. However, it depends on the number of exchange operations and is consequently much higher for algorithm amggs2, the algorithm that has the most levels and visits them most frequently due to the F-cycle. Second, certain hardware resources such as memory access are depleted since the computations on up to four processes take place on one node of the cluster; this effect is mainly responsible for the decrease of the parallel efficiency for computations on up to four processes; a detailed study of this effect may be read in Emans [7]. The third reason is the deterioration of the convergence that is due to imperfect parallelisation of the algorithms and that leads to an increase in the number of iterations of the solver. This effect depends on the algorithmic properties and is essentially only seen for algorithm ams1cg.

The opposed effect is a superlinear acceleration of the computation of smaller distributed

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

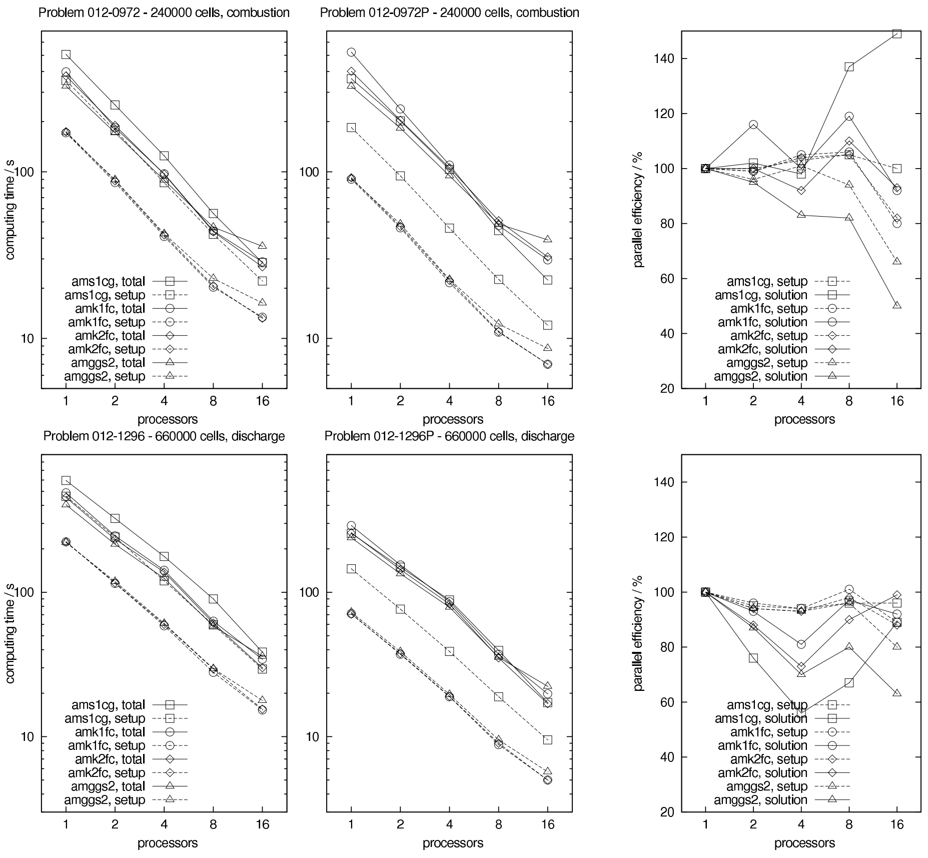

FIG. 5.2. *Computing times for cases 0972 and 1296, using SIMPLE (left) and PISO (center), and parallel efficiency for solution and setup phase (right) for SIMPLE.*

problems as reported by, e.g., Čiegis et al. [2], which is due to a decreased probability of cache misses. It is also very hard to predict this effect quantitatively. Some qualitative observations are reported in Emans [7]. However, the consequence is that the parallel efficiency of certain algorithms rises for computations on more than four nodes although a further decrease due to degraded convergence and additional communication cost would naturally be expected. The parallel efficiency exceeds 100% in such cases where the gain through cache effects is stronger than the losses through the parallelisation. Whereas for algorithm ams1cg obviously positive and negatives effects onto runtime partly annihilate each other and the positive ones prevail, for algorithm amggs2, characterised by the highest number of levels and the largest memory requirements, only in the largest case (0000) the positive effects dominate over the negative ones.

The operator complexity is a rough measure of the memory requirement. The operator complexities of both k-cycle methods are slightly higher than those of the ams1cg but still moderate. Due to the slow coarsening of amggs2 the operator complexities of this algorithm is significantly larger than that of the other algorithms. The main reason for this is that the first coarse-grid has about 50% of the number of rows of the fine grid, but the number of off-diagonal elements (per row) is significantly higher than the number of off-diagonal elements on the fine grid.

The significant increase of E_p of amk1fc and amk2fc for two processes is due to the instruction of Notay [19, 20] to use a stricter criterion to determine if a second preconditioning iteration is needed for the parallel case, leading to lower iteration numbers and consequently to lower solution times. A difference between amk1fc and amk2fc is observed, however, none of both outperforms the other throughout, but amk2fc is faster in most cases. Moreover, the number of iterations of these two algorithms is similar; the Krylov-acceleration on the coarse grids seems not to be effective since, if it is skipped (algorithm amk2fc), this has no significant negative effect on the convergence.

5.2. Performance of AMG algorithms within PISO. While the parallel efficiency is essentially the same as for SIMPLE (and is therefore not shown here), the computing times are different. One observes first that, as expected, the portion of setup time of the AMG solvers is reduced dramatically since expensively computed coarse-grid hierarchies can be reused several times due to the fact that several systems with identical right-hand-side operator are solved in each PISO iteration. Since the setup of ams1cg is much more expensive than that of amggs2, the difference between ams1cg and amggs2 for up to four processes has been reduced. Similar as for SIMPLE, amggs2 shows severe deficiencies at high numbers of processes. Again amk1fc and amk2fc are affected only mildly and perform significantly better than amggs2, but for 16 processes ams1cg remains the fastest solver for the linear systems within PISO.

6. Conclusions. In the benchmarks the k-cycle algorithms performed best among the established aggregation methods only in cases where the number of unknowns per node was below 20000 and where the coarse-grid hierarchy needs to be computed for each linear system. Other cases could be treated more efficiently by established methods. In particular, Smoothed Aggregation is an effective solver if systems with identical right-hand sides are to be solved. Basic aggregation AMG methods without Krylov-acceleration, on the other hand, are efficient whenever the setup should be cheap. It is remarkable that, although k-cycle methods are rather new and there exists consequently little experience in using them efficiently, their performance is fairly competitive.

REFERENCES

[1] B. BASARA, A. ALAJBEGOVIC, AND D. BEADER, *Simulation of single- and two-phase flows on sliding unstructured meshes using finite volume method*, Internat. J. Numer. Methods Fluids, 45 (2004), pp. 1137–1159.

[2] R. ČIEGIS, O. ILIEV, AND Z. LAKDAWALA, *On parallel numerical algorithms for simulating industrial filtration problems*, Comput. Methods Appl. Math., 7 (2007), pp. 118–134.

[3] I. DEMIRDŽIĆ AND S. MUZAFERIJA, *Numerical method for coupled fluid flow, heat transfer and stress analysis using unstructured moving meshes with cells of arbitrary topology*, Comput. Methods Appl. Mech. Engrg., 125 (1995), pp. 235–255.

[4] I. DEMIRDŽIĆ, Ž. LILEK, AND M. PERIĆ, *A collocated finite volume method for predicting flow at all speeds*, Internat. J. Numer. Methods Fluids, 16 (1993), pp. 1029–1050.

[5] M. EMANS, *Efficient parallel AMG methods for approximate solutions of linear systems in CFD applications*, SIAM J. Sci. Comput., 32 (2010), pp. 2235–2254.

[6] ———, *Performance of parallel AMG-preconditioners on CFD-codes for weakly compressible flows*, Parallel Comput., 36 (2010), pp. 326–338.

[7] M. EMANS AND A. VAN DER MEER, *Mixed-precision AMG as linear equation solver for definite systems*, in ICCS2010, Part I, G. v. Albada, P.M.A. Sloot and J. Dongarra, eds., vol. 1 of Procedia Computer Science, Elsevier, Amsterdam, 2010, pp. 175–183.

[8] R. FALGOUT, J. JONES, AND U. YANG, *Pursuing scalability for hypre's conceptual interfaces*, ACM Trans. Math. Soft., 31 (2005), pp. 326–350.

[9] R. FALGOUT AND U. YANG, *hypre: a library of high performance preconditioners*, in P.M.A. Sloot, C. Tan, J. Dongarra, and A. Hoekstra, eds., vol. 2331 of Computational Science, ICCS 2002 Part III, Lecture Notes on Computer Science, Springer, Heidelberg, 2002, pp. 632–641.

[10] J. FERZIGER AND M. PERIĆ, *Computational Methods for Fluid Dynamics*, Springer, Berlin Heidelberg, 1996.

[11] A. FUJII, A. NISHIDA, AND Y. OYANAGI, *Evaluation of parallel aggregate creation orders: smoothed aggregation algebraic multigrid method*, in High Performance computational science and engineering, M. Ng, A. Concescu, L. Yang, and T. Leng, eds., IFIP TC5 Workshop on High Performance Computational Science and Engineering (HPCSE), World Computer Congress, Toulouse, France, Springer, New York, 2004, pp. 201–213.

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

[13] R. ISSA, *Solution of the implicity discretised fluid flow equations by Operator-Splitting*, J. Comput. Phys., 62 (1985), pp. 40–65.

[14] W. JONES AND B. LAUNDER, *The prediction of laminarization with a two-equation model of turbulence*, Int. J. Heat Mass Transfer, 15 (1972), pp. 301–314.

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

[16] B. MAGNUSSEN AND B. HJERTAGER, *On mathematical modeling of turbulent combustion with special emphasis on soot formation and combustion*, in Proc. of the 16th Int. Symp. on Combustion 1976, Combustion Institute, Pittsburg, 1976, pp. 719–729.

[17] S. MARTHUR AND J. MURTHY, *A pressure based method for unstructured meshes*, Numer. Heat Transfer B, 31 (1997), pp. 195–215.

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

[19] ———, *An aggregation-based algebraic multigrid method*, tech. report, Service de Métrologie Nucléaire, Université Libre de Bruxelles, 2008.

[20] ———, *An aggregation-based algebraic multigrid method*, Electron. Trans. Numer. Anal., 37 (2010), pp. 123–146, `http://etna.math.kent.edu/vol.37.2010/pp123-146.dir`.

[21] S. PATANKAR, *Numerical Heat and Mass Transfer and Fluid Flow*, Hemisphere Publishing, New York, 1980.

[22] C. RHIE AND W. CHOW, *Numerical study of the turbulent flow past an airfoil with trailing edge separation*, AIAA J., 21 (1983), pp. 1525–1532.

[23] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid*, in Multigrid methods, vol. 3 of Frontiers Appl. Math., SIAM, Philadelphia, 1987, pp. 73–130.

[24] Y. SAAD, *Iterative Methods for Sparse Linear Systems*,third ed., SIAM, Philadelphia, 2003.

[25] H. D. STERCK, U. YANG, AND J. HEYS, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1–20.

[26] K. STÜBEN, *Algebraic multigrid (AMG): An introduction with applications*, tech. report, GMD Forschungszentrum Informationstechnik GmbH, St. Augustin (Germany), 1999.

[27] U. TROTTENBERG, C. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, Orlando, 2001.

[28] R. TUMINARO AND C. TONG, *Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines*, in Proceeding of 2000 ACM/IEEE conference on Supercomputing, Dallas, IEEE Computer Society, Washington, 2000.

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

[30] J. WEISS, J. MARUSZEWSKI, AND W. SMITH, *Implicit solution of preconditioned Navier-Stokes equations using algebraic multigrid*, AIAA J., 37 (1999), pp. 29–36.

[31] U. YANG, *Parallel algebraic multigrid methods - high performance preconditioners*, in Numerical Solution of Partial Differential Equations on Parallel Computers, A. Bruaset and A. Tveito, eds., vol. 51, Springer, Berlin, 2006, pp. 209–236.