AMG for Linear Systems in Engine Flow Simulations

Maximilian Emans

AVL List GmbH, Hans-List-Platz 1, 8020 Graz, Austria
maximilian.emans@gmx.at

Abstract. The performance of three fundamentally different AMG solvers for systems of linear equations in CFD simulations using SIMPLE and PISO algorithm is examined. The presented data is discussed with respect to computational aspects of the parallelisation. It indicates that for the compressible subsonic flows considered here basic AMG methods not requiring Krylov acceleration are faster than approaches with more expensive setup as well as recently presented k-cycle methods, but also that these methods will need special treatment for parallel application.

1 Introduction

The three-dimensional simulation of combustion engines requires the approximate solution of the Navier-Stokes equations and an energy equation for unsteady compressible flows in terms of pressure, temperature, and velocity fields. Turbulence treatment, models for combustion processes, formation of NOx and soot e.g., are attached to this kernel. Contemporary simulation tools such as AVL FIRE$^{(\mathrm{R})}$ 2009 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 computing times at an acceptable level.

A common approach to solve the Navier-Stokes equations in this context is the family of algorithms derived from SIMPLE, a general template of algorithms for pressure linked equations. For the solution of the appearing systems of linear equations a fast and sufficiently robust solver is needed. AMG methods are a reasonable option here. However, the development of these methods has not yet ceased and fundamentally new ideas have just been published recently, e.g. by De Sterck et al. [3] and Notay [8]. In this contribution we shall compare the performance of deliberately chosen AMG algorithms applied as linear solvers for systems that appear in the simulation of mainly subsonic flow at the example of a combustion engine. In contrast to related work, e.g. of Čiegis et al. [1] or Starikovičius [12] who have devised a detailed analysis of a single solver algorithm along with a method to predict the parallel performance, we shall merely observe the performance of different algorithms and discuss the opposed effects of increased parallel overhead and decreased memory requirement (per processor) with a growing degree of parallelism onto the effective computing time.

2 Pressure-Correction Equation

Any algorithm for the solution of the Navier-Stokes equations has to cater for the non-linearity of this system and has to provide a feasible way to couple the equations. Common in commercial tools are the iterative algorithms SIMPLE and PISO, both ensuring the pressure-velocity coupling by the solution of a pressure-correction equation which drives the velocity field towards the condition imposed by the continuity equation through an appropriate correction of the pressure field. For details of the SIMPLE algorithm we refer to Patankar [10], the extension to compressible flows has been provided by Demirdžić [2]; for the PISO algorithm see Issa [6]. PISO can be considered a refined variant of SIMPLE: Where in the correlation between pressure-correction and velocity update of SIMPLE certain terms are simply neglected, the same terms are approximated by an additional iteration in PISO, requiring the solution of linear systems, such that the latter algorithm guarantees continuity within each (outer) PISO iteration. The additional linear systems differ only in the right-hand side.

The most time consuming step of both algorithms is the approximate solution of these pressure-correction equations which have positive definite system matrices for the cases presented here. We know from experience that a reduction of any norm of the residual by a factor between 100 and 10000 is sufficient to allow the SIMPLE or PISO algorithm to converge; this distinguishes our task from the majority of applications of linear equation solvers that are characterised by a much lower residual tolerance.

3 AMG Algorithms

In this section we refer to papers providing detailed descriptions of the algorithms under examination in this contribution and describe our own necessary amendments e.g. for parallelisation.

3.1 General Aspects of Parallel AMG

The fundamental AMG algorithm is not repeated here; for this we refer the reader to the literature, e.g. to the appendix of Trottenberg et al. [13]. We follow the Galerkin approach, see Trottenberg et al. [13], to compute the coarse-grid operator. Given a system matrix A and a restriction operator R we use R^T as interpolation operator and compute the coarse-grid operator A^C explicitly as

$$A^C = RAR^T. \tag{1}$$

Accepting a deterioration of the parallel convergence of the algorithms, we allow only local (with respect to the domain assigned to a certain processor) interpolation to avoid the necessity to transfer parts of the matrix A and to reduce the data transfer during setup. A full parallelisation of the solution phase is also not feasible since the most appropriate smoothers are inherently sequential. The usual practice is to employ fully parallel Jacobi on the boundaries between two domains after the values on the boundaries have been exchanged, and to continue with a Gauß-Seidel scheme for the interior points. This technique is referred to as hybrid Gauß-Seidel smoother, see van Emden and Meier-Yang [4].

3.2 AMG Based on Smoothed Aggregation – ams1cg

This method divides the set of fine-grid points into a number of disjoint subsets that are called aggregates. These aggregates become the coarse-grid points. The tentative interpolation operator with constant interpolation is smoothed by application of one Jacobi step along the paths of the graph of the fine-grid matrix to improve the quality of the interpolation. This serial Smoothed-Aggregation method is described in detail in Vaněk et al. [15]. In the parallel case we do not permit aggregates that range over more than one subdomain and restrict the smoothing to local points since more complex approaches do not seem necessarily to result in better performance, in particular not for low processor numbers, see Tuminaro and Tong [14]. We follow the suggestion of Fujii et al. [5] and start the aggregation process at points adjacent to inter-domain boundaries.

This AMG method is used as a preconditioner for the conjugate gradient algorithm, see Saad [11]. We implemented a v-cycle scheme with two pre- and two post-smoothing hybrid Gauß-Seidel sweeps. In the parallel case grids with less than 200 nodes are merged to one of the neighbours. The equation system of the coarsest grid is solved directly by Gaußian elimination.

3.3 Aggregation-Based AMG with Krylov-Acceleration – amk1fc

This method has recently been suggested by Notay and Vassilevski [8]. To ensure that the computation of the coarse-grid operator is cheap, it splits the set of fine-grid nodes into aggregates of four nodes and uses constant interpolation. For details of the algorithm we refer to Notay [9].

In the solution phase of this algorithm the standard v-cycle is replaced by an adaptive algorithm that approximates the solution of the coarse-grid systems by one or two iterations of a Krylov-subspace method that is recursively preconditioned by itself, see again Notay [9] for details. Due to the adaptive preconditioning the "flexible conjugate gradient" method of Notay [7] (restarted after six iterations) is employed instead of a standard conjugate gradient method. Again two pre- and two post-smoothing hybrid Gauß-Seidel sweeps are performed. The grid hierarchy is complete once one grid has less than 200 cells, while the particular coarse-grid treatment suggested by Notay [9] is not applied since its parameters appeared to be somewhat arbitrary. The equations system of the coarsest grid is solved by a parallel block Gauß-Seidel with four iterations.

3.4 Basic AMG – amggs2

The most obvious application of algebraic (and also geometric) multigrid is its use as a "stand-alone" solver rather than as a preconditioner. As a matter of fact none of the AMG methods described above will show good performance when used as "stand-alone" solver, since convergence is poor. Better results are obtained with methods where the quality of the interpolation is rather high and the computation of the coarse-grid operator is very efficient. The second requirement precludes the application of expensive interpolation methods such that constant interpolation will be the method of choice. Then the first requirement can only be met if the number fine-grid points per aggregate or C-point is very low, e.g. 2 or 4.

For the coarse-grid selection in our basic AMG we use the algorithm of Notay [9] that produces aggregates comprising not more than two fine-grid nodes since it is documented in an excellent manner. The cycling scheme is an F-cycle, see Trottenberg et al. [13], i.e. recursively a w-cycle is followed by a v-cycle. Only two hybrid Gauß-Seidel sweeps are performed after the return to the finer grid, i.e. no pre-smoothing is done. The coarse-grid treatment is the same as that of the algorithm amk1fc.

4 Computations of Flows in an Engine

Our test cases are taken from the 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 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. Since a single simulation run on a parallel computer will still take a few hours computing time, we pick out four short periods of a few time steps, one from each of these four strokes of the cycle. Typical data of the engine cycle is shown in figure 1, the geometry and slices through the meshes can be seen in figure 2.

4.1 Description of the Numerical Experiments

The computationally relevant information about the cases is compiled in table 1 that contains e.g. the memory size of the matrix information, the time step dt,

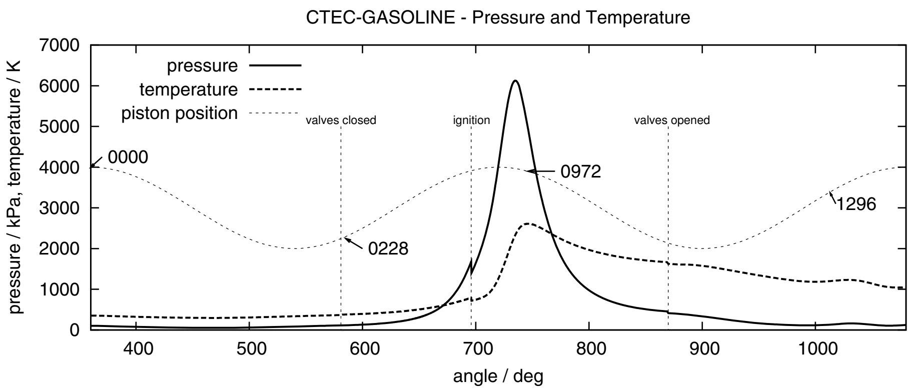

Fig. 1. Scheme of the engine cycle along with notation for the considered partial problems

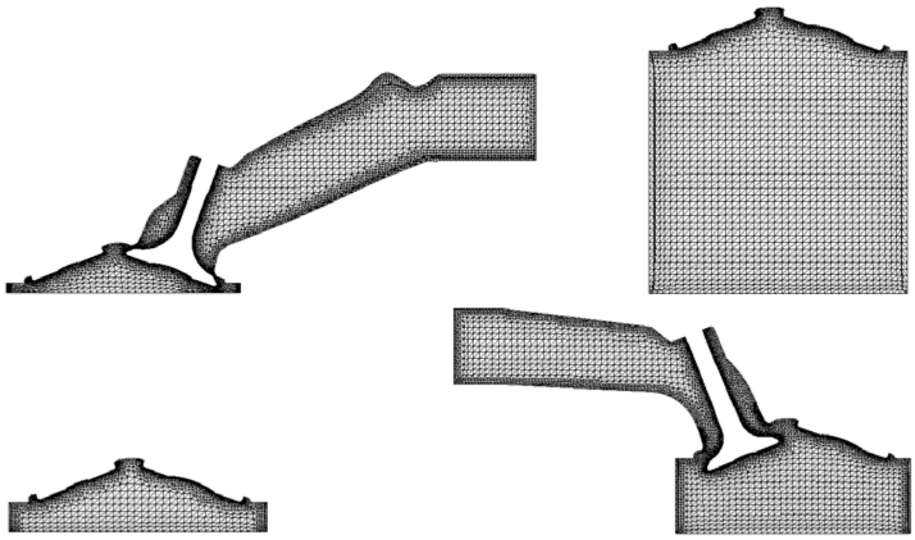

Fig. 2. Slices through the 3-dimensional meshes of the partial problems, from left top clockwise: load, compression, combustion, discharge

Table 1. Characterisation of the cases

no.	stroke	angle	size [MB]	dt [s]	n_t	boundaries	n_{sy} SIMPLE	n_{sy} PISO	n_{se} PISO
0000	load	360°	111.0	$3.03 \cdot 10^{-5}$	5	mass flow, wall	130	120	52
0228	compression	585°	23.0	$3.03 \cdot 10^{-5}$	20	wall	172	224	69
0972	combustion	746°	18.6	$6.06 \cdot 10^{-6}$	20	wall	378	512	201
1296	discharge	1013°	52.1	$3.03 \cdot 10^{-5}$	15	pressure, wall	175	130	58

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 (for SIMPLE it equals n_{sy}).

The AMG algorithms described in the previous section are compared to each other and to a reference, a conjugate gradient solver with incomplete Cholesky factorisation as preconditioner, referred to as ichpcg. We ran each case for each number of processors 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 however was different.

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 an Infiniband (Mellanox) network with an effective bandwidth of approximately 750 Gbit/s. The part of

Table 2. Number of iterations, operator complexity, and number of levels (in brackets) of the computations with the linear AMG solvers

Case n_p	1	4	16		1	4	16		1	4	16	
	amslcg			$c/(n_l)$	amk1fc			$c/(n_l)$	amggs2			$c/(n_l)$
0000	416	446	484	1.51	776	733	731	1.57	569	568	564	2.56
0000P	419	447	477	(3)	724	684	683	(7)	538	525	524	(14)
0228	316	348	400	1.52	480	464	464	1.59	419	417	424	2.60
0228P	427	498	534	(3)	673	659	659	(6)	555	554	574	(11)
0972	788	1125	1134	1.50	1414	1376	1313	1.59	1148	1149	1147	2.60
0972P	1534	1664	1817	(3)	2734	2211	1929	(6)	1761	1759	1747	(11)
1296	372	430	455	1.53	570	532	594	1.60	447	451	499	2.62
1296P	308	373	393	(3)	454	444	486	(6)	402	412	447	(12)

the program related to the solver was compiled by Intel-FORTRAN compiler 10.1, the communication is performed through calls to hp-MPI subroutines (C-binding). The test cases were run within the environment of the software AVL FIRE$^{(R)}$ 2009 on 1, 2, 4, 8, and 16 processors, where the domain decomposition was performed once for each case by the standard algorithm provided with this software. Computations with 1, 2, and 4 processors were done on a single node, for 8 and 16 processors we used 2 and 4 nodes respectively such that each processor had full access to 4 MB L2-cache since in preliminary experiments it has been found that the L2-cache is 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.

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 3 and 4. Furthermore, we present the operator complexity

$$c = \sum_{(l)} \frac{\text{number of matrix elements of level } l}{\text{number of matrix elements of level 1}}, \tag{2}$$

and the cumulative iteration count in table 2. 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 processors. The values of E_p for SIMPLE and PISO are very similar; for SIMPLE they may be found in figures 3 and 4.

4.2 Performance with SIMPLE Algorithm

The curves are influenced by two main opposed effects. On the one hand, E_p becomes worse as the number of processors 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

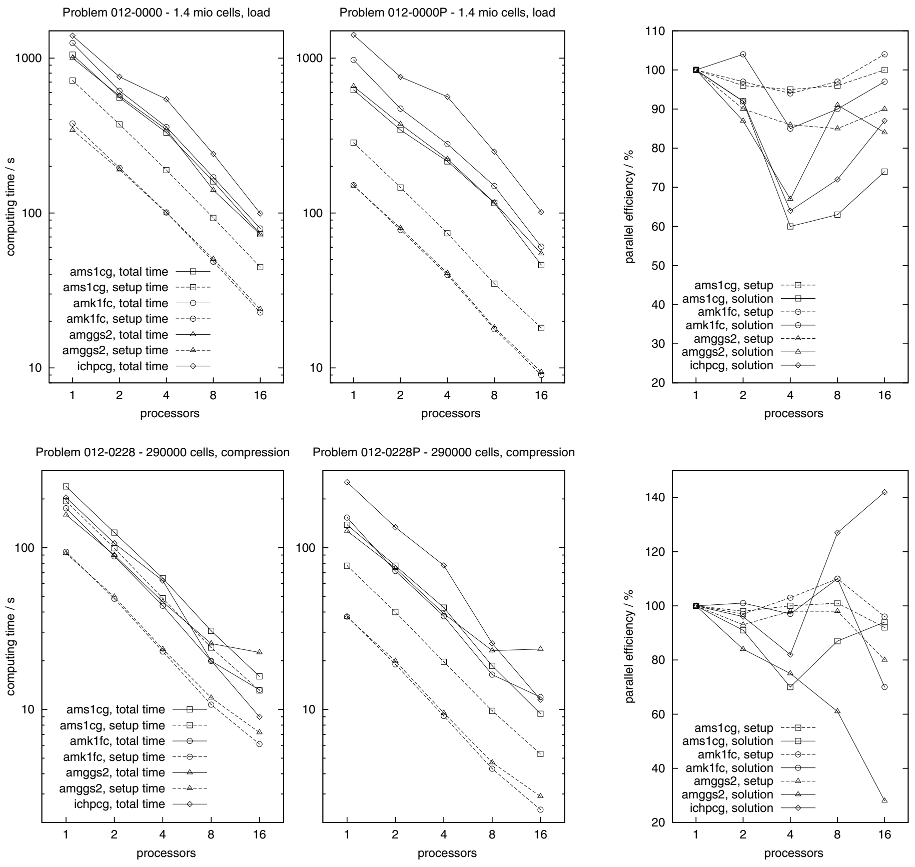

Fig. 3. Computing times for cases 0000 and 0228, using SIMPLE (left) and PISO (middle), and parallel efficiency for solution and setup phase (right) for SIMPLE

et al. [1] 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 processors 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 processors. The third reason is the deterioration of the convergence that is due to imperfect parallelisation; it leads to an increase in the number of iterations of the solver and is independent of the hardware. This effect is essentially only seen for algorithm ams1cg.

The opposed effect is a superlinear acceleration of the computation of the smaller distributed problems due to a decreased probability of cache misses. It is also very hard to predict this effect quantitatively, for a detailed description of a simple test case we refer to Čiegis et al. [1]. Here, the visible consequence is that

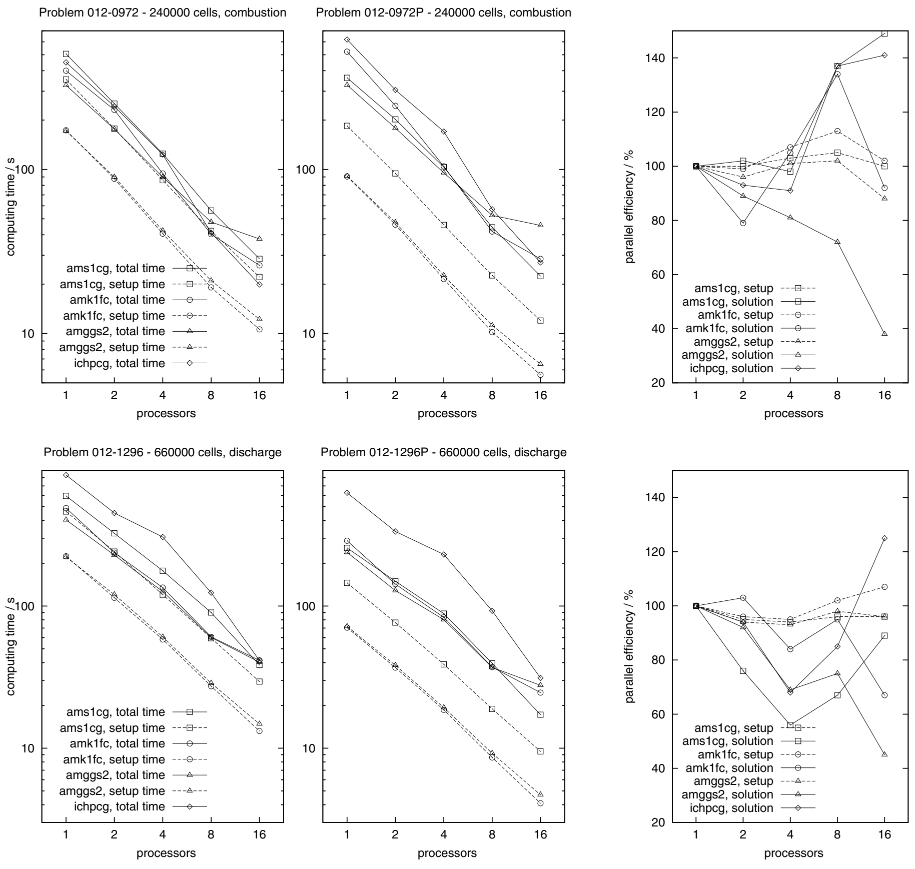

Fig. 4. Computing times for cases 0972 and 1296, using SIMPLE (left) and PISO (middle), and parallel efficiency for solution and setup phase (right) for SIMPLE

the parallel efficiency of certain algorithms rises for computations on more than four cores although one would expect it to decrease due to degraded convergence and additional communication cost. The parallel efficiency may exceed 100% in cases where the gain through cache effects is stronger than the loss through parallelisation. Whereas for algorithm ams1cg the obviously positive and negative effects onto the run-time partly annihilate each other and the positive ones prevail, for algorithm amggs2, characterised by the highest number of levels and the largest memory consumption, only in the largest case (0000) the positive effects dominate over the negative ones.

The significant increase of E_p of amk1fc for two processors is due to the instruction of Notay [9] 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. The parallel efficiency of amggs2 and amk1fc for 16 processors does not drop in the largest case 0000 the same way as in the other cases since the ratio of communication cost to cost of other tasks is significantly lower in case 0000; the reason is that this case is much larger than the other cases which entails a larger amount of internal work.

As mentioned, algorithm ams1cg is most vulnerable to the simplification of the parallel setup leading to degradation of the convergence in parallel runs; while this affects the performance only marginally, acceleration through reduction of cache misses and comparatively low impact of communication overhead cater for good parallel efficiency. However, for processor numbers lower than eight this algorithm is slower than the other ones since its setup is expensive.

In all four cases the basic AMG amggs2 is the fastest algorithm for computations on up to four processors, amk1fc comes close in some cases, whereas ams1cg is significantly slower (note the logarithmic scale in figures 3 and 4). For computations on more than four processors the parallelisation reduces the computing time reasonably for ams1cg, but not for amggs2 and amk1fc.

4.3 Performance with PISO Algorithm

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 the portion of setup time of the AMG solvers is reduced dramatically as expected since expensively computed coarse-grid hierarchies can be reused several times. As a consequence, ichpcg is significantly slower now than these algorithms for up to eight processors. For the same reason, the difference between ams1cg and amggs2 for up to four processors has been reduced. The performance of the new algorithm amk1fc lies between that of the other two AMG algorithms and ichpcg. Similar as for SIMPLE, amggs2 shows deficiencies at high processor numbers.

5 Conclusions

If this kind of problem is to be solved on less than eight processors, the fastest AMG algorithm of our selection is a basic multigrid method with a very lean setup and without Krylov-acceleration. Even if we apply PISO to permit the reuse of the grid hierarchies, algorithms with expensive setup such as ams1cg are not observed to be significantly quicker. For runs using more than eight processors, or if the coarse-grid hierarchy can be used several times, however, the Smoothed Aggregation algorithm should be preferred due to its good parallel efficiency.

Whereas the Smoothed Aggregation algorithm is slightly affected by the simplifications of the parallelisation, the basic multigrid method suffers from parallel overhead that is due to the high number of levels. It shares this principal problem with the recently presented k-cycle AMG of Notay [9]. Modifications to this kind of algorithms are needed such that they can be used efficiently for parallel computations if it is not possible to reuse the coarse-grid hierarchy. Finally, Smoothed Aggregation AMG is a good choice for modern Linux-clusters whenever the coarse-grid hierarchy can be used more than once.

References

1. Čiegis, R., Iliev, O., Lakdawala, Z.: On Parallel Numerical Algorithms for Simulating Industrial Filtration Problems. Computational Methods in Applied Mathematics 7, 118–134 (2007)
2. Demirdžić, I., Lilek, Ž., Perić, M.: A Collocated Finite Volume Method for Predicting Flows at All Speeds. International Journal for Numerical Methods in Fluids 27, 1029–1050 (1993)
3. De Sterck, H., Falgout, R.D., Nolting, J.W., Meier-Yang, U.: Distance-two Interpolation for Parallel Algebraic Multigrid. Numerical Linear Algebra with Applications 15, 115–139 (2008)
4. van Emden, H., Meier-Yang, U.: BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner. Applied Numerical Mathematics 41, 155–177 (2001)
5. Fujii, A., Nishida, A., Oyanagi, Y.: Evaluation of Parallel Aggregate Creation Orders: Smoothed Aggregation Algebraic Multigrid Method. In: Proc. of the Workshop on High Performance Computational Science and Engineering, HPCSE 2004, Toulouse, France, pp. 99–122 (2004)
6. Issa, R.I.: Solution of the Implicit Discretised Fluid Flow Equations by Operator Splitting. Journal of Computational Physics 62, 45–60 (1985)
7. Notay, Y.: Flexible Conjugate Gradients. SIAM Journal of Scientific Computing 22, 1444–1469 (2000)
8. Notay, Y., Vassilevski, P.S.: Recursive Krylov-based Multigrid Cycles. Numerical Linear Algebra with Applications 15, 473–487 (2008)
9. Notay, Y.: An Aggregation-based Algebraic Multigrid Method, Report No. GANMN 08-02, Service de Métrologie Nucléaire, Université Libre de Bruxelles (2000)
10. Patankar, S.V.: Numerical Heat Transfer and Fluid Flow. Hemisphere Publishing (1980)
11. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
12. Starikovičius, V., Čiegis, R., Iliev, O., Lakdawala, Z.: A parallel solver for the 3D simulation of flows through oil filters. In: Čiegis, R., Henty, D., Kagstrom, B., Žilinskas, J. (eds.) Parallel Scientific Computing and Optimization. Advances and Applications. Springer Optimization and Its Applications, vol. 27, pp. 181–192 (2009)
13. Trottenberg, U., Oosterlee, C., Schüller, A.: MULTIGRID. Elsevier Academic Press, London (2001)
14. Tuminaro, R.S., Tong, C.: Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines. In: Proc. of 2000 ACM/IEEE Conference on Supercomputing, Dallas, USA, article no. 5 (2000)
15. Vaněk, P., Brezina, M., Mandel, J.: Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. Computing 56, 179–196 (1996)