Proceedings of ALGORITMY 2012
pp. 141–150

AGGREGATION SCHEMES FOR K-CYCLE AMG

MAXIMILIAN EMANS*

Abstract. We examine different aggregation schemes for k-cycle AMG. With real-life examples from computational fluid dynamics, we confirm that using aggregates of mainly four nodes results in very efficient methods. However, we show also that the k-cycle aggregation AMG in its known form becomes even more efficient, if simpler aggregation schemes skipping the explicit calculation of intermediate coarse-grid operators are employed.

Key words. algebraic multigrid, fluid dynamics, finite volumes

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

1. Introduction.

Algebraic multigrid (AMG) based methods reflect the level of the art for sparse linear solvers in many areas of scientific computing, e.g. in computational fluid dynamics (CFD). The most recent development in the field of AMG solvers is the use of Krylov-accelerated aggregation methods (k-cycle methods), where we consider the paper of Notay [10] as a milestone. In previous examinations it could be shown that the performance of these methods is either equivalent or superior to that of conventional AMG, see Emans [3, 5]. Both, the adaptive preconditioning, and the Krylov-acceleration contribute to the increased efficiency of this class of methods.

In his original paper, Notay [10] suggests a particular aggregation technique that is efficient, easy to implement, and robust, also for parallel calculations. It is based on a pairwise aggregation. But in computational fluid dynamics, the methods of choice for the AMG solvers use a different aggregation strategy that allows for aggregates of different sizes, see e.g. Darwish et al. [1] or Weiss et al. [12]. In the present paper we compare systematically different aggregation techniques, starting from the structure of the multigrid operators that are obtained by different methods, continuing with aspects like memory requirement, and finishing with a performance analysis of benchmarks in CFD from engineering applications.

2. Aggregation AMG.

Within a Krylov method, the AMG algorithm shall be applied as a preconditioner to a symmetric positive definite or semi-definite system which we denote as

$$A\vec{x} = \vec{b} \tag{2.1}$$

where $A \in \mathbb{R}^{n\times n}$ is regular, $\vec{b} \in \mathbb{R}^n$ is some right-hand side vector and $\vec{x} \in \mathbb{R}^n$ the solution; n is the number of unknowns.

Any variant of the AMG algorithm requires the definition of a grid hierarchy with l_{max} levels $A_l \in \mathbb{R}^{n_l\times n_l}$ $(l = 1, ..., l_{max})$ with system size n_l where $n_{l+1} < n_l$ holds for $l = 1, ..., l_{max} - 1$ and $A_1 = A$ as well as $n_1 = n$. As it is common practice in algebraic multigrid, the coarse-grid operators are defined, starting on the finest grid, recursively by

$$A_{l+1} = P_l^T A_l P_l \qquad (l = 1, ..., l_{max} - 1). \tag{2.2}$$

*Johann Radon Institute for Computational and Applied Mathematics (RICAM), and Industrial Mathematics Competence Center (IMCC) GmbH, both Altenbergerstrasse 69, 4040 Linz, Austria (maximilian.emans@ricam.oeaw.ac.at).

where the prolongation operator P_l has to be determined for each level l while the restriction operator is P_l^T. It is the choice of the coarse-grid selection scheme that determines the elements of all P_l and consequently the entire grid hierarchy. The determination of the elements of P_l for all levels and the computation of the operators A_l ($l = 2, ..., l_{max}$) are referred to as setup phase of AMG.

2.1. Aggregation techniques. The class of aggregation methods we consider here splits the number of nodes on the fine grid into disjoint sets of nodes, the so-called aggregates that act as nodes on the coarse grid. The mapping from the coarse grid to the fine grid is than achieved by simply assigning the coarse-grid value of the aggregate to all the fine-grid nodes belonging to this aggregate. This corresponds to a constant interpolation; the consequence is that there is only one non-zero entry in each row of P_l with the value one such that the evaluation of eqn. (2.2) simplifies to an addition of rows of the fine-grid operator. This methods are particularly simple. To distinguish them from the Smoothed Aggregation method, that further refine the prolongation operator, see Vaněk et al. [11], we refer to this aggregation technique as plain aggregation.

2.1.1. Pairwise aggregation. The first step of the double-pairwise aggregation that has been suggested by Notay [10] to be used in k-cycle AMG, is the aggregation of the set of nodes into aggregates of pairs of nodes. For this Algorithm 1 is used.

For the pairwise aggregation that we refer to as algorithm NP2, the output of Algorithm 1, i.e. the aggregates G_i ($i = 1, ..., n_{l+1}$), is used to define the prolongation operator P_l. The calculation of the elements of the coarse-grid matrix $A_{l+1} = P_l^T A_l P_l$ with eqn. (2.2) is implemented as the addition of two rows in two steps: First, the rows are extracted from the matrix storage structure in a way that the corresponding elements of the data array are put in a single array and the row pointers in another array of the same size. Second, after the column pointers are replaced by the indices of

Algorithm 1 Pairwise aggregation (by Notay [10], simplified version)

```
Input:           Matrix $A = (a_{ij})$ with $n$ rows.
Output:          Number of coarse variables $g$ and aggregates $G_i$, $i = 1, ..., g$ (such
                 that $G_i \cap G_j = \emptyset$ for $i \neq j$).
Initialisation:  $U = \{i \in [1, n] | \exists a_{ik} \neq 0 \quad \text{with} \quad i \neq k\}$
                 $\forall i$: $S_i = \{j \in U \setminus \{i\} \mid a_{ij} < -0.25 \cdot \max_k |a_{ik}|\}$,
                 $\forall i$: $m_i = | \{j | i \in S_j\} |$,
                 $g = 0$.
Algorithm:
                  1: while $U \neq \emptyset$ do
                  2:   select $i \in U$ with minimal $m_i$;
                  3:   $g \leftarrow g + 1$
                  4:   select $j \in U$ such that $a_{ij} = \min_{k \in U} a_{ik}$
                  5:   if $j \in S_i$: $G_g = \{i, j\}$, else $G_g = \{i\}$
                  6:   $U \leftarrow U \setminus G_g$
                  7:   for all $k \in G_g$ do
                  8:     if $l \in S_k$: $m_l \leftarrow m_l - 1$
                  9:   end for
                 10: end while
```

the corresponding coarse-grid aggregates, both arrays have been sorted with respect to the new column pointers where matrix elements with the same column pointer are added. The parallel version of the method restricts the aggregates to nodes belonging to the same parallel domain.

For the double-pairwise aggregation, the aggregates G_i $(i = 1, ..., n_{l+1})$ are used to define the intermediate prolongation operator P_{l1}. With this, the elements of the corresponding intermediate coarse-grid matrix $A_{l+1/2} = P_{l1}^T A_l P_{l1}$, are determined exactly as in the pairwise aggregation. In order to obtain the matrix A_{l+1}, the double-pairwise aggregation applies the same procedure a second time, this time with input $A_{l+1/2}$ instead of A_l for Algorithm 1 which gives rise to the prolongation operator P_{l2}. A_{l+1} is calculated as $A_{l+1} = P_{l2}^T A_{l+1/2} P_{l2}$. The final prolongation operator is formally $P_l = P_{l2} P_{l1}$. Since it contains only the information to which aggregate a fine-grid node is assigned, it is sufficient to store it as an array of size n_l carrying the index of the coarse-grid node. The operators $A_{l+1/2}$, P_{l1}, and P_{l2} are discarded after A_{l+1} has been calculated. We refer to this method as NP4.

While the double-pairwise aggregation mainly forms aggregates of four fine-grid nodes, it is easily possible to extend the concept in such a way that a third pairwise aggregation (of the pairs-of-pairs) is added. This way aggregates of up to eight nodes are formed. The calculation of the coarse-grid operator requires the explicit calculation of two intermediate coarse-grid operators $A_{l+1/3} = P_{l1}^T A_l P_{l1}$ and $A_{l+2/3} = P_{l2}^T A_{l+1/3} P_{l2}$ where P_{l2} is the prolongation operator constructed on $A_{l+1/3}$; the final coarse-grid operator is then $A_{l+1} = P_{l3}^T A_{l+2/3} P_{l3}$ where P_{l3} is constructed on $A_{l+2/3}$. The prolongation operator on level l is $P_l = P_{l3} P_{l2} P_{l1}$. The operators apart from P_l and A_l are discarded. We refer to this triple-pairwise aggregation method as NP8.

2.1.2. Plain aggregation. This algorithm tries to form larger aggregates by a sort of greedy mechanism. In a first step it puts strongly connected neighbours into aggregates while in two further steps it is attempted to form additional aggregates out of the remaining unassigned nodes or to join them to existing ones, see Algorithm 2. It follows closely the aggregation algorithm used by Vaněk et al. [11] with the essential difference that we restrict the number of nodes per aggregate which gives rise to the parameter γ of this algorithm. In lines 12 and 23 we allow twice the number of nodes γ in order to avoid a large number of single-point aggregates. Usually only a few such enlarged aggregates are formed. In parallel, only nodes assigned to the same process are grouped into aggregates.

2.2. Multigrid cycling. Not long ago, Notay [10] has combined several ideas of previous years into a efficient novel multigrid scheme which is referred to as k-cycle. It has been shown in previous publications that the concept of this k-cycle AMG is particularly well suited for the requirements of important applications such as computational fluid dynamics, see e.g. Emans [3, 5]. Other than conventional multigrid methods that obtain a coarse-grid correction by recursive application of the multigrid method, the k-cycle employs a Krylov method on each level of the grid hierarchy and uses the grid hierarchy for the preconditioning of this Krylov method. Apart from the larger robustness of this scheme due to the additional Krylov method, the major advantage of the k-cycle is that it is adaptive: Either one or two iterations of the Krylov method on each grid level are done – comparable either to a v-cycle or a w-cycle in conventional AMG. The k-cycle is used as a preconditioner for the flexible conjugate gradient method, see Notay [9].

Although the aggregation variants discussed above can be combined with various

Algorithm 2 Simple aggregation (see Vaněk et al. [11], limited number of nodes per aggregate)

Input: Matrix $A = (a_{ij})$ with n rows, nodes per aggregate γ, connectivity threshold β

Output: Number of coarse variables g and aggregates G_i, $i = 1, ..., g$ (such that $G_i \cap G_j = \emptyset$ for $i \neq j$).

Initialisation: $U = \{i \in [1, n] | \exists a_{ik} \neq 0 \quad \text{with} \quad i \neq k\}$
$\forall i$: $S_i = \{j \in U \setminus \{i\} \mid a_{ij} < -\beta \cdot \max_{(k)} |a_{ik}|\}$,
$g = 0$.

Algorithm:

```
1:  i=1
2:  while $U \neq \emptyset$ and $i \leq n$ do
3:      if $S_i \neq \emptyset$: $g \leftarrow g + 1$
4:      for all $j \in S_i$ do
5:          if $|G_g| < \gamma$: $G_g \leftarrow G_g \cup \{j\}$; $U \leftarrow U \setminus \{j\}$
6:      end for
7:      while $i \notin U$ and $i < n$: $i \leftarrow i + 1$
8:  end while
9:  i=1
10: while $U \neq \emptyset$ and $i \leq n$ do
11:     while $i \notin U$ and $i < n$: $i \leftarrow i + 1$
12:     $J := \{j | \exists h : j \in (G_h \cap S_i), |G_h| < 2 \cdot \gamma\}$
13:     if $J \neq \emptyset$ then
14:         select $h$ such that $k \in G_h$ and $|a_{ik}| = \max_{k \in J} |a_{ik}|$
15:         $G_h \leftarrow G_h \cup \{i\}$; $U \leftarrow U \setminus \{i\}$
16:     end if
17: end while
18: i=1
19: while $U \neq \emptyset$ and $i \leq n$ do
20:     while $i \notin U$ and $i < n$: $i \leftarrow i + 1$
21:     $g \leftarrow g + 1$
22:     for all $j \in (S_i \cap U)$ do
23:         if $|G_g| < 2 \cdot \gamma$ : $G_g \leftarrow G_g \cup \{j\}$; $U \leftarrow U \setminus \{j\}$
24:     end for
25: end while
```

AMG components, we restrict ourselves for the purpose of this paper to the examination of k-cycle AMG with several of these aggregation variants. The smoother we apply in our schemes is a symmetric Gauß-Seidel smoother with two pre-smoothing sweeps and two post-smoothing sweeps. The parallel implementation relies on a domain decomposition; the values associated to nodes of external domains are exchanged at once, i.e. for the affected connections a Jacobi-like smoothing takes place. The parallel coarse-grid problem is treated by an agglomeration scheme, for details see Emans [4].

3. Applications. In this section we will discuss the algorithmic and computational properties of different aggregation schemes that are particularly well suited for the k-cycle AMG. These are, among the double-pairwise aggregation that is typically used together with the k-cycle AMG, see Notay [10], the triple-pairwise aggregation NP8, and plain aggregation schemes with a low limit of nodes per aggregate. In the first part of this section we will examine the grid hierarchy that results from different definitions of the prolongation operator. In the second part, we present results of benchmarks consisting of several linear systems of two typical CFD applications, focusing on computing time and convergence.

We apply the solvers within the framework of the CFD package FIRE$^{(R)}$ 2011, developed and distributed by AVL GmbH, Graz. This software calculates an approximate solution of the Navier-Stokes equations. The spatial discretisation is a 2nd order accurate finite volume scheme on unstructured meshes. The computationally most expensive task within this scheme is the solution of the pressure-correction equation. The matrix of this system is symmetric positive-definite for weakly compressible flow and symmetric semi-definite for incompressible flows, see Emans [2]. For this system the discussed AMG methods are employed. The problems we consider are real-world problems that stem from engineering application of such a software. Since it would require a too large amount of computational resources without bringing additional information, we do not run the calculation until a final solution of the system is obtained, but we stop the calculation after a certain number of SIMPLE iterations in the case of steady problems or a certain number of time steps in the case of time-dependent problems.

3.1. Algorithmic properties. The diagrams in Figure 3.1 show the maximum and average number of non-zero matrix elements of some plain aggregation methods with different γ and of the aggregation schemes derived from the pairwise aggregation. The latter type of aggregation schemes (NP2, NP4, and NP8) generally leads to leaner operators than the plain aggregation scheme with the corresponding γ. The reason is that the plain aggregation leads to a certain number of aggregates not having precisely γ nodes per aggregate. The pairwisely aggregating schemes, in contrast, produce aggregates that contain almost exclusively 2, 4, or 8 fine-grid nodes per aggregate, respectively. The fact that the pairwisely aggregating algorithms lead to grid hierarchies with one level more than the plain aggregation with the same number of nodes is due to presence of aggregates with more than 4 and 8 nodes per aggregate in the splittings of A004 and A008, respectively.

The grid hierarchies are characterised by the grid complexity c_g and the operator complexity c_o. The grid complexity is defined by

$$c_g := \sum_{l=1}^{l_{max}} n_l/n_1 \tag{3.1}$$

and the operator complexity by

$$c_o := \sum_{l=1}^{l_{max}} m_l/m_1 \tag{3.2}$$

where m_l is the number of non-zero entries of the matrix on level l ($l = 1, ..., l_{max}$). The grid complexity is a measure for the number of unknowns that needs to be recalculated during the multigrid cycle. With regard to the computational cost, the

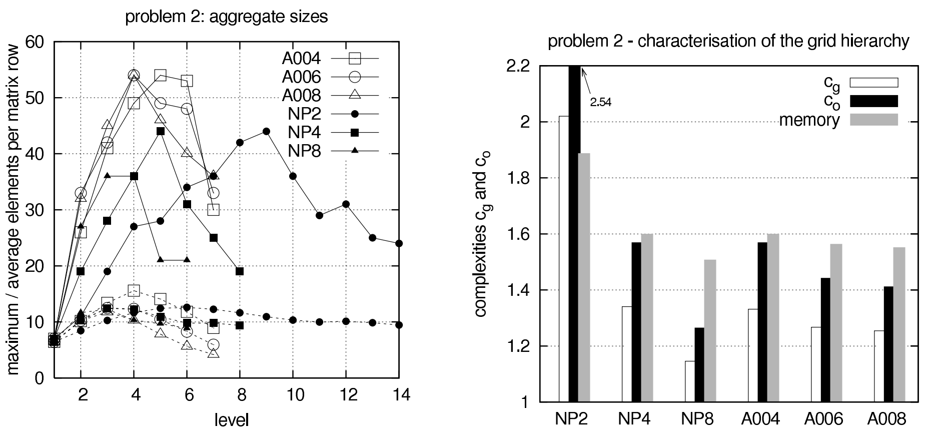


FIG. 3.1. *Left: number of non-zero elements per matrix row on different levels of the grid hierarchy obtained with different aggregation algorithms for problem 2, see section 3.2: maximum aggregate sizes (solid lines), average aggregate sizes (dashed lines); right: grid and operator complexity, memory requirement*

operator complexity is more relevant since it measures roughly the cost (in terms of both, elementary computational operations and memory requirement) of the multigrid cycle. However, we monitor the memory requirement also directly. In Figure 3.1 we show the grid complexity, the operator complexity, and the memory requirement for selected coarsening schemes applied to the first linear system solved in problem 2. Those schemes that have very small aggregates have generally a relatively high operator complexity, see the decreasing memory requirement for the pairwisely aggregating schemes and the plain aggregation schemes. The schemes NP4 and A004 are similar in all three values. Comparing NP8 and A008, NP8 is more favourable in terms of all three values.

3.2. Benchmarks. Problem 1 is a steady flow through the intake geometry of an engine cylinder. The flow is driven by a pressure difference that is imposed directly as a pair of boundary conditions. In this case, we consider the pressure-correction equations of the first five iterations of the SIMPLE algorithm. Problem 2 is part of an unsteady compressible full engine computation. In the period of time of the simulation that we consider, the cylinder is loaded by fresh air. A prescribed mass flow, known from experiments, flows into the cylinder while the piston starts to move downward from its top position. The mass flow gives rise to a Dirichlet boundary condition for the velocity. Besides this, only solid, but partially moving, wall boundary conditions are involved. In this case three time steps are calculated; the accumulated number of SIMPLE iterations is 88, i.e. 88 different linear systems are solved.

Table 3.1 gives an overview over the most important settings of the benchmark problems. In both problems, a standard k-ε turbulence model, see Jones and Launder [7], is added to the system of the Navier-Stokes equations. For both problems, the initial guess for each linear system that is solved by our AMG algorithms in this benchmark is the zero vector. The solver reduces the residual by a factor of 200 and then terminates.

3.2.1. Convergence. Figure 3.2 shows the cumulative iteration counts for the runs with selected aggregation schemes. The curves reveal that the difference in the

TABLE 3.1
Overview over the calculation cases

	problem 1: intakeport	problem 2: cylinder
fluid	air	air
initial condition	potential flow	rest
boundary conditions	wall, pressure	velocity, wall
discretisation	0.7 mio cells	1.4 mio cells
number of systems	5	88
mesh: hex/tet/other	91.4 / 0.2 / 8.4 %	1.1 / 80.1 / 18.8 %

convergence rate, i.e. the number of iterations, of the methods with aggregation based on the formation of pairs (NP4 and NP8) on the one side and the plain aggregation schemes on the other side, is generally small. Moreover, not surprisingly, the convergence is better if the number of nodes per aggregate is smaller.

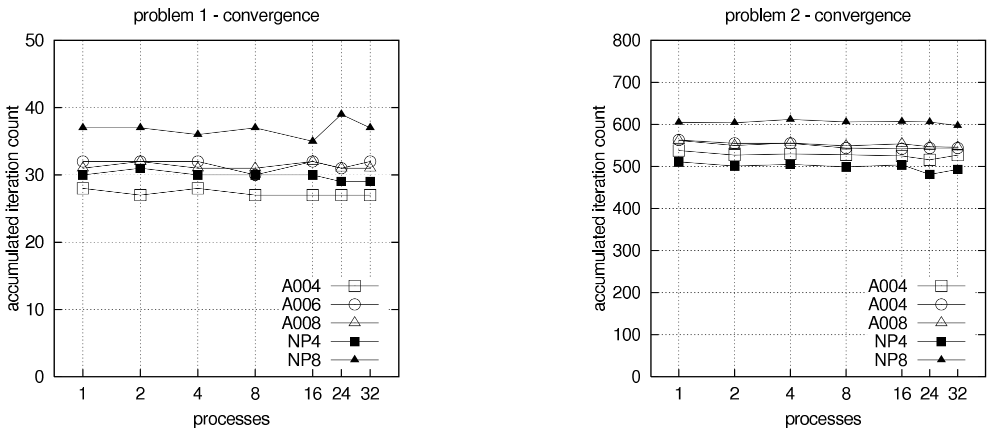


FIG. 3.2. *Cumulative iteration counts for problem 1 (left), for problem 2 (right)*

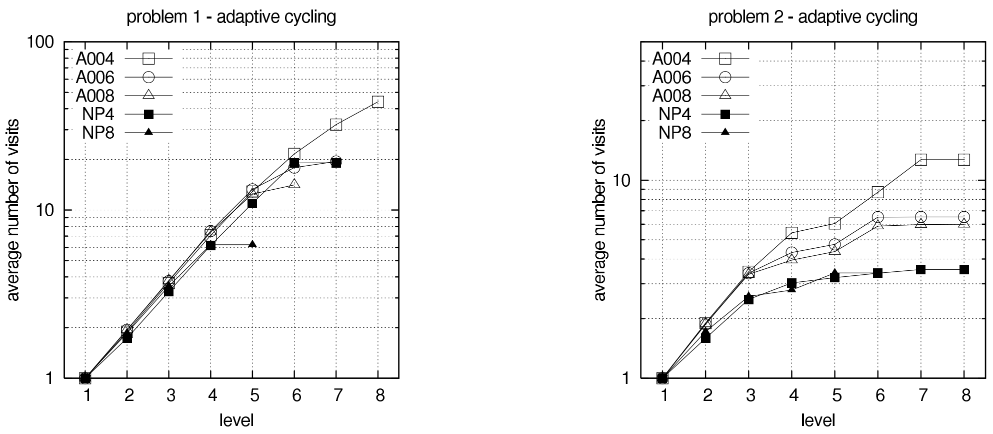


FIG. 3.3. *Cycling (average number of visits at the individual grids) for problem 1 (left), for problem 2 (right)*

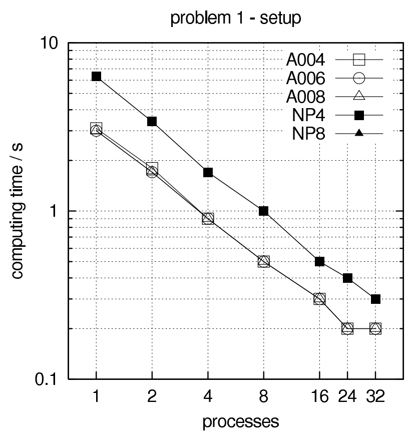
The substitution of NP4 by e.g. A004 in the k-cycle algorithm does not significantly worsen the global convergence in our problems, see Figure 3.2. It makes, however, the adaptive cycling of the k-cycle more expensive. To show this, we have plotted the averaged number of visits on each grid level l versus the grid level in Figure 3.3. The data indicates that the adaptivity of the k-cycle can compensate the slightly worse properties of the plain aggregation grids (compared to the grids of the pairwise aggregation schemes).

3.3. Computational cost. The problems were calculated on a cluster where each node is equipped with two Intel quad-cores Xeon X5365 (clock frequency 3.00 GHz, L2-cache 4 MB shared between two cores, Frontside-bus for on-chip data transfer) per node. The nodes of this machine are connected by a 4xDDR Infiniband interconnect of a previous generation. The effective bandwidth between two single processes on different nodes is around 750 MB/s and the latency is around 3.3 μs. We located up to eight processes on one node of the cluster.

The diagrams in Figure 3.4 show the time that is spent to setup the grid hierarchy. It is interesting to compare the setup times of the algorithms NP4 and NP8 to that of A004 and A008. Although the properties of the grid hierarchy appear to be similar, in particular in the case of A004 and NP4, the setup of A004 is around twice as fast as that of NP4. This is due to the fact that for NP4 an intermediate coarse-grid operator ($A_{l+1/2}$) is calculated. Since the convergence of these two algorithms is essentially the same, this indicates that using A004 instead of NP4 could be advantageous.

The diagrams in Figure 3.5 analyse the solution phase. One iteration is in general the faster, the lower the operator complexity of the method is. This is significant if we compare e.g. NP8 (with the lowest operator complexity) to A008 and the other plain aggregation methods, and also to NP4. Note that the scaling of the solution part of the k-cycle methods appears to be very favourable. Only from 8 to 16 parallel processes, a small bump in the timing curves is visible – this is due to the usage of the node-to-node interconnect which is not needed for smaller number of processes.

Finally the total computing times (time spent for the solver, i.e. setup and solution phase) is shown in Figure 3.6. With regard to the k-cycle algorithm, it can be seen that using the plain aggregation scheme A004 instead of the double-pairwise



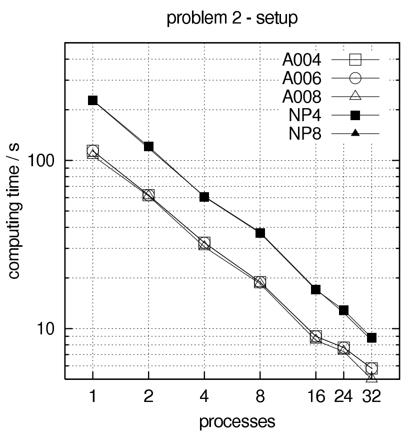
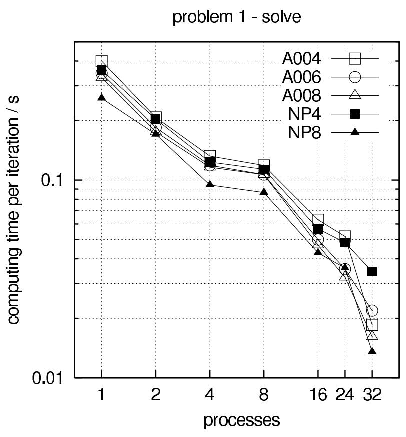


FIG. 3.4. *Computing time of setup phase for problem 1 (left), for problem 2 (right)*



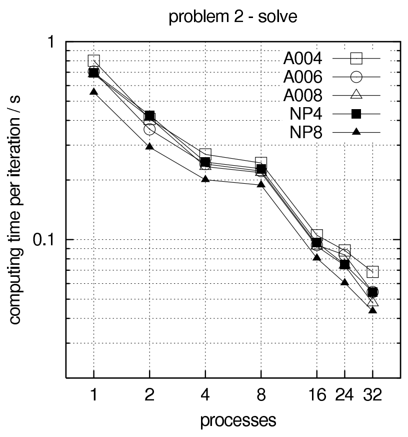
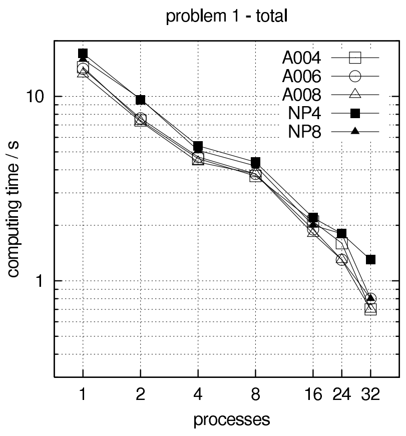


FIG. 3.5. *Computing time per iteration of solution phase for problem 1 (left), for problem 2 (right)*



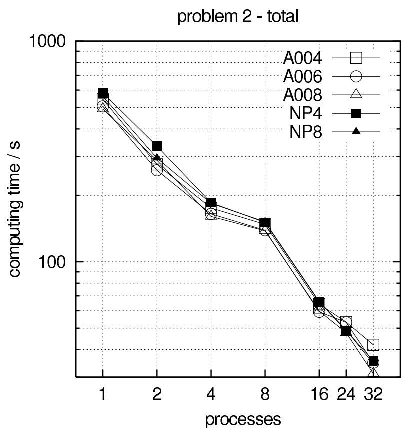


FIG. 3.6. *Total computing time for problem 1 (left), for problem 2 (right)*

aggregation improves the performance of the algorithm, mainly due to the skipped calculation of the intermediate operators $A_{l+1/2}$. Using A008 appears to be even more efficient than A004. This modification of Notay's k-cycle [10] contributes to the reduction of the setup time: It is therefore more relevant in situations where the setup phase significantly contributes to the total computing time. This is the case e.g. for implementations where the setup phase cannot be implemented as efficiently as the solution phase like on graphics processing units (GPU), see e.g. Haase et al. [6]. Moreover, we can confirm that using aggregates of around four nodes is a good choice with regard to the performance: While, on the one hand, smaller aggregates (e.g. NP2) lead to a too high operator complexity with obvious disadvantages, larger aggregates lead to methods with worse convergence such that the cost per iteration does not compensate the cost of the higher number of iterations in practical applications.

4. Conclusion and outlook.

In our examples, the double-pairwise aggregation scheme could be replaced by a plain aggregation scheme with a low number of nodes per aggregate. The convergence of the methods with the plain aggregation scheme is slightly worse, but the setup is almost twice as fast. The savings are therefore significant such that the methods with the plain aggregation scheme are in total faster. It will be interesting to compare the mathematical properties of the splittings obtained by both classes of aggregation techniques by the method recently provided for this purpose by Napov and Notay [8].

Acknowledgements. Part of this work was supported by the government of Upper Austria within the strategic program "Innovatives Oberösterreich 2010plus".

REFERENCES

[1] M. Darwish, T. Saad, and Z. Hamdan. Parallelization of an additive multigrid solver. *Numerical Heat Transfer*, 54:157–284, 2008.

[2] M. Emans. Efficient parallel amg methods for approximate solutions of linear systems in CFD applications. *SIAM Journal on Scientific Computing*, 32:2235–2254, 2010.

[3] M. Emans. Performance of parallel AMG-preconditioners in CFD-codes for weakly compressible flows. *Parallel Computing*, 36:326–338, 2010.

[4] M. Emans. Coarse-grid treatment in parallel AMG for coupled systems in CFD applications. *Journal of Computational Science*, 2:365–376, 2011.

[5] M. Emans. Krylov-accelerated algebraic multigrid for semi-definite and nonsymmetric systems in computational fluid dynamics. *Numerical Linear Algebra with Applications*, 19:210–231, 2012.

[6] G. Haase, M. Liebmann, C. Douglas, and G. Plank. A parallel algebraic multigrid solver on graphical processing unit. In W. Zhang, Z. Chen, C.C. Douglas, and W. Tong, editors, *High Performance Computing and Applications 2010*, volume 5938 of *Lecture Notes in Computer Science*, pages 38–47. Springer-Verlag Berlin Heidelberg, 2010.

[7] W.P. Jones and B.E. Launder. The prediction of laminarization with a two-equation model of turbulence. *International Journal of Heat and Mass Transfer*, 15:301–314, 1972.

[8] A. Napov and Y. Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM Journal on Scientific Computing*, 2012.

[9] Y. Notay. Flexible conjugate gradients. *SIAM Journal on Scientific Computing*, 22(4):1444–1460, 2000.

[10] Y. Notay. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis*, 37:123–146, 2010.

[11] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing 56*, pages 179–196, 1996.

[12] J.M. Weiss, J.P. Maruszewski, and W.A. Smith. Implicit solution of preconditioned Navier-Stokes equations using algebraic multigrid. *AIAA Journal*, 37(1):29–36, 1999.