NON-GALERKIN MULTIGRID BASED ON SPARSIFIED SMOOTHED AGGREGATION *

ERAN TREISTER† AND IRAD YAVNEH†

Abstract. Algebraic Multigrid (AMG) methods are known to be efficient in solving linear systems arising from the discretization of partial differential equations and other related problems. These methods employ a hierarchy of representations of the problem on successively coarser meshes. The coarse-grid operators are usually defined by (Petrov-)Galerkin coarsening, which is a projection of the original operator using the restriction and prolongation transfer operators. Therefore, these transfer operators determine the sparsity pattern and operator complexity of the multigrid hierarchy. In many scenarios the multigrid operators tend to become much denser as the coarsening progresses. Such behavior is especially problematic in parallel AMG computations where it imposes an expensive communication overhead. In this work we present a new algebraic technique for controlling the sparsity pattern of the operators in the AMG hierarchy, independently of the choice of the restriction and prolongation. Our method is based on the aggregation multigrid framework, and it "sparsifies" Smoothed Aggregation operators while preserving their right and left near null-spaces. Numerical experiments for problems of convection-diffusion and diffusion with discontinuous coefficients demonstrate the efficiency and potential of this approach.

Key words. Non-Galerkin Multigrid, Aggregation Multigrid, Smoothed Aggregation, Parallel AMG, Matrix Sparsification, Convection-Diffusion.

AMS subject classifications. 65N55, 65F08, 65N22

1. Introduction.

Multigrid methods are well known for their efficiency in solving linear systems arising from the discretization of elliptic partial differential equations (PDEs) [14, 44, 19, 52]. The discretization yields a sparse, large linear system,

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$, and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. AMG methods use two complementary components: *relaxation*, and *coarse-grid correction* (CGC). The relaxation is a simple and local iterative method, such as Jacobi or Gauss-Seidel, and is usually inefficient in handling certain error modes, called "algebraically smooth", which relate to the low energy eigenvectors of A. CGC aims at handling these modes, and is performed by solving smaller-size coarse-grid problems of the form,

$$A_c\mathbf{e}_c = \mathbf{r}_c. \tag{1.2}$$

In most AMG methods, problem (1.2) is defined by the Galerkin or Petrov-Galerkin coarsening,

$$(RAP)\mathbf{e}_c = R(\mathbf{b} - A\mathbf{x}), \tag{1.3}$$

which is a projection of the error equation, $A\mathbf{e} = \mathbf{b} - A\mathbf{x}$, onto the subspace defined by the full-rank prolongation and restriction operators, $P \in \mathbb{R}^{n\times n_c}$ and $R \in \mathbb{R}^{n_c\times n}$, respectively, with $n_c < n$[1].

*This research was funded (in part) by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

Eran Treister is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship.

† Department of Computer Science, Technion—Israel Institute of Technology, Haifa, Israel. (eran@cs.technion.ac.il, irad@cs.technion.ac.il.).

[1] In this paper we target both symmetric and nonsymmetric problems, and we therefore use the more general Petrov-Galerkin form RAP, where R is not necessarily P^T. For brevity, we use the term "Galerkin" also in this case.

Suppose that $\mathbf{e}_c$ is the solution of the smaller system (1.2). Then, we aim for $P\mathbf{e}_c$ to be a good approximation to the error of the original fine-grid system. The two-level CGC error-propagation operator is now given by

$$E_{TG} = I - P(A_c)^{-1}RA. \tag{1.4}$$

The coarsening process is applied recursively, employing successively coarser systems. A drawback of Galerkin coarsening is that the sparsity of RAP is determined by P and R. This often leads to relatively dense coarse-grid operators with increased operator complexity (defined as the total number of non-zeros in the entire multigrid hierarchy divided by the number of zeros in the fine-grid matrix). Thus, we must often compromise between quality of coarse-level operators and the aggressiveness of coarsening, both of which affect the rate of convergence and the operator complexity.

In addition, algorithms that employ high-quality transfer operators, such as classical AMG or Smoothed Aggregation (SA), typically lead to a severe stencil growth on coarse grids, even when the operator complexity remains moderate. That is, certain rows of the coarse-grid matrix become dense, even if the overall density of the matrix is moderate. In large-scale parallel computing, stencil growth leads to high communication overhead on coarse grids [37, 34] and loss of scalability, especially in 3D. In [38] it is suggested to apply aggressive coarsening on the finer levels to reduce the complexities. Such coarsening is defined by generalizing the strength of connection between neighboring variables to short-distance paths. Similar coarsening schemes were introduced in [37]. With this approach it is necessary to modify the prolongation. In particular, in the classical AMG interpolation there cannot be strong connections between $\mathcal{F}$-points without a mutual neighboring $\mathcal{C}$-point. This rule breaks if aggressive coarsening is applied, and to overcome this [50, 18] suggest modifications to support long-range interpolation. This in turn reduces the quality of the prolongation, and the resulting algorithms typically require more iterations to converge. For more information about parallel AMG see [4, 17, 1, 2, 3] and references therein.

Recently, there has been an effort to develop multigrid algorithms that explicitly control the sparsity pattern of the multigrid hierarchy [47, 43], or sparsify the Galerkin operators [34]. These ideas have yet to reach their full potential and were only applied for symmetric problems. We follow a similar framework, and propose a new method that is applicable also to non-symmetric problems. Other related works include [39, 26, 25], which apply similar sparsification techniques for solving graph-Laplacian problems that arise from computer vision applications. In these methods, the multilevel transfer operators are defined by a Schur complement, or "exact interpolation", and the resulting coarse operators are sparsified to maintain a reasonable coarsening rate and operator complexity.

To demonstrate our algorithm on non-symmetric problems, we focus on the convection-diffusion equation

$$-\epsilon\Delta u + \mathbf{v}\cdot\nabla u = f, \tag{1.5}$$

that is commonly studied in the context of geometric and algebraic multigrid [8, 9, 51, 53, 20, 13, 33, 30]. One popular AMG approach to treating this problem is by using aggregation-based AMG methods [45, 20, 11, 13, 33, 28, 29, 30], where the coarsening is done by clustering (aggregating) the grid unknowns. In the simple aggregation method (AGG), P and R are typically sparser than those obtained by most other AMG approaches, and the operator complexity of the multigrid hierarchy is usually

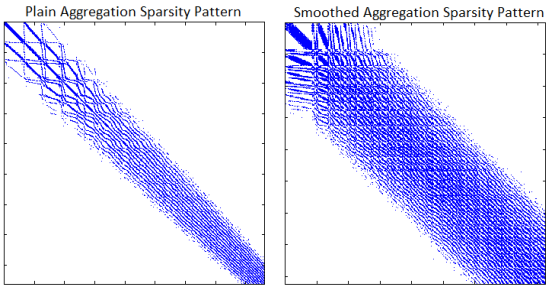

FIG. 1.1. *A comparison between the sparsity patterns of AGG and SA on level 3, for the 3D Poisson equation on a* 64^3 *grid, using a 7-point stencil. The left matrix corresponds to the AGG algorithm; It is of size* 6538×6538*, and contains* $99,878$ *non-zeros—about 15 non-zeros per row. The right matrix corresponds to the SA algorithm, and is smaller because of the more aggressive coarsening. It is of size* 1645×1645*, and contains* $120,299$ *non-zeros—about 73 non-zeros per row.*

well-bounded and attractive. However, it is difficult to obtain grid independent convergence using this approach, and therefore the approach of *Smoothed Aggregation* (SA) [45, 11, 13, 33, 22] is often preferred. In SA we smooth the simple aggregation operators by a relaxation operator. This improves the convergence properties of the multigrid solver, but it also increases the operator complexity. Therefore, when using SA we must make sure that our coarsening is aggressive enough to prevent exaggerated stencil growth on coarse grids. However, for a convection-dominated problem (1.5), such aggressive aggregation coarsening (say 3×3 for 2D problems on structured meshes) significantly harms the quality of the aggregation operators compared to moderate coarsening (say, 2×2) [20]. The latter, on the other hand, leads to an unbounded operator complexity using SA. We note that there are more sophisticated methods to improve SA by changing the values of the transfer operators P and R. For example, "energy-minimization" methods [27, 48, 10, 31, 35] optimize the non-zero entries of the transfer operators for some constrained energy measure given a predefined sparsity pattern. This improves the quality of the transfer operators without increasing the operator complexity. Also related are the adaptive and bootstrap AMG methods [12, 11, 13, 7] that involve adaptive learning of algebraically smooth errors.

As mentioned before, multigrid algorithms often exhibit a severe stencil growth on coarse grids, even though their operator complexity may remain moderate. Figure 1.1 compares between the sparsity patterns of AGG and SA on the third level, both using the aggregation algorithm of [45], for the solution of the 3D Poisson equation on a 64^3 cubic grid, using a 7-point stencil. The coarsening of SA is more aggressive, but the matrix of AGG is sparser than that of SA, where the stencil growth is evident. This stencil growth further increases on coarser grids. Nevertheless, both algorithms exhibit a well bounded operator complexity (1.25 and 1.6 for AGG and SA respectively). In other numerical results we observe much worse scenarios. We note that the stencil growth of SA in Figure 1.1 can be prevented if $3 \times 3 \times 3$ aggregates are chosen for the finest grid instead of the neighborhood aggregation of [45]. Then, a 27-point stencil is maintained on all levels.

In the recent paper [30], problem (1.5) is solved using plain aggregation with a moderate coarsening of factor close to 4. To overcome the slow convergence, the multigrid process includes acceleration on all levels of the hierarchy, requiring a more elaborate recursive structure (usually W-cycles with recursive Krylov accelerations, known as K-cycles). However, such cycles may be costly, especially when considering

parallel multigrid computations [37, 16].

In this paper, we present a new AMG algorithm that controls the sparsity pattern of the multigrid hierarchy, using ideas related to [43, 34]. The new algorithm is developed for non-symmetric problems but it preserves the symmetry property of the coarse-grid operators for symmetric fine-grid operators. Our algorithm uses the aggregation framework as a basis platform, by applying SA to define the prolongation and restriction, and AGG Galerkin coarse-grid operators for defining a sparse non-zero pattern of A_c. Once the transfer operators and the target sparsity pattern are set, our algorithm sparsifies the Galerkin SA coarse-grid operator to match the chosen sparsity pattern of A_c.

2. Motivation and Background. Similarly to [47, 43], we fix the sparsity pattern of A_c and, independently, use high quality transfer operators P and R. Denote the Galerkin operator by

$$A_g = RAP,$$

and assume that the current error, $\mathbf{e}$, is in the range of P, i.e., $\mathbf{e} = P\mathbf{e}_c$, where $\mathbf{e}_c \in \mathbb{R}^{n_c}$ is some coarse vector. Also, assume that our coarse operator, A_c, is constructed such that $A_c\mathbf{e}_c = A_g\mathbf{e}_c$. Then, the two-level cycle using A_c eliminates the error $\mathbf{e}$:

$$\begin{array}{rcccccc} \mathbf{e}_{new} & = & \left[I - P(A_c)^{-1}RA\right]\mathbf{e} & = & [I - P(A_c)^{-1}RA]P\mathbf{e}_c & & \\ & = & P[I - (A_c)^{-1}A_g]\mathbf{e}_c & = & P(A_c)^{-1}[A_c - A_g]\mathbf{e}_c & = & 0. \end{array} \tag{2.1}$$

Because A_c has fewer non-zeros than A_g, the equality $A_c\mathbf{e}_c = A_g\mathbf{e}_c$ cannot hold for all $\mathbf{e}_c$. However, because the CGC (coarse-grid correction) is applied after the relaxation, the error $\mathbf{e}$ before the CGC is typically smooth, and so is its coarse version $\mathbf{e}_c$. Motivated by this property and (2.1), [47, 43] suggest to define the operator A_c such that $A_c\mathbf{e}_c = A_g\mathbf{e}_c$ is satisfied with respect to algebraically smooth errors, which are not reduced efficiently by the relaxation. If A_c and A_g yield a similar coarse grid correction for those errors, the efficiency of our solution process will not degrade due to the use of A_c instead of A_g. Additionally, note that multigrid algorithms rely on the property that the algebraically smooth errors are in the range of P, and therefore they influence the definition of P. Therefore, when applying the non-Galerkin approach, the characterization of the smooth error modes should already be available to us from the construction of P. In some cases, the algebraically smooth errors are known. In the context of discretizations of scalar PDEs, they correspond to the constant and linear functions.

In addition, several existing algorithms show that the large Galerkin stencil is not necessary, at least for simple cases. The oldest of those is the geometric multigrid, where the PDE on the fine grid is re-descretized on coarse grids, retaining the stencil of the fine-grid operator. This approach is limited to numerical solution of PDEs on structured grids. Additionally, [28, 29, 30, 32, 36, 5, 46] apply simple aggregation, along with multilevel acceleration. This approach relies on the fact that the *two-level* AGG cycle has a mesh-independent convergence, and the convergence deteriorates only in multilevel cycles. This may suggest that the coarse grid *sparsity patterns* of AGG may suffice for mesh-independent convergence, so long as we adapt the values of the nonzero coefficients. This motivates our choice of using AGG for the sparsity pattern of A_c.

All the algorithms in [47, 43, 34, 39, 26, 25, 6] apply a sparsification mechanism within some multilevel framework, aiming to reduce the number of non-zeros of the

coarse-grid operators while maintaining "spectral equivalence" to the original coarse operator. The simplest of those is [6], which is limited to circulant matrices. It shows how to generate spectrally equivalent coarse operators using 5-pt stencils instead of 9-pt stencils in 2D, or 7-pt stencils instead of 27-pt stencils in 3D. The simplest example is as follows: let the fine level A be the 5-pt Laplacian operator, then the classical AMG coarsening generates a 9-pt stencil on the coarse grid (A_g). This stencil can be replaced by a spectrally equivalent 5-pt stencil (A_c) as follows:

$$A: \begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}, A_g: \frac{1}{64}\begin{pmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{pmatrix}, A_c: \frac{1}{16}\begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}. \quad (2.2)$$

This equivalence can be also seen from the fact that both A_g and A_c are discretizations of the '$-\Delta$' operator with an equal scaling of $\frac{1}{16}$, which comes from the multiplication of A with P and P^T. We note that our sparsification method in this paper yields the same A_c, as do the methods of [47, 25]. Paper [34] shows a condition for the non-Galerkin method to work, which is related to (2.1). It shows that spectral equivalence between the two SPD coarse operators can be measured by

$$\|I - (A_c)^{-1}A_g\|_2. \quad (2.3)$$

If (2.3) is smaller than 1, then the two-level cycle with A_c instead of A_g converges, and the performance of the two cycles becomes more similar as (2.3) tends to 0.

In the next section we describe our Sparsified Smoothed Aggregation (SpSA) framework, which uses SA for the transfer operators, and AGG for the sparsity pattern. Our version of SA is similar to the classical SA [45] except a few small changes that are summarized in Appendix A. Even though these are small, in some cases they dramatically change the properties of the SA solver compared to the classical one. Generally, our version of SA has better convergence, but also has a higher operator complexity than the original SA method. To set the stage for our framework, we first briefly describe the general SA method.

2.1. Smoothed Aggregation. An aggregation is a partitioning of the fine-grid index set $\{1, ..., n\}$ into n_c disjoint subsets $\{C_j\}_{j=1}^{n_c}$, called aggregates. Given these aggregates, a piece-wise constant tentative prolongation operator, which we denote by P_t (t for tentative), is defined:

$$(P_t)_{i,j} = \begin{cases} 1 & i \in C_j, \\ 0 & otherwise. \end{cases} \quad (2.4)$$

$R_t = P_t^T$ is the tentative restriction operator. We define our aggregation by a neighborhood approach, similar to the classical one of [45]. Other aggregation methods as in [41, 42, 29, 30] may also be suitable for our algorithm.

As mentioned above, we use SA to improve the convergence of AGG, by smoothing the operators P_t and R_t. More precisely, the SA operators P and R are defined by

$$P = (I - \omega Q A^F)P_t, \qquad R = R_t(I - \omega A^F Q), \quad (2.5)$$

where the matrix Q is a diagonal preconditioner of A, ω is a damping parameter, and A^F is a filtered version of the matrix A. The specific details of the aggregation algorithm, and specific choices of Q, ω, and A^F, are summarized in Appendix A.

3. The Sparsified Smoothed Aggregation (SpSA) Algorithm. Our algorithm, similarly to all the other non-Galerkin algorithms mentioned above, consists of three separate tasks:

1. Choose the transfer operators R and P.
2. Choose the sparsity pattern for A_c.
3. Calculate the entries in A_c.

The first step is essential to every algebraic multigrid method and the other two are needed only in non-Galerkin AMG.

As mentioned above, our algorithm uses the aggregation framework. For the first task, it uses transfer operators R and P based on SA as in (2.5). For the second and third tasks, we use two Galerkin operators

$$A_t = R_t A P_t, \quad \text{and} \quad A_g = RAP, \tag{3.1}$$

that are based on the non-smoothed and smoothed aggregation operators in (2.4) and (2.5), respectively. Once the target pattern A_t, and the Galerkin SA operator A_g are set, our algorithm sparsifies A_g to match the chosen sparsity pattern of A_t; the result is the coarse operator A_c. This setup process is described in Algorithm 1.

Algorithm: SpSA-Setup

1. Define the tentative prolongation P_t and restriction $R_t = P_t^T$.
2. Define SA operators P, R (see (2.5)).
3. Apply Galerkin Coarsening: $A_t = R_t A P_t$, $A_g = RAP$.
4. Sparsify A_g to the sparsity pattern of A_t: $A_c = \mathbf{Sparsify}(A_g, A_t)$
5. Apply recursion on A_c to generate the next levels.

Algorithm 1: Sparsified Smoothed Aggregation (SpSA) Setup

3.1. Sparsity Patterns in the Aggregation Framework. The operators in (2.5) and (3.1) have some unique properties that are related to their sparsity patterns. These will be used in our sparsening algorithm, described later. We denote the sparsity pattern of any matrix A as

$$\mathcal{S}_{\mathcal{P}}(A) = \{(i,j) : A_{i,j} \neq 0\}. \tag{3.2}$$

By (2.5), we have that

$$\mathcal{S}_{\mathcal{P}}(A_g) \supseteq \mathcal{S}_{\mathcal{P}}(A_t), \tag{3.3}$$

up to chance cancellations of elements which we ignore in our description. In addition, since $\mathcal{S}_{\mathcal{P}}(A) = \mathcal{S}_{\mathcal{P}}(I - QA)$ for any diagonal matrix Q, we have that

$$\mathcal{S}_{\mathcal{P}}(A_t) \supseteq \mathcal{S}_{\mathcal{P}}(RP_t) \quad \text{and} \quad \mathcal{S}_{\mathcal{P}}(A_t) \supseteq \mathcal{S}_{\mathcal{P}}(R_t P), \tag{3.4}$$

with equalities in the case of no filtering in (2.5).

As mentioned above, the tentative operators (2.4), and their associated Galerkin operator, may be efficient when used directly in a multigrid process with acceleration. This suggests that the sparsity pattern of A_t will not miss important non-zero entries. For example, if the graph of A_g is connected, i.e., there is a path from each node i to node j in the graph, then the graph of A_t is connected as well. To illustrate this, consider the grid-aligned anisotropic Laplacian using 5-point finite-difference

discretization as in [34]. The stencil of the fine-level A, and stencil A_g using classical AMG intepolation with semi-coarsening are given by:

$$A : \begin{pmatrix} & -\epsilon & \\ -1 & 2+2\epsilon & -1 \\ & -\epsilon & \end{pmatrix}, A_g : \frac{1}{4}\begin{pmatrix} -\epsilon & -6\epsilon & -\epsilon \\ -2+6\epsilon & 4+12\epsilon & -2+6\epsilon \\ -\epsilon & -6\epsilon & -\epsilon \end{pmatrix}.$$

Now, it is important that any non-Galerkin operator will have some of the weak ϵ entries in A_g. A stencil of $\hat{A}_c = \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$, for example, would split the graph of A_g into disconnected strong diffusion lines. As shown in [34], this would not be a good choice of a coarse-grid stencil, even though the norm of their difference is small: $\|\hat{A}_c - A_g\|_F = O(\epsilon)$, where $\|\cdot\|_F$ is the Frobenius norm. In our case, the stencil of A_t will always contain some of the non-zeros in the weak directions. More importantly, recall that $(a_t)_{ij} = \sum_{k\in\mathcal{C}_i, \ell\in\mathcal{C}_j} a_{k\ell}$, so if the graph of the fine-grid matrix A is connected, then so is the graph of A_t, given that no chance cancellations of elements occur, as in M-matrices.

3.2. The Sparsening Procedure. We next describe our sparsening procedure used to approximate A_g using the sparsity pattern of A_t. Our process aims to generate a coarse matrix A_c that, on top of its sparsity constraints, has the same product as A_g and $(A_g)^T$ with typical algebraically smooth error modes. As mentioned before, if $A_c\mathbf{e}_c = A_g\mathbf{e}_c$ for any smooth error $\mathbf{e}_c$, the efficiency of our solution process will not degrade due to the sparsification of A_g. The typical smooth error modes are also necessary in the construction of the prolongation and restriction, and in the context of elliptic PDEs, they are often chosen to be represented by the constant vector. Thus, motivated by (2.1), we impose

$$A_g\mathbf{1} = A_c\mathbf{1} \text{ and } (A_c)^T\mathbf{1} = (A_g)^T\mathbf{1}, \tag{3.5}$$

where $\mathbf{1}$ is the vector of ones.

We start by copying the values in the entries of A_g that belong to $\mathcal{S}_{\mathcal{P}}(A_t)$,

$$\textbf{if} \quad (A_t)_{k,i} \neq 0 \quad \textbf{then} \quad (A_c)_{k,i} \leftarrow (A_g)_{k,i}. \tag{3.6}$$

Otherwise, we have an entry satisfying $(A_t)_{k,i} = 0$ and $(A_g)_{k,i} \neq 0$; this is an entry that we wish to eliminate (set $(A_c)_{k,i} = 0$), while maintaining (3.5). Now, by setting $(A_c)_{k,i} = 0$, we break the equalities in (3.5), and thus we need to correct them by changing other entries as well. Our correction will be done by collapsing $(A_g)_{k,i}$ on other entries that correspond to variables that are "strongly connected" to k and i. In this context, the main heuristic that is used in multigrid is that if k and some variable m are strongly connected, then a typical algebraically smooth error $\mathbf{e}$ will satisfy $e_k \approx e_m$. We will use this heuristic in our algorithm. We elaborate on the choice of these entries in the next section.

3.2.1. Obtaining a surrogate path for eliminating the (k,i) entry. First, we remark that we must avoid the simplest choice of entries to make up for an elimination of a (k,i) entry—the diagonal and "mirror" entries, (k,k), (i,i) and (i,k). Considering that we might need to eliminate both $(A_g)_{k,i}$ and $(A_g)_{i,k}$, we have to satisfy four equations: the equalities in (3.5) for both i and k row and column. Since we have only the two diagonal entries (k,k), (i,i) at our disposal (the off-diagonals are zeroed), this task is impossible because we have four equations and only two variables to satisfy them. Thus, we conclude that, generally, additional or other entries need

to be changed. We note, however, that this is not true in the symmetric case where $(A_g)_{k,i} = (A_g)_{i,k}$. Then, (3.5) can be satisfied by adding the following 2×2 block

$$\begin{array}{c} \\ i \\ k \end{array} \begin{array}{c} \begin{array}{cc} i & k \end{array} \\ \begin{pmatrix} +(A_g)_{k,i} & -(A_g)_{k,i} \\ -(A_g)_{k,i} & +(A_g)_{k,i} \end{pmatrix} \end{array} \tag{3.7}$$

which changes only the diagonal entries, and has zero row and column sums. This property has led to various sparsification schemes in [34, 39, 26, 25]. Because we consider both symmetric and non-symmetric problems in this paper, we assume $(A_g)_{k,i} \neq (A_g)_{i,k}$ and do not use (3.7) in our sparsening scheme. In fact, we show that other corrections *must* be applied in this case.

We next describe a set of entries in A_t that are non-zeros and available to use for eliminating $(A_g)_{k,i}$. In Galerkin coarsening, every non-zero entry satisfies

$$(A_g)_{k,i} = \sum_j \sum_\ell (R)_{k,j} A_{j,\ell} (P)_{\ell,i}, \tag{3.8}$$

which means that every entry in A_g is generated by a sum of three-term multiplications. Now, if $(A_t)_{k,i} = 0$ and $(A_g)_{k,i} \neq 0$, then we have at least one non-zero term $(R)_{k,j} A_{j,\ell} (P)_{\ell,i}$ in the right hand side of (3.8), in which ℓ belongs to $\mathcal{C}_{m_1}$ and j belongs to the aggregate $\mathcal{C}_{m_2}$, and at least one of them is different from k and i (note that j and ℓ are fine-grid variables while the rest are coarse-grid variables). Also,

$$(RP_t)_{k,m_2} \neq 0, \quad (R_t P)_{m_1,i} \neq 0, \quad \text{and} \quad (A_t)_{m_2,m_1} \neq 0. \tag{3.9}$$

Furthermore, by (3.4), all these entries also belong to $\mathcal{S}_{\mathcal{P}}(A_t)$, and hence can be used together with their associated diagonal entries to eliminate $(A_g)_{k,i}$. Overall, in order to eliminate $(A_g)_{k,i}$ we add to A_g a submatrix of the form

$$\begin{array}{c} \\ i \\ m_1 \\ m_2 \\ k \end{array} \begin{array}{c} \begin{array}{cccc} i & m_1 & m_2 & k \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ -(A_g)_{k,i} & 0 & \times & 0 \end{pmatrix} \end{array} \tag{3.10}$$

where $\times$ denotes a non-zero entry, and require that it has zero row and column sums. We excluded (i,i) and (k,k) because they are singles in their row and column, respectively, and hence cannot be changed. Because the marked non-zero entries constitute a distance-three path $i \to m_1 \to m_2 \to k$ in A_t, we will denote by (i, m_1, m_2, k) the "surrogate path" for eliminating the (k,i) entry. Figure 3.1 demonstrates this path.

3.2.2. Setting values in the surrogate path for eliminating the (k,i) entry. We now describe how to set the values in the submatrix (3.10) so that (3.5) is satisfied, or equivalently, that (3.10) has zero row and column sums.

By imposing $(A_c)_{k,i} = 0$, or by setting $-(A_g)_{k,i}$ in the (k,i) entry of (3.10), we break the zero sum of the k-th row and i-th column. To satisfy them we must set:

$$\begin{aligned} (A_c)_{m_1,i} &\leftarrow (A_c)_{m_1,i} + (A_g)_{k,i} \\ (A_c)_{k,m_2} &\leftarrow (A_c)_{k,m_2} + (A_g)_{k,i}. \end{aligned} \tag{3.11}$$

Now, the corresponding zeros sum for the m_1-th row and the m_2-th column are broken, so we must satisfy them as well by applying

$$\begin{aligned} (A_c)_{m_1,m_1} &\leftarrow (A_c)_{m_1,m_1} - (A_g)_{k,i} \\ (A_c)_{m_2,m_2} &\leftarrow (A_c)_{m_2,m_2} - (A_g)_{k,i}. \end{aligned} \tag{3.12}$$

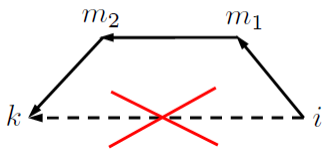

FIG. 3.1. *A surrogate path of* $(A_g)_{k,j}$. *The dashed arrow represents the eliminated entry, while the surrogate path is comprised of solid arrows.*

This again breaks the zero sum for the m_2-th row and m_1-th column, and to finally satisfy them both we must apply

$$(A_c)_{m_2,m_1} \leftarrow (A_c)_{m_2,m_1} + (A_g)_{k,i}. \tag{3.13}$$

Overall, following (3.11)-(3.13), the submatrix (3.10) is given by

$$\begin{array}{c} \\ i \\ m_1 \\ m_2 \\ k \end{array} \begin{array}{c} \begin{array}{cccc} i & m_1 & m_2 & k \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ (A_g)_{k,i} & -(A_g)_{k,i} & 0 & 0 \\ 0 & (A_g)_{k,i} & -(A_g)_{k,i} & 0 \\ -(A_g)_{k,i} & 0 & (A_g)_{k,i} & 0 \end{pmatrix} \end{array}, \tag{3.14}$$

and it has a zero row and column sum.

3.2.3. Partitioning the corrections onto several surrogate paths. In practice, there may be several paths from i to k, so we eliminate $(A_g)_{k,i}$ using all paths simultaneously, each eliminating a portion $0 < \theta_{(i,m_1,m_2,k)} < 1$ of $(A_g)_{k,i}$, where the portions sum to one. The weights $\theta_{(i,m_1,m_2,k)}$ are chosen proportionally to the strength of the connection in the associated path. More specifically, we use

$$\theta_{(i,m_1,m_2,k)} = |(R_tP)_{m_1,i}(A_t)_{m_2,m_1}(RP_t)_{k,m_2}| \tag{3.15}$$

as the strength of the path (i, m_1, m_2, k). However, there are situations where there is a distance-two path between i and k, featuring only one connector, i.e., $m_1 = m_2$. In such cases, which are also computationally easier to find, we choose the weight associated with the path (i, m, m, k) as $|(R_tP)_{m,i}(RP_t)_{k,m}|$ and set the weight of the distance-three paths to 0. We note that if we consider only symmetric problems, then our distance-two correction is similar to that of [25]. A motivation for our choice of the partitioning weight in (3.15) can be found in [40].

The accuracy of our sparsification depends on the local smoothness of the error. We wish to collapse entries on strongly connected neighbors of the two vertices i and k of the sparsified edge A_{ik} (we elaborate on this later in the discussion around (3.21)). For M-matrices, large values in the terms in (3.15) indicate strong connections between each couple along the path. For example, if $|(A_t)_{m1,m2}|$ is relatively large in the row m_1, then m_1 and m_2 are strongly connected. The same is true for R_tP and RP_t. Other definitions for strength of paths may be used instead of (3.15). A precise description of the sparsening algorithm appears in Algorithm 2. Our sparsening process treats the designated non-zero entries one by one, independently of their order, and hence it can be fully parallelized.

REMARK 1 (Treatment of non-constant near null-space). *The way Algorithm 2 is written in this paper, it is suitable only for matrices that have the constant vector as a single near null-space prototype. The same approach can be generalized for*

Algorithm: $A_c \leftarrow$ **Sparsify**(A_g, A_t, RP_t, R_tP)
% A_g - a Galerkin smoothed aggregation operator that needs to be sparsified
% A_t - a Galerkin aggregation operator that defines the desired sparsity pattern
% A_c - SpSA coarse-grid operator. Has the sparsity pattern of A_t.
foreach $(k,i) \in \mathcal{S}_{\mathcal{P}}(A_t)$ **do** $(A_c)_{k,i} \leftarrow (A_g)_{k,i}$.
foreach $(k,i) \in \mathcal{S}_{\mathcal{P}}(A_g) \backslash \mathcal{S}_{\mathcal{P}}(A_t)$ **do**
% Seek a set of distance-two surrogate paths:
$\mathcal{S}_{k\to i} = \{(i,m,m,k) : (R_tP)_{m,i} \neq 0 \quad \& \quad (RP_t)_{k,m} \neq 0\}$.
if $\mathcal{S}_{k\to i} \neq \emptyset$ **then**
foreach $(i,m,m,k) \in \mathcal{S}_{k\to i}$, associate a weight
$\theta_{(i,m,m,k)} = |(R_tP)_{m,i}(RP_t)_{k,m}|$
Normalize the weights: $\theta_{(i,m,m,k)} \leftarrow \left(\sum_p \theta_{(i,p,p,k)}\right)^{-1} \theta_{(i,m,m,k)}$
foreach $(i,m,m,k) \in \mathcal{S}_{k\to i}$ **do**
Define a portion of $(A_g)_{k,i}$: $\delta = (A_g)_{k,i} \cdot \theta_{(i,m,m,k)}$.
Collapse δ onto the path (i,m,m,k):
Add δ to the entries $(A_c)_{m,i}$ and $(A_c)_{k,m}$.
Subtract δ from the entry $(A_c)_{m,m}$.
end
else
% Define a set of distance-three surrogate paths:
$\mathcal{S}_{k\to i} = \{(i,m_1,m_2,k) :$
$(R_tP)_{m_1,i} \neq 0 \quad \& \quad (A_t)_{m_2,m_1} \neq 0 \quad \& \quad (RP_t)_{k,m_2} \neq 0\}$.
foreach $(i,m_1,m_2,k) \in \mathcal{S}_{k\to i}$, associate a weight
$\theta_{(i,m_1,m_2,k)} = |(R_tP)_{m,i}(A_t)_{m_2,m_1}(RP_t)_{k,m}|$
Normalize the weights: $\theta_{(i,m_1,m_2,k)} \leftarrow \left(\sum_{p,q} \theta_{(i,p,q,k)}\right)^{-1} \theta_{(i,m_1,m_2,k)}$
foreach $(i,m_1,m_2,k) \in \mathcal{S}_{k\to i}$ **do**
Define a portion of $(A_g)_{k,i}$: $\delta = (A_g)_{k,i} \cdot \theta_{(i,m_1,m_2,k)}$.
Collapse δ onto the path (i,m_1,m_2,k):
Add δ to the entries $(A_c)_{m_1,i}, (A_c)_{k,m_2}$ and $(A_c)_{m_2,m_1}$.
Subtract δ from the entries $(A_c)_{m_1,m_1}$ and $(A_c)_{m_2,m_2}$.
end
end
end

Algorithm 2: The Sparsening procedure

*matrices with a single **non-constant** near null-space by applying a diagonal scaling, similarly to the way it is applied for the restriction and prolongation in standard adaptive AMG [13, 42]. That is, let* $\mathbf{x}$ *and* $\mathbf{y}$ *be the (positive) right and left near null-space prototypes, respectively, and let* $X = diag(\mathbf{x})$ *and* $Y = diag(\mathbf{y})$ *be diagonal matrices whose diagonal elements are the entries in* $\mathbf{x}$ *and* $\mathbf{y}$ *respectively. Then the re-scaled matrix* YAX *has a constant right and left near null-space prototype and can be treated by Algorithm 2. The case of multiple near null-space prototypes, as in the elasticity problem, requires a different treatment—see [34] for a discussion about this matter in the context of non-Galerkin coarsening.*

Remark 2 (Computational cost). *The computational cost of Algorithm 2 is comparable to the cost of the Galerkin product RAP, because finding a distance 2*

path between two variables is similar to applying an inner product between two sparse vectors. For example, the non-zeros of the product RP correspond to all distance 2 paths in the first branch of Algorithm 2. In a way, the sparsening procedure tracks the three-term products of the Galerkin operator for sparsified entries and updates the entries of A_c accordingly. Our computational savings come from the third level on, since then all matrices are much sparser than in the non-sparsified SA. We note that most of the sparsified non-zeros are treated as distance 2 paths which are cheaper to process.

3.3. Theoretical results. In this section we state some of the theoretical properties of the sparsening procedure.

PROPOSITION 3.1. *If the fine matrix A is symmetric, $R_t = P_t^T$, and $R = P^T$, then A_c is symmetric as well.*

Proof. By the given symmetries and (3.1), we get that A_g and A_t are symmetric, and so are their sparsity patterns. Now, if the entry (k,i) needs to be eliminated, i.e., $(k,i) \in \mathcal{S}_{\mathcal{P}}(A_g) \backslash \mathcal{S}_{\mathcal{P}}(A_t)$, then the entry (i,k) needs to be eliminated as well. Furthermore, if there is a path from i to k such that

$$(P^T P_t)_{k,m_2} \neq 0, \quad (P_t^T P)_{m_1,i} \neq 0, \quad \text{and} \quad (A_t)_{m_2,m_1} \neq 0, \tag{3.16}$$

then the same path exists from k to i in the opposite direction (i.e., via m_2 and m_1), because $(P^T P_t)^T = (P_t^T P)$ and (A_t) is symmetric. Using this path, the submatrix (3.14) that corresponds to the elimination of $(A_g)_{i,k}$ is given by:

$$\begin{array}{c} \\ i \\ m_1 \\ m_2 \\ k \end{array} \begin{array}{c} \begin{array}{cccc} i & m_1 & m_2 & k \end{array} \\ \begin{pmatrix} 0 & (A_g)_{i,k} & 0 & -(A_g)_{i,k} \\ 0 & -(A_g)_{i,k} & (A_g)_{i,k} & 0 \\ 0 & 0 & -(A_g)_{i,k} & (A_g)_{i,k} \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} . \tag{3.17}$$

If we set $(A_g)_{i,k} = (A_g)_{k,i}$ in (3.17) and in (3.14), we can see that they are the transpose of each other. This means, that the sum of (3.14) and (3.17) is a symmetric submatrix under these conditions. Furthermore, under the above symmetries, also each portion $\theta_{(i,m_1,m_2,k)}$ equals $\theta_{(k,m_2,m_1,i)}$ according to (3.15). Thus, when eliminating each portion of a pair (i,k) and (k,i), we add a symmetric submatrix to A_g, and therefore A_c remains symmetric. ∎

In the next proposition, we consider diagonally dominant M-matrices. A matrix A is called an M-matrix if it is of the form of $A = sI - B$, where $B \geq 0$ and $s \geq \rho(B)$ ($\rho(B) = \max_i\{|\lambda_i|\}$ is the spectral radius of B). Furthermore, a matrix A is called diagonally dominant if every row i satisfies $A_{i,i} \geq \sum_{j \neq i} |A_{i,j}|$.

PROPOSITION 3.2. *If A_g is a diagonally dominant M-matrix, then A_c is a diagonally dominant M-matrix as well.*

Proof. Since A_g is a diagonally dominant M-matrix, then for every row k: $(A_g)_{k,k} \geq -\sum_{j \neq k} (A_g)_{k,j}$, or in matrix form: $(A_g)\mathbf{1} \geq 0$, where $\mathbf{1}$ is the vector of ones. By the sparsening construction in Algorithm 2, $(A_c)\mathbf{1} = (A_g)\mathbf{1} \geq 0$. Furthermore, any eliminated off-diagonal entry $(A_g)_{k,i}$ is non-positive, so the submatrix (3.14) for replacing it has only non-positive off-diagonal entries, except the (k,i) entry which cancels. Also, its diagonal entries are non-negative. This means that the sign-structure of A_c still corresponds to an M-matrix. This, together with $(A_c)\mathbf{1} \geq 0$

means that (A_c) is diagonally dominant. Finally, by the Gerschgorin theorem, A_c is also positive definite and hence an M-matrix. □

Our final observation deals with the energy preservation of our sparsification mechanism, and the spectral equivalence between A_g and A_c. It shows that the properties that were discussed in [6, 25, 47] are achieved also in our algorithm. Consider the following 5-point and 9-point 2D stencils that represent circulant matrices:

$$H_1 = \begin{pmatrix} c & b & c \\ a & -2(a+b)-4c & a \\ c & b & c \end{pmatrix}, H_2 = \begin{pmatrix} & b+2c & \\ a+2c & -2(a+b)-8c & a+2c \\ & b+2c & \end{pmatrix}. \quad (3.18)$$

These two stencils were shown to be spectrally equivalent in [6]. The algorithm of [47] would find the entries of H_2 given those of H_1 by imposing that the response of both of them to the monomials $\{1, x, y, x^2, y^2\}$ given by

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad (3.19)$$

respectively, is equal. That is, the values of H_1 and H_2 times the stencils of each of those basis functions is equal.

Algorithm 2, similarly to the work of [25], would find the entries of H_2 given those of H_1 by imposing an equal response to the constant vector from right and from left. Consider the sparsification of the northwest (NW) entry of H_1, which equals c. Since H_2 is a 5-pt stencil, the surrogate paths for elimination the NW entry are through the north entry (N), which equals b in H_1, and the west entry (W) which equals a in H_1. Each of these paths is a distance-two path. Furthermore, assume that the portions of both paths, $\theta_{NW,N,C}$ and $\theta_{NW,W,C}$, are both equal to 0.5 (C denotes the center). This would be the case if H_1 is a Galerkin operator, and the coefficients in the prolongation P also correspond to a similar stencil coming from the fine-level matrix. In this situation, when eliminating the NW edge, we add $\frac{1}{2}c$ to the N entry and to the W entry for each direction of this edge. Since the matrix that corresponds to H_1 is symmetric, this will add a full value of c to both N and W entries. An additional c will be added in a similar way when eliminating the other entries that relate to those paths (the SW and NE edges). Overall, for this correction, we add the following sub matrix for each path

$$\begin{pmatrix} 0 & \delta & -\delta \\ \delta & -2\delta & \delta \\ -\delta & \delta & 0 \end{pmatrix}. \quad (3.20)$$

This sub-matrix corresponds to the $[NW, N, C]$ entries or to the $[NW, W, C]$ entries, and in both options, $\delta = \frac{1}{2}c$. This matrix has a zero energy not only for the constant vector, but also for vectors in one of the two following subspaces:

$$\begin{pmatrix} NW \\ N/W \\ C \end{pmatrix} : \text{Span}\left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\} \text{ and Span}\left\{ \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\}. \quad (3.21)$$

These subspaces can span all the functions in Equation (3.19) for the $[NW, N/W, C]$ entries, which implies the equivalence between the above algorithms. Similar equivalence is also evident in 3D for spectral equivalence between 7-pt and 27-pt stencils.

Let us now show some measures that demonstrate the spectral equivalence achieved by our algorithm (and all the algorithms that produce (3.18)). We refer again to the example in (2.2), and test two measures for this example on a 32×32 grid: one is (2.3), and the other is the operator in (2.3) multiplied by a smoothing operator. The multiplication by a smoothing operator aims (roughly speaking) to restrict the spectral equivalence measure to smooth modes. In Table 3.3 we show the measures for A_g and A_c in (2.2), and also for the pure aggregation operator A_t, which is equal to $2A_c$ in this case. The table shows that all measures are below 1, as is expected, given that the two-level cycles with either operator converge. However, we get better equivalence using A_c. Moreover, introducing the relaxation operators S_J improves the equivalence measure of A_c up to a value close to 0, whereas the measure for A_t stagnates at 0.5. We note that 0.5 is indeed the convergence factor of the aggregation two level cycle for this example, and it deteriorates as more levels are used in a V-cycle. Further investigation of the required characteristics for spectral equivalence in the context of non-Galerkin multigrid is a subject of our future research.

	$B = A_c$	$B = A_t$
$\|I - B^{-1}A_g\|_2$	0.49887	0.74943
$\|(I - B^{-1}A_g)S_J\|_2$	0.15815	0.49907
$\|(I - B^{-1}A_g)S_J^2\|_2$	0.05611	0.49758

TABLE 3.1

Spectral equivalence measures. S_J denotes the damped Jacobi operator $(I - \frac{2}{3}D^{-1}A_g)$.

4. Numerical Results. In this section, the Sparsified Smoothed Aggregation (SpSA) method is compared to both Smoothed Aggregation (SA) and simple aggregation (AGG). We consider three groups of problems: non-symmetric 2D and 3D convection-diffusion problems, symmetric 2D and 3D diffusion problems with discontinuous coefficients, and symmetric 2D and 3D unstructured homogenous and non-homogenous graph-Laplacian problems on random graphs. For all problems, we compare the performance of the above algorithms as preconditioners to GMRES or PCG, and compare both iteration count and running time of our code. We consider only V-cycles in the work as it is the most suitable for parallel computations.

In the tables below, we present four measures for each run: '*it*' denotes the number of V-cycles needed to reduce the initial residual by a factor of 10^8, starting with a zero initial guess. '*op*' denotes the operator complexity, which is the total number of non-zero elements in the operators A on all the grids, divided by that of the fine-level operator. Our coarsening is performed until $n < 100$. '*st*' denotes the maximal stencil size, that is, the maximal number of non-zeros in any row in the whole hierarchy of operators. The maximal (or average) stencil size reflects the amount of communication required in parallel computations [49, 34]. Lastly, 'WU' denotes the number of work-units required for the solution—for the setup and solution phases combined. Each work-unit is a matrix-vector multiplication and the values in the table are calculated by measuring the time needed for the solution and dividing it by the time of one fine-level matrix-vector multiplication. We note that such timings are highly implementation and machine dependent, but still, they provide some indication for comparing between the algorithms. Our code is MATLAB based with several procedures written in C using the `mex` environment. The experiments were performed using MATLAB R2013b on a machine with an Intel core i7 quad-core CPU with 8 GB of RAM memory, running Windows 7.

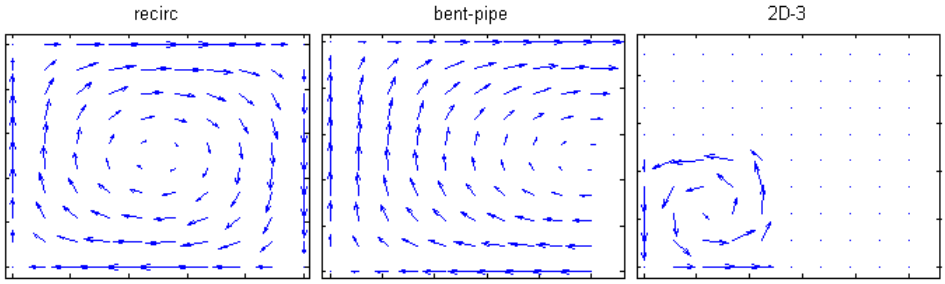

FIG. 4.1. *2D convection-diffusion problems: velocity fields.*

4.1. Non-symmetric test cases: convection-diffusion. We consider the 2D and 3D convection-diffusion equation (1.5) on the unit square/cube with Dirichlet boundary conditions. The problem is discretized using the first order upwind finite differences method, leading to a five-point stencil on a discrete domain. We use several of the harder problems in [21, 30, 33]:

$$
\begin{aligned}
&\texttt{recirc:} && \mathbf{v}^T = (x(1-x)(2y-1), -(2x-1)y(1-y)),\\
&\texttt{bent-pipe:} && \mathbf{v}^T = (x(x-2)(1-2y), -4y(y-1)(1-x)),\\
&\texttt{2D-3} : && \mathbf{v}^T = \begin{cases} \text{if} \quad x,y < 0.5, & (\cos(2\pi x)\sin(2\pi y), -\sin(2\pi x)cos(2\pi y))\\ \text{otherwise } 0, \end{cases}\\
&\texttt{3D-1} : && \mathbf{v}^T = (2x(1-x)(2y-1)z, (2x-1)y(y-1), (2x-1)(2y-1)z(z-1)),\\
&\texttt{3D-2} : && \mathbf{v}^T = (x(1-2y)(1-z), y(1-2z)(1-x), z(1-2x)(1-y)),\\
&\texttt{3D-3} : && \mathbf{v}^T = (x(1-y)(2-z), y(1-z)(2-x), z(1-x)(2-y)),
\end{aligned}
\tag{4.1}
$$

and following [33] generate f in (1.5) so that the solution is given by $u = \sin(\pi x)^2 + \sin(\pi y)^2$ for 2D or $u = \sin(\pi x)^2 + \sin(\pi y)^2 + \sin(\pi z)^2$ for 3D. Figure 4.1 shows the 2D velocity fields $\mathbf{v}$.

We use GMRES(10) acceleration, preconditioned with V-cycles. On the top level, we apply one pre-smoothing of forward Gauss-Seidel and one post-smoothing of backward Gauss-Seidel, and on coarser levels we apply one pre and post symmetric Gauss-Seidel smoothing.

Table 4.1 compares the three aggregation methods AGG, SA and SpSA for the 2D convection-diffusion (1.5) with the first three velocity fields in (4.1). Fields with '—' show that convergence was not achieved in 100 iterations. It is clear that the AGG method is not mesh independent and not efficient, especially for the `recirc` problem. It also struggles for the diffusion-dominated problems ($\epsilon = 10^{-2}$ and `2D-3` that has a large diffusive domain for all ϵ). The SA method shows good scalability for all combinations. It has a higher operator complexity than the other two methods, especially in the more convective problems, where the coarsening becomes less aggressive. The SpSA obviously has the low and attractive operator complexity of AGG, but also enjoys the scalable convergence behavior of SA for both the diffusive and convective problems. The maximal stencil sizes, 'st', are similar between AGG and SpSA, and are higher in SA—as are the operator complexities. Since these are only 2D problems, the stencil growth of SA is moderate. In terms of the computing time of the algorithms, which is measured in work-units, 'WU', both SpSA and SA beat AGG even though the setup phase of AGG is about 2-3 times faster (not shown in the tables). SA and SpSA have surprisingly similar timings in our implementation, and in both cases the timing of the setup and solution phases are quite similar, each

recirc:		*AGG*				*SpSA*				*SA*			
ϵ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
10^{-2}	256^2	80	1.3	8	1103	12	1.3	10	426	13	1.4	31	394
	512^2	—	1.3	8	—	13	1.3	10	287	13	1.4	34	269
	1024^2	—	1.3	10	—	14	1.3	10	293	15	1.4	35	297
	2048^2	—	1.3	9	—	14	1.3	10	300	16	1.4	36	309
10^{-4}	256^2	—	1.5	13	—	12	1.4	11	663	13	2.2	50	702
	512^2	—	1.4	12	—	13	1.4	12	341	13	1.9	52	338
	1024^2	—	1.4	11	—	15	1.4	17	370	14	1.8	51	367
	2048^2	—	1.3	12	—	17	1.3	11	358	17	1.5	52	360
10^{-6}	256^2	76	1.5	16	1439	18	1.5	13	565	17	2	70	600
	512^2	—	1.5	17	—	20	1.5	12	389	19	2.1	69	392
	1024^2	—	1.5	16	—	23	1.5	13	440	21	2.1	82	441
	2048^2	—	1.5	15	—	27	1.5	13	512	23	2.2	80	514
bent-pipe:		*AGG*				*SpSA*				*SA*			
ϵ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
10^{-2}	256^2	83	1.3	8	1299	12	1.3	10	455	12	1.4	36	419
	512^2	—	1.3	8	—	14	1.3	10	301	14	1.4	39	285
	1024^2	—	1.3	9	—	15	1.3	10	321	15	1.4	48	309
	2048^2	—	1.3	9	—	17	1.3	10	330	17	1.4	38	322
10^{-4}	256^2	25	1.5	11	542	15	1.5	12	670	14	2.4	57	661
	512^2	38	1.5	12	446	14	1.5	12	368	13	2.3	63	391
	1024^2	64	1.5	12	779	12	1.4	13	366	12	2.2	59	395
	2048^2	—	1.4	13	—	13	1.4	13	362	13	2	63	376
10^{-6}	256^2	22	1.5	13	509	16	1.5	13	649	14	2	69	650
	512^2	26	1.5	13	340	16	1.5	13	343	15	2	76	342
	1024^2	34	1.5	14	462	16	1.5	18	382	15	2	89	374
	2048^2	48	1.5	14	609	17	1.5	17	383	15	2.1	88	380
2D-3:		*AGG*				*SpSA*				*SA*			
ϵ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
10^{-2}	256^2	—	1.3	8	—	13	1.3	11	519	13	1.4	30	470
	512^2	—	1.3	8	—	14	1.3	10	289	14	1.4	36	283
	1024^2	—	1.3	8	—	15	1.3	11	304	17	1.4	41	315
	2048^2	—	1.3	9	—	17	1.3	11	331	18	1.4	36	331
10^{-4}	256^2	—	1.3	15	—	18	1.3	11	549	18	1.6	53	548
	512^2	—	1.3	13	—	18	1.3	12	377	18	1.6	52	360
	1024^2	—	1.3	11	—	18	1.3	12	370	18	1.6	54	380
	2048^2	—	1.3	13	—	18	1.3	13	357	18	1.5	54	358
10^{-6}	256^2	93	1.3	15	1518	23	1.3	11	650	21	1.5	56	582
	512^2	—	1.3	15	—	25	1.3	14	411	22	1.5	70	382
	1024^2	—	1.3	17	—	25	1.3	12	420	26	1.5	76	437
	2048^2	—	1.3	16	—	26	1.3	13	440	27	1.5	84	453

TABLE 4.1

2D convection-diffusion. '#it' denotes the number of V cycles, 'op' is the operator complexity, 'st' is the maximal stencil size in the hierarchy, and 'WU' is the work units measures.

about half of the WUs that are shown in the table.

Table 4.2 shows the results for the 3D problems. Here, we see even larger operator complexities for SA than in 2D, especially for the convection-dominated problems ($\epsilon = 10^{-4}, 10^{-6}$). In most of these cases, we also see a severe stencil growth in SA, which reaches thousands of non-zeros in some cases. By changing the strength of

3D-1:		*AGG*				*SpSA*				*SA*			
ϵ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
10^{-2}	64^3	39	1.3	28	456	11	1.3	26	360	11	1.6	149	324
	96^3	49	1.3	35	561	12	1.2	30	391	12	1.6	213	362
	128^3	59	1.3	30	626	12	1.3	34	362	13	1.6	248	358
	192^3	75	1.3	32	—	14	1.2	35	—	14	1.7	244	—
10^{-4}	64^3	20	1.5	51	316	12	1.5	47	695	11	4.8	837	847
	96^3	28	1.5	64	409	11	1.5	55	655	11	4.7	1092	840
	128^3	35	1.5	49	465	11	1.5	60	644	10	4.5	996	784
	192^3	48	1.5	53	—	11	1.5	54	—	10	4.2	1053	—
10^{-6}	64^3	19	1.5	67	313	14	1.5	67	554	13	3.7	1027	636
	96^3	21	1.5	83	338	14	1.5	83	532	13	3.8	1711	670
	128^3	23	1.5	81	327	14	1.5	98	482	12	3.9	2503	629
	192^3	27	1.5	106	—	13	1.5	112	—	12	3.9	3935	—

3D-2:		*AGG*				*SpSA*				*SA*			
ϵ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
10^{-2}	64^3	40	1.3	27	486	12	1.2	27	416	13	1.7	176	384
	96^3	55	1.3	31	625	13	1.2	29	403	13	1.6	229	378
	128^3	67	1.3	30	699	13	1.3	32	365	14	1.6	242	364
	192^3	84	1.3	31	—	15	1.2	33	—	15	1.7	241	—
10^{-4}	64^3	32	1.5	55	475	14	1.5	53	704	12	4.7	992	833
	96^3	40	1.5	61	552	14	1.5	64	694	12	5	1173	904
	128^3	48	1.5	67	616	13	1.5	80	683	12	5.1	1307	919
	192^3	70	1.5	65	—	14	1.5	66	—	12	5	1126	—
10^{-6}	64^3	86	1.5	58	1195	34	1.5	56	813	30	3.3	920	892
	96^3	71	1.5	79	941	22	1.5	72	602	18	3.3	1508	673
	128^3	77	1.5	101	938	24	1.5	73	600	24	3.3	2227	755
	192^3	91	1.5	103	—	25	1.5	96	—	21	3.3	2746	—

3D-3:		*AGG*				*SpSA*				*SA*			
ϵ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
10^{-2}	64^3	56	1.3	31	718	14	1.3	39	522	14	2.4	240	516
	96^3	79	1.3	30	872	15	1.3	38	458	15	2	266	474
	128^3	97	1.3	34	1005	15	1.3	32	422	16	1.8	253	437
	192^3	—	1.3	31	—	16	1.2	35	—	16	1.7	290	—
10^{-4}	64^3	37	1.5	29	515	15	1.5	29	550	13	3.5	424	564
	96^3	46	1.5	29	610	15	1.5	43	521	14	3.9	771	660
	128^3	54	1.5	36	665	16	1.5	51	548	14	4.4	892	741
	192^3	72	1.5	36	—	16	1.5	65	—	14	5.1	1824	—
10^{-6}	64^3	82	1.5	28	1052	20	1.5	28	530	17	2.8	117	503
	96^3	—	1.5	28	—	23	1.5	33	546	15	2.8	142	462
	128^3	—	1.5	30	—	20	1.5	34	488	14	2.8	170	438
	192^3	—	1.5	35	—	21	1.5	45	—	16	2.9	309	—

TABLE 4.2

Convection diffusion 3D. '#it' denotes the number of V cycles, 'op' is the operator complexity, 'st' is the maximal stencil size in the hierarchy, and 'WU' is the work units measures.

connection parameters in (A.2) we may control the stencil growth, but at the same time impair the convergence rate of SA, which is rather good and scalable, as in the 2D case. Unlike in 2D, the AGG method shows moderate convergence, albeit not mesh-independent in several cases. Its convergence is expected to further deteriorate as the problem gets bigger. Again, '—' denotes the cases where AGG failed to converge

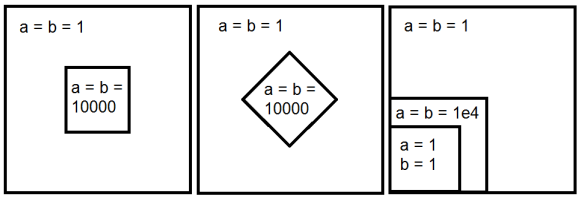

FIG. 4.2. *2D diffusion problems with discontinuous coefficients: square, diamond, and L shapes.*

in less than 100 iterations, or it shows that the timings in WUs of the run are not relevant. This happens in the 192^3 3D problems, where the memory in our machine ran out in the setup phase and the execution was dominated by memory swapping. As in 2D, SpSA has the low operator complexity and maximal stencil sizes of AGG, and convergence similar to SA. In terms of work-units, all algorithms are competitive with some advantage to SpSA in the majority of the cases. In 3D, the setup of AGG is about 3-5 times faster than the setup of SA and SpSA, because of the stencil growth, and the need to multiply much denser matrices, in the case of SA. In SpSA, more non-zeros need to be eliminated using more surrogate paths. In most cases where the operator complexity is high, the setup cost of SA and SpSA dominated their execution time. Other than that, the SpSA method again seems to be the most efficient. Although the timings of SA and SpSA are similar in our serial code, we expect to see advantage for SpSA in parallel settings, especially in the solution phase.

4.2. Diffusion with discontinuous coefficients. In the next group of test cases we consider the two- and three-dimensional diffusion equation on the unit square/cube, Ω, with Dirichlet boundary conditions

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= f, \qquad \text{in } \Omega \\ u &= g, \qquad \text{on } \partial\Omega. \end{aligned} \tag{4.2}$$

The diffusion problem is discretized by the vertex-centered finite differences method, leading to a five-point stencil on a discrete domain Ω_h with regular mesh size.

We consider three classical test cases of coefficient inhomogeneities, including jumps that are aligned with the grid lines and jumps that are not aligned (all the problems appear in [47]). Figure 4.2 shows the three choices of coefficients which are explicitly given by

$$\begin{gathered} \blacksquare : \kappa(\mathbf{x}) = \begin{cases} \|x - \frac{1}{2}\|_\infty < \frac{1}{4} & 10^4 \\ \text{otherwise} & 1 \end{cases} ; \qquad \blacklozenge : \kappa(\mathbf{x}) = \begin{cases} \|\mathbf{x} - \frac{1}{2}\|_1 < \frac{1}{\sqrt{8}} & 10^4 \\ \text{otherwise} & 1 \end{cases} \\ L : \kappa(\mathbf{x}) = \begin{cases} \frac{1}{4} < \|\mathbf{x}\|_\infty < \frac{1}{2} & 10^4 \\ \text{otherwise} & 1 \end{cases} . \end{gathered} \tag{4.3}$$

The resulting linear systems are symmetric.

We apply PCG, preconditioned using V-cycles with one pre- and post- Symmetric Gauss Seidel relaxations for all methods on all levels. Because the plain aggregation method, AGG, is known to struggle with such problems, we accelerate it using a multilevel overcorrection acceleration, which we denote by *AGG+*. That is, we interpolate $\mathbf{e} = P\mathbf{e}_c$, apply post relaxations for $A\mathbf{e} = \mathbf{r}$, and then calculate [5] $\mathbf{x} \leftarrow \mathbf{x} - \alpha\mathbf{e}$, such that $\alpha = \arg\min_\alpha \|\mathbf{x} - \alpha\mathbf{e}\|_A = \frac{\mathbf{r}^T\mathbf{e}}{\mathbf{e}^T A\mathbf{e}}$. Using this procedure we typically get $\alpha > 1$. This overcorrection technique may significantly accelerate the convergence of

2D-tests:		*AGG+*				*SpSA*				*SA*			
κ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
■	256^2	37	1.3	10	519	20	1.3	10	449	13	1.4	31	281
	512^2	60	1.3	10	628	20	1.3	10	338	14	1.4	37	276
	1024^2	72	1.3	11	797	22	1.3	10	370	16	1.4	44	313
	2048^2	—	1.3	11	—	25	1.3	11	406	17	1.4	41	319
♦	256^2	34	1.3	9	527	14	1.3	9	421	13	1.4	32	332
	512^2	57	1.3	9	616	16	1.3	10	293	14	1.4	42	272
	1024^2	86	1.3	10	941	18	1.3	10	340	17	1.4	38	316
	2048^2	—	1.3	10	—	20	1.3	11	346	16	1.4	41	305
L	256^2	35	1.3	10	461	14	1.3	11	252	10	1.4	34	206
	512^2	55	1.3	10	601	15	1.3	10	299	11	1.4	35	244
	1024^2	49	1.3	10	576	15	1.3	11	311	11	1.4	40	258
	2048^2	72	1.3	11	813	16	1.3	11	312	14	1.4	69	287
3D-tests:		*AGG+*				*SpSA*				*SA*			
κ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
■	64^3	33	1.3	31	383	19	1.3	32	415	15	1.7	198	375
	96^3	35	1.3	32	410	21	1.3	31	430	16	1.7	257	384
	128^3	46	1.3	31	517	22	1.3	30	420	17	1.6	245	388
	192^3	59	1.3	34	—	24	1.3	33	—	18	1.7	266	—
♦	64^3	30	1.3	29	412	16	1.3	30	452	12	1.6	144	347
	96^3	55	1.3	33	658	18	1.2	30	434	14	1.6	219	368
	128^3	49	1.3	32	545	19	1.3	30	387	15	1.6	219	360
	192^3	92	1.3	32	—	20	1.2	30	—	16	1.6	235	—
L	64^3	23	1.3	29	282	14	1.2	31	356	10	1.7	207	314
	96^3	29	1.3	32	353	14	1.2	30	361	11	1.7	249	335
	128^3	38	1.3	34	443	15	1.2	33	368	12	1.7	262	351
	192^3	45	1.3	34	—	16	1.2	34	—	13	1.7	278	—

TABLE 4.3

2D and 3D diffusion problems on structured meshes. '#it' denotes the number of V cycles, 'op' is the operator complexity, 'st' is the maximal stencil size, and 'WU' is the work units measures.

AGG. We note that this acceleration imposes an expensive communication overhead in parallel settings due to the inner products that are computed.

Table 4.3 summarizes the results for this group of problems. Because of the multilevel accelerations, the AGG+ method converged moderately fast, but again, its convergence is not mesh-independent. Again, '—' denotes failure to converge, or irrelevant timings in WUs. Like before, we see the best iteration counts in SA, with SpSA needing a bit more iterations. In terms of timings in WUs, SpSA and SA are comparable and generally better than AGG+, although the setup of AGG+ is much cheaper, like in the previous examples.

For the problems considered, SA has a rather low operator complexity, and it does not introduce a severe stencil growth. Its maximal stencil size is only in the low hundreds, also in 3D. As before, AGG+ and SpSA have similar low complexity, and maximal stencil size.

4.3. Random graph-Laplacian problems. In the next group of test cases we consider a graph-Laplacian problem on a 2D/3D random graph $G(V, E)$. We generate our graph by first generating n random points on the unit square/cube as the nodes V, and applying a Delaunay triangulation to generate edges in E. Such graphs were first suggested in [22]. A 2D example of such graph is shown in figure 4.3. Given the

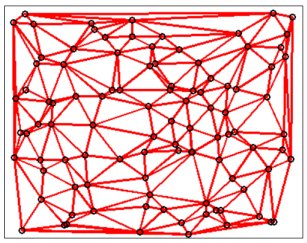

FIG. 4.3. *Unstructured random planar graph.*

edges $(i,j) \in E$ we create a matrix A which is defined by

$$\forall (i,j) \in E : A_{i,j} = -\Theta_{ij} \quad \text{and} \quad A_{ii} = -\textstyle\sum_{i \neq j} A_{ij}, \tag{4.4}$$

and if we set $\Theta_{ij} = 1$, it represents the homogenous graph-Laplacian operator on G. In the inhomogenous case, we set a weight $\Theta_{ij} > 0$ as the coefficient for each edge (i,j). The weights of the graph are symmetric, i.e $\Theta_{ij} = \Theta_{ji}$, and so is the associated matrix. We use random weights $\Theta_{i,j} = 10^{3u}$ where $u \in (0,1)$ is a random uniformly distributed number, so that $\Theta_{i,j} \in (1, 10^3)$, but obey a logarithmic distribution. This is a much harder problem than choosing the Θ's uniformly, because there are many weak connections in the matrix. We note that these problems have typically large stencils of non-uniform sizes. Therefore, we needed to change the aggregation procedure, as explained in the Appendix A. In these examples, we could not generate the 3D 192^3 problem because it exceeded the memory of our machine. We also note that this problem is singular, with the constant vector as a true null-space. To handle that, we apply a pseudo-inverse on the coarsest grid, and filter out the constant from the solution. See [42] for more discussion on that regard.

Table 4.4 summarizes the results for this group of problems, where *rand* denotes the case of random weights. Overall, all methods handle the problem well. *AGG+* presents reasonable iteration counts, although, again it converges more slowly than *SpSA* and *SA*. *AGG+* and *SpSA* have similar attractive operator complexity and maximal stencil sizes, while SA struggles somewhat to maintain low operator complexity and reasonable stencil size for the 3D non-homogenous problem. We note that in this set of examples, the sparsening mechanism in SpSA used many more quadrilateral correction paths than in the previous structured test cases. In terms of work-units, the methods are comparable in 2D while *AGG+* is the fastest in 3D, thanks to its fast setup phase. We again note that the recursive accelerations impose communication costs in parallel settings.

4.4. Communication complexity. In this section we provide measures for the communication that is involved in applying a V-cycle with SA and SpSA, using a simplified model from [2]. In this model, the communication cost is broken down into the start-up time α (latency) and the per-element send time β (inverse bandwidth). If a message has m elements, then the send cost is

$$T_{send} = \alpha + \beta m.$$

The constants α and β are machine dependent, and so we will focus on the factors that multiply them. Following the descriptions of [23], we define two measures of communication complexity as follows. Let p be the number of available computing

2D-tests:		*AGG+*				*SpSA*				*SA*			
Θ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
1	256^2	24	1.1	22	246	12	1.1	22	247	11	1.4	119	220
	512^2	29	1.2	42	332	15	1.1	42	321	13	1.4	228	257
	1024^2	29	1.2	66	317	17	1.1	66	303	14	1.4	319	274
	2048^2	33	1.2	55	347	20	1.1	62	334	15	1.4	659	289
rand	256^2	25	1.4	25	354	17	1.4	37	418	15	2.5	251	421
	512^2	33	1.4	53	407	18	1.4	41	395	16	2.5	453	418
	1024^2	51	1.4	40	662	21	1.4	36	479	16	2.5	583	437
	2048^2	37	1.4	58	478	22	1.4	38	461	19	2.5	1249	499
3D-tests:		*AGG+*				*SpSA*				*SA*			
Θ	n	it	op	st	WU	it	op	st	WU	it	op	st	WU
1	64^3	18	1.1	163	190	10	1.1	163	270	11	1.3	1067	249
	96^3	19	1.1	113	190	11	1.1	113	248	11	1.3	1118	249
	128^3	20	1.1	155	190	11	1.1	155	244	12	1.3	1826	254
rand	64^3	15	1.4	126	204	10	1.3	126	585	10	4.1	3117	755
	96^3	16	1.4	145	194	10	1.3	145	546	11	4.3	5501	751
	128^3	17	1.4	139	202	10	1.3	130	519	14	4.4	9108	842

TABLE 4.4
Unstructured Graph-Laplacian.

nodes. Define a partitioning of each matrix A in the multigrid hierarchy into p clusters of variables, with each cluster designated to a computing node. Each such node holds the corresponding rows in the matrices and the corresponding values of the iterate $\mathbf{x}^k$. We then define the communication required for applying a matrix-vector multiplication (MAT-VEC) on each level. The latency measure is the number of non-zero off-diagonal blocks in the matrix according to the cluster partitioning. It does not matter how many non-zeros there are in an off-diagonal block—it is the existence of at least one non-zero in A in that off-diagonal block that forces establishing communication between the two associated nodes. More precisely, assume that T is a matrix that corresponds to the p clusters just as (2.4) corresponds to the n_c aggregates; for a given multigrid hierarchy we define:

$$Latency = \frac{\sum_{\ell=0}^{L}(nnz(T_\ell^T A_\ell T_\ell) - p)}{(nnz(T_0^T A_0 T_0) - p)}, \tag{4.5}$$

where ℓ denotes the level, the matrices T_ℓ and A_ℓ are the clustering matrix and operator on level ℓ, respectively, p is the number of clusters and $nnz(\cdot)$ is the number of non-zeros. Next, we define the bandwidth that is needed for applying a MAT-VEC on each level—that is, the total number of values of $\mathbf{x}^k$ that are needed to be sent between nodes. This corresponds to the number of non-zero columns in the off-diagonal blocks according to the cluster partitioning. As in (4.5), we divide this number by the required bandwidth on the finest level. We preform the partitioning for each level using a graph partitioning software METIS [24] which aims at forming clusters such that the number of neighbors between the clusters is minimal. We use $p = 100$ clusters, and we measure the communication complexity only for levels with $n \geq 500$.

Table 4.5 summarizes the communication measures for eight sample hierarchies from the previous sections. It shows that in 2D, the latency measures of both SA and SpSA are quite comparable, and are mostly dictated by the number of levels,

Problem			SA		SpSA	
Name	n	Parameters	Latency	Bandwidth	Latency	Bandwidth
`2D Conv-Diff`	1024^2	`recirc`,$\epsilon = 10^{-6}$	5.98	3.02	5.26	1.97
`3D Conv-Diff`	96^3	`3D-1`,$\epsilon = 10^{-6}$	16.00	3.48	6.19	1.72
`2D Diffusion`	1024^2	$\kappa = \blacksquare$	4.19	1.91	3.99	1.59
`3D Diffusion`	96^3	$\kappa = \blacksquare$	10.08	1.82	4.39	1.41
`2D RandGraph`	1024^2	$\Theta = 1$	5.22	2.69	4.00	1.59
`3D RandGraph`	96^3	$\Theta = 1$	6.42	1.81	3.00	1.19
`2D RandGraph`	1024^2	$\Theta = rand$	7.05	4.79	5.00	2.05
`3D RandGraph`	96^3	$\Theta = rand$	12.93	4.70	4.15	1.66

TABLE 4.5
Communication complexity

which is equal for both algorithms in our tests. In 3D, on the other hand, there is a huge increase in the latency of SA, which is mostly due to the coarser levels. For SpSA, the latency measure is mostly equal to the number of levels, which is what we expect to achieve in this algorithm. In the bandwidth measures we see comparable complexities for diffusion and homogenous RandomGraph problems. This is also correlated with the similar operator complexity measures in the previous sections. A bigger difference is evident in the other problems, where we also saw an increase in the operator complexity of SA in the previous sections. In the bandwidth measures, we do not see a big difference between 2D and 3D problems (remember that the measures are relative to bandwidth on the finest grid). Obviously, SpSA required less communication in all the cases.

5. Conclusions. In this paper we have presented a new algebraic multigrid algorithm where the choice of the sparsity pattern of the coarse operators is independent of the choice of the high-quality transfer operators. This property makes the algorithm particulary worthwhile for parallel settings.

The new algorithm uses the well-known aggregation framework, adopting simple non-smoothed aggregation for determining the sparsity pattern of the coarse operators, and smoothed aggregation for high-quality transfer operators. It sparsifies the smoothed aggregation coarse operators onto the simple aggregation sparsity patterns. Numerical experiments show that the algorithm has promising capabilities for 2D and 3D convection-diffusion problems, diffusion problems with varying coefficients and unstructured graph-Laplacian problems. It seems scalable and robust and may be advantageous in cases where strict sparsity constraints prevent us from using high-quality Galerkin operators, as in parallel settings.

6. Acknowledgements. The authors would like to thank Prof. Raanan Fattal and Dr. Jacob Schroder for some fruitful conversations that inspired the non-symmetric sparsening approach. The authors would also like to thank Dr. Killian Miller for sharing his neighborhood aggregation code.

Appendix A. Smoothed Aggregation.

In this section we describe the details of the version of the SA method that we use in this paper as the "Galerkin" method. We note that for some of the non-symmetric problems that we tested, it performs significantly better than the original SA method [45], which was shown exhibit slow convergence for some convection-diffusion problems also in [33, 30, 13]. Our most significant change involves using two different parameters for strength of connections: one rather large in the aggregation procedure, and one rather small in the prolongation filtering procedure. In [45], these parameters are

equal. Using two different values in this way may result in rather smaller aggregates and rather large stencils on coarse grids, together with better convergence rates.

We note, however, that for isotropic problems we typically do not get a significantly different strength-of-connection matrix for the aggregation and also typically no filtering is performed. In particular, for a 5-point Laplacian operator, all the entries are considered strong and no filtering is applied. Therefore, for isotropic problems our version of SA behaves similarly to the original SA in terms of operator complexity and stencil sizes—this is evident in our numerical results. For other problems, e.g., convection dominated (1.5) and non-homogenous graph Laplacian, using the original version of SA (or, using just one parameter in our version) generates lower operator complexity. However, it also significantly degrades the convergence—especially for the 2D problems, which are harder because of their small mesh size. This is also evident in the literature mentioned earlier. Below, we choose the parameters for our method such that SA has mesh independent convergence for all problems, and those parameters were mostly dictated by the 2D problems.

To define the aggregations used in (2.4), we first define a strength of connection matrix, and then apply a neighborhood aggregation algorithm. To compute the strength of connection matrix, we first calculate:

$$S_{ij} = \begin{cases} 1 & i = j \\ \frac{-A_{i,j}}{\max_{k \neq i}\{-A_{i,k}\}} & i \neq j \\ 0 & otherwise \end{cases}, \tag{A.1}$$

and treat (i, j) as strongly connected if $S_{ij} > \theta$, where $0 < \theta < 1$ as a strength parameter. This definition corresponds to the strength condition that is used in classical AMG [19], which is usually written as: $-A_{i,j} \geq \theta \max_{k \neq i}\{-A_{i,k}\}$.

For the aggregation procedure, we use a symmetric version of the strength matrix, $S(\theta)$, such that:

$$S_{ij}(\theta) = \tfrac{1}{2}(S_{ij} + S_{ji}) : S_{ij} > \theta, \tag{A.2}$$

with a rather large $\theta = 0.5$. Generally, the larger θ we use, the smaller will be our aggregates, since less strong connections are considered.

Given the above strength of connection matrix, we found the original aggregation algorithm of [45] to be efficient, and most importantly—very fast to compute. However, for the unstructured problems that we test there are cases where some rows in the matrix have significantly more non-zeros than the rest of the rows. Such a row may result in a very large aggregate if it is chosen as a seed of an aggregate. It may also have the same effect even if it is not chosen as a seed. To solve this, in the first pass we ignore points with a relatively large number of strong connections, and treat them separately in a second identical pass. Algorithm 3 summarizes our aggregation scheme. In the original aggregation algorithm in [45], only the first and third passes are applied for $\mathcal{N}_1 = \{1, ..., n\}$ and $\mathcal{N}_2 = \emptyset$. We use $\tau = 3$ as the relative threshold for large neighborhoods, but note that any $\tau \in [1.5, 4]$ seems reasonable for us.

As noted before, the next step in the SA algorithm includes smoothing the tentative operators (Equation (2.5)), and for that, the filtered matrix A^F and the diagonal preconditioner Q need to be defined. The filtering aims at removing small entries from P and R, which may have little influence on their quality. Our filtering process to define A^F is similar to [45], only again we use a different strength of connection matrix, based on (A.1). Specifically, we define $\hat{S}_i(\epsilon) = \{j : |S_{i,j}| < \epsilon\}$. Then, A^F is

Algorithm: $\{\mathcal{C}_J\}_{J=1}^{n_c} \leftarrow$ **Neighborhood-Aggregation**(A, θ, τ)
% θ - Strength of connection parameter.
% τ - Relative neighborhood size parameter.

$S(\theta)$: Strength of connection matrix based on (A.2).
Let $S_i = \{j : S_{ij}(\theta) \neq 0\}$ denote the strong neighborhood of each point i
Compute average neighborhood size: $\bar{s} = \frac{1}{n}\sum_{j=0}^{n} |S_i|$
Set $\mathcal{N}_1 \leftarrow \{i : |S_i| \leq \tau\bar{s}\}$, $\mathcal{N}_2 = \{i : |S_i| > \tau\bar{s}\}$, and $J \leftarrow 0$

% First Pass: Assign only relatively small neighborhoods to aggregates.
foreach $i \in \mathcal{N}_1$ **do**
  if *each $j \in S_i$ does not belong to an aggregate* **then**
    Set $J \leftarrow J+1$, $C_J \leftarrow S_i \setminus \mathcal{N}_2$, and $\hat{\mathcal{C}}_J \leftarrow \mathcal{C}_J$.
  end
end

% Second Pass: Assign neighborhoods to aggregates for the rest of the points.
foreach $i \in \mathcal{N}_2$ **do**
  if *each $j \in S_i$ does not belong to an aggregate* **then**
    Set $J \leftarrow J+1$, $C_J \leftarrow S_i$, and $\hat{\mathcal{C}}_J \leftarrow \mathcal{C}_J$.
  end
end

% Third Pass: Assign remaining points to the aggregates.
$n_c \leftarrow J$
foreach *unassigned point i* **do**
  Set $\mathcal{C}_J \leftarrow \hat{\mathcal{C}}_J \cup \{i\}$, where $J \leftarrow \arg\max_{K=1,..,n_c} \left\{ \frac{1}{|\hat{\mathcal{C}}_K|} \sum_{j \in \hat{\mathcal{C}}_K} S_{ij} \right\}$
end

Algorithm 3: Neighborhood-based Aggregation

defined by

$$A^F_{i,j} = \left\{ \begin{array}{cc} A_{i,j} & \text{if } j \in \hat{S}_i(\epsilon) \\ 0 & otherwise \end{array} \right\}, \quad A^F_{i,i} = A_{i,i} + \sum_{j \notin S_i(\epsilon)} A_{i,j}. \tag{A.3}$$

In this work we use $\epsilon = 0.02$. Note that $A\mathbf{1} = A^F\mathbf{1}$.

For the smoothing preconditioner Q in (2.5), the inverse of the diagonal of A is usually chosen, and then $I - \omega QA$ is the error propagation matrix associated with the damped Jacobi relaxation. In this work, we use the SPAI diagonal preconditioner [15] for Q, i.e., Q is the diagonal matrix that minimizes $\|I - QA^F\|_F$ which leads to $Q_{ii} = \frac{A^F_{i,i}}{\sum_j (A^F_{i,j})^2}$ for $i = 1, ..., n$. This is a more sophisticated operator than Jacobi, and in the context of smoothing P_t it is related to the energy minimization diagonal preconditioner (EMIN) in [33]. As in [33], we also found that such a Q is more efficient for solving (1.5) than the Jacobi operator. We also found that additional weighting is needed. While minimizing $\|I - QA^F\|_F$ minimizes the mean squared singular values of the matrix $I - QA^F$, prolongation smoothing is often related to a minimization of the maximal eigenvalue of the Galerkin coarse operator A_g (at least in the symmetric case). Therefore, for symmetric problems we follow the classical prolongation smoothing weight which is based on the Chebychev polynomials, $\omega = \frac{4}{3\rho(QA^F)}$, and

for non-symmetric problems we use a lower value: $\omega = \frac{5}{4\rho(QA^F)}$, following [41]. In both cases, $\rho(QA^F)$ is approximated by $\|QA^F\|_\infty$.

REFERENCES

[1] A. Baker, R. Falgout, T. Kolev, and U. Yang, *Multigrid smoothers for ultraparallel computing*, SIAM J. Sci. Comput., 33 (2011), pp. 2864–2887.

[2] A. H. Baker, R. D. Falgout, H. Gahvari, T. Gamblin, W. Gropp, K. E. Jordan, T. V. Kolev, M. Schulz, and U. M. Yang, *Preparing algebraic multigrid for exascale*, Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-533076, (2012).

[3] A. H. Baker, R. D. Falgout, T. Gamblin, T. V. Kolev, M. Schulz, and U. M. Yang, *Scaling algebraic multigrid solvers: On the road to exascale*, in Competence in High Performance Computing 2010, Springer, 2012, pp. 215–226.

[4] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang, *Challenges of scaling algebraic multigrid across modern multicore architectures*, in Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, IEEE, 2011, pp. 275–286.

[5] R. Blaheta, *A multilevel method with overcorrection by aggregation for solving discrete elliptic problems*, J. Comput. Appl. Math., 24 (1988), pp. 227–239.

[6] M. Bolten and A. Frommer, *Structured grid AMG with stencil-collapsing for d-level circulant matrices*, Tech. Report BUW-SC 07/4, University of Wuppertal, 2007.

[7] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap amg*, SIAM J. Sci. Comput., 33 (2011), pp. 612–632.

[8] A. Brandt and I. Yavneh, *On multigrid solution of high-reynolds incompressible entering flows*, J. Comput. Phys., 101 (1992), pp. 151–164.

[9] ———, *Accelerated multigrid convergence and high-reynolds recirculating flows*, SIAM J. Sci. Comput., 14 (1993), pp. 607–626.

[10] J. Brannick, M. Brezina, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *An energy-based amg coarsening strategy*, Numerical linear algebra with applications, 13 (2006), pp. 133–148.

[11] M. Brezina, R. Falgout, S. Maclachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive smoothed aggregation (α SA)*, SIAM J. Sci. Comput., 25 (2004), pp. 1896–1920.

[12] ———, *Adaptive algebraic multigrid*, SIAM J. Sci. Comput., 27 (2006), pp. 1261–1286.

[13] M. Brezina, T. Manteuffel, S. McCormick, J. Ruge, and G. Sanders, *Towards adaptive smooth aggregation (α SA) for nonsymmetric problems*, SIAM J. Sci. Comput., 32 (2010), pp. 14–39.

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, SIAM, second ed., 2000.

[15] O. Brker and M. J. Grote, *Sparse approximate inverse smoothers for geometric and algebraic multigrid*, Applied Numerical Mathematics, 41 (2002), pp. 61–80.

[16] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang, *A Survey of Parallelization Techniques for Multigrid Solvers*, in Parallel Processing for Scientific Computing, 2006.

[17] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang, *A survey of parallelization techniques for multigrid solvers*, Parallel processing for scientific computing, 20 (2006), pp. 179–201.

[18] H. De Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang, *Distance-two interpolation for parallel algebraic multigrid*, Numerical Linear Algebra with Applications, 15 (2008), pp. 115–139.

[19] R. D. Falgout, *An introduction to algebraic multigrid*, IEEE: Computing in Science and Engineering, 8 (2006), pp. 24–33.

[20] H. Guillard and P. Vaněk, *An aggregation multigrid solver for convection-diffusion problems on unstructured meshes.*, Tech. Report UCD-CCM-130, University of Colorado at Denver, CO, USA, 1998.

[21] M. M. Gupta and J. Zhang, *High accuracy multigrid solution of the 3d convectiondiffusion equation*, Applied Mathematics and Computation, 113 (2000), pp. 249 – 274.

[22] H. De Sterck, T. A. Manteuffel, S. F. McCormick, K. Miller, J. Pearson, J. Ruge, and G. Sanders, *Smoothed aggregation multigrid for Markov chains*, SIAM J. Sci. Comput., 32 (2010), pp. 40–61.

[23] V. E. Henson and U. M. Yang, *¡ i¿ boomeramg¡/i¿: A parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics, 41 (2002), pp. 155–177.

[24] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular*

graphs, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.
[25] D. Krishnan, R. Fattal, and R. Szeliski, *Efficient preconditioning of Laplacian matrices for computer graphics*, ACM Trans. Graph., 32 (2013), pp. 142:1–142:15.
[26] D. Krishnan and R. Szeliski, *Multigrid and multilevel preconditioners for computational photography*, in ACM Transactions on Graphics (TOG), vol. 30, ACM, 2011.
[27] J. Mandel, M. Brezina, and P. Vaněk, *Energy optimization of algebraic multigrid bases*, Computing, 62 (1999), pp. 205–228.
[28] Y. Notay, *Aggregation-based algebraic multilevel preconditioning*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 998–1018.
[29] ———, *An aggregation-based algebraic multigrid method*, Electronic Transactions on Numerical Analysis, 37 (2010), pp. 123–146.
[30] ———, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM J. Sci. Comput., 34 (2012), pp. A2288–A2316.
[31] L. N. Olson, J. B. Schroder, and R. S. Tuminaro, *A general interpolation strategy for algebraic multigrid using energy minimization*, SIAM Journal on Scientific Computing, 33 (2011), pp. 966–991.
[32] C. W. Oosterlee and T. Washio, *Krylov subspace acceleration of nonlinear multigrid with application to recirculating flow*, SIAM J. Sci. Comput., 21 (2000), pp. 1670–1690.
[33] M. Sala and R. S. Tuminaro, *A new petrov-galerkin smoothed aggregation preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 31 (2008), pp. 143–166.
[34] J. Schroder and R. Falgout, *Non-Galerkin coarse-grid operators for AMG*, Submitted, (2013).
[35] J. B. Schroder, *Smoothed aggregation solvers for anisotropic diffusion*, Numerical Linear Algebra with Applications, 19 (2012), pp. 296–312.
[36] H. D. Sterck, K. Miller, E. Treister, and I. Yavneh, *Fast multilevel methods for Markov chains*, Numerical Linear Algebra with Application, 18 (2011), pp. 961–980.
[37] H. D. Sterck, U. M. Yang, and J. J. Heys, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM J. Matrix Anal. Appl, 27 (2006), pp. 1019–1039.
[38] K. Stüben, *Algebraic multigrid (AMG): an introduction with applications*, in Multigrid, O. C. Trottenberg, U. and A. Schuller, eds., Academic Press, 2001.
[39] R. Szeliski, *Locally adapted hierarchical basis preconditioning*, in ACM Transactions on Graphics (TOG), vol. 25, ACM, 2006, pp. 1135–1143.
[40] E. Treister, *Aggregation-based adaptive algebraic multigrid for sparse linear systems*, PhD Thesis, Technion, Israel Institute of Technology, Haifa 32000, Israel, September 2014.
[41] E. Treister and I. Yavneh, *Square and stretch multigrid for stochastic matrix eigenproblems*, Numerical Linear Algebra with Application, 17 (2010), pp. 229–251.
[42] ———, *On-the-fly adaptive smoothed aggregation multigrid for Markov chains*, SIAM Journal on Scientific Computing (SISC), 33 (2011), pp. 2927–2949.
[43] E. Treister, R. Zemach, and I. Yavneh, *Algebraic collocation coarse approximation (ACCA) multigrid*, Student paper, 12th Copper Mountain Conference on Iterative Methods, (2012).
[44] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, London and San Diego, 2001.
[45] P. Vanek, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.
[46] P. Vaněk and S. Míka, *Modification of two-level algorithm with overcorrection*, Appl. Math., 37 (1992), pp. 13–28.
[47] R. Wienands and I. Yavneh, *Collocation coarse approximation in multigrid*, SIAM J. Sci. Comput., 31 (2009), pp. 3643–3660.
[48] J. Xu and L. Zikatanov, *On an energy minimizing basis for algebraic multigrid methods*, Computing and Visualization in Science, 7 (2004), pp. 121–127.
[49] U. M. Yang, *Parallel algebraic multigrid methodshigh performance preconditioners*, Springer, 2006.
[50] ———, *On long-range interpolation operators for aggressive coarsening*, Numerical Linear Algebra With Applications, 17 (2010), pp. 453–472.
[51] I. Yavneh, *Coarse-grid correction for nonelliptic and singular perturbation problems*, SIAM J. Sci. Comput., 19 (1998), pp. 1682–1699.
[52] I. Yavneh, *Why multigrid methods are so efficient*, IEEE: Computing in Science and Engineering, 8 (2006), pp. 12–22.
[53] I. Yavneh, C. H. Venner, and A. Brandt, *Fast multigrid solution of the advection problem with closed characteristics*, To appear in SIAM J. Sci. Comput., (1998).