Smoothed aggregation for difficult stretched mesh and coefficient variation problems

Jonathan J. Hu | Christopher M. Siefert | Raymond S. Tuminaro

1Center for Computing Research, Sandia National Laboratories, Livermore, California, USA
2Center for Computing Research, Sandia National Laboratories, Albuquerque, New Mexico, USA

Correspondence
Jonathan J. Hu, Sandia National Laboratories, Livermore, CA, USA.
Email: jhu@sandia.gov

Funding information
Advanced Scientific Computing Research; Exascale Computing Program

1 of 26

1 INTRODUCTION

The smoothed aggregation algebraic multigrid (SA-AMG) algorithm was originally proposed over twenty years ago as an effective and scalable solution strategy for linear systems arising from discretized elliptic partial differential equations (PDEs). The basic form of the original algorithm has been employed without major mathematical modification within numerous applications to tackle a wide range of nontrivial problems. While generally successful, convergence difficulties can arise for some complex applications, even for matrices coming from elliptic PDEs.

Multigrid methods, including SA-AMG, are based on the idea that simple relaxation methods such as Jacobi generally smooth high frequency errors and that these smoothed errors can then be accurately represented and more efficiently reduced by relaxation iterations on a coarser grid representation of the linear system. This article considers modifications to the SA-AMG method, targeting some potentially vulnerable components that tend to be more fragile when irregular or anisotropic coarsening is needed. While simple relaxation methods generally smooth errors, they do not necessarily do so in a uniform or isotropic fashion. That is, errors after relaxation may be much smoother in certain directions than in others. In these cases, algebraic multigrid (AMG) coarsening must only occur in directions where errors are algebraically smooth (i.e., directions where the relaxation method significantly damps some local error components). These algebraically smooth errors are not necessarily geometrically smooth, but they do need to be well represented within the range of the coarse grid interpolation.

Irregular coarsening occurs when the strength-of-connection phase of the standard SA-AMG algorithm labels many nonzeros as weak. The presence of many weak connections accentuates some problematic facets of the
hierarchy construction process. Specifically, SA-AMG constructs a graph based only on the strong matrix connections (as determined by a strength-of-connection algorithm), which is then coarsened to generate coarse discretization operators. In general, sparser graphs (i.e., graphs with few connections) lead to less coarsening, larger coarse matrices, denser coarse matrices, and more total AMG levels. The overall effect is a potentially significant cost increase in both the setup and apply phases. Further, the presence of weak connections requires that a special filtered discretization matrix be devised in order to define the grid transfers. This filtering step can be somewhat fragile, and if not done properly can lead to suboptimal grid transfer operators. All of this is exacerbated by limitations in the standard strength-of-connection algorithms. The original strength-of-connection idea dates back to the 1980s and is motivated by $M$-matrix assumptions. While computationally inexpensive, it is easy to construct examples (see Section 1.2) where common strength criteria give a poor indication of the directions that correspond to algebraically smooth errors. To avoid the adverse convergence effects of coarsening in directions that are not aligned with algebraically smooth errors, often a somewhat large threshold or cutoff value is chosen to encourage the labeling of many connections as weak, and thus coarsen slowly. While there are some interesting alternatives to the classical coarsening approaches, that might reduce some of these ill effects, most of these alternatives are expensive and not fully robust. For this reason, these alternatives have not been generally adopted and are not considered further in this article.

Four algorithmic adaptations are proposed to address potential SA-AMG deficiencies that tend to arise in the presence of many weak connections. The new algorithms are algebraic and do not require any additional information or intervention from the application. Furthermore, these adaptations fit naturally into the existing SA-AMG setup workflow. The first idea considers an alternative diagonal approximation to the matrix inverse used within the Jacobi prolongator smoothing step. Relatively small diagonal entries can occur when formulating the filtered discretization operators. Unfortunately, these small entries are highly problematic for the Jacobi step. Effectively, we propose an alternative Jacobi-like step that uses the inverse of a diagonal matrix based on the 1-norm of individual rows of the filtered matrix. The second algorithm addresses how the filtered matrix is defined. Normally, the filtered diagonal entries are modified to reflect weak off-diagonal entries that are dropped from the original matrix. To prevent potentially small diagonal entries, we propose that dropped entries are accounted for by modifying off-diagonal nonzeros in some circumstances. The third variant introduces a set of constraints that the prolongator must normally satisfy (e.g., all entries lie between zero and one). The algorithm attempts to reformulate any SA-AMG prolongator row that violates the constraints by finding a suitable nearby row. The fourth and final algorithm introduces a second dropping or filtering stage that is performed after the standard SA-AMG coarsening algorithm. That is, the second filtering stage does not alter the coarsening process, but indirectly yields a sparser grid transfer operator by effectively sparsifying the matrix used within the prolongator smoothing step. Here, the aim is to reduce the cost of the solver while maintaining SA-AMG’s convergence properties. All four algorithms have been developed under an assumption that the linear system of interest is symmetric, though this is not a strict requirement for employing these schemes. All but one of the supporting numerical results involve the solution of symmetric linear systems. The one set of nonsymmetric systems (from a wind application) is mildly nonsymmetric, enough so that standard AMG preconditioning results in a convergent solver. In some cases, the four modifications do not significantly alter cost or convergence behavior (when compared to traditional SA-AMG). Under certain conditions, however, modified SA-AMG performs much better than the traditional version, especially for problems where irregular or anisotropic coarsening is needed.

The article is structured as follows. In Section 1.1 we give an overview of the SA-AMG method. In Sections 2.1–2.5, we present the new algorithmic variants for how SA-AMG generates the grid transfer. In Sections 3.1 and 3.2 we present results for Poisson and reaction-diffusion tests problem on a regular domain. In Section 3.5, we explore the effectiveness of the new algorithms on systems arising from a well-known reservoir model. In Section 3.6, we detail experiments on systems arising in a low-Mach computational fluid dynamics wind turbine simulation.

1.1 Smoothed aggregation

An example multigrid V cycle iteration is given in Algorithm 1 to solve

$$A_1 u_1 = f_1.$$ (1)
Algorithm 1. \( N_{\text{level}} \) multigrid V-cycle to solve \( A_\ell u_\ell = f_\ell \)

\[
\text{Vcycle}(A_\ell, f_\ell, u_\ell, \ell) \{
\text{if } \ell \neq N_{\text{level}} \{
    u_\ell = \hat{S}_\ell(A_\ell, f_\ell, u_\ell)
    \text{Vcycle}(A_{\ell+1}, P_\ell^T(f_\ell - A_\ell u_\ell), 0, \ell+1)
    u_\ell = u_\ell + P_\ell u_{\ell+1}
    u_\ell = \hat{S}_\ell(A_\ell, f_\ell, u_\ell)
\}
\text{else } u_\ell = A_\ell^{-1}f_\ell
\}
\]

To fully specify the algorithm, one must define relaxation procedures \( \hat{S}_\ell() \), \( \ell = 1, \ldots, N_{\text{level}} \) and the grid transfer operators \( P_\ell \), \( \ell = 1, \ldots, N_{\text{level}} - 1 \). Operator \( P_\ell \) interpolates solution updates from level \( \ell + 1 \) to level \( \ell \), while operator \( P_\ell^T \) restricts residuals from level \( \ell \) to level \( \ell + 1 \). The coarse grid discretization operators \( A_{\ell+1} \) (\( \ell \geq 1 \)) are defined by

\[
A_{\ell+1} = P_\ell^T A_\ell P_\ell. \tag{2}
\]

Smoothed aggregation is a particular type of AMG method\(^1,2,12\) to determine the \( P_\ell \)'s. Specifically, \( P_\ell \) is given by

\[
P_\ell = \left( I - \omega \bar{D}_\ell^{-1} \bar{A}_\ell \right) P^{(0)}_\ell, \tag{3}
\]

where the term in parenthesis is referred to as the prolongator smoothing step, \( \omega = \frac{4}{\lambda_{\ell,m}} \) is the damping factor, and \( P^{(0)}_\ell \) is the tentative prolongator. Here, \( \bar{A}_\ell \) is a filtered form of \( A_\ell \) with diagonal \( \bar{D}_\ell \), and \( \bar{A}_{\ell,j,m} \) is an approximation to the maximum eigenvalue of \( \bar{D}_\ell^{-1} \bar{A}_\ell \). By filtered, we mean that some entries are dropped, as described shortly. When all matrix connections are labeled as strong, \( \bar{A}_\ell = A_\ell \).

The tentative prolongator \( P^{(0)}_\ell \) must accurately interpolate certain near null space (kernel) components of the discrete operator \( A_\ell \). The prolongator smoothing step then improves the grid transfer operator by smoothing the basis functions associated with \( P^{(0)} \). When many off-diagonals of \( A_\ell \) are labeled as weak, \( \bar{A}_\ell \neq A_\ell \), and so a suitable \( \bar{A}_\ell \) must be defined. Before doing this, we first describe the coarsening process that defines \( P^{(0)}_\ell \). Though smoothed aggregation can be applied to PDE systems, this article focuses on scalar PDEs. In this case, \( P^{(0)}_\ell \) is an \( N_\ell \times N_{\ell+1} \) matrix with its sparsity pattern determined by a decomposition of the set of \( A_\ell \)'s graph vertices into \( N_{\ell+1} \) disjoint aggregates \( A^i_\ell \), such that

\[
\bigcup_{i=1}^{N_{\ell+1}} A^i_\ell = \{1, \ldots, N_\ell \} \quad A^i_\ell \cap A^j_\ell = \emptyset, \quad i \neq j. \tag{4}
\]

An ideal aggregate \( A^i_\ell \) is formed by grouping together a central or root vertex with its immediate neighbors (in the graph of \( \bar{A}_\ell \)) when both the central vertex and the neighbors are all unaggregated. Figure 1 illustrates two sets of ideal aggregates on a structured mesh. While ideal aggregates would only consist of a root vertex and its immediate neighbors, it is generally impossible to partition all of the vertices into ideal aggregates. Thus, some heuristics are needed to either create smaller aggregates or enlarge the ideal aggregates. Each aggregate on level \( \ell \) gives rise to one vertex on level \( \ell' + 1 \).

The nonzero values of \( P^{(0)}_\ell \) are defined by partitioning the near null space of \( A_\ell \) over the aggregates. In this article and for scalar PDEs, the near null space is simply the vector of all ones. This is related to the fact that constant functions lie in the null space of second order PDEs with the general form

\[
(a^{(1)}(x) u_x)_x + (a^{(2)}(x) u_y)_y + (a^{(3)}(x) u_z)_z = f, \quad x, y, z \in \Omega
\]

when only Neumann conditions are applied on the boundary of \( \Omega \). Here, \( a^{(k)}(x) \) are functions in \( H^1(\Omega) \). With this null space, \( P^{(0)}_\ell \) is given by
Two aggregation examples (aggregates indicated by blue enclosures) on a 2D structured mesh. Each red arrow indicates a strong $A_{ij}$ connection for vertex $i$ at the arrow base and vertex $j$ is the closest node in the direction pointed to by the arrow.

\[
(P^{(0)})_{ij} = \begin{cases} 
1 & \text{if } i \in \mathcal{A}_f^j \\
0 & \text{otherwise.}
\end{cases}
\] (5)

Near null space error components are not damped by conventional relaxation procedures and so this choice of $P^{(0)}$ guarantees that these constants are well represented on coarse levels. If the null space of $\tilde{A}_f$ is also defined by the space of constant vectors, then it is clear that the space of constants is contained not only in the range space of $P^{(0)}$ but also in the range space of $P_f$.

To complete the description of traditional SA-AMG for scalar PDEs, the filtered version $\tilde{A}_f$ of $A_f$ is given by

\[
(\tilde{A}_f)_{ij} = \begin{cases} 
(A_f)_{ij} & \text{if } i \neq j \text{ and } j \in S_{f,i} \\
(A_f)_{ii} + \sum_{k \in W_{f,i}} (A_f)_{ik} & \text{if } i = j \\
0 & \text{otherwise,}
\end{cases}
\] (6)

where $S_{f,i} \text{ and } W_{f,i}$ are the sets of strong (weak) neighbors of the $i$th vertex on level $f$. With this definition, the sum of the entries within each row of $A_f$ is equal to the sum of the entries within the corresponding $\tilde{A}_f$ row. Notice that if the constant is within the null space of $A_f$ (i.e., the sum of entries within a row is zero), it is also within the null space of $\tilde{A}_f$.

Finally, off-diagonals nonzeros are normally considered strong if and only if they satisfy

\[
| (A_f)_{ij} | \geq \theta \sqrt{(A_f)_{ii} \cdot (A_f)_{jj}},
\] (7)

for some user-defined threshold $\theta \in [0, 1]$. It should be noted that dropping entries and modifying the diagonal in this way does not necessarily preserve the SPD nature of a matrix. That is, $\tilde{A}$ is not guaranteed to be symmetric positive definite (SPD) when $A$ is SPD, though $\tilde{A}$ is guaranteed to be SPD when $A$ is an $M$-matrix. Generally, we do expect that the action of $A v \approx \tilde{A} v$ when $v$ corresponds to a smooth mode. This provides some expectation that the range space of a grid transfer produced by applying AMG to $\tilde{A}$ can accurately represent smooth modes associated with $A$.

1.2 | Prolongator smoothing and small diagonal entries

We now motivate Section 2 by considering some possible prolongator smoothing shortcomings. The prolongator smoothing step corresponds to one iteration of the damped Jacobi method applied to the matrix equation

\[
\tilde{A} P = 0,
\] (8)
starting with an initial guess of \( P = P^{(0)} \) and \( \bar{A} \) given by (6). Here, we now drop the subscript \( \ell' \) to simplify the notation for the remainder of the article. Of course, \( P = 0 \) is a trivial solution as is \( P = C \) when \( \bar{A} \)'s null space is given by the space of constant vectors. Here, \( C \) is a \( n \times m \) dense matrix with \( c_{ik} = c_{qk} \) for all \( i \) and \( q \) such that \( 1 \leq i, q \leq n \) and each \( k \) such that \( 1 \leq k \leq m \). While repeated Jacobi iterations converge to something uninteresting, one Jacobi step extends the nonzero support of the interpolation basis functions by one in the graph of \( \bar{A} \). When suitably damped, it also reduces the energy of these basis functions in the norm defined by \( ||p||_\lambda = \sqrt{p^T \bar{A} p} \) when \( \bar{A} \) is symmetric positive definite. Most AMG convergence theories rely on some bound for the energy of the interpolation basis functions. \( \bar{D}^{-1} \) can be viewed as an inexpensive approximation to \( \bar{A}^{-1} \) that will reduce high frequencies in \( P^{(0)} \)'s basis functions when used within a Jacobi step.

Undamped Jacobi is guaranteed to converge when \( \bar{A} \) is strictly diagonally dominant, that is, the magnitude of the diagonal entry in every row is greater than the sum of the absolute value of all other nonzeros in that row. When \( \bar{A} \) is symmetric positive definite but not strictly diagonally dominant, damping may be needed to ensure that the spectral radius of \( I - \omega \bar{D}^{-1} \bar{A} \) is less than one. For smoothed aggregation the damping parameter is typically chosen as \( \omega = 4/(3\lambda_m) \) and is based on a Chebyshev minimization principle. For strictly diagonally dominant matrices, it is easy to see that \( \lambda_m < 2 \) using the Gershgorin circle theorem. When the matrix is not strictly diagonally dominant, it is possible that \( \lambda_m \gg 2 \) or even undefined if some \( \bar{D}_{ii} = 0 \). Unfortunately, \( \lambda_m \) might be large even if only one row violates the strictly diagonally dominant condition. Obviously, the Jacobi step has little effect when \( \omega \) is small, which occurs for large \( \lambda_m \). Some discretization schemes can produce diagonally dominant matrices where the largest eigenvalue of \( \bar{D}^{-1} \bar{A} \), denoted \( \lambda_m \), is less than or equal to two. Here, \( \bar{A} \) is the discretization matrix and \( \bar{D} \) is the diagonal of the matrix. However, matrices arising from finite element discretization are generally not diagonally dominant, though often \( \lambda_m \) is still much larger than two. Unfortunately, dropping might lead to a \( \lambda_m \) that is much larger than \( \lambda_m \). In fact, a \( \bar{D}_{ii} \) might even become zero or negative. For example, a nodal linear finite element discretization of a Poisson operator on a mesh where all elements are \( 1 \times 1 \times h_c \) hexahedrons produces matrices with 27-point interior stencils. After scaling by \( 36h_c \), these identical stencils are given by \( 32 + 64h_c^2 \) (diagonal entry), \( -1 - 2h_c^2 \) (eight neighbors on cell corners), \( -4 - 2h_c^2 \) (eight neighbors sharing a \( x \) or \( y \) - \( z \) face), \( 16h_c^2 - 16 \) (two neighbors sharing a \( z \) edge), \( 2 - 8h_c^2 \) (four neighbors sharing a \( x \) - \( y \) face), and \( 8 - 8h_c^2 \) (four neighbors sharing a \( x \) or \( y \) edge). When \( h_c = \sqrt{7}/2 \), these stencils take on the values \( 144, -4.5, -7.5, 12, -12 \), and \(-6 \). For \( h_c > \sqrt{7}/2 \), the positive off-diagonal entry becomes the largest in magnitude value. Thus, any magnitude-based dropping criteria that employs a threshold that happens to retain the largest entry but drops all others will result in a negative \( \bar{D}_{ii} \). This follows from the fact that \( \bar{A} \)'s rows all sum to zero.

Further, small diagonal entries can be even more problematic for mildly nonsymmetric systems. While there may not be a damping parameter to guarantee Jacobi convergence for nonsymmetric matrices, it is still effective for matrices that are close to symmetric positive definite. In a wind simulation that will be described in Section 3.6, the discretization matrix includes a symmetric sub-block of only slightly smaller dimension than the entire matrix. However, a few of the nonsymmetric rows/columns have small diagonal entries, so that the nonsymmetric part of \( \bar{A} \) is emphasized in the matrix \( \bar{D}^{-1} \bar{A} \). We have observed that this causes severe eigenvalue convergence problems and very poor \( \lambda_m \) estimates coming from the typical power method used to provide these estimates.

## 2 | SMOOTHED AGGREGATION VARIATIONS

We now outline four algorithm variations to address some of the difficulties associated with weak connections and small aggregates.

### 2.1 | A 1-norm diagonal approximation

As noted, the \( \bar{D}^{-1} \) within the Jacobi iteration can be viewed as a diagonal approximation to \( \bar{A}^{-1} \). There are, however, alternatives, and one natural possibility is to replace \( \bar{D} \) by the diagonal matrix \( \bar{D} \) where \( \bar{D}_{ii} \) is the sum of the absolute values of nonzeros in the \( i \)th row of \( \bar{A} \). That is,

\[
\bar{D}_{ii} = \sum_j |\bar{A}_{ij}|,
\]  

(9)
which is the 1–norm of this ith row. One could argue that this 1–norm choice for \(D_{ii}\) better captures the scaling of the entire row and that an iterative procedure based on this will be less sensitive to the diagonal dominance properties of the matrix. This approximation is not new, and is related to a variant of the well-known pressure-correction algorithm, SIMPLE, that is referred to as SIMPLEC.\(^{14}\) As we will see, this relatively straight-forward change can have a significant convergence effect. Notice that the entries of this \(D\) matrix are larger (assuming more than one nonzero per row) than those of \(\tilde{D}\), and so the entries of \(\tilde{D}^{-1}\tilde{A}\) are smaller in magnitude than those of \(\tilde{D}^{-1}A\). Overall, however, it is the entries of \(\frac{1}{\lambda_m}\tilde{D}^{-1}A\) and \(\frac{1}{\lambda_m}\tilde{D}^{-1}A\) that appear within the Jacobi iteration, and these will be comparable when \(\tilde{A}\) is diagonally dominant, as \(\lambda_m\) will be smaller than \(\tilde{\lambda}_m\). For example, when every row of \(\tilde{A}\) is given by the stencil

\[
\begin{pmatrix}
-1 \\
-1 & 4 & -1 \\
-1
\end{pmatrix}
\]

then it is easy to see that \(\frac{1}{\lambda_m}\tilde{D}^{-1}A = \frac{1}{\lambda_m}\tilde{D}^{-1}A\). This case corresponds to a constant coefficient periodic Poisson problem where all diagonal entries of \(\tilde{D}\) are 4, all diagonal entries of \(D\) are 8, \(\tilde{\lambda}_m = 2\), and \(\lambda_m = 1\). Clearly, this exact equality will hold whenever \(\tilde{A}\) is defined by any circulant matrix. Thus, the proposed 1–norm diagonal modification will exactly reproduce the behavior of the traditional smoothed aggregation method in this case. We normally expect to see similar SA-AMG convergence behavior with either \(\tilde{D}\) or \(D\) when \(\tilde{A}\) is diagonally dominant. For more general problems, however, we will see that the behavior can be significantly different.

One interesting consequence of this \(\tilde{D}\) definition is that the magnitude of \(\tilde{\lambda}_m\), the maximum eigenvalue associated with \(\tilde{D}^{-1}\tilde{A}\), is always bounded by one, again as a consequence of the Gershgorin circle theorem. This implies that one can omit the eigenvalue calculation (which is problematic for the wind simulation in Section 3.6) and consider using one as an estimate for \(\tilde{\lambda}_m\). From the perspective of distributed memory parallel implementation, the 1-norm diagonal approximation requires no additional parallel communication, assuming that matrix rows are owned by a single MPI rank.

### 2.2 A safe guarded diagonal approximation

While this \(\tilde{D}_{ii}\) choice works well in practice, we propose one additional modification that is motivated by cases when the diagonal entry within a row of \(\tilde{A}\) is much larger than the sum of the magnitudes of the row’s off-diagonals. While this cannot happen for rows where the sum of the entries is zero, it might occur for matrix rows associated with or near Dirichlet boundary conditions. To better understand this large diagonal scenario, consider the extreme case when \(\tilde{A}\) is in fact a diagonal matrix. Then, \(\tilde{D}^{-1}\tilde{A} = I\), \(\tilde{\lambda}_m = 1\), and the prolongator smoothing step 3 becomes

\[
P = \left(I - \frac{4}{3\tilde{\lambda}_m}\tilde{D}^{-1}\tilde{A}\right)P^{(i)} = \left(I - \frac{4}{3}I\right)P^{(i)} = -\frac{1}{3}P^{(i)}.
\]

If instead the damping parameter had been defined as \(\omega = 1/(\tilde{\lambda}_m)\), which is not an unreasonable choice, then \(P\) would be identically zero. Essentially, the Jacobi iteration converges too rapidly to the \(P = 0\) solution when the diagonal is much larger than the sum of the magnitudes of the row’s off-diagonals. Instead of only reducing high frequencies, the whole spectrum is reduced, which is undesirable. If only a subset of rows have a very large diagonal, it would be better to use the corresponding row of \(P^{(i)}\) to somehow limit the damping. While this could be accomplished by setting \(\omega\) to a suitably small value, this would have the unintended consequence of limiting the smoothing effect for all rows, even those that are not so diagonally dominant. Instead, we prefer modifying or boosting the value of \(\tilde{D}_{ii}\) for any row where the modified Jacobi step significantly reduces low frequencies. This effectively corresponds to augmenting the definition of \(\tilde{D}_{ii}\) with a safeguard. To do this, we must have a criterion to detect the low frequency reduction.

To understand a possible remedy for low frequency reduction, consider the effect of the prolongator smoother step on just one row of the matrix. Specifically, assume that vertex \(i\) defines the center or root point of the \(k\)th aggregate. That is, \(i \in A_k\) and \(j \in A_k\) for any \(j\) such that \(\tilde{A}_{ij} \neq 0\). If we additionally take \(\tilde{\lambda}_m = 1\), then the \(i\)th row of the resulting prolongator
is given by

\[ P_{ij} = 1 - \frac{4}{3} \frac{s_i}{\tilde{D}_{ii}} \]

where \( s_i \) denotes the sum of \( \tilde{A} \)'s entries in the \( i \)th row. Note that if \( s_i = 0 \) and \( \tilde{D}_{ii} \neq 0 \), then \( P_{ij} = 1 \). One can argue that it is natural for a prolongator basis function to have a value of one at the aggregate’s central node and to then decay smoothly toward zero. This is analogous to the use of injection to interpolate points that are colocated on the fine and coarse grids. Here, a coarse unknown associated with an aggregate is viewed as being colocated with the root node. As the basis function should be largest at the root node, it should not have a negative value or be too small at this root vertex. In other words, we seek to enforce the condition that

\[ P_{ij} \geq \sigma, \]

where \( i \) is again the root node of the \( j \)th aggregate and \( \sigma \) is the minimum acceptable value for the prolongator basis function at the root vertex. This implies that \( \tilde{D}_{ii} \) must be chosen such that

\[ 1 - \frac{4}{3} \frac{s_i}{\tilde{D}_{ii}} \geq \sigma, \]

or

\[ \frac{4s_i}{3(1-\sigma)} \leq \tilde{D}_{ii}. \] (10)

Notice that when \( s_i = 0 \) and \( \sigma < 1 \), any non-negative \( \tilde{D}_{ii} \) will satisfy this condition. Thus, (9) can be used to define \( \tilde{D}_{ii} \). When \( s_i \) is not zero, an initial value computed via (9) is checked to see that it satisfies (10). If this second condition is not satisfied, (9) is discarded and instead the smallest \( \tilde{D}_{ii} \) satisfying (10) is used to define \( \tilde{D}_{ii} \). While the precise choice of \( \sigma \) is not obvious, we have found experimentally that \( \sigma = 1/3 \) works well. For this choice of \( \sigma \), (10) becomes

\[ 2s_i \leq \tilde{D}_{ii}. \] (11)

Thus, we first compute \( \tilde{D}_{ii} \) using (9). For \( D_{ii} \) that are identically zero (due to an entire zero row of \( \tilde{A} \)) we set \( \tilde{D}_{ii} = 1 \), and for those remaining rows violating (11) we set \( \tilde{D}_{ii} = 2s_i \). While the arguments were motivated by considering \( i \) to be a root node of an aggregate, we apply this criteria for all rows in the matrix, regardless as to whether or not the node is a root node. At nonroot nodes, the associated \( P_{ij} \) can be smaller than \( \sigma \) due to the fact \( s_i \) does not reflect the \( \tilde{A}P^{(t)} \) product. From the perspective of distributed memory parallel implementation, the safeguarded diagonal modification also requires no additional parallel communication, assuming that matrix rows are owned by a single MPI rank.

### 2.3 Alternative lumping strategies

As already noted, near null space vectors should be accurately represented in the range space of the interpolation operator. If the null space of \( A \) coincides with that of \( \tilde{A} \) and is also contained within the range space of \( P^{(t)} \), then it will additionally be contained within the range space of the smoothed prolongator. To force the two null spaces to coincide, \( \tilde{A} \)'s entries must be modified to account for dropped nonzeros. This is traditionally accomplished by only changing the diagonal. As discussed, any dropped off-diagonal \( A_{ij} \) is simply added to \( A_{ii} \) to define \( \tilde{A}_{ii} \). Unfortunately, this simple process can drastically alter the properties of the resulting filtered matrix. In extreme cases, it is possible that \( A_{ii} \) and \( \tilde{A}_{ii} \) have opposite signs. More generally, the diagonal dominance properties of a row might change. That is, we could have

\[ D(\tilde{A}_i) \gg D(A_i) \]  \text{ where } D(A_i) = \frac{\sum_{j \neq i} |A_{ij}|}{|A_{ii}|}.

Here, \( A_i \) refers to the \( i \)th row of \( A \) and \( D() \) measures the magnitude of off-diagonal entries relative to the diagonal. Notice that \( D(A_i) = 1 \) when all off-diagonals are negative and the sum of \( A_i \)'s entries are zero. In this case, any reasonable lumping
strategy will result in \( D(\bar{A}_i) = 1 \). However, this will not be the case when there are both positive and negative off-diagonals or when \( A_i \)’s entries do not sum to zero (e.g., at a boundary). For the most part, small values of \( D() \) are preferred by a damped Jacobi iterative method (i.e., the prolongator smoothing step) as its convergence rate is generally more rapid for matrices with small off-diagonal entries relative to the diagonal entries. This suggests that it might be effective to consider a strategy that restricts or limits the lumping of terms to the diagonal to avoid significant growth in the resulting \( D(\bar{A}_i) \). There are many possible such strategies. We now describe a scheme that is primarily oriented toward scalar diffusion-type PDEs, and so may not be appropriate for other operators.

For scalar diffusion-like PDEs, one can argue that positive off-diagonals entries are somewhat irregular. To see this, consider the simplified prolongator smoother step \( u = (I - D^{-1}A)v \). When all off-diagonals are negative and when the constant is in the null space of \( A \), then each \( u_i \) is just a weighted average of the \( v_i \)’s within its immediate neighborhood. In this way, the relaxation process mimics a diffusion process associated with a heat equation. When some off-diagonal entries are instead positive, then some weights will be negative, which no longer resembles a diffusion process. This can be made more rigorous by consider the relationship between Jacobi iterations and time marching for ordinary differential equations. Thus, a possible lumping algorithm might make decisions based on the sign of matrix entries. Our overall lumping strategy considers modifying or perturbing the retained positive off-diagonals, the diagonal, or the retained negative off-diagonals in this order of preference when the perturbation is negative. The general aim is to enforce \( D(\bar{A}_i) \leq \tau D(A_i) \) where \( \tau \) is a user-supplied growth factor.

Algorithm 2. Alternative lumping procedure for prolongator smoothing step

\[
\bar{A}_i = \text{Lump\_AvoidSmallDiag}(A_i, R_i, \tau)
\]

Input:
- \( A_i \) \( i \)th row of matrix with entries to be dropped
- \( R_i \) set of column indices in \( i \)th row to be removed
- \( \tau \) tolerance indicating that \( D(\bar{A}_i) \) should not exceed \( \tau D(A_i) \)

Output:
- \( A_i \) matrix row where \( A_{ij} = 0 \) for \( j \in R_i \) and \( \bar{A}_i v = A_i v \) where \( v \) is a constant vector

1. Let \( r_i \leftarrow \sum_{k \in R_i} A_{ik} \)
2. if \( r_i > 0 \) then \( \bar{A}_{ii} \leftarrow A_{ii} + r_i \) //decreases \( D(\bar{A}_i) \)
3. else {
   4. Let \( K^+ \leftarrow \{ k \mid A_{ik} > 0 \land k \neq i \land k \notin R_i \} \)
   5. Let \( K^- \leftarrow \{ k \mid A_{ik} < 0 \land k \neq i \land k \notin R_i \} \)
   6. Let \( \kappa^+ = \sum_{k \in K^+} A_{ik}; \kappa^- = \sum_{k \in K^-} A_{ik}; \)
   7. if \( |r_i| \leq \kappa^+ \) then \( \bar{A}_{ij} \leftarrow A_{ij}(1 + \delta_i) \) for \( j \in K^+ \) where \( \delta_i \leftarrow r_i / \kappa^+ \)
   8. else {
      9. \( \bar{A}_{ij} \leftarrow 0 \) for \( j \in K^+ \) //zero out the \( K^+ \) by distributing a
      10. \( \hat{r}_i \leftarrow r_i + \kappa^+ \) //portion of \( r_i \) (= \( \kappa^+ \)) to them
      11. if \( K^- = \emptyset \) then redistribute to \( K^+ \) if possible or if not possible do not modify row \( i \) and return
      12. else {
         13. find largest positive \( r_i^* < \min(d_{ii}, |\hat{r}_i|) \) such that \( D(\bar{A}_i) \leq \tau D(A_i) \)
         14. define \( \bar{A}_i \) such that its only nonzero values are
         15. \( \bar{A}_{ii} \leftarrow A_{ii} - r_i^* \)
         16. \( \bar{A}_{ij} \leftarrow A_{ij}(1 + \delta_i) \) for \( i \in K^- \)
         17. where \( \delta_i \leftarrow (\hat{r}_i + r_i^*) / \kappa^- \)
         18. }
   19. }
20. }
21. }
A detailed algorithm description is given in Algorithm 2. The algorithm is supplied a set of indices \( \mathcal{R}_i \) denoting the nonzero columns that should be removed from the \( i \)th row. The sum of these entries \( r_i \) must be then distributed to the kept entries of \( \tilde{A}_i \). If this sum is positive, then only the diagonal is modified in Line 2 as this lowers \( D(\tilde{A}_i) \). If instead \( r_i \) is negative, then more care is necessary. We first split the set of kept indices into two subsets \( \mathcal{K}^+ \) and \( \mathcal{K}^- \) corresponding to entries that have positive \( A_{ij} \) values or negative \( A_{ij} \) values, respectively. The sum of the nonzero values associated with these two sets is denoted by \( \kappa^+_i \) and \( \kappa^-_i \), respectively. Line 7 corresponds to the case when all of the lumping can be distributed to the positive kept entries without creating any new negative entries. If this is impossible, we distribute a portion of \( r_i \) equal to \( -\kappa^+_i \) to the \( \mathcal{K}^+ \), effectively zeroing them out. That is, we prefer not creating new negative entries as this might fundamentally change the equation's character. For the remaining \( \tilde{r}_i = r_i + \kappa^+_i \), we seek in Line 14 the largest magnitude perturbation, \( |\tilde{r}_i^*| \), to the diagonal that does not violate the \( D(\cdot) \) growth restriction. The remaining \( \tilde{r}_i - \tilde{r}_i^* \) is then distributed proportionally to the negative kept off-diagonals. In many cases, \( \tilde{r}_i^* = \tilde{r}_i \), so \( \tilde{r}_i \) is lumped entirely to the diagonal. When this is not true, \( \tilde{r}_i^* \) is generally given by

\[
\tilde{r}_i^* = \frac{\tilde{r}_i + \kappa^-_i + \tau D(A_i)A_{ii}}{1 - \tau D(A_i)}.
\]

This is obtained by some algebraic manipulations after first setting \( D(\tilde{A}_i) = \tau D(A_i) \) and recognizing that the sum of the absolute values of \( \tilde{A}_i \)'s off-diagonals is \((-\kappa^-_i)(1 + \delta)\) while its diagonal is \( A_{ii} + \tilde{r}_i^* \).

However, safe-guards must be added for situations where there is no suitable value of \( r_i^* \) satisfying the \( D(\cdot) \) growth restriction. This might occur if the set \( \mathcal{K}^- \) is empty. If it is instead possible to satisfy the \( D(\cdot) \) growth restriction by further lumping to \( \mathcal{K}^+ \), then this is done even though these off-diagonals now become negative. Otherwise, if there are no kept off-diagonal entries in the row, we skip the perturbation entirely, no longer preserving the row sums in these problematic rows. As with the other algorithm modifications, the alternative lumping strategy requires no additional parallel communication on a distributed memory parallel machine, assuming matrix rows are owned by a single MPI rank.

### 2.4 Prolongator constraints

From a geometric multigrid perspective, one can argue that the entries of \( P \) should lie between 0 and 1 inclusive. For example, consider the vector \( u = P e^{(j)} \) where \( e^{(j)} \) is a canonical basis vector with only one nonzero entry, the \( j \)th element, that is set to one. It is clear that all entries of \( u \) should be positive and not greater than one for any sort of geometric interpolation scheme (as opposed to an extrapolation scheme). For smoothed aggregation, this connection is a little less apparent. Obviously, the tentative prolongator \( P^{(i)} \) satisfies these constraints when the null space of \( A \) is given by the vector of all ones. As the objective of the prolongator smoother step is to produce low energy grid transfer basis functions, it can also be argued that these smoothed basis functions should decay smoothly from the peak value to zero. Thus, these basis functions should not be negative anywhere. Though somewhat less obvious, the peak will typically be either one or less than one. Specifically, smoothed prolongator basis functions and tentative prolongator basis functions will coincide for vertices that are not part of the aggregate boundary. This is a property of the null space of \( \tilde{A} \) being a constant function in the case of the Laplace operator and was discussed for ideal aggregates in Section 2.2. The remaining nonzero basis function entries will typically be less than one due to the energy minimization and smooth decay properties just mentioned. When instead prolongator values do not lie between zero and one inclusive, it is often an indicator that the smoothing of some basis functions is suboptimal. In these cases, one can consider enforcing a condition that all \( \bar{P}_{ij} \) lie between zero and one. Specifically, one can construct a minimization problem for the \( i \)th row of \( \bar{P} \)

\[
\begin{cases}
\bar{P}_i = \text{argmin} \ | | \bar{P}_i - P_i | |_2 \\
\text{subject to } \bar{P}_{ij} = 0 \text{ if } P_{ij} = 0, \quad \bar{P}_{ij} \geq 0, \quad \bar{P}_{ij} \leq 1, \quad \bar{P}_i v = P_i v
\end{cases}
\]

where \( v \) is the vector of all ones. That is, find a new prolongator row that is closest to \( P_i \) and satisfies the two bound constraints, has a sparsity pattern not extending beyond \( P_i \)'s pattern, and where the sum of \( P_i \)'s entries and \( \bar{P}_i \)'s entries are
identical. If the row sum of $P_i$ is negative, then there is no feasible solution to this minimization problem. If the $P_i$ row sum is zero, then the only solution is that $P_i$ is identically zero. There is also no feasible solution when the row sum is greater than the number of nonzeros in the sparsity pattern of $P_i$. When the minimization problem has a feasible solution, it can be obtained by the procedure described in Algorithm 3.

**Algorithm 3.** Constrain One P Row($P_i$, lowerBound, upperBound)

```plaintext
optimal ← false
k ← 0
ℓ_k ← arg min_{\ell_k} P_i[\ell_k] > lowerBound
u_k ← arg min_{u_k} P_i[u_k] > upperBound
while (optimal == false) {
    δ_k ← (\sum_{j=1}^{u_k} P_i[j] - \sum_{j=1}^{nnz(P_i)} P_i[j]) / (u_k - ℓ_k)
    // does $P_i[\ell_k] - \delta_k$ still satisfy lowerBound & does $P_i[u_k] - \delta_k$ still violate upperBound
    if $P_i[\ell_k] - lowerBound \leq P_i[u_k] - upperBound$, optimal ← true
    else {
        if $P_i[\ell_k] - lowerBound \leq P_i[u_k] - upperBound$, $\ell_k ← \ell_k + 1$
        else $u_k ← u_k + 1$
    }
    k ← k + 1
}  
for $j = 1 : \ell_k - 1$, $P_i[j] ← lowerBound$
for $j = \ell_k : u_k - 1$, $P_i[j] ← P_i[j] - \delta_k$
for $j = u_k : nnz(P_i)$, $P_i[j] ← upperBound$
```

The minimization task can be solved by a two-step process. The first step shifts all nonzero entries by $\delta$ while the second step enforces the bounds (i.e., sets any entry to the bound value if it violated this bound after step one). To understand why the two-step process yields an optimal solution, consider the simpler case where one seeks to only alter the sum of a row’s nonzero entries without introducing additional nonzeros. In this situation where there are no bound constraints, it is easy to show that $||P_i - \tilde{P}_i||_2$ is minimized by taking $\tilde{P}_i = P_i - \delta$ for each $j$ corresponding to $P_i$’s nonzeros. The desired row sum adjustment is given by $\delta \; nnz(P_i)$, where $nnz(P_i)$ is the number of nonzeros in $P_i$. In the case of our optimization problem, the minimum solution is also obtained by adjusting entries via a constant $\delta$ shift, which is now necessary to fix the row sum that is altered by satisfying bound constraints. The main complication is that the proper value of $\delta$ depends on knowing how many entries must be adjusted in the second step, which in turn is also a function of $\delta$ (as the $\delta$-shift might change the number of entries that violate bounds). Algorithm 3 is given and is based on guessing entries that will violate bounds after shifting by $\delta$, and computing a $\delta$ for this guess. We then either verify that this guess is correct or modify the set of entries representing bound violations (by adding or removing one entry). The process is repeated until the actual set of entries that must be adjusted in step two is identified. The initial guess corresponds to entries that violate the bounds when $\delta = 0$ in step one. To simply the Algorithm 3 description, it is assumed that a feasible solution exists, nonzero entries are sorted within an array denoted by $P_i[\ ]$, and $\delta$ is positive. We refer to $\delta_k$ as the $k$th guess shift value that restores the row sum assuming that only entries $1$ to $\ell_k - 1$ violate the lower bound and only entries $u_k$ to $nnz(P_i)$ violate the upper bound. Matlab notation is used so that $a : b$ refers to the consecutive sequence of integers from $a$ to $b$ inclusive. Some safeguards should be included to address cases where
\( \ell_k \) or \( u_k \) might be larger than \( nnz(P_i) \). The algorithm while loop must terminate as each iteration either determines that an optimal solution has been found or \( \ell_k \) or \( u_k \) is incremented, which eventually triggers a safeguard that terminates the loop. Further, the first summation in the \( \delta_k \) computation can be performed outside of the loop while the second summation can simply be updated. When the constraints cannot be satisfied within the \( i \)th row, then we simply take \( \overline{F}_i = P_t^{(i)} \). From the perspective of distributed memory parallel implementation, the prolongator constraints require no additional parallel communication, assuming that matrix rows are owned by a single MPI rank.

### 2.5 Further sparsification

To understand a possible further sparsification of the smoothed aggregation prolongator operator, we first review the aggregation process within smoothed aggregation. As noted, smoothed aggregation applies an algorithm to the graph of \( \overline{A} \) to construct aggregates \( A_j \) such that each fine mesh vertex belongs to only one aggregate. The basic idea is that a root node is first chosen and then an initial aggregate is defined as the root node and all of its strong neighbors. Each root node is chosen among vertices that have not yet been aggregated and are not adjacent (via strong connections) to any existing already aggregated vertex. This aggregation procedure is repeated until it is no longer possible to find such a root node as all unassigned vertices are adjacent to assigned vertices. At this juncture, some heuristics are needed to assign these remaining unassigned vertices by either creating new aggregates or enlarging existing aggregates. This implies that most aggregates are composed of a central root node and its strong neighbors, thus the shape of the aggregates is primarily governed by the strong neighbors of the root node. A typical aggregate will have a diameter of length 3. Figure 1 illustrates two aggregate scenarios on a regular mesh. In the leftmost image, all connections are strong and the corresponding aggregates happen to be perfect squares. In the rightmost image, only the vertical connections are strong with the exception of a few horizontal edges that in this contrived example never coincide with edges emanating from a root node. Once again, the aggregates are perfect, consisting of three points aligned in the vertical direction. That is, the aggregate shapes are determined by the root nodes, which only have strong vertical connections. The main issue is that the horizontal connections shown in the rightmost image will lead to nonzero fill-in within the coarse level discretization matrix due to the prolongator smoothing step. Specifically, additional nonzeros connections arise between coarse vertices associated with non-neighboring aggregates (distance two aggregates) in the horizontal direction. In particular, it is easy to show that for nontrivial \( \overline{A} \) matrices a nonzero \((i,j)\) entry occurs in the coarse matrix discretization whenever \((P^{(i)}A\overline{A}^T)_{ij}\) is nonzero where \(P^{(i)}\) refers to the \( i \)th row and \(A\overline{A}^T\) to the \( j \)th column of \( P^{(i)} \). This will certainly occur if there is a distance 3 path in the filtered graph (along the red arrows in Figure 1). More generally, fill-in between distant aggregates can occur due to a conflict between the characterization of strong connections between the root node and the nonroot nodes. Specifically, the root node indicates that connections to certain neighboring aggregates are weak while some member of the root node’s aggregate has a strong connection to this very same aggregate. To reduce fill-in, we can look for conflicts and reliable some conflicting connections.

As our conflict characterization is based on aggregate choices, it is most practical to only consider nonroot strong connections for relabeling as these do not alter the definition of the already-chosen aggregates. This relabeling corresponds to a further sparsification of \( \overline{A} \) that is performed immediately preceding the prolongator smoother step. This sparsification occurs aggregate-by-aggregate. First, the weak connections of the root node are examined to determine the neighboring aggregates (in the graph of \( A \)) associated with these weak connections. If these weak-neighbor aggregates have no strong connections to the root node, then these aggregates are put into a set \( \text{CandidatesForPruning} \). Second, we examine all the nonroot vertices in the aggregate looking for strong connections to any vertex within each \( \text{CandidatesForPruning} \) aggregate. If there is just one strong connection to a particular aggregate in \( \text{CandidatesForPruning} \), this strong connection is relabeled as weak and dropped from the \( \overline{A} \) that will be used in the prolongator smoothing step. The diagonal entry is modified to reflect the dropped entry following the usual method. Thus, strong connections to each aggregate in \( \text{CandidatesForPruning} \) are retained if there are multiple strong connections but dropped if there is only one strong connection. The full algorithm is described in Algorithm 4.
Algorithm 4. Further sparsification procedure for prolongator smoothing step

Let $agg_i$ be aggregate that node $i$ belongs to
Let $CFP_K$ be the set of candidate aggregates whose connections in $\tilde{A}$ to aggregate $K$ are considered for pruning from $\tilde{A}$.
Let $W_i$ be the set of nodes that are weakly connected to node $i$ w.r.t. graph of $A$.

for each aggregate $K$ do
  Let $r$ be the root node of aggregate $K$.
  Set $CFP_K = \emptyset$.
  for $i \in W_r$ do
    if there are no strong connections between $r$ and any node in agg, then
      $CFP_K \leftarrow CFP_K \cup \{agg_i\}$
    end
  end
  for each aggregate $C \in CFP_K$ do
    numStrongConnections $\leftarrow 0$
    for each node $i \in K\setminus\{r\}$ do
      for each node $j \in C$ do
        if $i$ and $j$ are strongly connected w.r.t. $A$ then
          numStrongConnections $\leftarrow$ numStrongConnections $+ 1$
          $\hat{i} \leftarrow i, \hat{j} \leftarrow j$
        end
      end
      if numStrongConnections == 1 then
        $A_{\hat{i}\hat{i}} \leftarrow A_{\hat{i}\hat{i}} + A_{\hat{j}\hat{j}}$
        Set $A_{\hat{j}\hat{j}} \leftarrow 0$
        $A_{\hat{j}\hat{i}} \leftarrow A_{\hat{i}\hat{j}} + A_{\hat{j}\hat{i}}$
        Set $A_{\hat{i}\hat{j}} \leftarrow 0$
      end
    end
  end
end

This leads to a further sparsification of $\tilde{A}$ that may now be nonsymmetric even if $\tilde{A}$ is symmetric. To remedy this, all $(j, i)$ entries are dropped if the associated $(i, j)$ entry was dropped in the relabeling phase. This restores the symmetry in the resulting filtered matrix, denoted as $\hat{A}$, that is then used in the prolongator smoothing step. As the final prolongator is now sparser, we can expect that the amount of fill-in will be reduced on coarse level matrices. However, this may also cause the convergence rate to be somewhat slower. From the perspective of distributed memory parallel implementation, sparsification of $\tilde{A}$ (symmetrized or not) requires no additional parallel communication, as long as aggregates do not cross processor boundaries.

3 | NUMERICAL RESULTS

Several sample problems are presented to examine the behavior of the four proposed SA-AMG algorithm variations. The first few correspond to somewhat academic, though potentially difficult, problems. The last two come from actual applications areas. One is the SPE10 benchmark the Society of Petroleum Engineers, which has highly heterogeneous material jumps. The second comes from a wind turbine simulation where the underlying mesh has problematic aspect ratios within some parts of the domain. For problems where timing information is presented, Sandia’s eclipse machine was used. Eclipse nodes each have two Intel Xeon E5-2695 (Broadwell) CPUs with 18 cores each and a total of 128 GB of RAM per node.
3.1 Randomly perturbed cube

The first test problem considers a Poisson equation

\[-\nabla^2 u(x) = f(x) \quad x \text{ in } \Omega,\]
\[u(x) = u_0(x) \quad x \text{ on } \partial \Omega,\]  \hspace{1cm} (12)

where the domain is defined by \(\Omega = [0, 1] \times [0, 1] \times [0, 100]\), the forcing function \(f(x)\) is identically zero, and \(u_0(x) = 1 + x_1 + x_2 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_1x_2x_3\). We discretize (12) using linear hexahedral finite elements on a tensor product mesh using 60 elements in each coordinate direction. The element sizes are given by a perturbation of a uniform spacing. Specifically, along each coordinate direction mesh points are randomly perturbed by up to 20% from the uniform spacing location (details given in Appendix A). This leads to a mesh where the spacing in the first two coordinate directions is generally finer than that in the third dimension. For our experiments, we generated 50 different test meshes using different random seeds.

The smoothed aggregation multigrid solver in MueLu\textsuperscript{15} is used as a preconditioner for the conjugate gradient (CG) method to solve the 50 linear systems. It is well-known that linear hexahedral elements on highly stretched meshes give rise to matrix coefficients that are problematic from a strength-of-connection perspective. That is, the magnitude of the matrix entries are not well correlated with mesh stretching. For this reason, an alternative matrix termed a distance Laplacian, \(L^{(d)}\), is used in this experiment for the aggregation/coarsening phase of the algorithm. Specifically, strong connections satisfy \(|L^{(d)}_{ij}| \geq \theta \sqrt{L^{(d)}_{ii} L^{(d)}_{jj}}\). The matrix \(L^{(d)}\) has the same nonzero pattern as the discretization matrix \(A\). The \(L^{(d)}\) off-diagonal values, however, are defined as the reciprocal of the negative distance between the \(i\)th and \(j\)th coordinate, requiring coordinates be supplied to the solver. The diagonal is then chosen so that sum of all entries within each row is identically zero. Here, \(\theta\) is taken as 0.025, which was determined experimentally to produce desirable aggregates that are primarily oriented along the first two coordinate directions. The smoothed aggregation hierarchy was generated so that the resulting discretization matrix is coarsened until only 1000 or fewer unknowns remain, at which point a direct solver is applied. Two sweeps of Chebyshev pre and postsmothing are applied on all other levels. The Chebyshev eigenvalue interval is given by \([\lambda^+ /10, \lambda^+]\) where \(\lambda^+\) is an estimate of the maximum eigenvalue of \(A\) obtained by 10 sweeps of the power method. The conjugate gradient iteration is terminated when the residual is reduced by a factor of \(10^{-10}\).

We now consider all 16 combinations of enalabling and disabling the four adaptations: the \(1\text{Norm}\) diagonal modification of Sections 2.1–2.2, the \(\text{OffLmp}\) lumping modification of Section 2.3, the \(\text{Cnstrnt}\) constraint enforcement of Section 2.4, and the \(\text{Sprsfy}\) sparsification of Section 2.5. Disabling all four options corresponds to traditional smoothed aggregation. In all experiments involving \(\text{OffLmp}\), we choose \(\tau = 1.1\) as the maximum allowable \(D()\) growth. Results can be found in Table 1.

We note first that traditional SA-AMG (all options off) fails in 16 of the 50 test cases. These failures arise from a negative eigenvalue estimate for \(\lambda_m\) in (3) due to the \(A\) matrix, which has poor diagonal dominance properties. These negative eigenvalues lead to catastrophic failures within the solver. Here, operator complexity is defined as the ratio of the number of nonzeros within all the hierarchy discretization matrices divided by the number of nonzeros for the finest level matrix. Three algorithms used by themselves (or in combination)—\(1\text{Norm}, \text{Sprsfy}, \text{and OffLmp}\)—prevent all

| Cnstrnt | \(\text{Sprsfy off}\) | \(\text{Sprsfy on}\) |
|---------|-----------------|-----------------|
| \(\text{OffLmp off}\) | \(26.1 \ (1.28)\) | \(27.9 \ (1.25)\) | \(26.1 \ (1.25)\) |
| \(\text{OffLmp on}\) | \(23.6 \ (1.28)\) | \(24.9 \ (1.25)\) | \(23.4 \ (1.25)\) |
| \(\text{Cnstrnt on}\) | \(19.3 \ (1.28)\) | \(21.2 \ (1.25)\) | \(18.2 \ (1.25)\) |

Note: Each algorithm denoted in red had 16 solution failures due to negative eigenvalues. These failures are ignored for iteration counts and operator complexities.
negative eigenvalues in the test problem. In the case of 1Norm or OffLmp, the diagonal entries are less sensitive to the dropping schemes that lead to small values in $\hat{D}$. The Sprsfy algorithm, indirectly removes some problematic entries from $\hat{A}$ which would otherwise lead to negative eigenvalues. The Cnstrnt algorithm has no effect on reducing the number of failures.

Second we note that OffLmp and 1Norm each modestly reduce the iteration counts whenever they are used (either individually or in combination), while Sprsfy tends to slightly increase the iteration counts on average. The Cnstrnt algorithm yields a substantial decrease in iterations, though it needs to be used in combination with at least one other option to avoid failures. For example, the combination of Cnstrnt and OffLmp reduced the iteration count from 26.1 with 32% failures to 16.0 with no failures. This corresponds to an average iteration reduction of 60% (not counting the failures). Finally, we note Sprsfy consistently leads to a small reduction in operator complexity, 1.28 to 1.25, whenever it is used.

3.2 Triaxially stretched cube

We take $(k_x, k_y, k_z) \in \{1, 5, 10\}^3$, restricted to $k_x \leq k_y \leq k_z$, yielding 10 different configurations. For each $(k_x, k_y, k_z)$ we define a domain $\Omega = [0, 3k_x + 3] \times [0, 3k_y + 3] \times [0, 3k_z + 3]$. On $\Omega$, we solve the reaction-diffusion equation

$$\sigma^{-1}u(x) - \nabla^2 u(x) = f(x) \quad x \text{ in } \Omega,$$

$$u(x) = u_D(x) \quad x \text{ on } \partial\Omega,$$

where the forcing function $f(x)$ is identically zero, $u_D(x) = 1 + x_1 + x_2 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_1x_2x_3$ and the reaction term, $\sigma$ is chosen to be $10, 10^2, 10^3, 10^4,$ or $10^5$. Notice that the nonzero $\sigma$ implies that the matrix has a positive row sum. We discretize (13) using linear hexahedral finite elements on a tensor product mesh using 60 elements in each coordinate direction. The element sizes linearly vary from 0.1 to $k_x/10$ in the $x$-dimension (and similarly in the $y$ and $z$ dimensions). Details are given in Appendix B.

Following Section 3.1, CG with a $10^{-10}$ tolerance is used with a MueLu preconditioner to solve the linear system. Again, the distance Laplacian criterion is used for the aggregation/coarsening phase of the algorithm with $\theta = 0.025$. Two sweeps of Chebyshev smoothing is applied (with identical parameters to Section 3.1) to all levels except the coarsest level, where a direct solver is used.

First we consider the traditional SA-AMG algorithm and 1Norm modifications. Figure 2a presents a probability histogram of the number of iterations taken by traditional SA-AMG minus the number of iterations taken with the 1Norm diagonal modification. We note that in all fifty cases, the 1Norm modification takes no more than the number of iterations taken by traditional SA-AMG. While 20 of the 1Norm modification runs are within three iterations of traditional SA-AMG, four runs have a difference of 15 or more iterations, which is a substantial savings. Similarly, Figure 2b shows the same
results for the OffLmp lumping modification, with 19 of the runs within three iterations of traditional SA-AMG, and four runs with a difference of 15 or more iterations. We do not show results for the Cnstnt constraint enforcement, but note that 40 of those runs are within three iterations of traditional SA-AMG while no runs had an iteration difference of 15 or more. More problematically, two of the 50 runs fail due to negative eigenvalues. We also do not show results for Sprsfy sparsification, as in three cases it increases the iteration count substantially. Thus we cannot recommend using Cnstnt or Sprsfy on this problem, unless they are paired with either one of the other modifications which can overcome their deficiencies.

3.3 Radial trisection problem

We now consider a class of stretched radial trisection meshes of a quarter of a cylinder, which are structured radial meshes far from the origin, but contain an unstructured “trisection” region near the origin. We show a close-up of the near-origin region for a particular mesh from our dataset in Figure 3. We solve (12) with the same solution as in Section 3.1, though on a quarter cylinder domain, with Dirichlet boundary conditions applied on all surfaces of the quarter cylinder. The z-extent of the domain is [0, 20] while the radial extent will be varied from 200 to 10,000. A small unstructured region of radius six is meshed near the origin, while the rest of the domain is a structured radial mesh. This mesh is graded with the radial length of the element adjacent to the trisection region of approximately 1, and the outermost element having a substantially larger radial length. The mesh is structured in the z direction. We discretize (12) using linear hexahedral finite elements on the mesh. Details are given in Appendix C. We choose 41 different values of radial length of the largest element in the outer region between 1.0658 and 100. The inner region is meshed in five different ways based on setting the trisection blocks parameter in the input deck in Appendix C to 1, 2, 3, 4, or 6. This leads to a total of 205 matrices tested.

Following Section 3.1, CG with a $10^{-10}$ tolerance is used with a MueLu preconditioner to solve the linear system. Again, the distance Laplacian criterion is used for the aggregation/coarsening phase of the algorithm with $\theta = 0.025$. Two sweeps of Chebyshev smoothing is applied (with identical parameters to Section 3.1) to all levels except the coarsest level, where a direct solver is used.

We now consider the traditional SA-AMG algorithm and 1Norm modifications. Figure 4a presents a probability histogram of the number of iterations taken by traditional SA-AMG minus the number of iterations taken with the 1Norm diagonal modification. We note that in all 205 cases, the 1Norm modification takes no more than the number of iterations taken by traditional SA-AMG. Of the 1Norm modification runs, 185 are within three iterations of traditional SA-AMG, five runs have a difference of seven or more iterations, which is a reasonable savings. Similarly, Figure 4b shows the same results for the OffLmp lumping modification, with the 182 runs within three iterations of traditional SA-AMG, and five with a difference of seven or more iterations. We do not show results for the Cnstnt constraint enforcement, but note that 165 of those runs are within two iterations of traditional SA-AMG while the remaining 40 runs crashed due to negative eigenvalues. We also do not show results for Sprsfy sparsification, as in 36 cases it increases the iteration count by at least seven iterations. Thus we cannot recommend using Cnstnt or Sprsfy on this problem, unless they are paired with either one of the modifications which can overcome their deficiencies.
3.4 Refined boundary mesh

We consider the exponentially refined mesh scenario depicted in Figure 5 that is intended to capture important features associated with domain boundaries. Such a mesh, which was generated with the IFISS package, might be used to address boundary layer applications using a logically regular mesh. Table 2 give results for a Poisson problem discretized using linear finite elements on a quadrilateral mesh where Dirichlet boundary conditions force the solution to be one along the top boundary and Neumann conditions are applied on the other boundaries. The underlying mesh corresponds to a 257 × 257 logically rectangular grid, which is a refined version of the left image shown in Figure 5. For this problem, the distance Laplacian matrix is again used to define the strong off-diagonal nonzeros with a drop tolerance of 0.05. A degree two Chebyshev smoother is used for pre and postrelaxation and convergence is declared when the initial residual is reduced by eight orders of magnitude. We can again see that the 1Norm option almost always reduces the number of iterations and generally improves the solve time, while only increasing the AMG setup cost by a negligible amount. The Sprsfy option is generally costly and does not help for this problem. In particular, it only modestly reduces the cost per iteration while increasing the total number of iterations. It also incurs a noticeable increase in AMG setup time. It should be noted that the implementations of the proposed new algorithms are relatively crude and there has been no attempt at code optimization/tuning. This is probably most detrimental to the Sprsfy setup timings as this is the most
TABLE 2  Poisson solve on refined boundary mesh showing AMG operator complexity, iterations, setup time (seconds), and solve time (seconds) for different AMG variants

| Algorithm choice | \( \theta = 0.05 \) |
|------------------|-------------------|
|                  | AMG complex | Its. | AMG setup | Solve |
| Cnstrnt          | 1.33        | 19   | 0.21      | 0.16  |
| OffLmp           | 1.33        | 20   | 0.23      | 0.18  |
| Sprsfy           | 1.33        | 15   | 0.27      | 0.14  |
| Cnstrnt          | 1.33        | 14   | 0.28      | 0.12  |
| Sprsfy           | 1.31        | 46   | 0.31      | 0.53  |
| Cnstrnt          | 1.31        | 45   | 0.36      | 0.52  |
| Sprsfy           | 1.31        | 28   | 0.35      | 0.27  |
| Cnstrnt          | 1.31        | 26   | 0.38      | 0.24  |
| 1Norm            | 1.33        | 14   | 0.22      | 0.12  |
| Cnstrnt          | 1.33        | 14   | 0.26      | 0.12  |
| OffLmp           | 1.33        | 14   | 0.26      | 0.12  |
| Cnstrnt          | 1.33        | 14   | 0.29      | 0.12  |
| Sprsfy           | 1.31        | 20   | 0.31      | 0.18  |
| 1Norm            | 1.31        | 21   | 0.34      | 0.19  |
| Sprsfy           | 1.31        | 25   | 0.36      | 0.23  |
| Cnstrnt          | 1.31        | 21   | 0.38      | 0.19  |

difficult algorithm to develop efficiently. Further, all of the AMG algorithms (including standard smoothed aggregation) are relatively effective for this problem and so the AMG setup phase is actually greater than the solve phase. Of course, it is often possible to amortize this setup time when repeated linear solves occur within some outer iterative process, for example, within a semi-implicit pressure Poisson time advancement scheme for incompressible fluid flow. For the most part, the Cnstrnt option does not have a significant impact on the iteration count, except in one case corresponding to the last row of the table. Finally, the OffLmp algorithm significantly improves iteration counts and run times when the 1Norm option is not used, but it does not have a big impact on iteration counts when 1Norm is used, except in the second to last row where it causes the iteration count to increase.

Table 3 gives results for a linear elasticity problem where the Poisson ratio is 0.3. The refined boundary mesh is again employed and the 2D elasticity-Q4 finite element solver code\(^{17}\) is used to generate matrices. The underlying mesh is again 257 \( \times \) 257. A sample solution on a coarser mesh is shown on the right side of Figure 5. Neumann boundary conditions are applied on all boundaries except for the four corner points. The two lower corners are restricted via Dirichlet boundary conditions to have zero displacements in both the x and y directions while the upper corners are only restricted to have zero displacements in the x direction. Forces are applied at two points on the top of the domain: one associated with pushing and the other associated with pulling. From Table 3 one can see a nice iteration and solve time gain due to the Cnstrnt option with a small gain from the 1Norm option. Specifically, the solve time is reduced from 2.18 s using standard smoothed aggregation to 1.56 s using the new 1Norm and Cnstrnt algorithm enhancements. In this case, the AMG setup time is not the dominate cost of the overall time to solution and increases slightly by about 7%. Here, results are only shown for 1Norm and Cnstrnt in conjunction with a distance Laplacian dropping criteria using a drop tolerance of 0.04. As with the previous problem, a degree two Chebyshev smoother is used for pre and postrelaxation and convergence is declared when the initial residual is reduced by 8 orders of magnitude. The results with OffLmp (which has primarily been developed for scalar PDEs) and Sprsfy were not beneficial. In these experiments, a tentative prolongator is again built based on piecewise constant interpolation for the x displacements and the y displacements. The rigid body rotation mode was not used in building the tentative prolongator nor was QR orthogonalization (sometimes both employed with smoothed aggregation on elasticity problems), as these options do not improve timings for any of the AMG solvers including standard smoothed aggregation. In general, it is unclear how to perform dropping such that the rotation mode remains within the null space.
TABLE 3 Elasticity solve on refined boundary mesh showing operator complexity, iteration setup time (seconds), and solve time (seconds) for different AMG variants

| Algorithm choice | θ = 0.05 |
|------------------|----------|
|                  | AMG complex | Its. | AMG setup | Solve |
|                  | 1.31      | 60   | 0.84      | 2.18  |
| Cnstrnt          | 1.31      | 50   | 0.89      | 1.66  |
| 1Norm            | 1.31      | 59   | 0.87      | 2.11  |
| 1Norm            | 1.31      | 48   | 0.90      | 1.56  |

FIGURE 6 Porosity of SPE10 benchmark problem

of the dropped matrix. The Cnstrnt algorithm is oriented toward grid transfers where the coefficients lie between zero and one, which is not necessarily true when employing the QR algorithm. The QR aspect of smoothed aggregation multigrid is only relevant when the tentative prolongator is chosen to exactly interpolate multiple nonorthogonal vectors, which would be the case if the rotation mode was included.

3.5 | Subsurface SPE10 problem

This model comes from a dataset associated with the 10th SPE comparative solution project (SPE10). It describes a reservoir simulation that models flow through porous media to predict well production from hydrocarbon deposits. The discrete matrix problem was formed using the open source toolbox MRST. The underlying PDE equations are defined by Darcy’s law for a single fluid along with external influences such as wells and are discretized using a two-point flux approximation method. The porosity is shown in Figure 6 and leads to large permeability variations that range up to 12 orders of magnitude. The large permeability variation requires that the multigrid solver coarsen irregularly. For these experiments, the same Chebyshev smoother choices are used as with the first example. A direct solver is also used on the coarsest grid. A standard smoothed aggregation strength-of-connection criteria is employed using the matrix coefficients (i.e., not the distance Laplacian) and three values of θ are considered. The number of multigrid levels is fixed at five where the finest level discretization matrix is 660 × 660 and the coarsest level matrix dimensions is always less than 14 K for the largest θ value and is much smaller for the other two θ. The conjugate gradient iteration is terminated when the residual is reduced by a factor of 10^-8.

Tables 4 and 5 illustrate results for different combinations of our proposed variants. All combinations that did not use the 1Norm variant failed and so these are not shown. For the smallest value of θ, the Cnstrnt and OffLmp procedures do not have too significant an effect on convergence. The Sprsly approach does reduce the operator complexity, but this comes at a fairly significant increase in iteration count (approximately double). For large values of θ, however, we see that some of the different algorithm choices do have a more pronounced effect. In particular, the best iteration counts employ OffLmp for θ = .1, which are about half those of the other methods when Sprsly is not used. The Sprsly algorithm more significantly improves the AMG operator complexity when θ = .1 and also improves the iteration counts in the case that OffLmp is not used, though the iterations are worse when both OffLmp and Sprsly are employed. The Cnstrnt option has
only a modest effect. For the most part, the run times mirror the iteration and complexity results. Here, one can see that the effect of a reduced AMG operator complexity associated with the Sprsfy algorithm by examining the $\theta = 0.1$ solve times. For example, the Sprsfy solve times are about 2.35 s while the two solve times without Sprsfy and OffLmp require approximately 3.65 s. While some of this is due to an approximately 20% reduction in the number of iterations, the rest comes from a reduction in the cost per iteration. Table 6 provides some of the AMG hierarchy details. Level 0, the finest AMG level, is not shown as it is identical in all cases. Specifically, the original matrix has 660,000 rows with approximately 7 nonzeros per row. All of the AMG variants significantly increase the number of nonzeros per row, though in general the Sprsfy algorithm leads to fewer nonzeros per row on coarser grids. Though not illustrated in these serial timings, large number of nonzeros per row can be particularly problematic on large parallel systems as they are typically associated with increased communication costs. Of course, the most important point is that the traditional SA-AMG algorithm failed to converge on this problem and so the diagonal 1Norm modification is critical in getting the solver to converge. The additional algorithm modifications can then provide some further benefit, though not always. As the choice of $\theta$ is unknown, it is reassuring that many of the other algorithm variants (as opposed to just using 1Norm) exhibit an iteration decrease as the cost/AMG operator complexity increases (associated with an increasing value of $\theta$).

### 3.6 Wind turbine

We now consider the effects of the various solver options within a wind turbine simulation run in the low Mach computational fluid dynamics (CFD) code NaluWind. Figure 7 depicts a 5-MW wind turbine that is composed of three turbine blades and a hub. The blades and background are meshed separately and coupled via constraints. NaluWind has two
TABLE 6  SPE10 AMG hierarchy details for the AMG-generated discretization matrices using two different AMG variants

| Level | 1Norm option only | 1Norm and Sprsfy options |
|-------|------------------|--------------------------|
|       | θ = 0.02 | θ = 0.05 | θ = 0.1 | θ = 0.02 | θ = 0.05 | θ = 0.1 |
|       | Rows | nnz/Row | Rows | nnz/Row | Rows | nnz/Row | Rows | nnz/Row | Rows | nnz/Row | Rows | nnz/Row |
| 1     | 113,785 | 3,099,635 | 27.24 | 123,128 | 3,279,932 | 26.64 | 180,866 | 4,362,292 | 24.12 |
| 2     | 12,413 | 630,339 | 50.78 | 18,775 | 891,695 | 47.49 | 45,104 | 1,731,822 | 38.40 |
| 3     | 1790 | 123,296 | 68.88 | 5309 | 282,291 | 53.17 | 19,267 | 765,097 | 39.71 |
| 4     | 553 | 27473 | 49.68 | 2967 | 135,833 | 45.78 | 13,335 | 467,091 | 35.03 |

FIGURE 7  Front and side views of NREL-5 MW (5 megawatt) turbine mesh, showing blades, and hub

main techniques for addressing these constraints. The first “coupled” technique can be viewed as an alternating Schwarz approach, in which each mesh has equally valid solutions. The corresponding linear systems have equations corresponding to constraints, and it is these equations which cause difficulties for the linear solver. In the second “decoupled” approach, the constraints are eliminated from the corresponding linear systems, which can be solved independently.

The simulation itself is time-dependent and consists of two main physics solve phases: momentum and pressure. While the momentum linear system is amenable to GMRES preconditioned with symmetric Gauss–Seidel, the pressure system requires a more robust, scalable solver. The current solver of choice is GMRES preconditioned by smoothed-aggregation multigrid, with a convergence criteria requiring a relative residual reduction of 10^{-5}. For this problem 5 AMG levels are used. The finest level matrix system has 23 million DOFs. The number of DOFs in the coarsest level AMG matrices varies, but is typically 9–15 k unknowns. One pre and one postsmoothing sweep using a degree 2 Chebyshev smoother is employed on all levels with the exception of the coarsest level where a direct solver is used. A standard SA-AMG strength-of-connection criteria is employed using the matrix coefficients (i.e., not the distance Laplacian) for a fixed θ threshold of 0.02. We will consider the effect of the proposed SA-AMG options on iteration counts over 10 time steps in the simulation. The left side of Figure 8 compares linear iteration totals for standard SA-AMG versus SA-AMG using various options for NaluWind run in decoupled mode. The best improvement in iteration counts comes with either the 1Norm or the OffLmp options. The Cnstrt option is only shown in one case as it generally produced very little improvement. The 1Norm option could not be improved further by using it in conjunction with any of the other algorithms, though it is worth noting that OffLmp and Cnstrt did not further degrade performance. As expected, Sprsfy generally increased the iteration count and unfortunately provides only a modest gain in AMG operator complexity from
FIGURE 8  Comparison of effect of prolongator options on linear solver iterations over 10 timesteps (four linear solvers per timestep). In the left plot, all combinations of the four algorithm variants were tested, but only the most effective strategies are shown. “Traditional” denotes standard SA-AMG. “DMod” is diagonal modification, “AltLmp” is alternative lumping, and “Sprsfy” is prolongator sparsification. In the right plot, only the effect of “DMod” is shown versus standard SA-AMG for an ExaWind 5MW linear system formulated either as a coupled system (containing constraints) or an uncoupled system

TABLE 7  ExaWind NREL-5MW average AMG preconditioner setup times, average Krylov solve times, and multigrid complexities using three different AMG variants

| Algorithm choice | AMG Setup | Solve | AMG Complexity |
|------------------|-----------|-------|----------------|
| Traditional      | 3.15      | 7.78  | 1.66           |
| 1Norm            | 3.52      | 4.41  | 1.65           |
| OffLmpSprsfyCnstrnt | 5.47      | 3.16  | 1.61           |

about 1.66 to about 1.61. When used with the traditional method by itself (shown in the plot) or in conjunction with 1Norm (not shown), it generally increases the average iteration count by about 1 iteration. Finally, when not using the 1Norm option, it is possible to improve the Sprsfy iteration counts and make it fairly competitive with the best runs by applying both OffLmp and Cnstrnt. Most of the improvements are due to the OffLmp (on average a 2.575 iteration improvement) while the Cnstrnt gains are more modest (on average an additional 0.75 iteration improvement). As the 1Norm option alone provides the biggest improvement for this problem, the right side highlights the impact of only the 1Norm option (all others are turned off) for the coupled formulation. For the sake of comparison, the same information is repeated for the decoupled formulation. Here, one can see that the coupled mode leads to generally harder linear systems. For both coupled and decoupled, there is a nice reduction in the number of iterations using the 1Norm option. Further, the iteration count is generally less erratic as well with the 1Norm option.

Table 7 shows AMG setup times and Krylov solve time for traditional SA and for the two most effective option combinations, averaged over 40 linear solves in the NREL-5 MW simulation shown in the left plot of Figure 8. For each option combination, the multigrid complexities were consistent across the forty solves. We observe that traditional SA AMG has the fastest setup times, and that the new options add to the setup costs. However, the new options are effective in reducing solve times, both due to decreased iteration counts and decreased multigrid complexities.

Table 8 shows some details from representative AMG hierarchies from the twentieth linear solve in the left plot of Figure 8. Only the traditional SA and two most effective option combinations are shown. Level 0 is not shown, as it is the discretization matrix generated by the application and is the same for all hierarchies. It can be seen that the 1Norm option does not appreciably affect the coarse level matrix sizes or sparsities. The use of the Sprsfy option reduces the number of nonzeros in the coarse level matrices, starting on Level 2.

Figure 9 considers a robustness study for one representative linear system within a McAllister fixed wing simulation using the coupled formulation. In this study, iteration counts are shown as a function of a varying drop threshold $\theta$ parameter. Without the 1Norm option, we again see erratic behavior including three threshold choices where the solver
### Table 8
Comparison of AMG-generated discretization matrices from twentieth solve in ExaWind simulation

| Level | traditional SA | 1Norm option | Sprsfy+OffLmp+Cnstrnt |
|-------|----------------|---------------|----------------------|
|       | Rows           | nnz           | nnz/Row              | Rows           | nnz           | nnz/Row              | Rows           | nnz           | nnz/Row              |
| 1     | 3,313,139      | 92,264,951    | 27.85                | 3,313,139      | 92,264,951    | 27.85                | 3,313,139      | 91,873,076    | 27.73                |
| 2     | 424,524        | 15,391,732    | 36.26                | 424,516        | 15,391,244    | 36.26                | 424,645        | 13,113,049    | 30.88                |
| 3     | 106,264        | 10,694,754    | 100.64               | 103,797        | 10,341,763    | 99.63                | 107,098        | 6,367,280     | 59.45                |
| 4     | 10,130         | 3,474,098     | 342.95               | 9300           | 2,607,786     | 280.41               | 13,355         | 1,808,421     | 135.41               |

Note: The traditional AMG preconditioner and the two most effective variants are shown.

### Figure 9
Left: Sample mesh sections for 5 MW turbine. Middle: GMRES/AMG iterations for time sequence on 5 MW simulation (coupled formulation being more challenging). Right: GMRES/AMG iterations for user-supplied strong/weak threshold for a McALISTER fixed wing configuration

...does not converge. However, the iteration counts vary smoothly without any failures when the 1Norm option is used. That is, one benefit of 1Norm appears to be enhanced robustness.

### 4 Conclusion

In this article, we have presented four new algorithmic variants to SA-AMG that focus on improving the smoothed prolongator grid transfer, especially for problems with many weak connections. Such systems commonly arise in practice, and can lead to poor SA-AMG performance, which can manifest as high operator complexity, increased iteration counts, and even failure to converge (due to iteration matrices with negative eigenvalues). Whereas other SA-AMG research has focused on developing new strength-of-connection measures to mitigate these issues, we have assumed a standard scalar strength measure is utilized, and in this article present algorithms aimed at improving the final smoothed prolongator. These algorithms are algebraic in nature and build naturally on the existing SA-AMG machinery.

We have demonstrated the efficacy of these new algorithms on a suite of problems that are challenging for standard SA-AMG to solve: manufactured scalar Poisson problems with severe variable mesh stretching and lack of diagonal dominance, a linear elasticity problem, a standard oil reservoir benchmark, and linear systems arising from a low-Mach CFD application. The main take-away is that the four new variants generally yield improvements over standard SA-AMG. The 1Norm and OffLmp variations very rarely take more iterations than traditional SA-AMG. While there are cases where the convergence behavior is similar to that of SA-AMG, there are other cases where 1Norm and OffLmp are significantly faster and more robust than SA-AMG. The results with Cnstrnt are a bit mixed. Sometimes it helps dramatically but other times it is not so robust. The Sprsfy results do help a modest amount with the multigrid operator complexity, but in most cases convergence does suffer. However, we do note that Sprsfy was robust on the random cube problem, which was not true for SA-AMG. Exploring the use of these algorithms in the context of a new weak-connection threshold approach will be the subject of a forthcoming paper, where Sprsfy’s ability to reduce multigrid operator complexity is more significant. Another potential topic for future research is the adaptation of one or more of these algorithms to systems of PDEs.
ACKNOWLEDGMENTS
This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program; Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing program through the FASTMath Institute under Contract No. DE-NA0003525 at Sandia National Laboratories; and by the exascale computing project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation’s exascale computing imperative. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under grant DE-NA-0003525. This article describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

CONFLICT OF INTEREST
This study does not have any conflicts to disclose.

DATA AVAILABILITY STATEMENT
Data available on request from the authors.

ENDNOTE
*Diagonally dominant matrices are defined by replacing the greater than the sum condition in the strictly diagonally dominant definition by a condition greater than or equal to the sum.

ORCID
Jonathan J. Hu https://orcid.org/0000-0003-4636-3619

REFERENCES
1. Vaněk P, Mandel J, Brezina M. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. Computing. 1996;56:179–96.
2. Vaněk P, Brezina M, Mandel J. Convergence of algebraic multigrid based on smoothed aggregation. Numer Math. 2001;88:559–79.
3. Brandt A, McCormick S, Ruge J. Algebraic multigrid (AMG) for sparse matrix equations. In: Evans D, editor. Sparsity and its applications. Cambridge, UK: Cambridge University Press; 1984. p. 257–84.
4. Ruge J, Stüben K. Algebraic multigrid (AMG). In: McCormick S, editor. Multigrid methods. Frontiers in Applied Mathematics. Volume 3. Philadelphia, PA: SIAM; 1985. p. 73–130.
5. Livne O. Coarsening by compatible relaxation. Numer Linear Algebra Appl. 2004;11:205–27.
6. Bröker O. Parallel multigrid methods using sparse approximate inverses [PhD thesis]. Swiss Federal Institute of Technology, Zurich, Zurich, Switzerland; 2003.
7. Brannick J, Zikatanov L. Algebraic multigrid methods based on compatible relaxation and energy minimization. In: Widlund OB, Keyes DE, editors. Domain decomposition methods in science and engineering XVI. Lecture Notes in Computational Science and Engineering. Volume 55. New York, NY: Springer-Verlag; 2006. p. 15–26.
8. Brannick J, Brezina M, MacLachlan S, Manteuffel T, McCormick S. An energy-based AMG coarsening strategy. Numer Linear Algebra Appl. 2006;13:133–48.
9. Brandt A, Brannick J, Kahl K, Livshits I. Algebraic distance for anisotropic diffusion problems: multilevel results. Electron Trans Numer Anal. 2015;44:472–96.
10. Brannick JJ, Falgout RD. Compatible relaxation and coarsening in algebraic multigrid. SIAM J Sci Comput. 2010;32:1393–416. Available from. http://dblp.uni-trier.de/db/journals/siamsc/siamsc32.html#BrannickF10
11. Olson L, Schroder J, Tuminaro R. New perspective on strength measures in algebraic multigrid. Numer Linear Algebra Appl. 2010;17:713–33.
12. Mandel J, Brezina M, Vaněk P. Energy optimization of algebraic multigrid bases. Computing. 1999;62:205–28.
13. Brezina M. Robust iterative methods on unstructured meshes [PhD thesis]. University of Colorado at Denver; 1997.
14. Doormaal JPV, Raithby GD. Enhancements of the simple method for predicting incompressible fluid flows. Numer Heat Transf. 1984;7:147–63. https://doi.org/10.1080/01495728408961817
15. Prokopenko A, Hu JJ, Wiesner TA, Siefert CM, Tuminaro RS. MueLu user’s guide 1.0, Technical report no. SAND2014-18874, Sandia National Laboratories; 2014.
16. Elman H, Ramage A, Silvester D. IFISS: a computational laboratory for investigating incompressible flow problems. SIAM Rev. 2014;56:261–73.
17. Orhan E. 2D elasticity - Q4 finite element solver, version 1.0.0; 2021. Available from: https://www.mathworks.com/matlabcentral/fileexchange/69708-2d-elasticity-q4finite-element-solver. Accessed 15 Dec 2021.
18. SPE comparative solution project; May 2000. Available from: https://www.spe.org/web/csp/datasets/set02.htm. Accessed 10 Feb 2021.
19. Lie K-A. An introduction to reservoir simulation using MATLAB/GNU octave: user guide for the MATLAB reservoir simulation toolbox (MRST). Cambridge, UK: Cambridge University Press; 2019. https://doi.org/10.1017/9781108591416
20. Sprague MA, Ananthan S, Vijayakumar G, Robinson M. Exawind: a multi-fidelity modeling and simulation environment for wind energy. Proceedings of the NAWEA WindTech Conference; 2020.
21. Hensinger D, Drake R, Foucar J, Gardiner T. Pamgen, a library for parallel generation of simple finite element meshes, Technical report no. SAND2008-1933, Sandia National Laboratories, 2008.

How to cite this article: Hu JJ, Siefert CM, Tuminaro RS. Smoothed aggregation for difficult stretched mesh and coefficient variation problems. Numer Linear Algebra Appl. 2022;29(6):e2442. https://doi.org/10.1002/nla.2442

APPENDIX A. RAND CUBE INPUT

The results from Section 3.1, were generated from meshes using the following Pamgen\textsuperscript{21} template. The mesh is uniform in x and y and stretched with 100 : 1 in the z-direction. The nodes are each given a random perturbation up to 20% of the distance to the neighboring node in each direction. The random number generator was seeded with 50 different seeds in order to generate the meshes considered.

\begin{verbatim}
mesh  
brick  
  zmin = 0.0  
xmin = 0.0  
ymin = 0.0  
umz 1  
  zblock 1 1.0 interval 60  
umx 1  
  xblock 1 1.0 interval 60  
umy 1  
  yblock 1 1.0 interval 60  
end  
set assign  
  sideset, ilo, 1  
  sideset, jlo, 2  
  sideset, klo, 3  
  sideset, ihi, 4  
  sideset, jhi, 5  
  sideset, khi, 6  
end  
user defined geometry transformation  
  ',  
  outxcoord = (inxcoord + 0.2*drand()/60)*1.;  
  outycoord = (inycoord + 0.2*drand()/60)*1.0;  
  outzcoord = (inzcoord + 0.2*drand()/60)*100.0;  
  ',  
end
\end{verbatim}
APPENDIX B. TRIAXIALLY STRETCHED CUBE INPUT

The results from Section 3.2, were generated from meshes using the following Pamgen template. The elements linearly vary in size in each dimension, depending on the parameters $KX$, $KY$, and $KZ$, which must be substituted into the input deck below (quantities inside braces are replaced).

```
mesh
    brick
    zmin = 0.0
    xmin = 0.0
    ymin = 0.0
    numx 1
        xblock 1 \{3.0*(\{K\}X_{+}1)\}, first size .1, last size \{\{K\}X_/10\}
    numy 1
        yblock 1 \{3.0*(\{K\}Y_{+}1)\}, first size .1, last size \{\{K\}Y_/10\}
    numz 1
        zblock 1 \{3.0*(\{K\}Z_{+}1)\}, first size .1, last size \{\{K\}Z_/10\}
end
set assign
    sideset, ilo, 1
    sideset, jlo, 2
    sideset, klo, 3
    sideset, khi, 6
    sideset, khi, 6
end
```

APPENDIX C. RADIAL TRISECTION INPUT

The results from Section 3.3, were generated from meshes using the following Pamgen template. The mesh represents a quadrant of the cylinder which is meshed uniformly in the $Z$ direction and uniformly in the $\theta$ direction outside of the inner trisection region. In the outer region starts with elements with their radial extent near their $Z$ extent, but the elements increase in radial extent up to the value of first_last. The extent of the outer region is varied with first_last to keep the number of elements roughly constant. The inner region’s transition type is controlled by trisct_blks parameter.

```
mesh
    radial trisection
    trisection blocks, \{trisct_blks\}
    transition radius, 6.
    numz 1
        zblock 1 20. interval 20
    numr 2
        rblock 1 8.0 interval 4
        rblock 2 \{200*floor((1+first_last)/2)\} first size 1.0 last size \{first_last\}
    numa 1
        ablock 1 90. interval 24
end
set assign
    sideset, jlo, 2
    sideset, klo, 3
    sideset, khi, 4
```
sideset, jhi, 5
sideset, khi, 6
end
end