SIAM J. SCI. COMPUT.
Vol. 38, No. 5, pp. S332–S357

REDUCING PARALLEL COMMUNICATION IN ALGEBRAIC MULTIGRID THROUGH SPARSIFICATION*

AMANDA BIENZ†, ROBERT D. FALGOUT‡, WILLIAM GROPP†, LUKE N. OLSON†, AND JACOB B. SCHRODER‡

Abstract. Algebraic multigrid (AMG) is an $\mathcal{O}(n)$ solution process for many large sparse linear systems. A hierarchy of progressively coarser grids which utilize complementary relaxation and interpolation operators is constructed. High-energy error is reduced by relaxation, while low-energy error is mapped to coarse-grid matrices and reduced there. However, large parallel communication costs often limit parallel scalability. As the multigrid hierarchy is formed, each coarse matrix is formed through a triple matrix product. The resulting coarse grids often have significantly more nonzeros per row than the original fine-grid operator, thereby generating high parallel communication costs associated with sparse matrix-vector multiplication (SpMV) on coarse levels. In this paper, we introduce a method that systematically removes entries in coarse-grid matrices after the hierarchy is formed, leading to improved communication costs. We sparsify by removing weakly connected or unimportant entries in the matrix, leading to improved solve time. The main trade-off is that if the heuristic identifying unimportant entries is used too aggressively, then AMG convergence can suffer. To counteract this, the original hierarchy is retained, allowing entries to be reintroduced into the solver hierarchy if convergence is too slow. This enables a balance between communication cost and convergence, as necessary. In this paper we present new algorithms for reducing communication and present a number of computational experiments in support.

Key words. multigrid, algebraic multigrid, non-Galerkin multigrid, high performance computing

AMS subject classifications. 65F50, 65N55, 65F08, 65F10, 65Y05, 68Q10

DOI. 10.1137/15M1026341

1. Introduction.

Algebraic multigrid (AMG) [22, 9, 24] is an $\mathcal{O}(n)$ linear solver for standard discretizations of elliptic differential equations [2, 30, 14]. We consider AMG as a solver for the symmetric positive-definite (SPD) matrix problem

$$Ax = b, \tag{1}$$

with $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$. AMG consists of two phases, a setup phase and a solve phase. The setup phase defines a sequence or *hierarchy* of L coarse-grid and interpolation operators, $A_1, \ldots, A_L$ and $P_0, \ldots, P_{L-1}$, respectively. The solve phase iteratively improves the solution through relaxation and coarse-grid correction.

*Received by the editors June 16, 2015; accepted for publication (in revised form) June 16, 2016; published electronically October 27, 2016.

http://www.siam.org/journals/sisc/38-5/M102634.html

Funding: This work is partially supported by the National Science Foundation Graduate Research Fellowship award DGE-1144245 and the Air Force Office of Scientific Research under grant FA9550-12-1-0478. This research is also part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-JRNL-673388).

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (bienz2@illinois.edu, http://web.engr.illinois.edu/~bienz2; wgropp@illinois.edu, http://wgropp.cs.illinois.edu; lukeo@illinois.edu, http://lukeo.cs.illinois.edu).

‡Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550 (rfalgout@llnl.gov, http://people.llnl.gov/falgout2; schroder2@llnl.gov, http://people.llnl.gov/schroder2).

The focus of this paper is on the communication complexity of AMG in a distributed memory, parallel setting. To be clear, we refer to the *communication complexity* as the time cost of interprocessor communication, while referring to the *computational complexity* as the time cost of the floating-point operations. The *complexity* or *total complexity* is then the cost of the algorithm, combining the communication and computational complexities.

There is a trade-off between per-iteration complexity and the resulting convergence; this is controlled by the setup phase in AMG. Indeed, more accurate interpolation leads to more entries in P_ℓ, and this often improves the overall convergence. However, this also leads to more entries in $A_{\ell+1}$ through the Galerkin product, $A_{\ell+1} = P_\ell^T A P_\ell$, which is the most common way to form $A_{\ell+1}$ in AMG. As we will see, the number of entries in $A_{\ell+1}$ is a good proxy for the complexity of level $\ell+1$. In contrast, sparser interpolation and fast coarsening reduce the complexity of a single iteration of an AMG cycle through fewer entries in $A_{\ell+1}$ and fewer overall AMG levels but can lead to a deterioration in convergence [30, 14].

The sparse matrices, $A_1, \ldots, A_L$, in the multigrid hierarchy are, by design, smaller in dimension yet are often more dense as the level increases. As an example of this, Table 1 shows the properties of a hierarchy for a three-dimensional (3D) Poisson problem with a 27-point finite element stencil on a $100 \times 100 \times 100$ grid. Classical AMG (Ruge–Stüben) is used for this example.[1] As the problem size decreases on coarse levels, the average number of nonzero entries per row increases. Here we denote by `nnz` the number of nonzero entries in the respective matrix. Figure 1 highlights this example, where we see that both the density and pattern of nonzeros increase on lower levels in the hierarchy.

TABLE 1
Matrix properties using classical AMG for a 3D Poisson problem.

Level ℓ	Matrix size n	Nonzeros `nnz`	Nonzeros per row $\frac{\texttt{nnz}}{n}$
0	1 000 000	26 463 592	26
1	124 984	3 645 644	29
2	23 042	1 466 006	64
3	2991	198 043	66
4	570	32 680	57
5	117	4705	40

In parallel, coarse levels that are more dense correlate with an increase in parallel communication costs [5]. Figure 2 shows this by plotting the time spent on each level in an AMG hierarchy during the solve phase. The time grows substantially on coarse levels, which is attributed to increased communication costs from a decrease in sparsity. This effect is common in AMG methods; Figure 2 shows two examples.

In this paper we introduce a method for controlling the communication complexity in AMG. The method increases the sparsity of coarse-grid operators (A_ℓ, $\ell = 1, \ldots, L$) by eliminating entries in A_ℓ. This results in an improved balance between convergence and per-iteration complexity in comparison to the standard algorithm. In addition, we develop an adaptive method which allows nonzero entries to be reintroduced into the AMG hierarchy, thus recovering convergence if entry elimination is too aggressive.

In the context of this paper, we define *sparsity* and *density* in terms of the average

[1] See PyAMG [6] using `ruge_stuben_solver(poisson((100,100,100), type='FE'))` for more details. Similar results are seen using a 7-point finite difference discretization.

FIG. 1. *Matrix sparsity pattern using classical AMG for three levels in the hierarchy:* $\ell = 3$, 4, 5. *The full matrix properties are given in Table* 1.

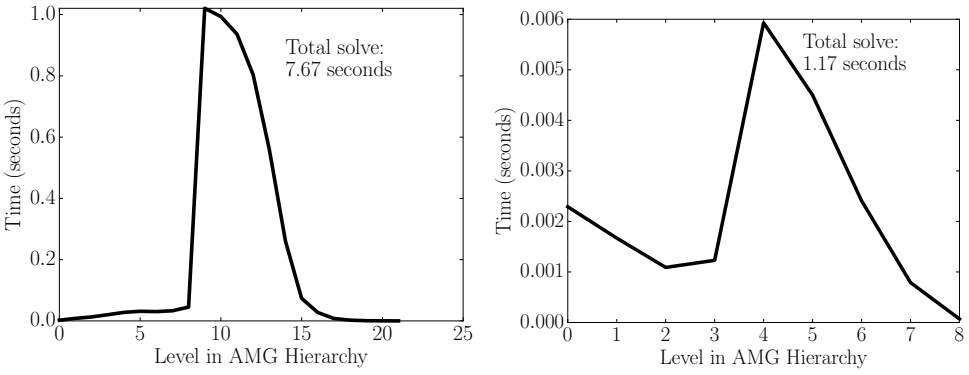


FIG. 2. *Left: Time spent on each level of the hierarchy during a single iteration of classical parallel AMG for a 3D Poisson problem with hypre using Falgout coarsening* [10] *and hybrid symmetric Gauss–Seidel relaxation. Right: Repeat experiment, but using aggressive HMIS coarsening. The total time is much lower; however, the qualitative feature of expensive coarse levels remains.*

number of nonzeros per row (or equivalently, the average degree of a node in the graph of the matrix). In particular, density of a matrix A_ℓ of size n_ℓ is defined to be $\texttt{nnz}(A_\ell)/n_\ell$. The performance of AMG is closely correlated with this metric, especially the communication costs. In addition, note that if a matrix A_ℓ is "sparser" or "denser" under this definition, it is also the case under the more traditional density metric, $\texttt{nnz}(A_\ell)/n_\ell^2$. Another advantage is that this measure yields a meaningful comparison between matrices of different sizes. For example, a goal of our algorithm is to generate coarse matrices as close as possible in terms of sparsity structure to the fine-grid matrix.

There are a number of existing approaches to reducing per-iteration communication complexity at the cost of convergence. Aggressive coarsening, such as HMIS [14] and PMIS [14], rapidly coarsens each level of the hierarchy, leading to a reduction in both the number and the density of coarse operators. While these coarsening strategies reduce the cost of each iteration or cycle in the AMG solve phase, they do so at the cost of accuracy, often resulting in reduced convergence. Likewise, interpolation schemes such as *distance-two interpolation* [13] improve the convergence for aggressive coarsening but also result in an increase in complexity.

Figure 2 shows that the time per level during the solve phase is reduced in comparison to standard coarsening, even though the same number of processes and problem size per core are used. The use of HMIS coarsening is the only difference in problem settings between these two runs. Regardless, while aggressive coarsening may reduce the total work required during an iteration of AMG, the problem of expensive coarse levels still persists.

Another strategy for reducing communication complexity in AMG consists of systematically improving sparsity in the interpolation operators [13]. Removing nonzeros from the interpolation operators reduces the complexity of the coarse-grid operators; however, this process can also have an unpredictable impact on coarse-level performance if used too aggressively. Sparsity can alternatively be improved by considering a sparse approximation to the transformation that relates the fine-grid matrix A purely injected to the coarse grid with the less sparse but generally more desirable Galerkin coarse-grid matrix. This transformation along with the injected matrix A go on to form a coarse grid that is sparser than Galerkin but still yields good multigrid convergence for several cases [8].

The typical approach to building coarse-grid operators, A_ℓ, is to form the Galerkin product with the interpolation operator: $A_{\ell+1} = P_\ell^T A_\ell P_\ell$. This ensures a *projection* in the coarse-grid correction process and a guarantee on the reduction in error in each iteration of the AMG solve phase (although the factor by which the error is reduced may be small for a poorly converging method). Yet it is the triple matrix product in the Galerkin construction that leads to a growth in the number of nonzeros in coarse-grid matrices. As such, there are several approaches to constructing coarse operators that do not use a Galerkin product and are termed *non-Galerkin* methods. These methods have been formed in a classical AMG setting [15] and also in a smoothed aggregation [27] context. In general, these methods selectively remove entries from coarse-grid operators, reducing the complexity of the multigrid cycle. Assuming the appropriate entries are removed from coarse-grid operators, the result is a reduction in complexity with little impact on convergence.

An alternative to limiting communication complexity is to directly determine the coarse-grid stencil, an approach used in geometric multigrid. For instance, simply rediscretizing the PDE on a coarse level results in the same stencil pattern as for the original finest-grid operator, thus avoiding any increase in the number of nonzeros in coarse-grid matrices. More sophisticated approaches combine geometric and algebraic information and include BoxMG [11, 12] and PFMG [4], where a stencil-based coarse-grid operator is built. Additionally, collocation coarse grids (CCAs) [29] have been used on coarse levels to effectively limit the number of nonzeros. Yet, all these methods rely on geometric properties of the problem being solved. One exception is the extension of CCAs to AMG (ACCA) [28], which has shown similar performance to smoothed aggregation AMG.

Another approach is to consider sparsification of the graph of the sparse matrix. There are several major approaches, grouped by how the difference between the original graph and the sparsified graph is measured. One major approach finds graphs where the weight of cuts in the original graph and the sparsified graph are close [16]. In another, called spectral sparsification [25, 26], edges of the graph are removed if the resulting graph Laplacian remains spectrally close to the original. Sparsification of graphs has been an active area of research recently [20], concentrating on developing *near* linear-time algorithms for the sparsification. While the immediate impact on the coarse-level matrices in a multigrid hierarchy has not been studied, this could point to an additional improvement on the methods presented in this paper.

The approach developed in this paper is to form a coarse-level operator that does not satisfy a Galerkin product by modifying *existing* hierarchies. The novel benefit of the proposed approach is that it is applicable to most AMG methods, requires no geometric information, and provides a mechanism for recovery if the dropping heuristic is chosen too aggressively (see section 6). This paper is organized as follows. Section 2 describes standard AMG as well as the method introduced in [15]. Section 3 introduces two new methods for reducing the communication complexity of AMG: Sparse Galerkin and Hybrid Galerkin. Parallel performance models for these methods are described in section 4, and the parallel results are displayed in section 5. An adaptive method for controlling the trade-off between communication complexity and convergence is described in section 6. Finally, section 7 makes concluding remarks.

2. Algebraic multigrid. In this section we detail the AMG setup and solve phases. We let the *fine-grid* operator A be denoted with a subscript as A_0.

Algorithm 1 describes the setup phase and begins with `strength`, which identifies the strongly connected edges[2] in the graph of A_ℓ to construct the strength-of-connection matrix S_ℓ. From this, P_ℓ is built in `interpolation` to interpolate vectors from level $\ell+1$ to level ℓ, with the goal of accurately interpolating (smooth) error not sufficiently reduced by relaxation. For classical AMG, `interpolation` first forms a disjoint splitting of the fine-level index set $\{1,\dots,n\} = C \cup F$, where C is the set of indices on the coarse level and where F is the set of indices for variables that reside only on the fine level. The size of the coarse grid is then given by $n_{\ell+1} = |C|$, and an interpolation operator, $P_\ell : \mathbb{R}^{n_{\ell+1}} \to \mathbb{R}^{n_\ell}$, is constructed using S_ℓ and A_ℓ to compute sparse interpolation formulas that are accurate for algebraically smooth functions. Finally, the coarse-grid operator is created through a Galerkin triple-matrix product, $A_{\ell+1} = P_\ell^T A_\ell P_\ell$. In a two-level setting, this ensures the desirable property that the coarse-grid correction process $I - P_\ell A_{\ell+1}^{-1} P_\ell^T A_\ell$ is an A_ℓ-orthogonal projection that ensures the best coarse-grid correction of the error in the range of interpolation (in the A_ℓ norm). When an approximation to $A_{\ell+1}$ is introduced, this projection property is lost, leading to a possible deterioration in convergence.

The density of each coarse-grid operator $A_{\ell+1}$ depends on that of the interpolation operator P_ℓ. Even interpolation operators with modest numbers of nonzeros typically lead to increasingly dense coarse-grid operators [15, 17]. Algorithm 1 addresses this with the optional step `sparsify`, which triggers the sparsification steps developed in this paper. The approach [15] also fits within this framework, which we detail in section 2.1.

The solve phase of AMG, described in Algorithm 2 as a V-cycle, iteratively improves an initial guess x_0 through the use of the residual equation $A_0 e_0 = r_0$, where e_0 and r_0 are the fine-grid error and residual, respectively. High-energy error in the approximate solution is reduced through relaxation in `relax`—e.g., Jacobi or Gauss–Seidel. The remaining error is reduced through coarse-grid correction: a combination of restricting the residual equation to a coarser level, followed by interpolating and correcting with the resulting approximate error. The coarsest-grid equation is computed with `solve`, using a direct solution method.

The dominant computational kernel in Algorithm 2 is the sparse matrix-vector (SpMV) product, found in `relax` and interpolation/restriction. Typically relaxation

[2] A degree of freedom i is strongly connected to j if algebraically smooth error (error not effectively reduced by relaxation) varies slowly between them. Strength information critically informs AMG how to coarsen [3] and how to interpolate [24]; while a variety of strength measures abound [23], the standard strength measure is sufficient for the problems tested.

Algorithm 1: `amg_setup`

Input: A_0: fine-grid operator
`max_size`: threshold for max size of coarsest problem
`nongalerkin`: (optional) non-Galerkin method
$\gamma_1, \gamma_2, \ldots$ (optional) drop tolerances for each level

Output: $A_1, \ldots, A_L$,
$P_0, \ldots, P_{L-1}$

while `size`$(A_\ell) >$ `max_size`
  $S_\ell =$ `strength`(A_ℓ) {Strength-of-connection of edges}
  $P_\ell =$ `interpolation`(A_ℓ, S_ℓ) {Construct interpolation and injection}
  $A_{\ell+1} = P_\ell^T A_\ell P_\ell$ {Galerkin product}

  if `nongalerkin` {(optional) described in section 2.1}
    $\hat{A}_{\ell+1} =$ `sparsify`$(A_{\ell+1}, A_\ell, P_\ell, S_\ell, \gamma_\ell)$ {Remove nonzeros in $A_{\ell+1}$}
    $A_{\ell+1} = \hat{A}_{\ell+1}$

Algorithm 2: `amg_solve`

Input: x_0: fine-level initial guess
b_0: fine-level right-hand side
$A_0, \ldots, A_L$
$P_0, \ldots, P_{L-1}$

Output: x_0, fine-level approximation

for $\ell = 0, \ldots, L-1$ **do**
  `relax`$(A_\ell, x_\ell, b_\ell, \nu_1)$ {Presmooth ν_1 times}
  $b_{\ell+1} = P_\ell^T (b_\ell - A_\ell x_\ell)$ {Restrict residual}
$x_L =$ `solve`(A_L, b_L) {Coarsest-level direct solve}
for $\ell = L-1, \ldots, 0$ **do**
  $x_\ell = x_\ell + P_\ell x_{\ell+1}$ {Interpolate and correct}
  `relax`$(A_\ell, x_\ell, b_\ell, \nu_2)$ {Postsmooth ν_2 times}

dominates since A_ℓ is larger and denser than P_ℓ. Thus, the performance on level ℓ of the solve phase depends strongly on the performance of a single SpMV with A_ℓ.

When performing parallel sparse matrix operations, a matrix A is distributed evenly across processes using a contiguous, row-wise partition, as shown in Figure 3. The local portion of the matrix is split into two groups: the diagonal block, containing all columns of A that correspond to local elements of the vector, and the off-diagonal block, corresponding to elements of the vector that are stored on other processes. For an SpMV, all off-process elements in the vector that correspond to matrix nonzeros must be communicated. Therefore, the density of a matrix contributes to the cost of communication complexity in the SpMV operation. This implies that the less sparse AMG coarse levels yield large communication costs and, often, an inefficient solve phase [15, 17].

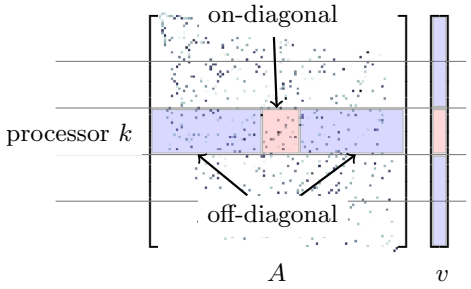


FIG. 3. *Matrix A and vector v distributed across processes in a row-wise partition.*

2.1. Method of non-Galerkin coarse grids. In this section we introduce terminology related to [15], which we term the method of *non-Galerkin coarse grids* or non-Galerkin for the remainder of the paper. In this case, coarse-level operators do not satisfy the Galerkin relationship where $A_{\ell+1} = P_\ell^T A_\ell P_\ell$ for each level ℓ. The coarse-grid operators are first formed through the Galerkin product, followed by a sparsification step that generates $\hat{A}_{\ell+1}$—see the call to `sparsify` in Algorithm 1. As motivated in the previous section, fewer nonzeros in the coarse-grid operator reduce the communication requirements. The sparser matrix $\hat{A}_{\ell+1}$ replaces $A_{\ell+1}$ and is then used when forming the remaining levels of the hierarchy, creating a dependency between $\hat{A}_{\ell+1}$ and all successive levels, as shown in Figures 6a and 6b. Thus, this approach does not preserve a coarse-grid correction corresponding to an A-orthogonal projection, as described in section 2.

In the following we use $\texttt{edges}(A)$, for a sparse matrix A, to represent the set of edges in the graph of A. That is, $\texttt{edges}(A) = \{(i,j) \text{ such that } A_{i,j} \neq 0\}$, where $A_{i,j} = (A)_{i,j}$ is the (i,j)th entry of A. In addition, we denote $\hat{P}_\ell$ as the *injection* interpolation operator that injects from level $\ell+1$ to the C points on level ℓ so that $\hat{P}_\ell$ is defined as the identity over the coarse points, leaving $\hat{P}_\ell$ zero over the F-points.

The `sparsify` method for reducing the nonzeros in a matrix is described in Algorithm 3, where the level subscripts are dropped for readability. The algorithm selectively removes small entries outside a minimal sparsity pattern[3] given by $\mathcal{M}_\ell$ where $\texttt{edges}(\mathcal{M}_\ell) = \texttt{edges}(\hat{P}_\ell^T A_\ell P_\ell + P_\ell^T A_\ell \hat{P}_\ell)$. For a given tolerance γ, any entry $A_{i,j}$ with $(i,j) \notin \mathcal{M}$ and $|A_{i,j}| < \gamma \max_{k \neq i} |A_{i,k}|$ is considered insignificant and is removed. When entry $A_{i,j}$ is removed, the value of $A_{i,j}$ is lumped to other entries that are strongly connected to $A_{i,j}$, and $A_{i,j}$ is set to zero. This reduces the per-iteration communication complexity and heuristically targets spectral equivalence between the sparsified operator and the Galerkin operator [15, 27].

There is a trade-off between the communication requirements and the convergence rate. Each entry in the matrix has a communication cost that is dependent on the number of network links that the corresponding message travels in addition to network contention. In addition, each entry in the matrix also influences convergence of AMG, with large entries generally having larger impact (although this is not uniformly the case). Any entry that has an associated communication cost outweighing the impact on convergence should be removed. However, while it is possible to

[3]The goal of the minimal sparsity pattern is to maintain, at the minimum, a stencil as wide for the coarse grid as exists for the fine grid. This is a critical heuristic for achieving spectral equivalence between the sparsified operator and the Galerkin operator. The current $\mathcal{M}$ achieves this in many cases. It is possible in some cases to reduce $\mathcal{M}$ further. See [15] for more details.

Algorithm 3: `sparsify` from [15]

Input: A_c: coarse-grid operator
A: fine-grid operator
P: interpolation
$\hat{P}$: injection
S: classical strength matrix
γ: sparse dropping threshold parameter

Output: $\hat{A}_c$, a sparsified A_c

$\mathcal{M} = \texttt{edges}(\hat{P}^T A P + P^T A \hat{P})$ {Edges in the minimal sparsity pattern}
$\mathcal{N} = \emptyset$ {Edges to keep in A_c}
$\hat{A}_c = \mathbf{0}$ {Initialize sparsified A_c}

for $(A_c)_{i,j} \neq 0$ **do**
  if $(i,j) \in \mathcal{M}$ `or` $\left|(A_c)_{i,j}\right| \geq \gamma \max_{k \neq i} \left|(A_c)_{i,k}\right|$
    $\mathcal{N} \leftarrow \mathcal{N} \cup \{(i,j),(j,i)\}$ {Add strong edges or the required pattern}

for $(A_c)_{i,j} \neq 0$ **do**
  if $(i,j) \in \mathcal{N}$
    $(\hat{A}_c)_{i,j} = (A_c)_{i,j}$
  else
    $\mathcal{W} = \{k \,|\, S_{j,k} \neq 0, (i,k) \in \mathcal{N}\}$ {Find strong neighbors in the keep list}
    for $k \in \mathcal{W}$ **do**
      $\alpha = \frac{|S_{j,k}|}{\sum_{m \in \mathcal{W}} |S_{j,m}|}$ {Relative strength to k}
      $(\hat{A}_c)_{i,k} \leftarrow (\hat{A}_c)_{i,k} + \alpha (A_c)_{i,j}$
      $(\hat{A}_c)_{k,i} \leftarrow (\hat{A}_c)_{k,i} + \alpha (A_c)_{i,j}$
      $(\hat{A}_c)_{k,k} \leftarrow (\hat{A}_c)_{k,k} - \alpha (A_c)_{i,j}$

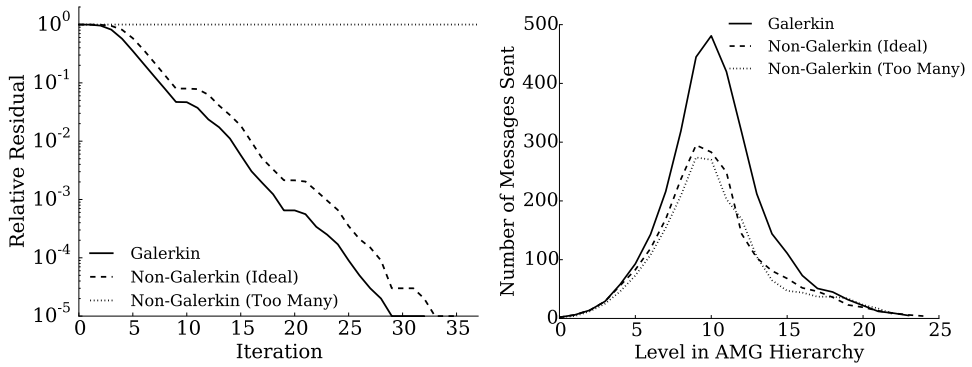
Algorithm 3b: Diagonal lumping—alternative **for** loop (section 3.1)

↳ **for** $(A_c)_{i,j} \neq 0$ **do**
1 $\texttt{ismax} \leftarrow \left|(A_c)_{i,j}\right| = \max_{k \neq i} |(A_c)_{ik}|$ `and` $(i,k) \notin \mathcal{N} \,\forall\, k \neq i$ `and` $\sum_j A_{i,j} = 0$
  if $(i,j) \in \mathcal{N}$ `or ismax` {Keep if entry is the single, maximum nonzero}
2    $(\hat{A}_c)_{i,j} = (A_c)_{i,j}$
  else {Otherwise add to the diagonal}
    $(\hat{A}_c)_{i,i} \leftarrow (\hat{A}_c)_{i,i} + (A_c)_{i,j}$

predict this communication cost based on network topology and message size, the entry's contribution to convergence cannot be easily predetermined. When dropping via non-Galerkin coarse grids, if the chosen drop tolerance is too large, too many entries are removed and convergence deteriorates. Because the ideal drop tolerance

is problem dependent and cannot be predetermined, it is likely that the chosen drop tolerance is suboptimal.

Figures 4a and 4b show the convergence and communication complexity, respectively, of various AMG hierarchies for solving a 3D Poisson problem with the method of non-Galerkin coarse grids. For both figures, the 27-point Laplacian was solved on 8192 processes with $10,000$ degrees of freedom per process. The original Galerkin hierarchy converges in the fewest number of iterations but has the highest communication complexity. Non-Galerkin removes an ideal number of nonzeros from coarse-grid operators (labeled *ideal*) when no entries are removed from the first coarse level, and all successive levels have a drop tolerance of 1.0. In this case, the communication complexity of the solver is greatly reduced with little effect on convergence. However, if the first coarse level is also created with a drop tolerance of 1.0, essential entries are removed (labeled *too many*). While the complexity of the hierarchy is further reduced, the method fails to converge.



(a) Relative residual per iteration

(b) Number of MPI sends per level

FIG. 4. *Sparsity is improved in the AMG hierarchy for the* 27*-point Laplacian on* 8192 *processes with* 10,000 *degrees of freedom per core. The time spent on middle levels of the AMG hierarchy is decreased (right) with little change to the residual after each iteration (left).*

If a large drop tolerance is chosen in the non-Galerkin method, the effect on convergence can be determined after one or two iterations of the solve phase. At this point, if convergence is poor, eliminated entries can be reintroduced into the matrix. However, with this method, convergence improvements cannot be guaranteed. As shown in Algorithm 1, sparsifying on a level affects all coarser-grid operators. Hence, adding entries back into the original operator does not influence the impact of their removal on all coarser levels. Figure 5 shows how re-adding entries is ineffective by plotting the required communication costs versus the achieved convergence for both Galerkin and non-Galerkin AMG solve phases for the same 3D Poisson problem. The data set *Non-Galerkin (added back)* is generated by removing entries with a drop tolerance of 1.0 (everything outside of $\mathcal{M}$) on the first coarse-grid operator and 0.0 (retaining everything) on all successive levels. This results in a nonconvergent method. We then add these removed entries back into the first coarse-grid operator, but this does not reintroduce the entries which were removed from coarser grid operators as a result of the non-Galerkin triple-matrix product $P_\ell^T \hat{A}_\ell P_\ell$. Figure 5 shows that this hierarchy requires little coarse-level communication after all entries have been reinstated to the first coarse-grid operator. However, as the required entries are not

added back into all coarser grid operators, the method still fails to converge.

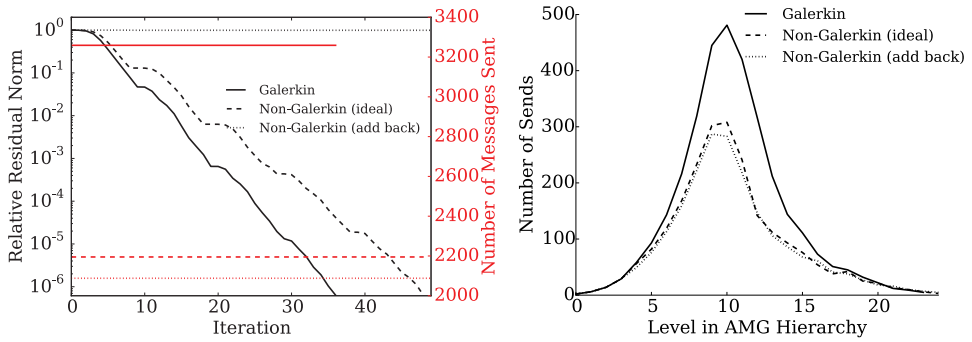


FIG. 5. *Convergence versus communication of Galerkin and non-Galerkin hierarchies for the 27-point Poisson problem on 8192 processes, with 10,000 degrees of freedom per process. Relative residual per AMG iteration (black) versus the number of message passing interface (MPI) sends per iteration (red) (left), and (maximum) number of sends per level in AMG hierarchy (right).*

3. Sparse and Hybrid Galerkin approaches. In this section we present two methods as alternatives to the method of non-Galerkin coarse grids. The methods consist of forming the entire Galerkin hierarchy before sparsifying each operator, yielding a lossless approach for increasing sparsity in the AMG hierarchy. The first method, which is called the Sparse Galerkin method, is described in Algorithm 4 (see line 1). Sparse Galerkin creates the entire Galerkin hierarchy as usual. The hierarchy is then thinned as a postprocessing step to remove relatively small entries outside of the minimal sparsity pattern $\mathcal{M} = \hat{P}^T AP + P^T A\hat{P}$ using `sparsify`.

Algorithm 4: `sparse_hybrid_setup`

Input: A_0: fine-grid operator
`max_size`: threshold for max size of coarsest problem
$\gamma_1, \gamma_2, \ldots$ drop tolerances for each level
`sparse_galerkin`: Sparse Galerkin method
`hybrid_galerkin`: Hybrid Galerkin method

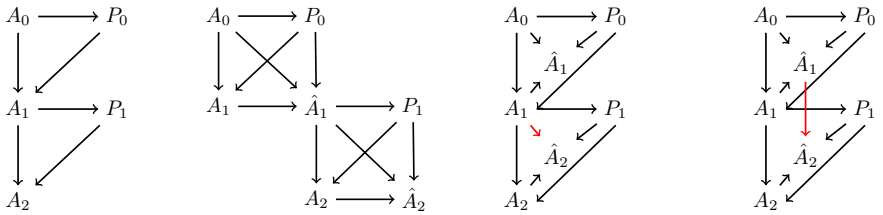
Output: $\hat{A}_1, \ldots, \hat{A}_L$

$A_1, \ldots, A_L, P_0, \ldots, P_{L-1} =$ `amg_setup`$(A_0,$ `max_size`, `False`$)$
$\hat{A}_0 = A_0$
for $\ell \leftarrow 1$ **to** L **do**
  if `sparse_galerkin`
1 $\hat{A}_{\ell+1} =$ `sparsify`$(A_{\ell+1}, A_\ell, P_\ell, S_\ell, \gamma_\ell)$ {Increase using the Sparse method}
  else if `hybrid_galerkin`
2 $\hat{A}_{\ell+1} =$ `sparsify`$(A_{\ell+1}, \hat{A}_\ell, P_\ell, S_\ell, \gamma_\ell)$ {Increase using the Hybrid method}

The second method that we introduce is called Hybrid Galerkin since it combines elements of Galerkin and Sparse Galerkin to create the final hierarchy. The method is again lossless, and is outlined in Algorithm 4 (see line 2). After the Galerkin hierarchy is formed, small entries outside are removed, this time using a modified,

minimal sparsity pattern of $\mathcal{M} = \hat{P}^T \hat{A} P + P^T \hat{A} \hat{P}$.

The Sparse and Hybrid Galerkin methods retain the structure of the original Galerkin hierarchy. Consequently, these methods introduce error only into relaxation and residual calculations. The remaining components of each V-cycle in the solve phase (see `amg_solve`), such as restriction and interpolation, are left unmodified. Therefore, the grid transfer operators do not depend on any sparsification, as shown in Figure 6. Here, we see that the Sparse Galerkin method does not use the modified (or sparsified) operators to create the next coarse-grid operator in the hierarchy. Conversely, Hybrid Galerkin uses the newly modified operator to compute the sparsity pattern $\mathcal{M}$ for the next coarse-grid operator.



(a) Galerkin (b) Non-Galerkin (c) Sparse Galerkin (d) Hybrid Galerkin

FIG. 6. *Dependencies for forming each operator in the various AMG hierarchies. The difference between Sparse and Hybrid Galerkin dependencies is highlighted in red.*

The new Sparse Galerkin and Hybrid Galerkin methods reduce the per-iteration cost in the AMG solve cycle as less communication is required by each sparse, coarse-grid operator. However, high-energy error may also be relaxed at a slower rate, yielding a reduction in the convergence factor. As a result, the solve phase is more efficient when the reduction in communication outweighs the change in convergence factor.

As with the method of non-Galerkin, it is difficult to predict the impact of removing entries from A_c in Algorithm 3 on the relaxation process. However, as the structure of the Galerkin hierarchy is retained, the convergence factor of the solve phase can be controlled on the fly. In our approach, differences between A_c and $\hat{A}_c$ are stored while forming the sparse approximations. Subsequently, if the convergence factor falls below a tolerance, entries can be reintroduced into the hierarchy, allowing improvement of the convergence factor up to that of the original Galerkin hierarchy (see section 6).

3.1. Diagonal lumping. A significant amount of work is required in Algorithm 3 to improve the sparsity of each coarse operator. When forming non-Galerkin coarse grids, this additional setup cost is hidden by the reduced cost of the triple-matrix product with the sparsified matrix $P^T \hat{A} P$. However, as the entire Galerkin hierarchy is initially formed as usual in our new methods (Algorithm 4), the additional work greatly reduces the scalability of the setup phase, as shown in section 5.2. This significant cost suggests using an alternative method for sparsification of coarse-grid operators. When reducing the number of nonzeros from coarse-grid operators with Sparse Galerkin or with Hybrid Galerkin, the structure of the Galerkin hierarchy remains intact, allowing a more flexible treatment of increasing sparsity in the matrix. For instance, one option is to remove entries by lumping to the diagonal rather than strong neighbors, as described in Algorithm 3b. This variation of `sparsify` is beneficial for several reasons, including a much cheaper setup phase when compared

to Algorithm 3; potential to reduce the cost of the solve phase; reduced storage constraints for adaptive solve phases (see Algorithm 5); and retaining positive-definiteness of coarse operators.

Algorithm 3b replaces the **for** loop in Algorithm 3. For each nonzero entry in the matrix, the algorithm first checks if the entry is the maximum element in the row and if all other entries in the row are selected for removal (see line 1). In this case, the nonzero entry is not removed if there is a zero row sum.

The method of diagonal lumping (Algorithm 3b) results in a cheaper setup phase than Algorithm 3. The original non-Galerkin `sparsify` requires each removed entry to be symmetrically lumped to significant neighbors. As a result, the process of calculating the associated strong connections requires a large amount of computation through a costly loop over neighbors of neighbors. Furthermore, to maintain symmetry, all matrix entries that are not stored locally must be updated, requiring a significant amount of interprocessor communication. Lumping these entries to the diagonal eliminates both the computational and communication complexities.

Eliminating the requirement of lumping to strong neighbors yields potential for removing a larger number of entries from the hierarchy, further reducing the communication costs of the solve phase. The original version of Algorithm 3 requires that an entry must have strong neighbors to be removed, as its value is lumped to these neighbors.

While relaxing the restrictions of the original non-Galerkin `sparsify` provides more opportunity to remove entries, the diagonal lumping also negatively influences convergence in some cases. Indeed, the energy of the operator can be increased as a result of the diagonal lumping, leading to a decrease in (spectral) equivalence with the original operator. To mitigate this, if convergence suffers, entries can be easily reintroduced into the hierarchy, improving convergence during the solve phase.[4] As removed entries are only added to the diagonal, the storage of both the sparse matrix along with removed entries is minimal. In addition, these entries can be restored by inserting their values into the original positions, and subtracting these values from the associated diagonal entries as shown in Algorithm 5. The process of reintroducing these entries requires no interprocessor communication as well as a small amount of local computational work.

Diagonal lumping also preserves matrix properties such as symmetric positive-definiteness. As described in the following theorem, if the sparsity of a diagonally dominant SPD matrix is increased using diagonal lumping, the resulting matrix remains SPD. Consequently, Sparse and Hybrid Galerkin with diagonal lumping can be used in preconditioning many methods such as conjugate gradient (CG). It is important to note that, while SPD matrices are an attractive property for AMG, AMG methods do not guarantee diagonal dominance of the coarse-grid operators. Yet, in many instances this property is preserved—for example, for more standard elliptic operators.

THEOREM 1. *Let A be SPD and diagonally dominant. If $\hat{A}$ is produced by Algorithm* 3b*, then it is symmetric positive semidefinite and diagonally dominant.*

[4] Another mitigating factor occurs when diagonal lumping is done after the construction of the hierarchy, when $\hat{A}_\ell$ will be used only for relaxation. At this point, $\hat{A}_\ell$ must only effectively damp high energy modes and leave low energy modes largely untouched. Since diagonal lumping of relatively small entries (as controlled by γ) largely impacts spectral equivalence for low energy modes, this mitigates any effect from reduced spectral equivalence.

Proof. Let A be SPD with diagonal dominance,

$$|A_{i,i}| \geq \sum_{k \neq i} |A_{i,k}| \quad \forall i. \tag{2}$$

Symmetry of $\hat{A}$ is guaranteed from the symmetry of both A and the $\mathcal{N}$ from Algorithm 3. For all off-diagonal entries $(i,j),(j,i) \in \mathcal{N}$,

$$\hat{A}_{i,j} = A_{i,j} = A_{j,i} = \hat{A}_{j,i}, \tag{3}$$

by line 2 in Algorithm 3b and the symmetry of A.

The positive-definiteness is guaranteed by the diagonal dominance and by a Gershgorin disc argument. The proof proceeds by starting with the matrix A and then considering the change made to A by the elimination of each entry. Initially, all the Gershgorin discs of A are strictly on the right side of the origin, thus implying that all eigenvalues are nonnegative. Then, assume that we eliminate some arbitrary entry $A_{i,j}$, $(i,j) \in \mathcal{N}$. This results in row i being updated:

$$A_{i,i} \leftarrow A_{i,i} + A_{i,j} \quad \text{and} \quad A_{i,j} \leftarrow 0. \tag{4}$$

If $A_{i,j} > 0$, then the center of the Gershgorin disc is shifted to the right, and the radius shrinks, thus keeping the disc to the right of the origin and preserving definiteness. If $A_{i,j} < 0$, then the center of the disc is shifted to the left by $|A_{i,j}|$, but the radius of the disc also shrinks by $|A_{i,j}|$. This also keeps the disc to the right of the origin and preserves semidefiniteness. Furthermore, since each disc is never shifted to the left half plane, diagonal dominance is also preserved. The proof then proceeds by considering all of the entries to be eliminated. ∎

Remark 3.1. If any row of A is strictly diagonally dominant, as often happens with Dirichlet boundary conditions, then $\hat{A}$ will be positive definite. Essentially, Algorithm 3b never shifts a Gershgorin disc to the left, so $\hat{A}$ can have no 0 eigenvalue.

4. Parallel performance. In this section we use a parallel performance model to illustrate the per-level costs associated with each of the six methods:

Galerkin: Classic coarsening in AMG, as outlined in Algorithm 1.
Non-Galerkin: The base algorithm presented in [15].
Sparse Galerkin: The new Algorithm 4 with `sparse_galerkin` and full lumping from Algorithm 3.
Sparse Galerkin (Diag): The new Algorithm 4 with `sparse_galerkin` and diagonal lumping from Algorithm 3b.
Hybrid Galerkin: The new Algorithm 4 with `hybrid_galerkin` and full lumping from Algorithm 3.
Hybrid Galerkin (Diag): The new Algorithm 4 with `hybrid_galerkin` and diagonal lumping from Algorithm 3b.

The solve phase of AMG (see Algorithm 2) is largely comprised of sparse matrix-vector multiplication; thus we model each method by assessing the cost of performing an SpMV on each level of the hierarchy. We focus on the operators A_ℓ, as the work required for this matrix is more costly than the restriction and interpolation operations. Specifically, we employ an α–β model to capture the cost of the parallel SpMV based on the number of nonzeros in A. We denote p as the number of processors, α as the latency or startup cost of a message, and β as the reciprocal of the network bandwidth [17, 18]. In addition, $\texttt{nnz}_p$ represents the average number of nonzeros local

to a process, while s_p and n_p are the maximum numbers of message passing interface (MPI) sends and message size across all processors. Finally, we use c to represent the cost of a single floating-point operation. With this we model the total time as

$$T = 2\,c\,\mathtt{nnz}_p + \max_p s_p(\alpha + \beta n_p). \tag{5}$$

For the model parameters above we use the Blue Waters supercomputer at the University of Illinois at Urbana-Champaign [1, 7]. The latency and bandwidth were measured through the HPCC benchmark [21], yielding $\alpha = 1.8 \times 10^{-6}$ and $\beta = 1.8 \times 10^{-9}$. Since the achieved flop rate depends on matrix size, we determine the value of c by timing the local SpMV. Specifically, letting $\mathtt{nnz}_{\text{local}}$ be the number of nonzeros local to the processor and T_{local} the time to perform the local portion of the SpMV, we compute $c = {}^{T_{\text{local}}}/_{2\mathtt{nnz}_{\text{local}}}$ for each matrix in the hierarchy.

The minimal per-level cost associated with the non-Galerkin and Sparse/Hybrid Galerkin methods occurs when entries are removed with a drop tolerance of $\gamma = 1.0$. Using the model, (5), this is highlighted in Figure 7 for both the Laplace and rotated anisotropic diffusion problems (a full description of these problems is given in section 5). We see that both non-Galerkin and Hybrid Galerkin have the potential to minimize the per-level cost. However, when the per-level cost is minimized, the convergence of AMG often suffers. Therefore, less aggressive drop tolerances such as $\gamma < 1.0$ may remove fewer entries, increasing the per-level cost, but due to better convergence will improve the overall cost of the solve phase. Indeed, for the rotated anisotropic diffusion problem, this is the case, where we reduce γ at fine levels in the hierarchy in order to retain convergence (not shown for brevity).

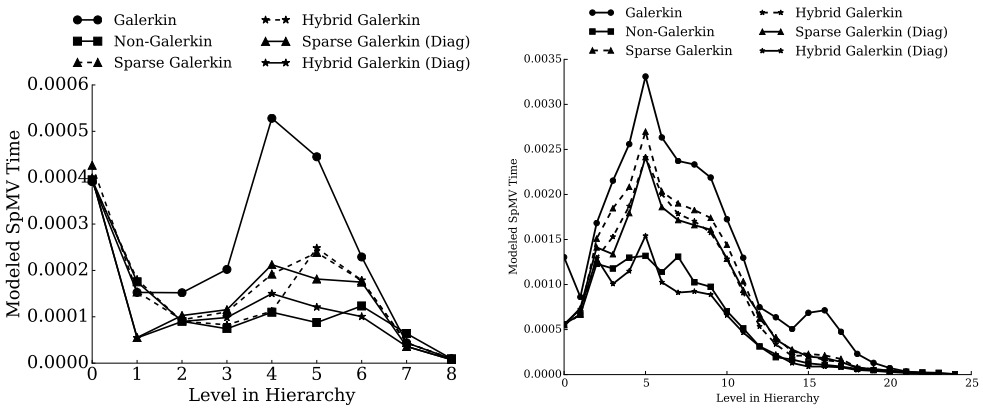


FIG. 7. *Modeled minimal cost of a single SpMV on each level of the AMG hierarchy for Laplace (left) and rotated anisotropic diffusion (right), for an aggressive drop tolerance of* 1.0 *on each level.*

5. Parallel results for Sparse and Hybrid Galerkin. In this section we highlight the parallel performance of the Sparse and Hybrid Galerkin methods. We consider scaling tests on the familiar 3D Laplacian since this is a common multigrid problem used to establish a baseline. In order to test problems where AMG convergence is suboptimal, we consider a two-dimensional (2D) rotated anisotropic diffusion problem. Finally, we test our methods on a suite of matrices from the Florida Sparse Matrix Collection. All computations were performed on the Blue Waters system at the University of Illinois at Urbana-Champaign [1]. Each method was implemented

and solved with *hypre* [2, 19], using default parameters unless otherwise specified. In summary, we compare the solve and setup times for the six methods considered in section 4 while preconditioning a Krylov method such as CG or GMRES in each test.

The drop tolerances for each method vary by level, using a combination of 0.0, 0.01, 0.1, and 1.0 across the coarse levels. Six combinations of these drop tolerances are tested for the various test cases, and the series yielding the minimum solve time for each is selected. Note that at $100,000$ cores, the best drop tolerances from the second largest run size are used due to large costs associated with running six drop tolerances at this core count. Details of the drop tolerances used in all the tests below are given in the Supplemental Materials (nongalerkin_paper_supplement.pdf [local/web 132KB]) due to length.[5]

Generally, the drop tolerances are aggressive for the simple isotropic 3D Laplacian example, where, for instance, in Figures 8 and 9, the Sparse Galerkin (Diag) method used the values of $[0.0, 0.1]$, i.e., no dropping on the first coarse level and then 0.1 on all subsequent levels. For the more complex examples such as rotated anisotropic diffusion, the drop tolerances are less aggressive and usually begin on coarser levels. For instance, in Figures 8 and 9, the Sparse Galerkin (Diag) method used values of $[0.0, 0.0, 0.0, 0.1]$; i.e., no dropping occurs until the third coarse level, where 0.1 is used from that point onward on all coarser levels. Overall, the drop tolerance is similar to other multigrid parameters, such as the strength-of-connection drop tolerance. Some experimentation with a problem type is required, but thereafter a general, conservative parameter choice can be made for subsequent use.

We consider the diffusion problem

$$-\nabla \cdot K \nabla u = 0, \tag{6}$$

with two particular test cases for our simulations. Also, as problems with less structure result in increased density on coarse levels, we consider a subset from the Florida Sparse Matrix Collection. The test problems are as follows.

Laplace: Here, we use $K = I$ on the unit cube with homogeneous Dirichlet boundary conditions. Q1 finite elements are used to discretize the problem using a uniform mesh, leading to a familiar 27-point stencil. The preconditioner formed for the 3D Laplacian uses aggressive coarsening (HMIS) and distance-two (extended classical modified) interpolation. The interpolation operators were formed with a maximum of five elements per row, and hybrid symmetric Gauss–Seidel was the relaxation method.

Rotated Anisotropic Diffusion: In this case, we consider a diffusion tensor with homogeneous Dirichlet boundary conditions of the form $K = Q^T D Q$, where Q is a rotation matrix and D is a diagonal scaling defined as

$$Q = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}, \qquad D = \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix}. \tag{7}$$

Q1 finite elements are used to discretize a uniform, square mesh. In the following tests we use $\theta = \frac{\pi}{8}$ and $\epsilon = 0.001$. In each case, the preconditioner uses Falgout coarsening [10], extended classical modified interpolation, and hybrid symmetric Gauss–Seidel.

[5] In addition, the modifications to *hypre* `v2.11.0` are stored at https://github.com/lukeolson/hypre/releases/tag/SISC-sparse-hybrid-galerkin and https://github.com/lukeolson/hypre/releases/tag/SISC-sparse-hybrid-galerkin-adaptive.

Florida Sparse Matrix Collection: We consider a subset of all real, SPD matrices from the Florida Sparse Matrix Collection with size over 1,000,000 degrees of freedom. In addition, we consider only the cases where GMRES preconditioned with Galerkin AMG converges in fewer than 100 iterations. Each problem uses HMIS coarsening and so-called *extended+i* interpolation if possible. In some cases, however, Galerkin AMG does not converge with these options; in these cases Falgout coarsening and modified classical interpolation are used. Relaxation for all systems is hybrid symmetric Gauss–Seidel. Note that, when necessary for convergence, some *hypre* parameters, such as the minimum coarse-grid size and strength tolerance, vary from the default.

The following results demonstrate that the diagonally lumped Sparse and Hybrid Galerkin methods are able to perform comparably to non-Galerkin. Non-Galerkin and Sparse/Hybrid Galerkin all significantly reduce the per-iteration cost by reducing communication on coarse levels. Since the method of non-Galerkin is multiplicative in construction, the setup times are often much lower in comparison to standard Galerkin. However, Sparse and Hybrid do not observe this benefit since they are constructed in a postprocessing step. While the per-iteration work is decreased for all methods, the convergence suffers for the case of rotated anisotropic diffusion problems with non-Galerkin at large scales. However, Sparse and Hybrid Galerkin converge at rates similar to the original Galerkin hierarchy, yielding speedup in total solve times. Moreover, in a strong scaling study, we observe that Hybrid Galerkin is competitive, particularly at large core counts.

5.1. Improving sparsity in AMG hierarchies. The significant number of nonzeros on coarse levels creates large, relatively dense matrices near the middle of the AMG hierarchy, yielding large communication costs for each SpMV performed on these levels. As the solve phase of AMG consists of many SpMVs on each level of the hierarchy, the time spent on coarse levels can increase dramatically. Sparse, Hybrid, and non-Galerkin can all reduce both the cost associated with communication as well as the time spent on each level during a solve phase.

Figure 8 shows the time spent on each level of the hierarchy during a single iteration of AMG for both test cases with 10,000 degrees of freedom per core using 8192 cores. Both the method of non-Galerkin coarse grids and the Sparse and Hybrid Galerkin methods reduce the time required on levels near the middle of the hierarchy. Non-Galerkin has a larger impact on the time spent on middle levels of the hierarchy for the Laplace problem than Sparse and Hybrid Galerkin. However, for the anisotropic problem, diagonally lumped Hybrid Galerkin reduces level-time similarly to non-Galerkin. This is due to a large reduction in the number of messages required in each SpMV, as shown in Figure 9. The reduction in total size of all messages communicated is relatively small.

The increase in time spent on each level, as well as the associated communication costs of these levels, becomes more pronounced at higher processor counts in a strong scaling study. Figure 10 illustrates this by plotting the per-level times required during a single iteration of AMG, as well as the number of messages communicated during an SpMV for the rotated anisotropic diffusion problem with 1,250 degrees of freedom per core using 8192 cores. Compared with the 10,000 degrees of freedom per core in Figures 8 and 9, there is a sharper increase in time required for levels near the middle of the hierarchy due to the increasing dominance of communication complexity.

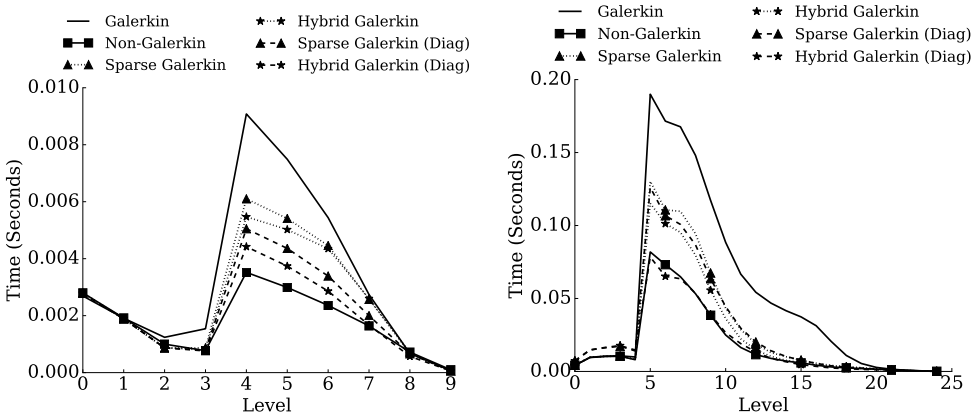


FIG. 8. *Time spent on each level of the AMG hierarchy during a single iteration of the solve phase for* Laplace (left) *and* Rotated Anisotropic Diffusion (right)*, each with* 10,000 *degrees of freedom per core.*

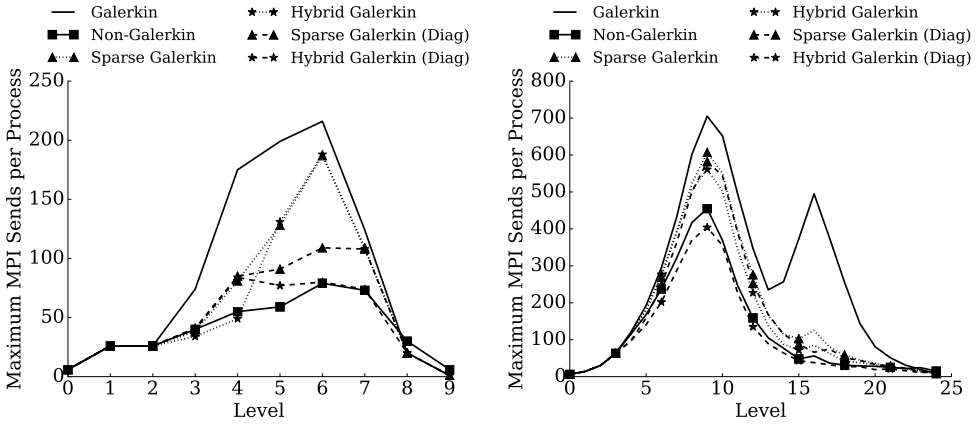


FIG. 9. *Number of sends required to perform a single SpMV on each level of the AMG hierarchy for:* Laplace (left) *and* Rotated Anisotropic Diffusion (right)*, each with* 10,000 *degrees of freedom per core.*

5.2. Costs of weakly scaled setup phases. Each sparsification method can lead to reduced communication costs in the middle of the hierarchy. However, removing insignificant entries from coarse-grid operators requires additional work in the setup phase. In the non-Galerkin method, setup times are reduced since the increased sparsity is used directly in the triple-matrix product required to form each successive coarse-grid operator. However, for the new methods, Sparse and Hybrid Galerkin, the entire Galerkin hierarchy is first constructed so that the sparsify process on each level requires additional work. Figure 11 shows the times required to set up an AMG hierarchy for rotated anisotropic diffusion, with Laplace setup times scaling in a similar manner. While there is a slight increase in setup cost associated with the Sparse and Hybrid Galerkin hierarchies, this extra work is nominal. Therefore, while the majority of this additional work is removed when using diagonal lumping, the difference in work required in the setup phase of these two lumping strategies is

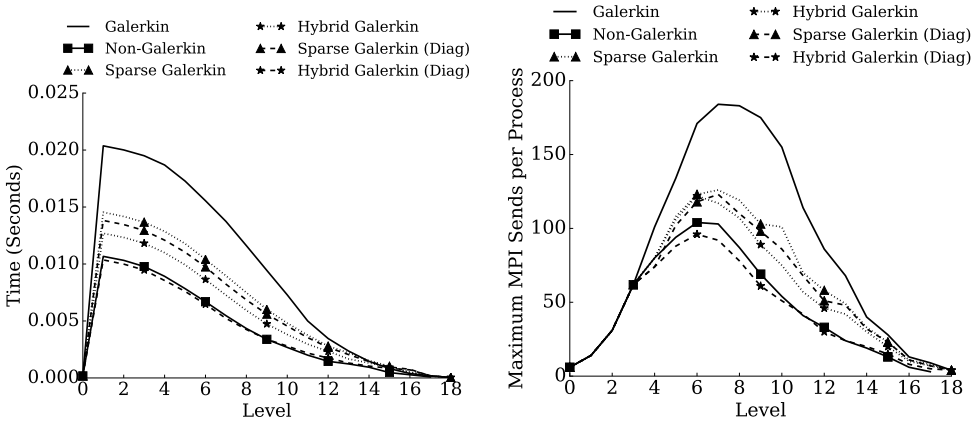


FIG. 10. *For each level of the AMG hierarchy, time per iteration of AMG (left) and number of messages sent during a single SpMV (right) for the* Rotated Anisotropic Diffusion *problem with* 1,250 *degrees of freedom per core.*

insignificant for the problems being tested.

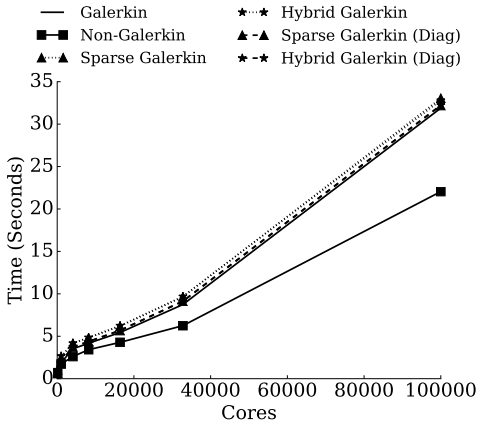


FIG. 11. *Time required to set up AMG hierarchy for* Rotated Anisotropic Diffusion *with* 10,000 *degrees of freedom per core.*

5.3. Weak scaling of GMRES preconditioned by AMG. In this section we investigate the *weak* scaling properties of the methods. Figure 12 shows both the average convergence factor and the total time spent in the solve phase for a weak scaling study with rotated anisotropic diffusion problems at 10,000 degrees of freedom per core using GMRES preconditioned by AMG. GMRES is used instead of CG because Algorithm 3 guarantees symmetry but not positive-definiteness of the preconditioner. In many cases, positive-definiteness is preserved, but when using more aggressive drop tolerances, we have observed that this property is lost. While the convergence of both diagonally lumped Sparse and Hybrid Galerkin remains similar to that of Galerkin, the non-Galerkin method converges more slowly. Therefore, while non-Galerkin and diagonally lumped Hybrid Galerkin yield similar communication requirements, GMRES preconditioned by Hybrid Galerkin performs significantly better

as fewer iterations are required.

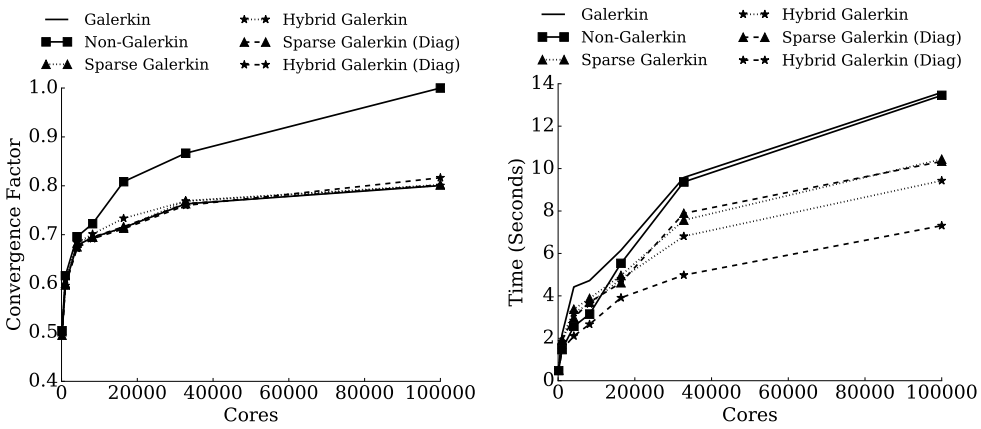


FIG. 12. *Convergence factors (left) and times (right) for weak scaling of* Rotated Anisotropic Diffusion *(*10,000 *degrees of freedom per core), solved by preconditioned GMRES. For large problem sizes, non-Galerkin AMG does not converge, and timings indicate when the maximum iteration count was reached.*

Remark 5.1. With the chosen drop tolerances, non-Galerkin does not converge for this anisotropic problem at 100,000 cores. In this case, nothing was dropped from the first three coarse levels of the hierarchy. On the fourth coarse level, a drop tolerance of 0.01 was used, and the fifth was sparsified with a tolerance of 0.1. The remaining levels were sparsified with a drop tolerance of 1.0. This was determined to be the best tested drop tolerance sequence for smaller run sizes, and multiple drop tolerance sequences were not tuned at this large problem size due to the significant costs. However, a better drop tolerance could yield a convergent non-Galerkin method at this scale.

The efficiency of weakly scaling to p processes is defined as $E_p = \frac{T_1}{T_p}$, where T_1 is the time required to solve the problem on a single process and T_p is the time to solve on p processes. The efficiency of solving weakly scaled rotated anisotropic diffusion problems with non-Galerkin, Sparse Galerkin, and Hybrid Galerkin, relative to the efficiency of Galerkin AMG, is shown in Figure 13. While both the original and diagonally lumped Sparse and Hybrid Galerkin methods scale more efficiently than Galerkin, the poor convergence of non-Galerkin on large run sizes yields a reduction in relative efficiency.

5.4. Strong scaling of GMRES preconditioned by AMG. We next consider the rotated anisotropic diffusion system with approximately 10,240,000 unknowns using cores ranging from 128 to 100,000. Therefore, the simulation is reduced from 80,000 degrees of freedom per core when run on 128 cores to just over 100 degrees of freedom per core on 100,000 cores. Computation dominates the total cost of solving a problem partitioned over relatively few processes, as each process has a large amount of local work. However, as the problem is distributed across an increasing number of processes, the local work decreases while communication requirements increase. Therefore, the time required to solve a problem is reduced with strong scaling, but only to the point where communication complexity begins to dominate. The efficiency of strongly scaling to p processes is defined as $E_p = \frac{T_1}{pT_p}$. Figure 14

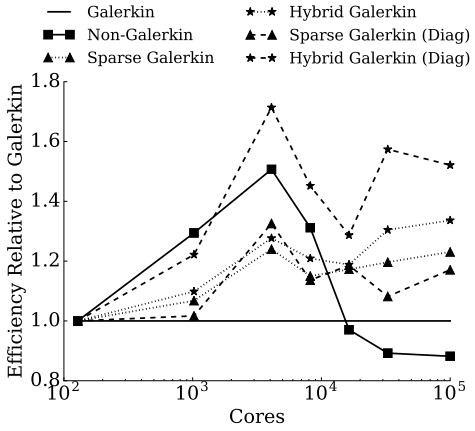


FIG. 13. *Efficiency of solving weakly scaled* Rotated Anisotropic Diffusion *at* 10,000 *degrees of freedom per core with various methods,* relative to that of the Galerkin hierarchy.

shows the efficiency of solving a strongly scaled rotated anisotropic diffusion problem with GMRES preconditioned by the various sparse methods. In each case we observe improvements over standard Galerkin.

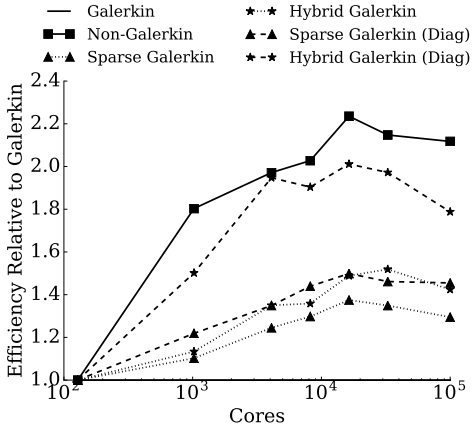


FIG. 14. *Efficiency of non-Galerkin and Sparse/Hybrid Galerkin methods in a strong scaling study,* relative to Galerkin AMG *for* Rotated Anisotropic Diffusion.

A strong scaling study is also performed on the subset of matrices from the Florida Sparse Matrix Collection. These problems were tested on 64, 128, 256, and 512 processes. Figure 15 shows the time required to perform a single V-cycle for each of the matrices in the subset, relative to the time required by Galerkin AMG. All methods reduce the per-iteration times for each matrix in the subset. Furthermore, the total time required to solve each of these matrices is also reduced, as shown in Figure 16. While Sparse Galerkin provides some improvement, the Hybrid and non-Galerkin methods are comparable, particularly at high core counts.

5.5. Diagonal lumping alternative and PCG. Diagonal lumping retains positive definiteness of diagonally dominant coarse-grid operators as described in The-

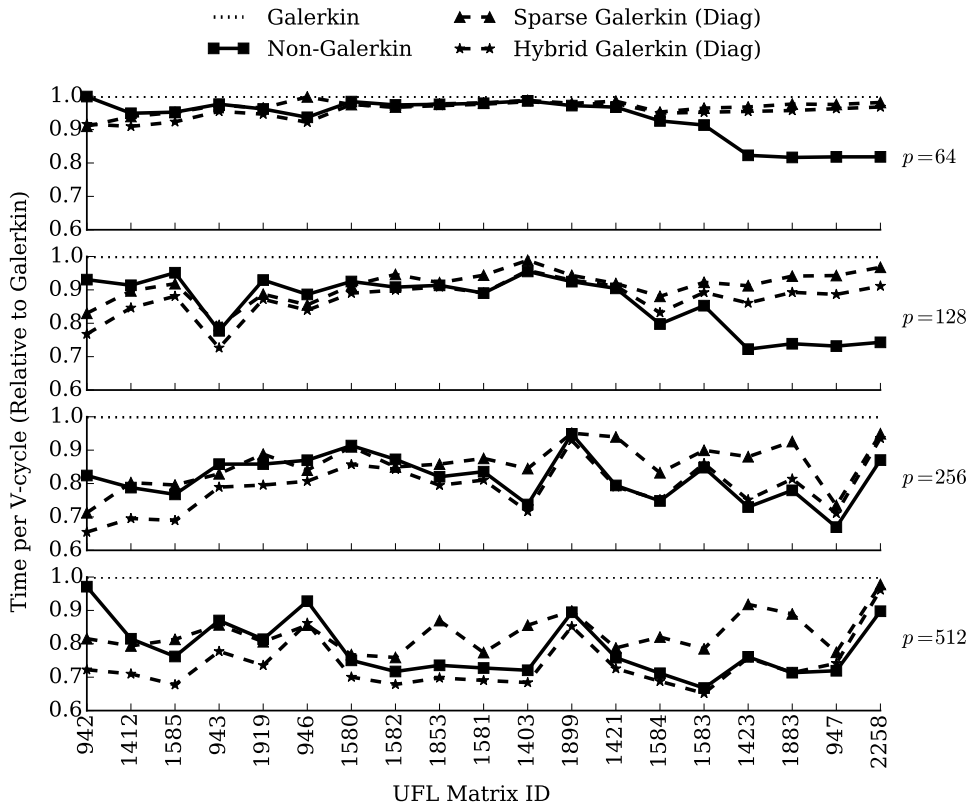


FIG. 15. *Time (relative to Galerkin) per iteration for each matrix in the Florida Sparse Matrix Collection, using* $p = 64$*,* 128*,* 256*, and* 512*.*

orem 1. Therefore, as the preconditioned conjugate gradient (PCG) method requires both the matrix and preconditioner to be symmetric and positive-definite, the Laplace and anisotropic diffusion problems are solved by CG preconditioned by the diagonally lumped Sparse and Hybrid Galerkin hierarchies. Figure 17 shows the solve phase times for solving the weakly scaled rotated anisotropic diffusion problem with PCG. As with GMRES, both the Sparse and Hybrid Galerkin preconditioners decrease the time required in the AMG solve phase during a weak scaling study.

6. Adaptive solve phase. The previous results describe the case where good drop tolerances were known a priori for `sparsify`. However, as the appropriate drop tolerance changes with problem type, problem size, and even level of the AMG hierarchy, a good drop tolerance is often not easily realized. When the drop tolerance is too small, few entries are removed from the hierarchy, and the communication complexity remains the same. However, if the drop tolerance is too large, the solver is nonconvergent, as described in section 2.1.

In this section we consider an *adaptive* method that attempts to add entries back into the hierarchy as a deterioration in convergence is observed. This is detailed in Algorithm 5. The algorithm initializes a Sparse or Hybrid Galerkin hierarchy and proceeds by executing k iterations of a preconditioned Krylov method—e.g., PCG. If the convergence is below a tolerance, the coarse levels are traversed until a coarse

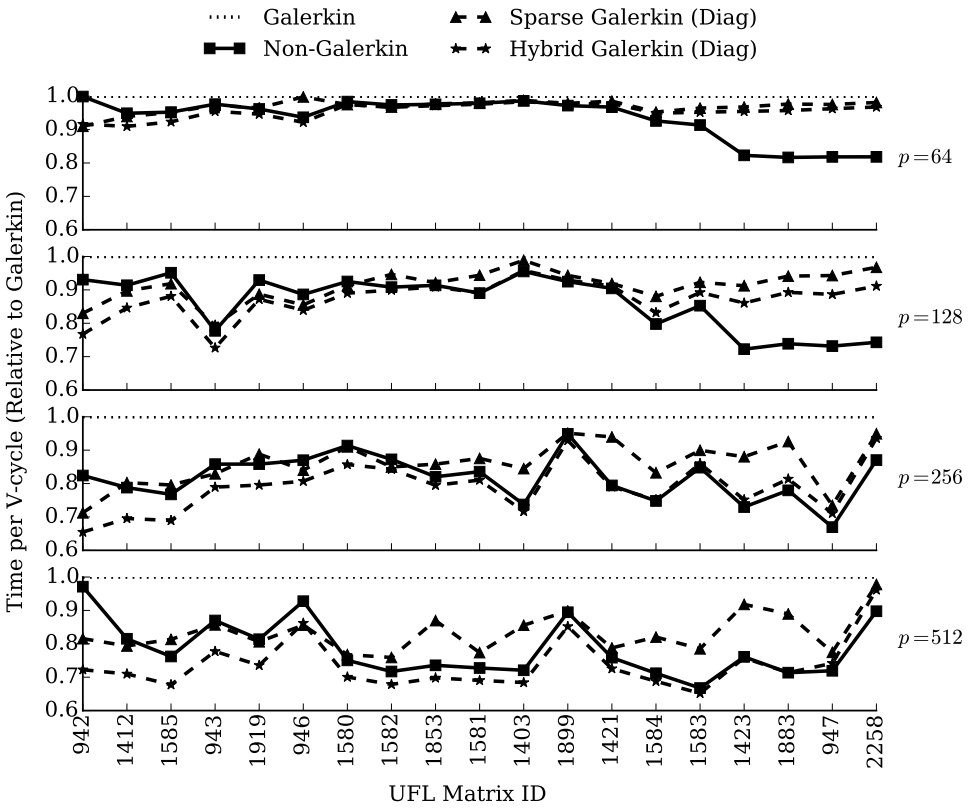


FIG. 16. *Time (relative to Galerkin) per AMG solve for each matrix in the Florida Sparse Matrix Collection, using* $p = 64$, 128, 256, *and* 512.

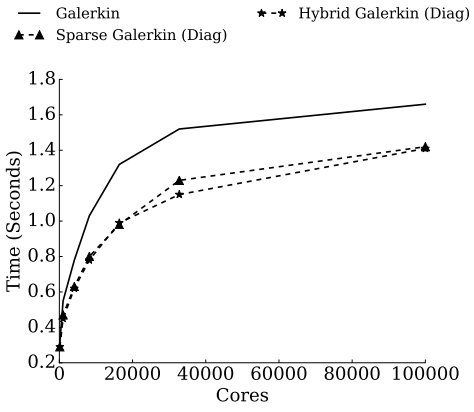


FIG. 17. *Weak scaling solve time for* Rotated Anisotropic Diffusion*, solved by PCG preconditioned by various AMG hierarchies.*

grid operator is found on which entries were removed with a drop tolerance greater than 0.0. Entries are then added back to this coarse-grid operator, reducing the drop tolerance by a factor of 10. Any new drop tolerance below $\gamma_{\min} = 0.01$ is rounded

down to 0.0. This continues until entries have been reintroduced into s coarse-grid operators. At this point, the Krylov method continues, using the most recent value for x unless the previous iterations diverged from the true solution.

This entire process is then repeated until convergence. The adaptive solve phase requires additional iterations over Galerkin AMG, as initial iterations of this method may not converge. However, the goal of this solver is to guarantee convergence similar to Galerkin AMG. Speedup over Galerkin AMG is still dependent on choosing reasonable initial drop tolerances.

If the Krylov method is not flexible, such as PCG or GMRES, then it must be restarted after the preconditioner has been edited. On the other hand, a flexible method, such as FGMRES, would not have to be restarted but requires greater memory storage (and possibly also more computational work) than PCG. While the new algorithms are agnostic to the Krylov scheme, we use restarted PCG in order to directly compare schemes.

Example 6.1. As an example, consider the case of a hierarchy with six levels using drop tolerances of $[0, 0.01, 0.1, 1.0, 1.0, 1.0]$; i.e., $\hat{A}_1$ retains all entries from A_1, while $\hat{A}_2$ and $\hat{A}_3$ result from `sparsify` with $\gamma = 0.01$ and $\gamma = 0.1$, etc. Suppose that `adaptive_solve` with $k = 3$ and $s = 2$ results in three iterations of PCG and a large residual. The adaptive solve finds the first level containing a sparsified coarse grid matrix, namely $\hat{A}_2$. The drop tolerance on this level is changed from 0.01 to 0.0, and the original coarse matrix A_2 is sparsified with the new drop tolerance. Furthermore, since $s = 2$, the drop tolerance on level 3 is reduced from 0.1 to 0.01, and A_3 is also sparsified. PCG then restarts with the new hierarchy. If convergence continues to suffer after three iterations, the hierarchy is updated again, but since $\hat{A}_2$ has $\gamma_2 = 0.0$, entries are reintroduced into coarse matrices $\hat{A}_3$ and $\hat{A}_4$ instead.

Using Algorithm 5, Figure 18 shows both the relative residual of the system after each iteration as well as the communication costs of PCG using three different AMG hierarchies: standard Galerkin, Sparse Galerkin with diagonal lumping and aggressive dropping, and Sparse Galerkin with diagonal lumping modified with adaptivity. For the adaptive case, we purposefully choose an overly aggressive initial drop tolerance so that entries can be added back multiple times and one coarse level at a time to show the effect on convergence and communication. Initially, when the drop tolerance is aggressive, the associated communication costs are low, but the resulting PCG iterations do not converge; this provides a baseline. As sparse entries are reintroduced into the hierarchy, convergence improves, while only slightly increasing the associated communication cost. When entries are reintroduced into the hierarchy, the preconditioner for PCG changes, and hence the method must be restarted. After restarting the method, convergence improves.

7. Conclusion. We have introduced a lossless method to reduce the work required in parallel algebraic multigrid by removing weak or unimportant entries from coarse-grid operators after the multigrid hierarchy is formed. This alternative to the original method of non-Galerkin coarse grids is similarly capable of reducing the communication costs on coarse levels, yielding an overall reduction in solve times. Furthermore, this method retains the original Galerkin hierarchy, allowing many of the restrictions of non-Galerkin to be relaxed. As a result, removed entries are easily lumped directly to the diagonals, greatly reducing setup costs, while also reducing communication complexity during the solve phase. Furthermore, as entries are added to the diagonal, entries removed from the matrix are stored and adaptively reintro-

Algorithm 5: `adaptive_solve`

Input: A, b, x_0

$\hat{A}_1, \ldots, \hat{A}_L$	Sparse/Hybrid Galerkin coarse grid matrices
$A_1, \ldots, A_L$	original Galerkin coarse grid matrices
$P_0, \ldots, P_{L-1}$	
k	PCG iterations before convergence test
s	AMG levels per update
$\gamma_0, \ldots, \gamma_L$	sparsification drop tolerance used at each level
`tol`	convergence tolerance
`sparse_galerkin`	Sparse Galerkin method
`hybrid_galerkin`	Hybrid Galerkin method

Output: x

$x = x_0$
$r_0 = b - Ax_0$

while $\|r\|/\|r_0\| \leq$ `tol`
  $M =$ `preconditioner`$(\texttt{amg\_solve}, \hat{A}_1, \ldots, \hat{A}_L, P_0, \ldots, P_{L-1})$
  $x = \mathrm{PCG}(A, b, x, k, M)$ {Call k steps of preconditioned CG}
  $r = b - Ax$
  if $\frac{\|r\|}{\|r_0\|} \leq$ `tol`
    `continue`
  else
    for $\ell = 0, \ldots, L$ **do**
      if $\gamma_\ell > 0$
        $\ell_{\text{start}} \leftarrow \ell$ {Find finest level that uses dropping}
    for $\ell = \ell_{start} \ldots \ell_{start} + s$ **do**
      $$\gamma_\ell = \begin{cases} \frac{\gamma_\ell}{10} & \text{if } \frac{\gamma_\ell}{10} > \gamma_{\min}, \\ 0 & \texttt{otherwise} \end{cases}$$ {Determine new dropping parameter}
      {$\gamma_{\min}$ is the $\min(\gamma_0 \ldots \gamma_L)$}
      if `sparse_galerkin` {Re-add entries at the new dropping tolerance}
        $\hat{A}_\ell =$`sparsify`$(A_\ell, A_{\ell-1}, P_{\ell-1}, S_{\ell-1}, \gamma_\ell)$
      else if `hybrid_galerkin` {Re-add entries at the new dropping tolerance}
        $\hat{A}_\ell =$`sparsify`$(A_\ell, \hat{A}_{\ell-1}, P_{\ell-1}, S_{\ell-1}, \gamma_\ell)$

duced into the hierarchy if necessary for convergence. Hence, the trade-off between convergence and the communication costs is controlled at solve-time with little additional work.

REFERENCES

[1] *Blue Waters*, https://bluewaters.ncsa.illinois.edu/.
[2] *HYPRE: High Performance Preconditioners*, http://www.llnl.gov/CASC/hypre/.
[3] D. M. Alber and L. N. Olson, *Parallel coarse-grid selection*, Numer. Linear Algebra Appl., 14 (2007), pp. 611–643, doi:10.1002/nla.541.
[4] S. F. Ashby and R. D. Falgout, *A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations*, Nuclear Sci. Engrg., 124 (1996), pp. 145–159.

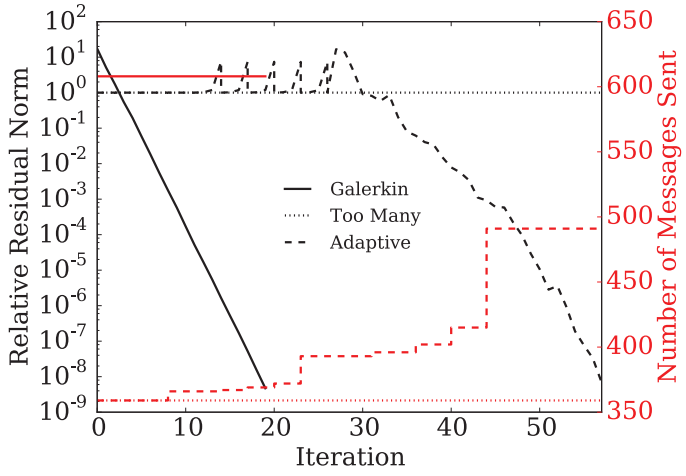


FIG. 18. *Relative residual (black) and number of MPI sends (red) per iteration when solving the* Laplace *problem with* (1) *PCG using Galerkin AMG;* (2) *Hybrid Galerkin with aggressive dropping (labeled Too Many);* (3) *Hybrid Galerkin solved with Algorithm* 5*, using* $k = 3$*,* $s = 1$*, and* $\gamma_0 \dots \gamma_L$ *set to the same drop tolerances as the aggressive case.*

[5] A. H. BAKER, M. SCHULZ, AND U. M. YANG, *On the performance of an algebraic multigrid solver on multicore clusters*, in Proceedings of the 9th International Conference on High Performance Computing for Computational Science, VECPAR'10, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 102–115, doi:10.1007/978-3-642-19328-6_12.

[6] W. N. BELL, L. N. OLSON, AND J. B. SCHRODER, *PyAMG: Algebraic Multigrid Solvers in Python v*3.0, Release 3.0, 2015, http://www.pyamg.org.

[7] B. BODE, M. BUTLER, T. DUNNING, T. HOEFLER, W. KRAMER, W. GROPP, AND W. HWU, *The Blue Waters super-system for super-science*, in Contemporary High Performance Computing: From Petascale toward Exascale, CRC Computational Science Series 1, J. S. Vetter, ed., Taylor and Francis, Boca Raton, FL, 2013, pp. 339–366.

[8] M. BOLTEN, T. K. HUCKLE, AND C. D. KRAVVARITIS, *Sparse matrix approximations for multigrid methods*, Linear Algebra Appl., 502 (2016), pp. 58–76, doi:10.1016/j.laa.2015.11.008.

[9] A. BRANDT, S. F. MCCORMICK, AND J. W. RUGE, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, UK, 1984, pp. 257–284.

[10] E. CHOW, R. D. FALGOUT, J. J. HU, R. S. TUMINARO, AND U. M. YANG, *A survey of parallelization techniques for multigrid solvers*, in Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, 2006, pp. 179–201.

[11] J. DENDY, *Black box multigrid*, J. Comput. Phys., 48 (1982), pp. 366–386, doi:10.1016/0021-9991(82)90057-2.

[12] J. DENDY, *Black box multigrid for nonsymmetric problems*, Appl. Math. Comput., 13 (1983), pp. 261–283, doi:10.1016/0096-3003(83)90016-4.

[13] H. DE STERCK, R. D. FALGOUT, J. W. NOLTING, AND U. M. YANG, *Distance-two interpolation for parallel algebraic multigrid*, Numer. Linear Algebra Appl., 15 (2008), pp. 115–139, doi:10.1002/nla.559.

[14] H. DE 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, doi:10.1137/040615729.

[15] R. D. FALGOUT AND J. B. SCHRODER, *Non-Galerkin coarse grids for algebraic multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. C309–C334, doi:10.1137/130931539.

[16] W. S. FUNG, R. HARIHARAN, N. J. HARVEY, AND D. PANIGRAHI, *A general framework for graph sparsification*, in Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC '11, ACM, New York, 2011, pp. 71–80, doi:10.1145/1993636.1993647.

[17] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp, *Modeling the performance of an algebraic multigrid cycle on HPC platforms*, in Proceedings of the International Conference on Supercomputing, ICS '11, ACM, New York, 2011, pp. 172–181, doi:10.1145/1995896.1995924.

[18] H. Gahvari, M. Hoemmen, J. Demmel, and K. Yelick, *Benchmarking sparse matrix-vector multiply in five minutes*, in SPEC Benchmark Workshop 2007, Austin, TX, 2007.

[19] V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177, doi:10.1016/S0168-9274(01)00115-5.

[20] I. Koutis, A. Levin, and R. Peng, *Faster spectral sparsification and numerical algorithms for SDD matrices*, ACM Trans. Algorithms, 12 (2016), 17, doi:10.1145/2743021.

[21] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, *The HPC Challenge (HPCC) benchmark suite*, in Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (Tampa, FL), ACM, New York, 2006, 213, doi:10.1145/1188455.1188677.

[22] S. F. McCormick and J. W. Ruge, *Multigrid methods for variational problems*, SIAM J. Numer. Anal., 19 (1982), pp. 924–929, doi:10.1137/0719067.

[23] L. N. Olson, J. Schroder, and R. S. Tuminaro, *A new perspective on strength measures in algebraic multigrid*, Numer. Linear Algebra Appl., 17 (2010), pp. 713–733, doi:10.1002/nla.669.

[24] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods, Frontiers Appl. Math. 3, S. F. McCormick, ed., SIAM, Philadelphia, 1987, pp. 73–130.

[25] D. A. Spielman and N. Srivastava, *Graph sparsification by effective resistances*, SIAM J. Comput., 40 (2011), pp. 1913–1926, doi:10.1137/080734029.

[26] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC '04, ACM, New York, 2004, pp. 81–90, doi:10.1145/1007352.1007372.

[27] E. Treister and I. Yavneh, *Non-Galerkin multigrid based on sparsified smoothed aggregation*, SIAM J. Sci. Comput., 37 (2015), pp. A30–A54, doi:10.1137/140952570.

[28] E. Treister, R. Zemach, and I. Yavneh, *Algebraic collocation coarse approximation (ACCA) multigrid*, in 12th Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, 2012.

[29] R. Wienands and I. Yavneh, *Collocation coarse approximation in multigrid*, SIAM J. Sci. Comput., 31 (2009), pp. 3643–3660, doi:10.1137/08074461X.

[30] U. M. Yang, *On long-range interpolation operators for aggressive coarsening*, Numer. Linear Algebra Appl., 17 (2010), pp. 453–472, doi:10.1002/nla.689.