ARTICLE TYPE

Parallel Performance of Algebraic Multigrid Domain Decomposition (AMG-DD)

Wayne Mitchell* | Robert Strzodka1 | Robert Falgout2 | Stephen McCormick3

1ZITI, Universität Heidelberg, Baden-Württemberg, Germany
2CASC, Lawrence Livermore National Lab, California, USA
3Dept. Applied Math, University of Colorado, Boulder, Colorado, USA

Correspondence
*Wayne Mitchell, ZITI, Im Neuenheimer Feld 368, 5. OG, 69120 Heidelberg. Email: wayne.b.mitchell@gmail.com

Present Address
Present address

Abstract

Algebraic multigrid (AMG) is a widely used scalable solver and preconditioner for large-scale linear systems resulting from the discretization of a wide class of elliptic PDEs. While AMG has optimal computational complexity, the cost of communication has become a significant bottleneck that limits its scalability as processor counts continue to grow on modern machines. This paper examines the design, implementation, and parallel performance of a novel algorithm, Algebraic Multigrid Domain Decomposition (AMG-DD), designed specifically to limit communication. The goal of AMG-DD is to provide a low-communication alternative to standard AMG V-cycles by trading some additional computational overhead for a significant reduction in communication cost. Numerical results show that AMG-DD achieves superior accuracy per communication cost compared to AMG, indicating that it should provide significant speedups for computational environments where the cost of communication is high compared to the cost of computation, such as very large clusters or GPU accelerated systems.

KEYWORDS:
low-communication algorithms, algebraic multigrid, parallel performance

1 INTRODUCTION

Algebraic multigrid (AMG) is a widely used scalable solver and preconditioner for large-scale linear systems resulting from the discretization of a wide class of elliptic PDEs. AMG scales well to very large problem sizes due to its optimal computational complexity: application of an AMG cycle has $O(N)$ computational cost, where $N$ is the size of the linear system to be solved. The cost of communication, however, has become a significant bottleneck that limits the scalability of AMG as processor counts continue to grow on modern machines. In fact, it is well documented that for large processor counts, an AMG V-cycle may spend more time on coarse levels of the hierarchy than the finest level. This difficulty is due to increased communication costs on the coarse levels even though there is exponentially less computational work there compared to the finest level. This result is in direct conflict with one of the most general guiding principles of multigrid methods: that the coarse grids should be inexpensive. Thus, in order to maintain the efficiency of AMG on modern supercomputers with ever increasing parallelism, work must be done to ameliorate its communication costs. This paper continues development of algebraic multigrid domain decomposition (AMG-DD), a novel algorithm built on top of AMG and designed with the express purpose of achieving similar or better convergence with significantly reduced communication at the cost of some redundant storage and computation.

There are several approaches available when considering the reduction of communication costs in AMG that may be divided into three broad categories that may each be employed in conjunction with one another: optimization and algorithm development
for parallel implementation; modification of the AMG algorithm itself through various changes to the AMG hierarchy; and use of entirely different cycling structures or algorithms based on the AMG hierarchy such as AMG-DD.

The first category includes changes to the parallel implementation that avoid any changes to the underlying AMG algorithm itself and thus exactly preserve convergence and other algorithmic properties. In \textsuperscript{6} for example, the authors develop an assumed partition algorithm for efficiently determining interprocessor communication patterns when global partitioning information is unavailable. The applicability of this algorithm is quite general, but, in particular, it may be used to speed up the AMG setup phase where communication patterns for coarse-grid matrices must be established. In \textsuperscript{2}, the authors develop node-aware communication patterns for general sparse matrix-vector multiplication that may be applied to speed up the solve phase of AMG. These approaches to communication reduction are the least invasive in that there is no change to the AMG algorithm itself, so speedup may be achieved with no fear of degraded convergence.

A more invasive category of approaches involves changing the AMG hierarchy itself. The cost of communication is generally most problematic on coarse grids in the middle of the AMG hierarchy. On the finest grids, communication patterns are typically regular and fairly sparse (if the problem comes from, for example, the discretization of a PDE that is then decomposed among processors), and the amount of computational work is typically very large compared to the required communication. On the coarsest grids, on the other hand, the total number of degrees of freedom can be significantly less than the number of processors, meaning that many processors are inactive and the total amount of communication is small. That leaves the mid-range of the hierarchy, where increasing complexity of the coarse-grid operators leads to large communication stencils and, thus, large communication costs that dominate the relatively small computational cost on these levels. On systems where communication is a dominant cost, AMG may actually spend more time on these mid-range coarse levels than the fine grid (see, for example, numerical results from \textsuperscript{5}), contradicting a basic heuristic of multigrid methods that the coarse grids should be cheap relative to the fine grid. Thus, efforts to ameliorate communication cost through modification of the AMG hierarchy have focused on sparsification of the coarse-grid operators. Modified coarsening and interpolation schemes are developed in \textsuperscript{10,11} that are designed to produce sparse coarse-grid operators with correspondingly sparser communication patterns. In \textsuperscript{12}, aggressive coarsening is used in conjunction with improved long-distance interpolation in order to both sparsify the coarse grids and reduce the total number of grid levels. More direct approaches to sparsification through elimination of matrix entries resulting in non-Galerkin coarse-grid operators are explored in \textsuperscript{6,14} In each of these cases, the sparsified hierarchies yield faster AMG V-cycles at the cost of a degraded V-cycle convergence factor. Thus, more (but faster) V-cycles may be required to solve a given problem to a desired tolerance. This tradeoff means that the benefits of using these modified AMG hierarchies may be very problem dependent: significant speedup is observed for some problems, whereas convergence is destroyed for others.

Finally, there has been exploration of altogether different algorithms based on AMG (or multigrid methods in general) that aim to replace typical multigrid cycling. A main underlying principle here is the attempt to add parallelism and/or reduce communication by decoupling pieces of the multigrid cycle. Additive multigrid methods, for example, add extra concurrency compared to traditional multiplicative methods by decoupling the levels of the multigrid hierarchy. Traditional additive multigrid methods suffer from worse convergence compared to their multiplicative counterparts, though more advanced variants on additive multigrid that recover good convergence exist, such as the asynchronous fast adaptive composite grid methods described in \textsuperscript{14} or the additive multigrid methods using modified smoothed interpolation operators described in \textsuperscript{15}. Algebraic multigrid domain decomposition (AMG-DD) retains a multiplicative multigrid cycling approach, but it decouples the spatial decomposition of the problem among processors, employing independent cycling on each processor to reduce communication by storing fully overlapping composite grid structures. Similar algorithms employing overlapping composite grids have been explored in the context of geometric multilevel methods and adaptive mesh refinement. The parallel meshing algorithm from \textsuperscript{19} uses the idea of storing additional overlapping information on each processor in order to determine adaptively refined meshes with reduced communication, but relies on subsequent conventional domain decomposition or multigrid iterations to perform the solve. In \textsuperscript{17,13}, a multigrid solver based on full domain partitions and a hierarchical basis is developed that requires communication only twice per cycle. The range-decomposition algorithm developed in \textsuperscript{19} enables fully independent nested-iteration solves with adaptive mesh refinement on each processor. In contrast to these algorithms, AMG-DD operates in a purely algebraic context, providing a linear solver that requires only the fine-grid matrix equation with no need for additional discretization information or infrastructure.

AMG-DD was originally presented in \textsuperscript{7} in which the authors provide some basic motivating theory, performance models, and test results using a model serial code for small problem sizes. This paper continues the development of AMG-DD through further modification and design of the algorithm, implementation of a scalable parallel code, and empirical study of algorithm behavior and parallel performance. Important algorithmic developments presented here include an efficient algebraic fast adaptive
composite (AlgFAC) cycle with minimal storage and communication requirements and a new hybrid variant of AMG-DD that can reduce the computational overhead of the algorithm while maintaining most of the communication reduction. In addition, the communication algorithm proposed in Section 3 is implemented and modified to avoid the redundant communication present in the original algorithm. Section 4 gives a brief review of AMG and is followed by a description of AMG-DD in Section 5. Section 6 explains the newly developed AlgFAC cycle used as the primary computational routine of AMG-DD, and Section 7 gives further detail on the primary communication routine of AMG-DD. Finally, Section 8 studies optimal parameter choices for AMG-DD, proposes the hybrid variant of the algorithm, and studies the scaling behavior and potential for speedup for AMG-DD.

2 REVIEW OF AMG

Algebraic multigrid (AMG) generates a multigrid hierarchy and then solves a given linear system, \( Au = f \), using only the information given in the matrix \( A \). The given linear system comprises the finest level of the hierarchy, level \( l = 0 \), so denote \( A = A_0 \), \( u = u_0 \), and \( f = f_0 \), and let \( N = N_0 \) be the number of degrees of freedom in this system. Generating a set of coarse-grid degrees of freedom may be done either by splitting the fine-grid degrees of freedom into disjoint sets of fine points (\( F \)-points) and coarse points that are repeated on the coarse grid (\( C \)-points), as done in “classical” AMG, or by grouping several degrees of freedom and treating them as a single point on the coarse grid, as done in “smoothed aggregation” AMG. This paper treats only classical AMG, although nothing prevents AMG-DD from being applied to smoothed aggregation. In classical AMG, the splitting of the degrees of freedom into \( C \)- and \( F \)-points is typically accomplished via a coloring algorithm that seeks to generate a maximal subset of \( C \)-points such that \( C \)-points do not strongly influence each other but do strongly influence the remaining \( F \)-points, where “strong influence” is determined by the relative size of entries in \( A_0 \). Given such a splitting with \( N_1 \) \( C \)-points, an \( N_0 \times N_1 \) interpolation operator, \( P_0 \), is constructed based on connections in the matrix, \( A_0 \). Typically, an interpolated value at a \( C \)-point is simply injected from the coarse grid, while a value at an \( F \)-point is obtained by applying interpolation weights (determined based on connections in \( A_0 \)) to values at \( C \)-points in the \( F \)-point’s interpolatory set. Classically, the interpolatory set contains \( C \)-points within distance 1 through the graph of \( A_0 \), but interpolation operators based on longer-distance interpolatory sets are also now commonly used. An \( N_1 \times N_0 \) restriction operator, \( R \), may also be generated separately, but often is set as \( R_0 = P_0^T \). Once the interpolation and restriction operators are defined, the coarse-grid operator is then formed by the triple matrix product, \( A_1 = R_0 A_0 P_0 \). This process may then be repeated recursively on level \( l = 1, 2, ..., L \), splitting the level \( l \) degrees of freedom into \( C \)- and \( F \)-points, defining \( P_l \) and \( R_l \), and then forming \( A_{l+1} = R_l A_l P_l \), until a sufficiently coarse level, \( L \), on which \( A_L \) is small enough to be easily handled by relaxation or a direct method such as Gaussian elimination. Construction of the interpolation, restriction, and coarse-grid operators comprises the setup phase of AMG. The solve phase then involves cycling on the hierarchy of grids by performing some relaxation on a grid level (usually some variation on an easily applied, pointwise iterative method like Jacobi or Gauss-Seidel), then either restricting a residual equation or interpolating a correction to move up or down through the hierarchy. Algorithm 1 shows pseudocode and lists communications for an AMG \( V(v_1, v_2) \)-cycle, the most commonly used cycle structure, which is simply a downsweep performing \( v_1 \) relaxations on each level followed by an upsweep performing \( v_2 \) relaxations on each level. For simple relaxation methods like Jacobi and Gauss-Seidel, the cost of relaxation on level \( l \) is equivalent to performing a matrix-vector multiplication (mat-vec) with \( A_l \). Thus, the cost associated with an AMG \( V(v_1, v_2) \)-cycle is the cost of \((1 + v_1 + v_2)\) mat-vecs with \( A_l \) \((v_1 + v_2)\) relaxations and a residual calculation plus a mat-vec with \( P_l \) for interpolation and a mat-vec with \( R_l \) for restriction on each level, \( l \). When running an AMG cycle on a parallel system, degrees of freedom on each level are partitioned among processors. In order to perform a mat-vec, each processor must then communicate with nearest processor neighbors to exchange “halo” data - vector data at points that are distance 1 (through the graph of the matrix) away from the owned, on-processor points. As stated in the introduction, on the finest levels of the hierarchy, this communication cost is usually dominated by computational cost, but growing operator complexities on coarser levels can lead to a sharp increase in the number of neighbors each processor must communicate with in order to perform a mat-vec. Thus, these coarse-level communication costs may dominate the total cost of the entire cycle.

3 AMG-DD ALGORITHM DESCRIPTION

Algebraic multigrid domain decomposition (AMG-DD) seeks to reduce the communication cost of the AMG solve phase by constructing global composite grids on each processor based on the underlying AMG hierarchy and that processor’s owned partition of the degrees of freedom, referred to here as that processor’s subdomain. These composite grids are generated by...
moving through the graphs of the operators, $A_l$, on each level of the AMG hierarchy, as illustrated in Figure 1. First, choose some padding value for each level, denoted $\eta_l$. Then, beginning on the finest level, a processor’s composite grid includes its subdomain (the black points in the left pane of Figure 1) plus additional points within distance $\eta_0$ through the graph of $A_0$ (the white points in the left pane of Figure 1). All $C$-points from the resulting set are then included in the composite grid on level 1 plus all points within distance $\eta_1$ through the graph of $A_1$ (the white points in the middle pane of Figure 1). This process of coarsening and then extending through the graph of $A_l$ by distance $\eta_l$ is then repeated for all of the coarse grids. The coarsest-level padding, $\eta_L$, should be chosen large enough that each processor stores the entire coarsest grid (depending on the paddings chosen and the size of the coarsest grids, this may be true for a few of the coarsest grids). Thus, each processor stores a composite grid that has finest-level information over its subdomain plus additional information at composite-grid points that extend further away over the global domain on coarser levels. Processors may independently perform cycling on these composite grids through an algebraic fast adaptive composite (AlgFAC) cycling routine (described in detail in Section 4).

Algorithm 2 shows pseudocode and lists communications for an AMG-DD cycle. The cycle provides a correction to the residual equation, $A_0 \delta u_0 = f_0 - A_0 u_0$, for some current fine-grid iterate, $u_0$. The first step is to calculate a current residual and restrict that residual to all levels of the hierarchy (using regular, distributed mat-vecs). Then a more complicated algorithm (described in detail in Section 5) is used to communicate residuals such that each processor obtains current residuals at all of their composite grid points (note that before this step, processors only have access to current residuals in their subdomain). These residuals are then used as the right-hand sides for a local solve via AlgFAC cycles on the composite grids in which each processor, $p$, obtains its own correction term, $\delta u_0^p$. The correction to the global solution is applied locally on each processor, that is, each processor updates $u_0 \leftarrow u_0 + \delta u_0^p$ over its fine-grid subdomain, $\Omega_0^p$, using fine-grid values from its locally calculated composite-grid solution. Since the composite grids stored on each processor cover the global domain at some coarse level, the local solves performed on each processor may be thought of as solving a composite representation of the global problem. This composite representation is (hopefully) accurate to the global problem in the processor’s subdomain, where it has finest-level information, and so the global correction produced by AMG-DD is patched together from these processor subdomains.

Whereas AMG V-cycles require communication for each operation on each level of the hierarchy, AMG-DD requires communication only to calculate, restrict, and communicate the residual values between local AlgFAC solves. Algorithms 1 (AMG) and 2 (AMG-DD) allow a high level comparison of the different communication requirements for each algorithm. The reduction in communication provided by AMG-DD comes with the added cost of redundant composite-grid storage and computation: any additional points in a processor’s composite grid that were not in that processor’s subdomain represent additional storage and computation cost. Due to the graded nature of the composite grids, this additional overhead may be kept to a manageable level (as studied in detail in Section 6).

4 ALGEBRAIC FAST ADAPTIVE COMPOSITE CYCLING

In the proof-of-concept implementation of AMG-DD used in [2], the local composite-grid problems on each processor were solved by explicitly constructing composite-grid matrix operators and solving the resulting matrix equations with AMG. The additional
overhead of forming the local composite-grid matrices (and subsequent associated AMG hierarchies) may be avoided, however, by cycling directly on the pieces of the original AMG hierarchy that make up the composite grid via an algebraic variant of the fast adaptive composite (FAC) method. FAC cycling has previously been applied to geometric multigrid hierarchies, and a detailed description of this can be found in [22]. When attempting to apply the ideas behind FAC to an algebraic multigrid hierarchy, extra care must be taken in order to ensure proper calculation of values at the borders of each grid level. In the geometric case, these borders correspond to changes in the level of mesh refinement, and calculations there are based on information from the underlying discretization and mesh. In the algebraic setting, such geometric information is not available, and the AMG coarse grids may be quite irregular with arbitrary connectivity. Thus, this section discusses the necessary generalizations of previous geometric techniques for handling the borders of composite grids in order to achieve algebraic fast adaptive composite cycling (AlgFAC) on general AMG hierarchies.

Classically, FAC cycles are used for geometric problems where high resolution of the grid is required only over some subset of the domain. A first step towards a more efficient algorithm for dealing with such a problem is to simply suppress relaxation where it is not necessary (i.e., outside of the region requiring high resolution). This saves the computational effort of relaxing where coarser representation of the solution is sufficient. The storage and computational cost can then be further reduced by removing the fine-grid representation of the solution where it is not needed altogether, resulting in a grid with varying levels of refinement similar to the AMG-DD composite grids. The resulting representation of the solution and multigrid cycling on this composite grid should be exactly equivalent to the algorithm that retains a fine-grid representation everywhere and simply suppresses relaxation. This will be the guiding principle used to develop AlgFAC cycling on algebraic grids: the result obtained via the AlgFAC cycle should be exactly the same as that obtained by cycling on the entire grid while suppressing relaxation at certain points. This equivalence is shown in the following subsections by breaking the multigrid cycle down into its component parts: relaxation, interpolation, and restriction.

### 4.1 Relaxation

Denote the set of degrees of freedom that comprise grid level \( l \) of the global AMG hierarchy as \( \Omega_l \), and denote the composite-grid points on level \( l \) as \( D_l \subseteq \Omega_l \). The points in \( D_l \) undergo relaxation as part of the AlgFAC cycle and are referred to as “real” points, whereas the remaining points on the level, referred to as “ghost” points, are not relaxed. In designing the AlgFAC cycling algorithm, the goal is to reproduce the action of a global AMG cycle (operating on the full \( \Omega_l \) grids) where only points in \( D_l \) are relaxed and as few ghost points as possible are actually stored or involved in any computation.

Applying simple, pointwise algebraic relaxation schemes like Jacobi or Gauss-Seidel to a real point, \( i \in D_l \), requires solution values at all points, \( j \), connected through the row stencil of \( A_l \), i.e., a nonzero matrix entry, \( a_{ij} \in A_l \), connects points \( i \) and \( j \). Denote the extension of \( D_l \) through the row stencil of \( A_l \) as

\[
\tilde{D}_l = D_l \cup \{ j : a_{ij} \in A_l, \ a_{ij} \neq 0, \ i \in D_l \}.
\]

Thus, in order to perform correct relaxation on each grid level, it is necessary to store solution values at all points in \( \tilde{D}_l \).

### 4.2 Interpolation

Since relaxation requires correct solution values to be maintained at points in \( \tilde{D}_l \), it is necessary to interpolate coarse-grid corrections to these points. Denote the coarse-grid interpolatory set for \( \tilde{D}_l \) as \( P_l(\tilde{D}_l) \), i.e., points connected through the row stencil of \( P_l \). For typical AMG hierarchies, most points in \( P_l(\tilde{D}_l) \) are in \( D_{l+1} \) if \( \eta_{l+1} \geq 1 \), so their contributions to interpolation will be correctly accounted for. A point that is in \( P_l(\tilde{D}_l) \) but not in \( D_{l+1} \) is, by assumption, not a real degree of freedom and is thus initialized to zero and not relaxed on level \( l + 1 \). The solution value at such a point will remain zero (and thus does not need to be accounted for in interpolation) so long as it also receives zero coarse-grid correction. That is, the set formed by recursively moving through the interpolatory sets of this point all the way to the coarsest level should not include any real degrees of freedom on any level. This is true for many hierarchies in practice including all test cases examined in this paper, but is not necessarily guaranteed in general. Assuming this property, all points in \( P_l(\tilde{D}_l) \) are either accounted for on the composite grid on the next coarse level or have zero contribution (and may thus be simply omitted), and interpolation may proceed as normal by applying \( P_l \) to correction values on composite-grid level \( l + 1 \).
4.3 | Restriction

As described above, the implementation for relaxation and interpolation over the composite grid is identical to that for a global multigrid cycle. Relaxation is simply not done for points outside \( D_i \), and interpolation is not done for points outside \( \bar{D}_i \). A similarly straightforward implementation of restriction presents problems, however. Since the coarse composite grids expand to cover the entire computational domain, storing the restriction stencils for all points in the composite grid is a recursive way such that it is always possible to restrict from fine to coarse grids in the standard way results in storing the entire global hierarchy. Thus, restriction must be somehow rewritten for composite-grid points whose restriction stencils do not reside in the composite grid.

A key realization is that actual intergrid restriction is not necessary when finer-grid relaxation does not impact the residual. For the discussion here, assume the use of a V-cycle, though all ideas are easily extended to arbitrary cycle types. Consider restriction to composite-grid points whose restriction stencils include only points that are greater than distance 1 away from real points on all finer grid levels. Since corrections are initialized to zero on the downsweep and no relaxation occurs to change those values outside the real points, residual calculation simply yields the currently stored right-hand side for all coarse grids. Thus, setting the right-hand side, \( f_{i+1,j} \), on level \( l+1 \) for V-cycle iteration \( i \) by restricting the residual from level \( l \) to level \( l+1 \) may be rewritten as a restriction of the finest-grid residual after the previous V-cycle iteration as follows:

\[
f_{i+1,i} = R_i f_{i,i}
\]

\[
f_{i+1,i} = R_i R_{i-1} f_{i-1,i}
\]

\[
\vdots
\]

\[
f_{i+1,i} = R_i \ldots R_0 (f_0 - A_0 u_{0,i-1}).
\]

Note that the fine-grid right-hand side, \( f_{0,j} = f_0 \), remains constant from iteration to iteration. Here, \( u_{0,i-1} \) is the fine-grid solution at the end of the previous V-cycle iteration \( i-1 \). Again considering only composite-grid points on levels \( 0, \ldots, l \) that are distance 1 away from real points means that \( u_{0,i-1} \) is unaffected by relaxation on levels \( 0, \ldots, l \) and may thus be expressed as the previous solution, \( u_{0,i-2} \), plus the interpolated corrections from coarser levels generated on iteration \( i-1 \) all the way down to level \( l \) as follows:

\[
f_{i+1,i} = R_i \ldots R_0 (f_0 - A_0 u_{0,i-1})
\]

\[
f_{i+1,i} = R_i \ldots R_0 (f_0 - A_0 (u_{0,i-2} + P_0 u_{1,i-1}))
\]

\[
\vdots
\]

\[
f_{i+1,i} = R_i \ldots R_0 (f_0 - A_0 (u_{0,i-2} + P_0 \ldots P_l u_{l+1,i-1})).
\]

Now, rearranging and invoking the definition of the coarse-grid operators as \( A_{i+1} = R_i A_i P_i \) shows that restricting the residual may, in fact, be rewritten as a residual recalculation on the coarse grid using the previous solution/correction value and right-hand side as follows:

\[
f_{i+1,i} = R_i \ldots R_0 (f_0 - A_0 u_{0,i-2}) - R_i \ldots R_0 A_0 P_0 \ldots P_l u_{l+1,i-1}
\]

\[
f_{i+1,i} = f_{i+1,i-1} - A_{i+1} u_{i+1,i-1}.
\]

Thus, away from the real points where relaxation occurs, restricting the residual can be replaced with a residual recalculation on the coarse grid, removing the necessity for storing information on the grids above. Meanwhile, restriction may occur as usual for points whose restriction stencils do include the influence of finer-grid relaxation. Note that this implementation still requires some additional layers of ghost points to be stored and computed on. Specifically, ghost points within distance 2 plus twice the interpolation distance are required, i.e., distance 4 or 6 neighbors when distance 1 or 2 interpolation is used, respectively.

While the above method for implementing restriction is feasible, it still demands somewhat large overhead by requiring storage and computation at ghost points within distance 4 to 6 of the real points. Furthermore, initial right-hand side data must be communicated at all of these ghost points between each outer AMG-DD iteration, so the ghost points represent a significant overhead not only in terms of storage and computation but also in terms of communication cost. Restriction may actually be correctly performed without these additional ghost points, however, if the idea of coarse-grid residual recalculation is applied to all points, even those influenced by finer-grid relaxation.
Rewriting restriction in this way requires some additional notation. Define \( v_{l,i} \) as the value stored in the solution/correction vector just before restriction occurs from level \( l \) to level \( l+1 \) on iteration \( i \). Define \( u_{l,i} \) as the value stored in the solution/correction vector just before interpolation occurs from level \( l \) to level \( l-1 \) on iteration \( i \). Respectively define \( \delta^1_{l,i} \) and \( \delta^2_{l,i} \) as the effects of pre and post relaxation on level \( l \), iteration \( i \). The following equations express the relationship between each of these terms during a V-cycle:

\[
v_{l,i} = \begin{cases} 
    u_{0,i-1} + \delta^1_{0,i}, & l = 0 \\
    \delta^1_{l,i}, & \text{else} 
\end{cases}
\]  
(13)

\[
u_{l,i} = v_{l,i} + P_l u_{l+1,i} + \delta^2_{l,i}.
\]  
(14)

The right-hand sides, \( f_{l,i} \), on level \( l \), iteration \( i \) are then written as follows:

\[
f_{l,i} = \begin{cases} 
    f_0, & l = 0 \\
    R_{l-1}(f_{l-1} - A_{l-1}v_{l-1,i}), & \text{else.} 
\end{cases}
\]  
(15)

The goal is to reexpress restriction in a form that requires as little computation and storage as possible outside of the real degrees of freedom. Begin by rewriting restriction to the first coarse grid as follows:

\[
f_{1,i} = R_0(f_{0,i} - A_0v_{0,i}) = R_0(f_0 - A_0(u_{0,i-1} + \delta^1_{0,i})) = R_0(f_0 - A_0(u_{0,i-1} + P_0u_{1,i-1} + \delta^2_{0,i-1} + \delta^1_{0,i})) = R_0(f_{0,i-1} - A_0v_{0,i-1}) - R_0A_0P_0u_{1,i-1} - R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}) = f_{1,i-1} - A_1u_{1,i-1} - R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}).
\]  
(16)

Thus, restricting the residual to the first coarse grid may be written as a recalculcation of a residual on the coarse grid using the currently stored right-hand side and correction vector, \( f_{1,i-1} - A_1u_{1,i-1} \), plus an update term that restricts the effect of fine-grid relaxation, \(-R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i})\).

It is possible to write restriction to the second coarse grid in a similar way as follows, recursively using the expression above for \( f_{1,i} \) in order to rewrite \( f_{2,i} \):

\[
f_{2,i} = R_1(f_{1,i} - A_1v_{1,i}) = R_1(f_{1,i-1} - A_1u_{1,i-1} - R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}) - A_1\delta^1_{1,i}) = R_1(f_{1,i-1} - A_1u_{1,i-1} + P_1u_{2,i-1} + \delta^2_{1,i-1}) - R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}) - A_1\delta^1_{1,i}) = R_1(f_{1,i-1} - A_1v_{1,i-1}) - R_1A_1P_1u_{2,i-1} - R_1A_1(\delta^2_{1,i-1} + \delta^1_{1,i}) - R_1R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}) = f_{2,i-1} - A_2u_{2,i-1} - R_1A_1(\delta^2_{1,i-1} + \delta^1_{1,i}) - R_1R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}).
\]  
(17)

Thus, restriction to any level of the multigrid hierarchy may be rewritten as follows as a residual recalculation on the coarse level plus some restricted updates due to finer-level relaxation:

\[
f_{l,i} = f_{l,i-1} - A_lu_{l,i-1} - R_{l-1}A_{l-1}(\delta^2_{l-1,i-1} + \delta^1_{l-1,i-1}) - R_{l-1}R_{l-2}A_{l-2}(\delta^2_{l-2,i-1} + \delta^1_{l-2,i-1}) - R_{l-1}R_{l-2}...R_0A_0(\delta^2_{0,i-1} + \delta^1_{0,i}).
\]  
(18)

The restricted update terms above may be accumulated in a recursive way during the downsweep of the AlgFAC cycle such that obtaining the right-hand side on level \( l \) involves only a single application of \( R_l \) to a vector on level \( l-1 \) that has collected update terms from all finer levels.

### 4.4 AlgFAC cycle

With restriction rewritten as in the previous subsection, it is possible to write the AlgFAC V-cycle as a whole as shown in Algorithm 3. Auxiliary vectors, \( t_l \) and \( s_l \), are introduced in order to accumulate and restrict the effects of relaxation.
For other cycle types, such as W or F cycles, a similar description is possible. The update vector, \( t_i \), accumulates changes due to relaxation, \( s_i \), accumulates updates from finer grids, and these updates are combined and passed to coarser grids during restriction, after which they are reset to zero. Rewriting things in this way requires only a single layer of ghost points and, furthermore, does not require right-hand side information there. Right-hand side information is needed at real points in order to perform relaxation and coarse-grid residual recalculation but is no longer required in the restriction stencils of real points. The update term, \( t_i \), is nonzero only at real points, so \( A_i t_i \) is nonzero only in \( D_i \). Some assumptions on the connectivity of the AMG hierarchy are required to ensure the stored composite grids correctly account for contributions to \( s_i \) throughout the cycle. In fact, this requirement is very similar to the requirement for correct interpolation discussed in Section 4.2 and is equivalent to that requirement if \( R_i = P_i^T \). Denote the restriction transpose stencil for \( D_i \) as \( R_i^T(D_i) \), i.e., points on level \( l + 1 \) that are connected through the columns of \( R \) to points in \( D_i \). Most points in \( R_i^T(D_i) \) will be naturally contained in \( D_{i+1} \), and contributions to \( s_{i+1} \) will be accounted for at these points. If a point in \( R_i^T(D_i) \) is not in \( D_{i+1} \), then the value of \( s_i \) is not needed here since no relaxation occurs and, subsequently, no right-hand side is needed. Moreover, the value of \( s_i \) at this point need not be restricted to coarser levels, provided that moving recursively further through the restriction transpose stencils all the way to the coarsest level includes no real points. If \( R_i = P_i^T \), then the restriction transpose stencils are exactly the interpolation stencils (or interpolatory sets), and this is the same requirement discussed in Section 4.2.

Compared to a standard AMG V-cycle, the AlgFAC V-cycle shown in Algorithm 3 contains a few extra operations, but the extra computational cost of these operations is not too significant. Simple operations on each point that do not involve multiplication by a matrix (e.g., accumulation of the updates due to relaxation, or resetting vectors to zero) have small cost relative to mat-vecs. Most of the mat-vecs in Algorithm 3 also occur in a standard AMG V-cycle (i.e., relaxation, interpolation, and restriction) with the same cost. The only additional mat-vec required in Algorithm 3 is the coarse-grid residual recalculation, \( f_{i+1} \leftarrow f_{i+1} - A_{i+1} u_{i+1} \). This additional mat-vec need not occur on the initial downswep, however, since \( u_{i+1} \) is initialized to zero, and it never happens on the finest grid, meaning that this is a relatively small extra computational expense.

5 | RESIDUAL COMMUNICATION ALGORITHM

Apart from the local AlgFAC solves, the other main piece of the AMG-DD algorithm that requires special attention is the residual communication algorithm by which processors receive right-hand side values at composite-grid points outside their subdomains. Denote processor \( p \)'s subdomain on level \( l \) as \( \Omega_p^l \) and its composite grid on level \( l \) as \( D_p^l \). Then, during the residual calculation and restriction steps of Algorithm 2, a given processor obtains updated residuals at points in \( \Omega_p^l \), and then these updated values must be distributed to other processors that use these points in their composite grids, that is, processors \( q \) such that \( D_q^l \cap \Omega_p^l \neq \emptyset \). This is a non-trivial process, since the processor that owns the updated information does not have global information about which other processors’ composite grids overlap its subdomain (and thus need to receive information from this processor). In addition, even if this information were available, a direct broadcast from the owning processor to all other processors that must receive information from that processor would result in inefficient communication with far too many messages being sent. Some kind of intelligent accumulation and distribution of information is needed, analogous to performing an efficient Allgather collective communication as opposed to naive point-to-point communication.

The AMG-DD residual communication algorithm was originally proposed (but not implemented) in [2], and this paper provides empirical study of a parallel implementation of this algorithm in practice. The algorithm functions by recursively building up composite grids starting from the coarsest level of the AMG hierarchy and moving to the finest level. On each level, \( l \), processors communicate with distance \( \eta_l \) processor neighbors, where \( \eta_l \) is the padding on level \( l \) and distance is measured through the graph of \( A_l \). It is assumed that distance 1 processor neighbors are known, since this information is required to perform mat-vecs with \( A_l \) (and if this information is not available, algorithms exist to efficiently obtain it). From this starting point, arbitrary-distance neighbors may be found by recursively communicating with processor neighbors at longer and longer distances (growing the set of processor neighbors with each communication).

Once distance \( \eta_l \) processor neighbors are known, the AMG-DD residual communication algorithm proceeds as follows. Beginning on the coarsest level, \( L \), each processor, \( p \), with a non-empty subdomain (note that if the coarsest level has fewer points than there are processors, some processors will remain inactive here) communicates with each of its distance \( \eta_l \) processor neighbors, \( q \), sending residuals at points in the set \( \Psi = \{ i \in \Omega_p^L : \text{dist}(i, \Omega_q^L) \leq \eta_L \} \), to processor \( q \), where \( \text{dist}(\cdot, \cdot) \) measures distance through the graph of \( A_L \). This routine then repeats on the next finer level, \( L - 1 \), but in addition to points in the set \( \Psi = \{ i \in \Omega_p^{L-1} : \text{dist}(i, \Omega_q^{L-1}) \leq \eta_{L-1} \} \), each processor, \( p \), sends a composite-grid structure based on \( \Psi \) to processor \( q \). More
precisely, take the $C$-points of $\Psi$ on the next coarse level, add points within the padding distance, and call the resulting set $\Psi_c$. Then residuals for all points in $\Psi \cup \Psi_c$ are communicated. After this communication, each processor has updated residuals in a composite grid based on its subdomain from level $L - 1$ down. The recursion is now apparent. The same process is repeated moving up to the finest level, with each processor sending distance $\eta_l$ points, $\Psi$, on level $l$ plus a composite-grid structure, $\Psi_c$, containing points on coarser levels. By recursive argument, after communication on each level, $l$, all processors have correctly updated residual values in a composite grid based on their subdomain on level $l$. A more detailed proof of the correctness of this algorithm is given in\cite{numres}. Pseudocode for this residual communication algorithm is given in Algorithm\cite{rescom}.

Note that the communication stencils used for the residual communication algorithm are based on the padding, $\eta_l$, used on each level. In the case where $\eta_l = 1 \forall l$, the communication stencils are exactly those used for mat-vects on each level with $A_l$ and the $\Psi$ sets exactly comprise the mat-vec halo points (message sizes will be larger due to the need to send the $\Psi_c$ sets as well). Using larger $\eta_l$ may result in expanded communication stencils and subsequently more overall messages as well as a larger total volume of data being communicated, as shown in more detail in Section\cite{comcom}. On finer levels, reaching further through the graph of $A_l$ may not jump to more distant processors, but on coarse levels this is likely: on a coarse level where each processor has single degree of freedom, for example, any increase to $\eta_l$ will expand the set of processor neighbors, potentially leading to a significant increase in the total number of messages sent. This highlights the importance of rewriting restriction in the AlgFAC cycle as described in Section\cite{numres}. The more straightforward approach to restriction demands right-hand side information at multiple layers of ghost points (typically 4 to 6 layers), which in turn demands that the residual communication algorithm communicate information at distance $\eta_l$ plus the number of ghost layers, resulting in significantly more messages and total volume sent. By rewriting restriction, however, right-hand side information is only needed at real degrees of freedom, enabling the residual communication algorithm to communicate information only at the distance of the padding as described above.

Note also that the residual communication algorithm does not require processors to have global information about off-processor composite grids (only distance $\eta_l$ processor neighbors and corresponding distance $\eta_l$ points are needed), nor does it even require a given processor to know the size and shape of its own composite grid a priori. Thus, this algorithm may also be used to setup the composite grids in an efficient manner. The top down process of expanding by the padding, coarsening, and repeating is a convenient way to define composite grids, but is not an efficient way to construct them in practice in a distributed setting. Thus, the residual communication algorithm may be employed to construct the composite grids simply by passing matrix information (rows of $A_l$, $P_l$, and $R_l$) instead of residual values. Note that this does require communicating information at distance $\eta_l + 1$ on each level in order to obtain the matrix information at the ghost points.

One important caveat about the residual communication algorithm is that it communicates a significant amount of redundant information if implemented as described above. This redundancy has two causes. First, processor $p$ may communicate information at some point on level $l$ to processor $q$ and then send the same information to the same destination again later if that point is included in a $\Psi_c$ composite-grid structure being sent to processor $q$ on some finer level. Second, it is possible that two processors $p$ and $r$ have accumulated updated information at some point and both processors then send information at this point to processor $q$. The first source of redundancy is easily rectified, since the redundant information is sent from the same processor: when constructing a $\Psi_c$ composite-grid structure to send, the sending processor may simply remove points previously sent to the same destination. The second source of redundancy is less trivial and requires a second round of communication during the setup phase, during which each processor sends back a list of redundantly received points to all processors it received from. This extra communication step need only happen once during the setup phase, however, and then all redundancy is avoided during the solve phase residual communications.

6 | NUMERICAL RESULTS

The model problems used throughout this section are variations on the diffusion problem,

\begin{equation}
-\nabla \cdot K \nabla u = 1, \quad \Omega \tag{31}
\end{equation}

\begin{equation}
u = 0, \quad \partial \Omega. \tag{32}
\end{equation}

The equation is discretized using linear $H^1$ conforming finite elements. Two versions of the problem are considered: 3D Poisson, where $K = I$ and $\Omega$ is the unit cube; and 2D rotated anisotropic diffusion, where $\Omega$ is the unit square, $K = Q^T D Q$, and $Q$ and
\(D\) are the rotation and scaling matrices respectively defined by

\[
Q = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix},
\]

with \(\theta = \pi/8\) and \(\epsilon = 0.001\). These problems are chosen because they are well-known to have significant growth in communication stencils on the coarse levels of classical AMG hierarchies, leading to significant communication costs.

All implementation of AMG-DD discussed in this paper is done in hypre on top of BoomerAMG. Parameter choices for the underlying AMG hierarchy as well as the decomposition of the problem among processors all influence the size of the composite grids and subsequently the storage and computation overhead and overall performance of AMG-DD. Since the composite grids are constructed based on movement through the graph of \(A_l\) on each level, the complexity of these coarse-grid operators has a direct impact on the size of the composite grids. Classical AMG and its initial extensions to the parallel setting generally yield higher complexity than current “best practices” for AMG [10,11]. Significantly larger composite grids result from Falgout coarsening and classical modified interpolation (classical AMG) compared to HMIS coarsening and extended + i interpolation (best practices AMG), which are designed to reduce coarse-grid complexity. The tests shown here use the settings for best practices.

The size of the composite grids relative to the size of the subdomain on a processor also depends on the partitioning of the problem. Assuming the partitioning of the degrees of freedom among processors corresponds to a spatial partitioning of the underlying problem geometry, the size of a processor’s composite grid is roughly dependent on the surface area of its subdomain. Thus, the overhead of the composite grids depends on the surface-area-to-volume ratio of the processor subdomains. The dimensionality of the underlying problem as well as the number of degrees of freedom per processor both impact the surface-area-to-volume ratio: higher dimensional problems and a smaller number of degrees of freedom per processor both result in larger surface-area-to-volume ratio and, consequently, larger overheads for AMG-DD, as shown throughout the results below.

### 6.1 AMG-DD parameter choice

AMG-DD introduces a few new important parameters to choose (on top of the many parameter choices involved in setting up the underlying AMG hierarchy for a given problem). One main question of interest, however, is how accurately each processor’s composite-grid solution represents the desired correction to the global solution. There are two sources of error here: that due to the approximation of the global fine grid with a composite grid; and that due to inexact solution of the composite problem. The two primary parameters that affect these sources of error are the padding chosen in constructing the composite grids and the number and type of AlgFAC cycles used to solve on the composite grid between each global residual recalculation. The use of larger padding grows the overall size of the composite grid on each processor, allowing for the composite problem to more accurately represent the global problem but also incurring the additional cost of communicating and computing on more composite-grid degrees of freedom. Performing additional AlgFAC cycles between each global residual recalculation should generate solutions on each processor that are more accurate to their corresponding composite problems at additional computational cost but at no additional communication cost.

Figure 2 shows convergence of the relative residual by iteration for AMG vs. AMG-DD with different levels of padding for each test problem with 256 processors and 12,000 and 18,000 degrees of freedom per processor on the fine grid for the 3D Poisson problem and the rotated anisotropic problem, respectively. Here, the same padding is chosen on each level, and the local composite problems on each processor are solved accurately (using 10 AlgFAC V-cycles) in order to isolate the effect of padding on accuracy of the composite problems to the global problem. As shown in Figure 2, larger paddings yield better AMG-DD convergence, as expected, but recall that larger paddings also incur larger costs both in terms of communication and computation.

Figure 3 shows the overhead incurred by the AMG-DD composite grids with different paddings compared to AMG by plotting the total number of nonzeros of \(A_l\) for all levels, \(l\), stored across all composite grids (i.e., accounting for points stored redundantly in many composite grids) normalized by the total number of nonzeros stored in AMG (i.e., with no redundant storage). The number of nonzeros in \(A_l\) serves here as a proxy for storage and computational costs of each method. As shown in Figure 3, increasing the padding dramatically increases the storage and computational cost of AMG-DD relative to AMG, with significantly worse overheads in 3D due to higher surface-area-to-volume ratios for each processor’s subdomain.

Communication costs can also grow significantly with increased padding, as shown in Figure 4. The maximum number of messages sent from a single processor is used as a measure of latency costs and the total volume of data communicated is used as a measure of bandwidth cost. The highest volume of data is communicated on the fine levels, and, for larger paddings, AMG-DD
communicates significantly more data due to the need to communicate deeper halos. The increase in the number of messages on coarse grids for AMG is observed here and mirrored by AMG-DD. Recall that the AMG-DD residual communication algorithm (which is the dominant communication cost) uses the same communication stencils as a mat-vec with \( A_l \) on each level in the padding 1 case, but these communication stencils expand with larger paddings. The growth in the number of messages for AMG-DD with higher padding may be alleviated in some cases by replacing coarse-level point-to-point communication with a collective Allgather of the global coarse grid (assuming a grid coarse enough that it is reasonable to redundantly store on all processors) or by grouping processors together on coarse levels, treating the group as a single rank in the residual communication algorithm, and doing some additional local collective communications. While these techniques can limit the growth in the number of messages for AMG-DD with higher padding, higher padding always incurs additional messages, and the volume of data communicated cannot be reduced.

For the test problems shown here, the convergence benefit does not justify the cost incurred by increased padding for AMG-DD. Figure 5 combines the convergence and cost results described above by plotting the residual reduction per cost, where the costs are measured by the total number of nonzeros in \( A_l \) for all \( l \) and all processors (computational cost), the maximum number of messages sent by one processor summed over each communication stage (latency cost), and the total volume of data communicated (bandwidth cost). As shown, AMG-DD with padding 1 achieves better residual convergence with less cost compared to higher paddings. Also, AMG-DD with padding 1 achieves significantly better accuracy per communication cost than AMG.
FIGURE 4 Maximum number of messages sent from a single processor (top) and total volume of communicated data (bottom) on each level for AMG vs. AMG-DD for different paddings for 3D Poisson (left) and rotated anisotropic diffusion (right).

The number of inner AlgFAC cycles per AMG-DD iteration determines how well the composite grid problems are solved on each problem. While additional AlgFAC cycles incur no additional communication cost, they do come at additional computational expense. Thus, it is desirable to optimize the number of AlgFAC cycles in terms of accuracy per computational cost, where accuracy is measured by outer AMG-DD convergence rather than AlgFAC convergence to the composite solution. Figure 6 shows residual convergence by iteration and by computational cost for AMG vs. AMG-DD with padding 1 and different numbers of AlgFAC V(1,1)-cycles. For the problems here, AMG-DD convergence changes very little beyond 4 AlgFAC cycles, indicating that the inner problem has sufficiently converged. More accurate composite solutions (obtained through more AlgFAC cycles) generally yield better AMG-DD convergence, but interestingly, for the rotated anisotropic diffusion problem, initial convergence is actually better with less accurate composite solves, while asymptotic convergence is better with more accurate composite solves. For both problems, however, the additional computational cost of multiple AlgFAC cycles outweighs any benefits to convergence. AMG-DD achieves the best efficiency in terms of accuracy per computational cost for a single AlgFAC cycle and is only slightly less computationally efficient than AMG in that case.

6.2 Hybrid AMG-DD

Up to this point, AMG-DD has been considered as an alternative cycling method to AMG V-cycles. It may be beneficial, however, to employ a hybrid method that uses regular AMG cycling on the finer levels and AMG-DD starting on a coarse level, $\tilde{L}$. As previously mentioned, it is common for the finest levels of an AMG hierarchy to be compute limited while the coarser levels are communication limited. This is due to the fact that the finer grids have simultaneously many more degrees of freedom and usually far smaller communication stencils than the coarse grids. Thus, the real need for communication reduction in an AMG cycle exists primarily (if not entirely) on the coarse levels. In a hybrid AMG-DD cycle, AMG-DD acts as a kind of coarse-grid solver for the AMG cycle, where the transition level, $\tilde{L}$, between AMG and AMG-DD is somewhere in the middle of the hierarchy. Algorithm 5 shows this hybrid AMG-DD idea applied to a a V-cycle, but this idea can be applied to other cycle structures as
FIGURE 5 Residual convergence by computational cost (top), latency cost (middle), and bandwidth cost (bottom) for AMG vs. AMG-DD for different paddings for 3D Poisson (left) and rotated anisotropic diffusion (right).

well, meaning that hybrid AMG-DD may enable more efficient versions of W or F cycles, where coarse-grid communication costs are even larger. Using AMG-DD in this way retains communication reduction on the coarse levels while also reducing the storage and computation overheads of the composite grids. As shown in the following weak scaling results, hybrid AMG-DD starting on a coarse level as opposed to level 0 still significantly reduces latency costs compared to AMG while achieving significantly less storage and computation overhead, especially in the 3D case and for a small number of degrees of freedom per processor, where fine-grid halos have significant size relative to the size of the subdomain.

6.3 Weak scaling results

With the above discussion on parameter choices and problem setup in mind as well as the possibility of hybrid AMG-DD, weak scaling results comparing AMG and AMG-DD with padding 1 and a single AlgFAC V(1,1) cycle for the composite solve
are presented below. These tests were run on the bwForCluster MLS&WiSC
and scale up to 4,096 cores for the largest runs. Two regimes were tested: small and large problem size per processor. Small problem sizes per processor are used in order to deemphasize the cost of computation and enter a communication-dominated regime on the bwForCluster. AMG-DD suffers from large computational overhead in this regime, however, due to large surface-area-to-volume ratios for processor subdomains, and so the hybrid AMG-DD scheme is employed in order to alleviate these large overheads. When large problem size per processor is used, the total time is dominated by computation on the finest grids and communication is no longer a significant cost on the bwForCluster. Thus, AMG-DD does not produce overall speedup in this regime: AMG-DD only reduces the time spent on communication, and it does so at the expense of additional computation. Larger problem size per processor does allow AMG-DD starting on level 0 to achieve acceptable overheads, however, due to much better surface-area-to-volume ratios for the subdomains. Hybrid AMG-DD results are also shown, and the starting level is chosen as the level in the AMG hierarchy where communication begins to dominate the cost of mat-vecs with $A_l$. AMG-DD and hybrid AMG-DD both provide convergence at significantly reduced communication cost at the expense of quite moderate computational overhead in this regime. Thus, for computing environments with relatively higher communication cost compared to computational cost, AMG-DD is expected to achieve significant speedup.

For the communication-dominated regime of small problem sizes per processor, roughly 650 and 1,900 degrees of freedom per processors are used for the 3D Poisson problem and the 2D rotated anisotropic problem, respectively. Figure 7 shows that this regime is, in fact, communication-dominated by plotting the time to perform a mat-vec with $A_l$ (the dominant cost on each level of the AMG hierarchy) on each level, $l$, as well as just the time to perform the communication required for the mat-vec. With such a small number of degrees of freedom per processor, AMG-DD must be started on a coarse level in order to avoid generating composite grids many times the size of the original AMG hierarchy. Thus, hybrid AMG-DD is used, starting on level 3 and 4 for the 3D Poisson and 2D rotated anisotropic problems respectively.

The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1134-1 FUGG.
FIGURE 7 Total time and communication time required to perform a mat-vec with $A_l$ on each level, $l$, of the AMG hierarchy with 4,096 processors in the communication-dominated regime for 3D Poisson (left) and rotated anisotropic diffusion (right).

FIGURE 8 Grid size overheads for hybrid AMG-DD in the communication-dominated regime starting on level 3 and 4, respectively, for 3D Poisson (left) and rotated anisotropic diffusion (right).

Figure 8 shows how the composite grid overhead grows with larger processor counts. By starting on a coarse enough level, excessively high overhead is avoided, but the growth of the overhead as the number of processors increases is significant due to the high surface-area-to-volume ratio of processor subdomains in this regime. As shown in Figure 9, the number of messages per iteration for hybrid AMG-DD remain significantly lower compared to AMG as processor counts increase, though the communication volume is nearly the same as for AMG due to the dominance of the fine-grid halos communicated by the AMG portion of hybrid AMG-DD. Finally, Figure 10 shows the total time for a single solve iteration for AMG vs. hybrid AMG-DD, and modest speedups are observed in this communication-dominated regime.

Having more degrees of freedom per processor results in a regime where computational cost on the finest grids is the dominant cost, as shown in Figure 11. Here, the 3D Poisson problem has roughly 42,000 degrees of freedom per processor and the rotated anisotropic problem has roughly 30,500. On some coarse level, communication clearly becomes the dominant cost, but that cost is not large in absolute terms compared to the time spent on the fine grids. For significantly larger processor counts or machines where communication is relatively much more expensive compared to computation (e.g. for GPU accelerated systems), the coarse-grid communication cost is expected to once again dominate the cost of fine-grid computation. Further study of the performance of AMG-DD on such communication-limited systems is a topic of future research.

Results for both AMG-DD starting on the finest level and hybrid AMG-DD starting on a coarse level are shown for the computation-dominated regime. Hybrid AMG-DD starts on level 2 for 3D Poisson and level 5 for 2D rotated anisotropic diffusion. Figure 12 shows much more agreeable overheads for AMG-DD (even starting on the finest level) for this regime compared to the results for few degrees of freedom per processor. Especially in the case of hybrid AMG-DD, the overhead is quite small and grows slowly with increasing number of processors. Figure 13 shows that both AMG-DD and hybrid AMG-DD maintain
FIGURE 9 Weak scaling of number of messages (top) and communication volume (bottom) for hybrid AMG-DD scaled by the costs for AMG in the communication-dominated regime with hybrid AMG-DD starting on level 3 and 4, respectively, for 3D Poisson (left) and rotated anisotropic diffusion (right).

FIGURE 10 Weak scaling of solve times for one iteration of AMG vs. hybrid AMG-DD in the communication-dominated regime starting on level 3 and 4, respectively, for 3D Poisson (left) and rotated anisotropic diffusion (right).

significant reductions in communication cost compared to AMG as the number of processors grows, with hybrid AMG-DD trading smaller overhead for slightly higher communication costs compared to AMG-DD.

Since the overall time for an iteration of the AMG or AMG-DD solve phase is dominated by fine-grid computation in this regime, AMG-DD does not provide any wall clock speedup. It is still important to note, however, that AMG-DD provides significant benefits in terms of accuracy per communication cost, both in terms of latency and bandwidth costs. Figure 14 shows that both AMG-DD and the hybrid version achieve residual reduction with far fewer messages sent and less total volume of
FIGURE 11 Total time and communication time required to perform a mat-vec with $A_l$ on each level, $l$, of the AMG hierarchy with 4,096 processors in the computation-dominated regime for 3D Poisson (left) and rotated anisotropic diffusion (right).

FIGURE 12 Grid size overheads for AMG-DD and hybrid AMG-DD in the computation-dominated regime with hybrid AMG-DD starting on level 2 and 5, respectively, for 3D Poisson (left) and rotated anisotropic diffusion (right).

data communicated compared AMG. Thus, for computational environments where communication is a larger cost relative to computation, AMG-DD is expected to provide significant wall clock speedup.

7 | CONCLUSIONS

This paper continues the development of algebraic multigrid domain decomposition (AMG-DD), a novel, low-communication algorithm built on top of an algebraic multigrid (AMG) hierarchy and designed to achieve similar or better convergence with reduced communication cost compared to AMG V-cycles. An algebraic variant of fast adaptive composite (AlgFAC) cycling is developed that allows for efficient local solution of the composite problems with minimal storage and communication requirements due to a rewriting of traditional multigrid restriction. Numerical results are shown for the first full parallel implementation of the algorithm, and a new hybrid version of AMG-DD starting on a coarser level of the hierarchy is also presented. Optimal parameters for AMG-DD are discovered via empirical study and then used to produce weak scaling studies of the algorithm’s performance. For the tests shown here, communication is only a significant cost for a very small number of degrees of freedom per processor, and AMG-DD is able to achieve modest speedups in this regime despite the large computational overheads for AMG-DD also produced by this regime. For a higher number of degrees of freedom per processor, AMG-DD is shown to achieve quite moderate computational overheads along with superior accuracy per communication cost compared to AMG, that is, it achieves lower error with significantly fewer messages and less total volume of data communicated. Thus, AMG-DD is
FIGURE 13 Weak scaling of number of messages (top) and communication volume (bottom) for AMG-DD and hybrid AMG-DD scaled by the costs for AMG in the computation-dominated regime with hybrid AMG-DD starting on level 2 and 5, respectively, for 3D Poisson (left) and rotated anisotropic diffusion (right).

expected to provide significant speedup over AMG V-cycles in computational environments where communication represents the dominant cost, such as clusters with a very high number of processors or GPU accelerated systems.

References

1. Brandt A, McCormick SF, and Ruge JW. Algebraic multigrid (AMG) for sparse matrix equations. In: Sparsity and its applications. Cambridge Univ. Press, Cambridge; 1984. p. 257–284.

2. Stüben K, and Ruge JW. Algebraic Multigrid. In: McCormick SF, editor. Multigrid Methods, vol. 3, Frontiers in Applied Mathematics. Philadelphia: SIAM; 1987. p. 73–130.

3. Briggs WF, Henson VE, and McCormick SF. A Multigrid Tutorial, 2nd ed. Edition. SIAM; 2000.

4. Baker AH, Falgout RD, Kolev TV, and Yang UM. Scaling hypre’s multigrid solvers to 100,000 cores. In: Berry M, editor. High Performance Scientific Computing Algorithms and Applications. Springer; 2012.

5. Gahvari H, Baker AH, Schulz M, Yang UM, Jordan KE, and Gropp W. Modeling the performance of an algebraic multigrid cycle on HPC platforms. In: Proceedings of the 25th International Conference on Supercomputing (ICS 2011). Tuscon, AZ; 2011. p. 172–181.

6. Bienz A, Falgout RD, Gropp W, Olson LN, and Schroder JB. Reducing parallel communication in algebraic multigrid through sparsification. SIAM Journal on Scientific Computing. 2016;38(5):S332–S357.

7. Bank RE, Falgout RD, Jones TM, Manteuffel TA, McCormick SF, and Ruge JW. Algebraic multigrid domain and range decomposition (AMG-DD/AMG-RD). SIAM Journal on Scientific Computing. 2015;37(5):S113–S136.
FIGURE 14 Residual convergence by latency (top) and bandwidth cost (bottom) for AMG-DD and hybrid AMG-DD vs. AMG with hybrid AMG-DD starting on level 2 and 5, respectively, for 3D Poisson (left) and rotated anisotropic diffusion (right) with 4,096 processors.

8. Baker AH, Falgout RD, and Yang UM. An assumed partition algorithm for determining processor inter-communication. Parallel Computing. 2006;32(5-6):394–414.

9. Bienz A, Gropp WD, and Olson LN. Node aware sparse matrix-vector multiplication. Journal of Parallel and Distributed Computing. 2019;.

10. De Sterck H, Falgout RD, Nolting JW, and Yang UM. Distance-two interpolation for parallel algebraic multigrid. Numerical Linear Algebra with Applications. 2008 Mar;15(2-3):115–139.

11. De Sterck H, Yang UM, and Heys JJ. Reducing Complexity in Parallel Algebraic Multigrid Preconditioners. SIAM Journal on Matrix Analysis and Applications. 2006 Jan;27(4):1019–1039.

12. Yang UM. On long-range interpolation operators for aggressive coarsening. Numerical Linear Algebra with Applications. 2010;17(2-3):453–472.

13. Falgout RD, and Schroder JB. Non-Galerkin coarse grids for algebraic multigrid. SIAM Journal on Scientific Computing. 2014;36(3):C309–C334.

14. Lee B, McCormick SF, Philip B, and Quinlan DJ. Asynchronous Fast Adaptive Composite-Grid Methods for Elliptic Problems: Theoretical Foundations. SIAM Journal on Numerical Analysis. 2004 Jan;42(1):130–152.

15. Vassilevski PS, and Yang UM. Reducing communication in algebraic multigrid using additive variants. Numerical Linear Algebra with Applications. 2014;21(2):275–296.

16. Bank RE, and Holst M. A new paradigm for parallel adaptive meshing algorithms. SIAM Journal on Scientific Computing. 1999;.
17. Mitchell WF. Parallel adaptive multilevel methods with full domain partitions. ANACM Applied Numerical Analysis and Computational Mathematics. 2004;1(1-2):36–48.

18. Mitchell WF. A parallel multigrid method using the full domain partition. Electronic Transactions on Numerical Analysis. 1998;6:224–233.

19. Appelhans DJ, Manteuffel T, McCormick S, and Ruge J. A low-communication, parallel algorithm for solving PDEs based on range decomposition. Numerical Linear Algebra with Applications. 2017;24(3):e2041.

20. Mitchell WB, and Manteuffel TA. Advances in implementation, theoretical motivation, and numerical results for the nested iteration with range decomposition algorithm. Numerical Linear Algebra with Applications. 2018;25(3):e2149–26.

21. Vaněk P, Mandel J, and Brezina M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing. 1996 Sep;56(3):179–196.

22. McCormick SF. Multilevel Adaptive Methods for Partial Differential Equations. Cambridge University Press; 1989.

**How to cite this article:** Mitchell W., R. Strzodka, R. Falgout, and S. McCormick (2020), Parallel Performance of Algebraic Multigrid Domain Decomposition (AMG-DD), . .
Algorithm 1 AMG \( V(v_1, v_2) \)-cycle

\[
\text{for Downswep}: \ l = 0, \ldots, (L - 1) \ \text{do}
\]
\[
\quad \text{Relax} \ v_1 \ \text{times on} \ A_j u_l = f_l.
\]
\[
\quad \text{Restrict residual:} \ f_{l+1} \leftarrow R_l(f_l - A_j u_l).
\]
\[
\quad \text{Initialize:} \ u_{l+1} \leftarrow 0.
\]
\text{end for}

\[
\text{Coarse solve:} \ u_L \leftarrow A_L^{-1} f_L.
\]

\text{for Upsweep:} \ l = (L - 1), \ldots, 0 \ \text{do}
\[
\quad \text{Interpolate correction:} \ u_l \leftarrow u_l + P_l u_{l+1}.
\]
\[
\quad \text{Relax} \ v_2 \ \text{times on} \ A_j u_l = f_l.
\]
\text{end for}

(mat-vec communications with \( A_l \))

Algorithm 2 AMG-DD cycle

\[
\text{Calculate fine-grid residual:} \ r_0 \leftarrow f_0 - A_0 u_0.
\]
\[
\text{Restrict residual to all levels:} \ r_{l+1} \leftarrow R_l r_l, \forall l.
\]
\[
\text{Obtain residual values at all composite-grid points.}
\]
\[
\text{Perform local AlgFAC cycles to obtain} \ \delta u_0^p.
\]
\[
\text{Add fine-grid correction,} \ u_0 \leftarrow u_0 + \delta u_0^p, \ \text{in} \ \Omega_0^p.
\]

(mat-vec communication with \( A_0 \))

Algorithm 3 AlgFAC V-cycle

\text{Initialize solution/corrections:} \ u_l \leftarrow 0, \forall l
\text{Initialize updates:} \ t_l \leftarrow 0, s_l \leftarrow 0, \forall l
\text{for Iterate:} \ i = 0, 1, \ldots \ \text{do}
\text{for Downswep}: \ l = 0, \ldots, (L - 1) \ \text{do}
\text{If} \ l \neq 0, \ \text{initialize}: \ u_l \leftarrow 0
\text{Relax:} \ u_l \leftarrow u_l + \delta_l^1
\text{Accumulate update:} \ t_l \leftarrow t_l + \delta_l^1
\text{Restrict update:} \ s_{l+1} \leftarrow R_l(s_l + A_l t_l)
\text{Recalculate coarse residual:} \ f_{l+1} \leftarrow f_{l+1} - A_{l+1} u_{l+1}
\text{Subtract restricted update:} \ f_{l+1} \leftarrow f_{l+1} - s_{l+1}
\text{Reset updates:} \ t_l \leftarrow 0, s_l \leftarrow 0
\text{end for}
\text{Coarse solve:} \ u_L \leftarrow A_L^{-1} f_L
\text{for Upswep}: \ l = (L - 1), \ldots, 0 \ \text{do}
\text{Interpolate:} \ u_l \leftarrow u_l + P_l u_{l+1}
\text{Relax:} \ u_l \leftarrow u_l + \delta_l^2
\text{Accumulate update:} \ t_l \leftarrow t_l + \delta_l^2
\text{end for}
\text{end for}
Algorithm 4 AMG-DD residual communication algorithm

\begin{algorithm}
\begin{algorithmic}
\For{Loop over levels: $l = L, \ldots, 0$}
  \For{Loop over distance $\eta_l$ processor neighbors: $q = 1, \ldots$}
    \Let $\Psi = \{ i \in \Omega_{l_q}^p : \text{dist}(i, \Omega_{l_q}^p) \leq \eta_l \}$
    \Form a composite grid $\Psi_c$ based on $\Psi$
    \Send residuals at points in $\Psi \cup \Psi_c$ to processor $q$
  \EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

Algorithm 5 Hybrid AMG-DD $V(\nu_1, \nu_2)$-cycle

\begin{algorithm}
\begin{algorithmic}
\For{Downsweep: $l = 0, \ldots, (\tilde{L} - 1)$}
  \Relax $\nu_1$ times on $A_l u_l = f_l$.
  \Restrict residual: $f_{l+1} \leftarrow R_l(f_l - A_l u_l)$.
  \Initialize: $u_{l+1} \leftarrow 0$.
\EndFor
\AMG-DD cycle: $u_{\tilde{L}} \leftarrow \AMG-DD(A_{\tilde{L}}, f_{\tilde{L}})$.
\For{Upsweep: $l = (\tilde{L} - 1), \ldots, 0$}
  \Interpolate correction: $u_l \leftarrow u_l + P_l u_{l+1}$.
  \Relax $\nu_2$ times on $A_l u_l = f_l$.
\EndFor
\end{algorithmic}
\end{algorithm}