SIAM J. SCI. COMPUT.
Vol. 41, No. 5, pp. S196–S219

A COMPARISON OF CLASSICAL AND AGGREGATION-BASED ALGEBRAIC MULTIGRID PRECONDITIONERS FOR HIGH-FIDELITY SIMULATION OF WIND TURBINE INCOMPRESSIBLE FLOWS*

S. J. THOMAS†, S. ANANTHAN†, S. YELLAPANTULA†, J. J. HU‡, M. LAWSON†, AND M. A. SPRAGUE†

Abstract. This paper presents a comparison of parallel strong scaling performance of classical and aggregation algebraic multigrid (AMG) preconditioners in the context of wind turbine simulations. Fluid motion is governed by the incompressible Navier–Stokes equations, discretized in space with control-volume finite elements and in time with an inexact projection scheme using an implicit integrator. A discontinuous-Galerkin sliding-mesh algorithm captures rotor motion. The momentum equations are solved with iterative Krylov methods, preconditioned by symmetric Gauss–Seidel (SGS) in Trilinos and ℓ_1 SGS in *hypre*. The mass-continuity equation is solved with GMRES preconditioned by AMG and can account for the majority of simulation time. Reducing this continuity solve time is crucial. Wind turbine simulations present two unique challenges for AMG preconditioned solvers: the computational meshes include strongly anisotropic elements, and mesh motion requires matrix reinitialization and computation of preconditioners at each time step. Detailed timing profiles are presented and analyzed, and best practices are discussed for both classical and aggregation-based AMG. Results are presented for simulations of two different wind turbines with up to 6 billion grid points on two different computer architectures. For moving mesh problems that require linear-system reinitialization, the well-established strategy of amortizing preconditioner setup costs over a large number of time steps to reduce the solve time is no longer valid. Instead, results show that faster time to solution is achieved by reducing preconditioner setup costs at the expense of linear-system solve costs. Standard smoothed aggregation with Chebyshev relaxation was found to perform poorly when compared with classical AMG in terms of solve time and robustness. However, plain aggregation was comparable to classical AMG.

Key words. Navier–Stokes, iterative solvers, algebraic multigrid, wind turbines, exascale

AMS subject classifications. 65F08, 65F10, 65F50, 76D05, 76D25

DOI. 10.1137/18M1179018

1. Introduction.

For wind energy, cost-competitiveness with fossil fuels, without subsidies, is being achieved by moving to larger, multimegawatt wind turbines that reside in large, many-turbine wind farms. Current barriers to reducing the cost of wind energy include an incomplete understanding of the typical flow dynamics in wind farms, including wake formation, turbine-turbine wake interactions, and wake evolution under different atmospheric stability conditions. High-fidelity modeling, combined with high-performance computing, provides an excellent opportunity for understanding the complex, multiscale flow dynamics around wind turbines and in wind farms. Predictive simulations can expose new pathways to extracting energy

*Received by the editors April 5, 2018; accepted for publication (in revised form) May 13, 2019; published electronically October 29, 2019.
https://doi.org/10.1137/18M1179018

Funding: This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

†National Renewable Energy Laboratory, Golden, CO 80401 (Stephen.Thomas@nrel.gov, Shreyas.Ananthan@nrel.gov, Shashank.Yellapantula@nrel.gov, Michael.Lawson@nrel.gov, Michael.A.Sprague@nrel.gov).

‡Sandia National Laboratories, Livermore, CA 94551 (jhu@sandia.gov).

through technology innovations, new control algorithms, and plant layouts. Such simulations will also be the foundation for computationally efficient, reduced-order models used in wind turbine and plant design and optimization.

Predictive wind turbine simulations will require computational fluid dynamics (CFD) with body-resolving meshes (for the tower, nacelle, and moving blades) and subgrid-scale turbulence modeling [15]. A blade-tower-nacelle resolved simulation for a megawatt-scale turbine that accurately predicts wake formation and evolution will require extreme mesh refinement. Such a simulation would require at least $\mathcal{O}(1)$ billion degrees of freedom (DOFs) and small time steps, e.g., 10^{-3} seconds or smaller [31], and thus represents a challenging problem for many CFD solvers. Overall, the keys to high-fidelity wind energy simulation are accuracy and short time to solution. Accuracy is achieved with appropriate mathematical and numerical models combined with sufficient grid resolution in space and time. Short time to solution is obtained by effectively exploiting high-performance computing, minimizing the wall clock time per time step by employing optimized linear-system solvers, and running simulations with the largest time step size allowed by accuracy requirements and stability constraints.

In the current study, the focus is on single-turbine CFD methods and simulations for the incompressible-flow Navier–Stokes equations. For wind turbine flows, the incompressibility assumption is justified in that the maximum Mach number is approximately $0.2-0.3$ near the blade tips. The CFD algorithms are based on implicit time integration, where mass continuity is maintained through operator-split pressure projection. In that configuration, the governing equations consist of the mass-continuity equation for pressure and momentum equations for velocity. A particular challenge for wind turbine simulations with body-resolved meshes is the movement of meshes required for rotating blades and, in general, other moving components (e.g., pitching blades, yawing nacelle). The computational cost associated with simulations with moving meshes is significant, including reinitialization and reconstruction of matrices (numerical values of matrix elements may change, along with the sparsity structure and communication buffer indices) and recomputation of preconditioners at *every time step*. These costs are important when comparing linear-system solvers, and it may be advantageous to obtain a lower setup time for the preconditioner in exchange for a higher solve time.

Our open-source CFD platform is Nalu-Wind, a wind-focused variant of the Nalu code [10] that was developed at Sandia National Laboratories. While the focus of this paper is on constant-density problems, Nalu-Wind is equipped to solve the low-Mach-number equations as well. Both model systems are acoustically incompressible and require an elliptic-system solve at every time step as part of the pressure-projection scheme that maintains mass continuity. Nalu-Wind is built on the Trilinos Sierra Toolkit and coupled to the Trilinos [17] and *hypre* [13] linear-system solver libraries. The spatial discretization is a second-order control-volume finite element method on generalized topological meshes (i.e., with hexahedral, tetrahedral, and pyramid elements). For time advancement, an approximate pressure-projection scheme is applied with an implicit backward-differentiation-formula (BDF) time integrator. Splitting errors are controlled by fixed-point iteration and the overall scheme is capable of second-order accuracy [10]. Advection stabilization is applied in order to mitigate spurious oscillations. A discontinuous-Galerkin sliding-mesh algorithm based on an interior penalty method has been implemented to capture turbine rotor motion [10]. In such a configuration, a prescribed mesh interface is defined to handle the rigid-body rotation. For the simulations described in this paper, a wall-adapting local-eddy-viscosity-model

(WALE) subgrid-scale turbulence model is applied in order to represent the turbulent energy cascade and dissipation range [20].

The pressure-projection scheme results in separate momentum and pressure linear systems that need to be solved at every time step. Krylov iterative methods are employed to solve both systems. A symmetric Gauss–Seidel relaxation is employed as the preconditioner for momentum, whereas algebraic multigrid (AMG) is the pressure linear-system preconditioner. In this paper, our focus is on comparing the two fundamental AMG algorithms as preconditioners for the pressure linear system: classical Ruge–Stüben AMG [29] (C-AMG) and aggregation AMG [34] (smoothed aggregation (SA-AMG) and plain aggregation (PA-AMG)). SA-AMG and PA-AMG are provided by the Trilinos MueLu solver library [27]. Classical Ruge–Stüben AMG is implemented by the *hypre*-BoomerAMG library [16]. The Trilinos solver library Belos [2] provides the iterated classical Gram–Schmidt GMRES (ICGS-GMRES) algorithm, which requires less communication than the modified Gram–Schmidt GMRES (MGS-GMRES) algorithm as implemented in *hypre*. High-aspect-ratio element meshes are handled differently by C-AMG and aggregation AMG (SA-AMG and PA-AMG). The strength threshold in C-AMG is row-oriented and is well suited to stretched elements. For aggregation AMG a distance Laplacian is applied to the strength threshold in order to coarsen. An extensive study comparing C-AMG and aggregation AMG for model problems was conducted by Yang [35].

In order to assess GMRES solver performance with C-AMG, SA-AMG, and PA-AMG preconditioners, high-fidelity wind turbine simulations were performed on the Cori and Mira supercomputers, which are at the National Energy Research Scientific Computing Center (NERSC) and the Argonne Leadership Computing Facility (ALCF), respectively. Specifically, wake vortex formation is simulated around Vestas V27 (225 kW) and NREL 5-MW wind turbines (blades, tower, and nacelle). The V27 is a smaller turbine with a 27 m diameter rotor. The NREL 5-MW turbine is a notional reference turbine defined in the open domain [19]. It has a 126 m diameter rotor and is representative of large modern turbines with flexible blades.

The paper is organized as follows. Section 2 describes the Navier–Stokes pressure-projection, time integration, and mesh motion algorithms. An overview of classical and aggregation AMG algorithms is provided in section 3. Strong scaling studies with run times and detailed timing profiles are presented in section 4.

2. Incompressible Navier–Stokes equations and numerical discretization.

Our fluid model is the incompressible-flow Navier–Stokes equations. For the wind turbine flow problems described here, the fluid density is constant and the continuity equation is

$$\rho\frac{\partial u_j}{\partial x_j} = 0\,, \tag{2.1}$$

where ρ is the fluid density, u_j is the velocity, x_j is the spatial coordinate, and Einstein notation (summation over repeated indices) is employed. The momentum equation is

$$\rho\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(\,\rho u_j\, u_i - \sigma_{ij}\,) = S_i\,, \tag{2.2}$$

where t is time and source term S_i. The Cauchy stress σ_{ij} at pressure p is given by

$$\sigma_{ij} = \tau_{ij} - p\,\delta_{ij} + \frac{2}{3}\,\mu\,\frac{\partial u_k}{\partial x_k}\,\delta_{ij}\,, \tag{2.3}$$

where τ_{ij} is the viscous stress tensor, μ is the viscosity, and δ_{ij} is the Kronecker delta function. Following Domino [8], the control-volume finite element method defines a dual mesh constructed within each element. To obtain the weak variational form, each equation is multiplied by a test function w and integrated over the domain. The schemes are Petrov–Galerkin because the test function differs from the interpolation basis. For the momentum equation, the stress and advection terms are integrated by parts.

The time integration schemes in Nalu-Wind can be expressed in the general form for a fluid field variable ϕ, with time step Δt and weights γ_i, as

$$\int \frac{\partial \rho\,\phi}{\partial t} dV = \int \frac{\gamma_1 \rho^{n+1}\phi^{n+1} + \gamma_2 \rho^n \phi^n + \gamma_3 \rho^{n-1}\phi^{n-1}}{\Delta t}\, dV\,, \tag{2.4}$$

where the superscript denotes the time step. For simulations in this paper an implicit BDF time integrator was employed with an adaptive (variable) time step that can increase toward a target Courant–Friedrichs–Lewy (CFL) number. The momentum and pressure solutions obtained during time stepping are calculated by an incremental approximate pressure-projection algorithm. The momentum and pressure equations are segregated and solved sequentially with implicit advection/diffusion.

To describe the approximate pressure-projection algorithm, consider the block (indefinite, saddle-point) matrix form of the discrete equations

$$\begin{bmatrix} F & G \\ D & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{2.5}$$

where the matrix F will depend on the predicted, current, and earlier time-levels. F contains discrete contributions to the momentum equations from the time derivative, diffusion, and linearized advection terms [4]. $F = I/\Delta t + \mu L + N$, where L is the Laplacian and N linearized advection [26]. The discrete gradient and divergence matrices are G and D, respectively. The vector f contains the additional terms for the momentum equations, e.g., body force terms, lagged stress tensor terms. The right-hand side contains the appropriate terms, e.g., for a nonsolenoidal velocity field. For more details, see [7]. In order to derive a projection scheme, consider the time-split system of equations,

$$\begin{bmatrix} F & G \\ D & 0 \end{bmatrix} \begin{bmatrix} u^{n+1} \\ \Delta p^{n+1} \end{bmatrix} + \begin{bmatrix} I & G \\ D & 0 \end{bmatrix} \begin{bmatrix} 0 \\ p^n \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{2.6}$$

where $\Delta p^{n+1} = p^{n+1} - p^n$. Consider the block factorization of the matrix

$$\begin{bmatrix} F & G \\ D & 0 \end{bmatrix} = \begin{bmatrix} F & 0 \\ D & -DF^{-1}G \end{bmatrix} \begin{bmatrix} I & F^{-1}G \\ 0 & I \end{bmatrix}. \tag{2.7}$$

Inversion of F to form the Schur complement matrix $M = -DF^{-1}G$ would be costly. The splittings described in [11], [25], and [28] approximate the inverse with the diagonal matrix $[\mathrm{diag}(F)]^{-1}$. Nalu-Wind employs an approximate projection scheme that introduces auxiliary time-scale matrices. The factor $B_2 = (\tau/\rho)\, I$ determines the projection time scale. The factor $B_1 = -\tau L$ defines the linear system for pressure and approximates M. L is the Laplacian matrix obtained from the discrete form of the Gauss divergence theorem

$$\int L\phi\, dV = \int \nabla\phi \cdot \mathrm{d}A. \tag{2.8}$$

These matrices are introduced into an approximate block factorization as follows:

$$\begin{bmatrix} F & 0 \\ D & B_1 \end{bmatrix} \begin{bmatrix} I & B_2 G \\ 0 & I \end{bmatrix} \begin{bmatrix} u^{n+1} \\ \Delta p^{n+1} \end{bmatrix} + \begin{bmatrix} I & 0 \\ DB_2 & -B_1 \end{bmatrix} \begin{bmatrix} I & G \\ 0 & I \end{bmatrix} \begin{bmatrix} 0 \\ p^n \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} .$$

The time scale $\tau = \Delta t$ is chosen for stabilization [7]. The equations are solved for $\Delta\hat{u} = \hat{u} - u^n$, and Δp^{n+1} at each outer nonlinear iteration of the time step

$$F\Delta\hat{u} = f - Gp^n - Fu^n \,, \tag{2.9}$$

$$-\tau\, L\Delta p^{n+1} = -D\,(\,\rho\hat{u} + \tau\, G\, p^n\,) + \tau\, L\, p^n \,, \tag{2.10}$$

$$u^{n+1} = \hat{u} - \frac{\tau}{\rho}\, G\Delta p^{n+1} \,. \tag{2.11}$$

The momentum (2.9) and pressure continuity (2.10) equations are solved separately within a fixed-point iteration for the incompressible-flow system of equations [33]. An iteration predicts a velocity field that is not divergence free and then projects to a divergence-free subspace. The matrix F and the right-hand side of the momentum equation are functions of the solution at time step $n+1$ due to the choice of an implicit BDF time integrator. At each iteration, a better estimate for the solution is computed at time step $n+1$ and hence a better estimate for F.

For mesh motion associated with rotation of the turbine blades, a sliding-mesh interface based on the discontinuous-Galerkin interior penalty method was implemented by Domino [10]. The control volumes adjacent to the interface contribute nonsymmetric terms to the pressure matrix. Following the notation in [10], two domains, Ω^A and Ω^B, with a common interface, Γ^{AB}, and a set of interfaces not in common, $\Gamma\backslash\Gamma^{AB}$, are displayed in Figure 1. In particular, the numerical flux terms have a skew-symmetric form $\hat{C}(A,B) = -\hat{C}(B,A)$, at the interface Γ^{AB} between adjacent elements A and B at the sliding interface where the continuity equation is modified

$$\int_{\Gamma\backslash\Gamma^{AB}} \rho\, \hat{u}\, n_j\, d\,\Gamma + \int_{\Gamma^{AB}} \hat{C}(A,B)\, d\,\Gamma = 0 \,, \tag{2.12}$$

$$\int_{\Gamma\backslash\Gamma^{BA}} \rho\, \hat{u}\, n_j\, d\,\Gamma + \int_{\Gamma^{BA}} \hat{C}(B,A)\, d\,\Gamma = 0 \,. \tag{2.13}$$

The flux terms are computed as follows:

$$\hat{C}(\alpha,\beta) = \frac{1}{2}(\, m_j^\alpha\, n_j^\alpha - m_j^\beta\, n_j^\beta\,) + \lambda_C^{\alpha\beta}\,(\, p^\alpha - p^\beta\,), \quad \lambda_C^{\alpha\beta} = \frac{\tau}{2}\left(\frac{1}{L^\alpha} + \frac{1}{L^\beta}\right) ,$$

where m_j is the mass flowing into an element and n_j is the unit normal vector. The penalty term is $\lambda_C^{\alpha\beta}$ and L^α is the mesh length scale.

3. Algebraic multigrid.

AMG methods are effective scalable solvers that are well suited for high-performance computer architectures [35, 22, 23]. When employed as a stand-alone solver or as a preconditioner for a Krylov iteration such as GMRES [30], AMG can in theory solve a linear system with n unknowns in $\mathcal{O}(n)$ operations. It is common practice now to use AMG as a preconditioner to a Krylov method, even though it was originally developed as a solver.

An AMG method accelerates the solution of a linear system

$$Ax = b \tag{3.1}$$

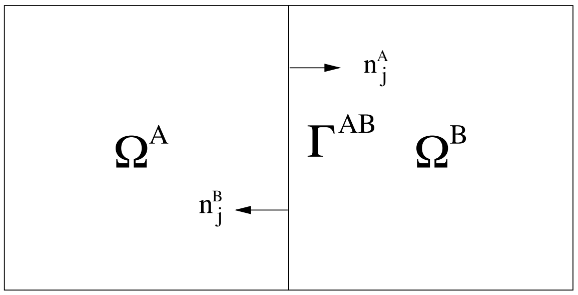

FIG. 1. *Two-block example, Ω^A and Ω^B, with common interface, Γ^{AB}.*

through error reduction by using a sequence of coarser matrices called a *hierarchy*. We will refer to the sequence of matrices as A_k, where $k = 0 \dots m$, and A_0 is the matrix from (3.1). Each A_k has dimensions $m_k \times m_k$, where $m_k > m_{k+1}$ for $k < m$. For the purposes of this paper, we will assume that

$$A_k = R_k A_{k-1} P_k \tag{3.2}$$

for $k > 0$, where P_k is a rectangular matrix with dimensions $m_{k-1} \times m_k$. P_k is referred to as a *prolongation matrix* or *prolongator*. R_k is the *restriction matrix* and $R_k = P_k^T$ in the Galerkin formulation of AMG. Associated with each A_k, $k < m$, is a solver called a *smoother*, which is usually an inexpensive iterative method, e.g., Gauss–Seidel, polynomial, or incomplete factorization. The *coarse solver* associated with A_m is often a direct solver, although it may be an iterative method if A_m is singular.

The setup phase of AMG is nontrivial for several reasons. Each prolongator P_k is developed algebraically from A_{k-1} (hence the name of the method). Once the transfer matrices are determined, the coarse matrix representations are recursively computed from A through sparse matrix-matrix multiplication.

In the AMG solve phase, a few steps of a smoother are applied to the finest-level linear system with a zero initial guess. This is referred to as *presmoothing*. A residual is calculated and restricted to the next coarser level, for which it becomes the right-hand side for the next coarser linear system. This process repeats recursively until the coarsest level is reached. The coarsest-level system is usually solved with a direct method. The solution of the coarsest system solve is then interpolated to the next finer level, where it becomes a correction for that level's previous approximate solution. A few steps of *postsmoothing* are applied after this correction. This process is repeated until the finest level is reached. This describes a V-cycle, the simplest complete AMG cycle. See Algorithm 3.1 for the complete description. AMG methods achieve optimality (constant work per DOF in A_0) through complementary error reductions by the smoother and solution corrections propagated from coarser levels.

3.1. AMG and strong scaling.

For the current wind turbine application, the problem mesh has a fixed size, and the goal is to simulate a fixed number of turbine rotations as quickly as possible, using as many computer processes as are practicable. This is so-called parallel *strong scaling*, which stands in contrast to *weak scaling*. In the latter case, the work per process is fixed, and the global problem grows as the number of processes is increased.

Strong scaling is challenging for AMG for a few reasons. First, the work per process decreases as the number of processes increases. This is true for every matrix

Algorithm 3.1 Multigrid single-cycle algorithm ($\nu = 1$ yields V-cycle) for solving $Ax = b$. The hierarchy has $m + 1$ levels.

```
//Solve $Ax = b$.
Set $x = 0$.
Set $\nu = 1$ for $V$-cycle.
call Multilevel$(A, b, x, 0, \nu)$.

function MULTILEVEL$(A_k, b, x, k, \nu)$
    // Solve $A_k x = b$ ($k$ is current grid level)
    // Presmoothing step
    $x = S_k^1(A_k, b, x)$
    if $(k \neq m)$ then
        // $P_k$ is the interpolant of $A_k$
        // $R_k$ is the restrictor of $A_k$
        $r_{k+1} = R_k(b - A_k x)$
        $A_{k+1} = R_k A_k P_k$
        $v = 0$
        for $i = 1 \ldots \mu$ do
            MULTILEVEL$(A_{k+1}, r_{k+1}, v, k + 1, \nu)$
        end for
        $x = x + P_k v$
        // Postsmoothing step
        $x = S_k^2(A_k, b, x)$
    end if
end function
```

in the multigrid hierarchy. While the computational work for a process is decreasing, the communication pattern and overhead remain the same. Second, as the number of unknowns per process decreases, the coarsening quality will degrade. This can be ameliorated if either the coarsening algorithm is allowed to cross processor boundaries or data is otherwise migrated between processes.

3.2. Ruge–Stüben AMG. We now give a brief overview of classic Ruge–Stüben AMG, starting with some notation that will be used in the subsequent discussions. Point j is a neighbor of i if and only if there is a nonzero element a_{ij} of the matrix A. Point j strongly influences i if and only if

$$|a_{ij}| \geq \theta_C \max_{k \neq i} |\, a_{ik} \,| , \tag{3.3}$$

where θ_C is the strength of connection threshold, $0 < \theta_C \leq 1$. This strong influence relation is used to select coarse points. The selected coarse points are retained in the next coarser level, and the remaining fine points are dropped. Let C_k and F_k be the coarse and fine points selected at level k, and let m_k be the number of grid points at level k ($m_0 = n$). Then, $m_k = |C_k| + |F_k|$, $m_{k+1} = |C_k|$, A_k is a $m_k \times m_k$ matrix, and P_k is a $m_{k-1} \times m_k$ matrix. Here, the coarsening is performed rowwise by interpolating between coarse and fine points. The coarsening generally attempts to fulfill two contradictory criteria. In order to ensure that a chosen interpolation scheme is well defined and of good quality, some close neighborhood of each fine point must contain a sufficient amount of coarse points to interpolate from. Hence the set of coarse points must be rich enough. However, the set of coarse points should be sufficiently small in order to achieve a reasonable coarsening rate. The interpolation should lead to a reduction of roughly five times the number of nonzeros at each level of the V-cycle.

There has been extensive research on variants of AMG since the development of the first AMG algorithms [29] with details presented in [32]. One of the drawbacks with the C-AMG method is that, despite rapid convergence, it often generates excessive operator complexities, in particular for three-dimensional problems. This problem is exacerbated for parallel implementations of AMG when the coarsening algorithm is applied within an MPI rank (subdomain) and the smoother is local. At the interface between MPI ranks the smoother is a pointwise Jacobi iteration. Consequently, efforts were made to coarsen more aggressively to reduce operator complexities [35]. More aggressive coarsening leads to often considerably reduced convergence rates because it violates conditions required for classical interpolation. Convergence can be improved again by combining more aggressive coarsening with long-distance interpolation [36]. Non-Galerkin coarse grids, for which the coarse-level operator does not satisfy the relationship $A_k = P_k^T A_{k-1} P_k$ for each level k, were introduced in [12].

3.3. Smoothed aggregation AMG. As in C-AMG, the main challenge for SA-AMG is the formation of an appropriate prolongation matrix P. SA-AMG coarsens a matrix A by grouping unknowns into *aggregates* that correspond to the unknowns used on the next level [34]. An aggregate $N_i(\theta)$ is formed by starting with a *root* unknown i and selecting from $A(i,:)$'s row stencil those unknowns with matrix entries that are sufficiently large. Specifically,

$$N_i(\theta) = \left\{ \, j : |a_{ij}| \geq \theta \sqrt{a_{ii}\, a_{jj}} \, \right\} , \tag{3.4}$$

where θ is the problem-dependent scalar-valued strength threshold. Instead of using entries of A for this *strength of connection measure*, an alternative strategy is to use entries from an auxiliary matrix with entries that reflect distances between mesh points.

A key point of SA-AMG (or any AMG method, for that matter) is ensuring that slow-to-converge error modes are represented on coarse-level problems. For an elliptic problem, these are the modes in the (near) nullspace[1] of the matrix. For canonical problems these modes are known. For other types of problems, these modes may be different and should be supplied by the application. Most SA-AMG codes default to using the constant vector, i.e., vector of all ones. Regardless, the specified near nullspace modes are rewritten as a matrix $\widetilde{P}$ so that they have local support over the aggregates $N_i(\theta)$. After orthonormalization of its columns, $\widetilde{P}$ is called the *tentative prolongator* and corresponds to piecewise-constant interpolation.

The tentative prolongator can be improved by applying Jacobi smoothing:

$$P = (\, I - \omega\, D^{-1}\, A^F \,)\widetilde{P}, \text{ where } \omega = \frac{4}{3\,\lambda_{max}(\, D^{-1}\, A)}, \tag{3.5}$$

where $D = \text{diag}(A)$ and λ_{max} is the maximum eigenvalue. This step serves to lower the energy in the basis vectors of the grid transfer. The matrix A^F reflects connections

[1]The nullspace of the Laplacian matrix with Neumann boundary conditions is the constant vector. With boundary conditions applied, this vector is the near nullspace and still a low energy error mode. Efficiently reducing errors associated with this and nearby modes is key to good multigrid performance.

that may have been dropped via (3.4). Elements a_{ij}^F of A^F are given by

$$a_{ij}^F = \begin{cases} a_{ij}, & j \in N_i(\theta), j \neq i, \\ 0, & j \notin N_i(\theta), \\ a_{ii} + \sum_{k \notin N_i(\theta)} a_{ik}, & i = j. \end{cases} \tag{3.6}$$

In the numerical experiments described in section 4, SA-AMG is provided by the MueLu library [27], part of the Sandia Trilinos project [17]. The aggregation algorithm is performed local to an MPI rank or subdomain.

3.4. Metrics for assessing AMG quality. There are two important measures that determine the quality of an AMG algorithm. The first is the convergence factor, which indicates how fast the method converges. The second is the operator complexity, which affects the number of operations and the memory usage. Operator complexity C is defined as

$$C = \sum_{k=0}^{m} \text{nnz}(A_k)/\text{nnz}(A)\,, \tag{3.7}$$

where nnz(A) is the number of nonzero elements of A. The complexity measure indicates the amount of memory required. To reduce memory utilization, the complexity C should remain small. The complexity is also an indicator of the number of operations per V-cycle in the solver, assuming an iterative smoother. Small operator complexities lead to small V-cycle times. Even when the stencil sizes $S(A)$ of the original matrix are small, very large stencil sizes can occur on coarser levels. Large stencil sizes can lead to large setup times, even if the operator complexity is small. This is because the coarsening algorithms, and to some degree interpolation, visit neighbors of neighbors, resulting in superlinear or even quadratic growth in the number of operations when evaluating the coarse system or the interpolation matrix. Large stencil sizes can also increase parallel-communication cost, because they may require the exchange of larger sets of data. Both convergence factors and complexities need to be considered when evaluating coarsening and interpolation algorithms, as they often influence each other. Higher complexities can improve convergence, and lower complexities lead to degradation in convergence rates. A degradation in convergence rate due to lower complexity often can be overcome by Krylov methods such as GMRES.

3.5. Considerations for AMG parallel performance. SA-AMG, in which the damped Jacobi prolongator improvement step is skipped, is often referred to as *unsmoothed aggregation* or *unsmoothed prolongation.* As described earlier, we denote it by plain aggregation, or PA-AMG. PA-AMG typically has a lower operator complexity C than standard SA-AMG, resulting in lower setup time and lower cost per iteration. The trade-off, however, is that the convergence rate is worse than when SA-AMG employs a smoothed prolongator, and theoretical bounds on grid independence are lost. As will be seen in section 4, the loss in convergence is made up for in lower overall setup times. The coarse matrix sparsity is directly related to the multigrid complexity C. To a large extent C determines the solve and setup times. In the flow simulations presented here, parallel maximal independent set coarsening is used in C-AMG, due to its high degree of parallelism. Aggressive coarsening is applied on the first two levels. The remaining levels employ extended+i interpolation to construct the prolongation; "extended+i" refers to a coarse-to-fine point interpolation scheme in the C-AMG coarsening algorithm. The paper by Yang [36] describes such long-range interpolation operators. In this approach, we use not only connections a_{jk} from strong fine neighbors j of i, to points k of the interpolation set, but also connections a_{ji} from j to i itself. This often leads to improved convergence compared to other distance-two interpolation operators [5].

Another important consideration for parallel performance is load balancing of the linear systems in the multigrid hierarchy. In standard AMG, the levels in the hierarchy are visited sequentially. If the work per process is not approximately equal, then some processes will be left idle, while other processes are still computing. For coarse levels in the multigrid hierarchy, load-balancing to a subset of MPI processes can also reduce pressure on MPI, thus improving communication performance.

Mesh-element anisotropy, load-balancing, and cost per iteration are important factors in the comparison of the C-AMG, SA-AMG, and PA-AMG variants. Elements can be severely stretched and distorted near the surface of the wind turbine blades. For SA-AMG, a graph distance Laplacian dropping scheme is applied in the strength of connection test to handle anisotropic meshes [14], [27]. The dropping is applied during aggregate construction. In the case of classical AMG the row-based strength of connection threshold is well suited to anisotropic problems. As will be seen in section 4, the test problem contains regions of the mesh with elements of very large aspect ratios.

4. Results.

In this section we investigate simulation timing results and strong scaling performance with C-AMG, SA-AMG, and PA-AMG as pressure equation preconditioners in a simulation of the Vestas V27 (225 kW) and NREL 5-MW wind turbines (nacelle, blades, and tower), in uniform inflow modeling the tip vortex formation. A lower-resolution study of the V27 wind turbine was presented in Domino [10]. We consider a rotating-blade configuration on unstructured boundary-layer resolving meshes. The V27 experiments are conducted on three increasingly refined meshes, denoted R0, R1, and R2. The NREL-5-MW-turbine simulations are conducted on two increasingly refined meshes, denoted G1 and G2. Details for the five meshes are given in Table 1. DOFs (equivalent to element grid points) are provided. Figure 2 provides an illustration of the surface mesh in the vicinity of the hub for the V27 R0 mesh. For the V27 mesh, a rotor rotation rate of 43 rpm and an inflow wind speed of 7.6 m/s are specified. The NREL 5-MW rotation rate is 9 rpm with 8 m/s wind speed. The WALE subgrid-scale turbulence model is applied. The nonlinear stabilization operator is applied together with a first-order accurate implicit time integration scheme (BDF-1).

TABLE 1

Details for the three mesh refinement levels of the V27 and two mesh refinement levels of the NREL 5-MW turbine.

Name	Number of elements	Number of pressure DOF
V27 R0	166M	46M
V27 R1	500M	229M
V27 R2	4.3B	1.8B
NREL 5-MW G1	1.4B	761M
NREL 5-MW G2	11.4B	6.0B

Performance studies were conducted on the Cori Cray XC40 supercomputer at NERSC on the Intel Haswell partition and on the Mira IBM Blue Gene/Q at Argonne ALCF. Cori Haswell nodes consist of E5-2698 v3 Haswell processors clocked at 2.3 GHz, with 2×16 core sockets, and 128 GB DDR4 memory. Cori's interconnection

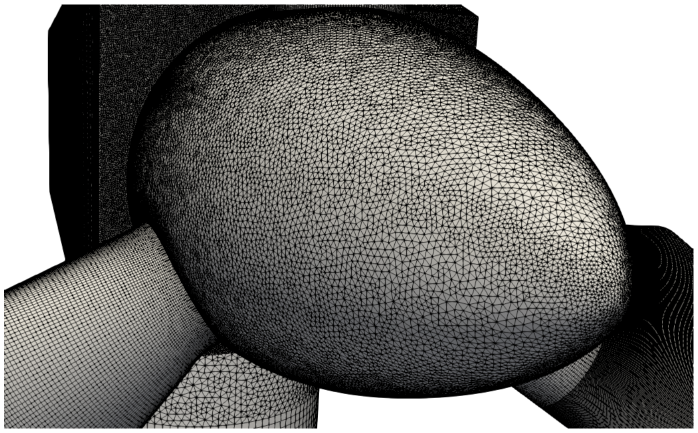

Fig. 2. *Close-up of Vestas $V27$ hub section and nacelle for the $V27$ $R0$ mesh; surface mesh lines are in black.*

network is a Dragonfly topology. The node-to-node network latency is 0.7 microseconds. There are three levels of network connection. The rank-1 point-to-point bandwidth in a chassis is 15.4 GB per second. The rank-2 interchassis bandwidth is also 15.4 GB per second, as is the rank-3 all-to-all bandwidth. The Mira IBM BG/Q consists of 49,152 nodes with a 16-core 1.6 GHz IBM PowerPC A2 processor per node. There are a total of 786,432 cores and 760 TB of memory. The interconnect is a 5D torus, with 2 GB/s chip-to-chip links and from 80 nano-seconds to 3 microseconds latency. Although Nalu-Wind can be run with MPI and threading enabled simultaneously, for the purposes of these runs, one MPI rank per core was employed.

The pressure continuity solver contribution to the overall simulation wall clock time is determined by several factors. The number of GMRES solver iterations required to achieve a given relative residual $\|b-Ax\|_2/\|b\|_2$ tolerance level may be small. However, the smoothing sweeps applied to the coarse matrices in the V-cycle can be expensive if these matrices are not sufficiently sparse. The sparsity (or complexity) of these matrices is determined by the number of nonzeros per row and coarsening rates. SA-AMG ideally coarsens by 3^d times, whereas C-AMG coarsens by 2^d times, where d is the problem dimension. The C-AMG interpolation stencil width changes the complexity and aggressive coarsening can change the rate to over 4 times. Unsmoothed prolongation in PA-AMG (with $\omega = 0$; see (3.5)) can further improve sparsity. The number of V-cycle levels and size of the coarse system also affect the GMRES convergence rate, amount of parallel communication, and run time.

4.1. V27 simulation results. For the present strong scaling study, we have made significantly different algorithmic choices for SA-AMG than for previous weak scaling studies. A prior large-scale study of Nalu (the precursor to Nalu-Wind) and Trilinos/MueLu employed smoothed aggregation and polynomial smoothing for an open-jet low-Mach-number simulation [22]. While we still use a multigrid V-cycle for the present simulations, we have found that PA-AMG is effective for convergence, while also leading to sparser coarse matrices than SA-AMG. The aggregation scheme is a greedy, processor-local algorithm, and a distance-Laplacian dropping scheme is applied with threshold drop tolerance $\theta = 0.03$. The restriction matrix R_k is calculated explicitly, rather than being applied implicitly. Minimum and maximum aggregate sizes are specified to be 3 and 8, respectively, which leads to lower (restarted) GMRES iterations. Rebalancing is applied to the coarse levels to improve parallel performance. In particular, the restriction matrix R_k is explicitly rebalanced, in order to avoid extra communication during the solve phase. This slightly increases the setup cost when applied to the coarse levels, where $k > 1$, and reduces the solve time. Additionally, when rebalancing does occur, we specify that each process get 10,000 rows, which is a significantly higher target than in [22]. The smoother is ℓ_1 Gauss–Seidel, which is a nonoverlapping block Jacobi iteration, with damped Gauss–Seidel for processor-local unknowns near interprocess boundaries. Two pre- and postsweeps of ℓ_1 Gauss–Seidel are performed at each multigrid level except at the coarsest level, which is solved directly with SuperLU [21]. The full input deck specification can be found in [18].

The best performing *hypre*-BoomerAMG parameter choices are as follows. Coarsening for *hypre*-BoomerAMG is based on the parallel maximal independent set algorithm of Luby [6, 5, 24], allowing for a parallel setup phase. A transposed prolongation operator is retained for triple-matrix RAP products. The strength of connection threshold is set to $\theta_C = 0.25$. Aggressive coarsening is applied on the first two V-cycle levels with multipass interpolation and a stencil width of two elements per row. The remaining levels employ extended+i interpolation. Interpolation truncation level 0.25 is specified together with a maximum interpolation stencil width of two matrix elements per row. Sparsification techniques with non-Galerkin operators and drop tolerances applied to specific levels were introduced in [3] to further reduce the complexity C. The truncation is applied on the first three levels with drop tolerances $\gamma = [\, 0.0,\ 0.01,\ 0.01 \,]$. The smoother is two sweeps of a hybrid Gauss–Seidel relaxation scheme. The *hypre*-BoomerAMG smoother is hybrid with Gauss–Seidel applied locally and then Jacobi smoothing for globally shared DOFs. The coarsening rate for the V27 problem is roughly $4\times$ with eight levels in the V-cycle for *hypre*. Operator complexity C is close to 1.1, indicating more efficient V-cycles with aggressive coarsening; however, an increased number of (restarted) GMRES iterations are required compared to standard coarsening.

Here we present a detailed comparison of C-AMG with PA-AMG and SA-AMG. The SA-AMG V-cycle employs a Chebyshev polynomial smoother and follows Domino [10]. A similar study for model problems was conducted by Yang [35], who examined finite difference and finite element discretizations of the Laplacian. The reported results demonstrated the correlation between the complexity C and solve time, along with stencil-width S and setup time. Unless otherwise noted, in the following test cases the momentum matrix is solved with GMRES with symmetric Gauss–Seidel preconditioning from Trilinos. In order to avoid initial transients in the linear system solves, simulations were begun from a restart file generated after the initial 20 steps starting from $t = 0$ seconds.

Navier–Stokes simulation times with *hypre* and MueLu pressure solve and setup times are reported in Tables 2 and 3 for simulations on Cori. The tables also report the number of smoother sweeps (sw) at each multigrid level, strength of connection threshold θ, multigrid operator complexity C, and maximum matrix row stencil size $S_{max} = \max_{1 \le k \le m} S(A_k)$.

The original SA-AMG settings lead to large $S_{max} = 113$ stencil widths and, together with the original Chebyshev smoother settings, result in much higher GMRES iteration counts and solve times compared to the PA-AMG algorithm. The continuity solver times for SA-AMG are 10 times larger than PA-AMG. The overall simulation time is $3\times$ larger than with PA-AMG and almost $4\times$ greater than with C-AMG. The GMRES(50) solver with SA-AMG and Chebyshev smoother does not converge after 300 iterations for the V27 R2 mesh.

More recently, however, we have found that adjusting the Chebyshev smoother parameters can dramatically improve the performance of SA-AMG. For example, changing the interval of convergence for Chebyshev results in a V27 R1 SA-AMG total simulation at 4096 cores of 450 seconds. While not as fast as PA-AMG, this does indicate that simulations based on the V27 mesh sequence may be amenable to using SA-AMG for the continuity solve.

Table 2
Continuity solver settings; iterations; and solve, preconditioner setup, total simulation times (in seconds) and speed-up Sp for 10 *time steps of* $V27$ *R*1 *simulation on Cori.*

(a) Trilinos continuity & momentum solvers SA-AMG, Chebyshev									
Cores	*sw*	θ	C	S_{max}	Iterations	Solve	Setup	Total	*Sp*
2048	4	0.03	1.30	113	152	1642	129	2521	1
4096	4	0.03	1.30	113	134	808	105	1325	1.9
8192	4	0.03	1.30	113	132	546	65	985	2.6
12288	4	0.03	1.30	113	140	468	61	891	2.8
(b) Trilinos continuity & momentum solvers PA-AMG, ℓ_1 Gauss–Seidel									
Cores	*sw*	θ	C	S_{max}	Iterations	Solve	Setup	Total	*Sp*
2048	2	0.03	1.15	19.6	25	157	44	810	1
4096	2	0.03	1.15	19.6	26	77	24	435	1.9
8192	2	0.03	1.15	19.6	28	49	20	290	2.8
12288	2	0.03	1.15	19.6	28	36	17	259	3.1
(c) *hypre*-BoomerAMG continuity solver & Trilinos momentum solver C-AMG, hybrid Gauss–Seidel									
Cores	*sw*	θ_C	C	S_{max}	Iterations	Solve	Setup	Total	*Sp*
2048	2	0.25	1.15	21.4	18	132	29	753	1
4096	2	0.25	1.15	21.4	17	75	19	428	1.8
8192	2	0.25	1.15	21.4	18	52	21	282	2.7
12288	2	0.25	1.15	21.4	17	43	29	245	3.1

Table 3
Continuity solver settings; iterations; and solve, preconditioner setup, and total simulation times (in seconds) for 10 *time steps of* $V27$ *R*2 *simulation on Cori.*

(a) Trilinos continuity & momentum solvers PA-AMG, ℓ_1 Gauss–Seidel									
Cores	*sw*	θ	C	S_{max}	Iterations	Solve	Setup	Total	*Sp*
12288	2	0.03	1.14	19.8	36	369	214	2202	1
24576	2	0.03	1.14	19.8	29	148	59	1560	1.4
(b) *hypre*-BoomerAMG continuity solver & Trilinos momentum solver C-AMG, hybrid Gauss–Seidel									
Cores	*sw*	θ_C	C	S_{max}	Iterations	Solve	Setup	Total	*Sp*
12288	2	0.25	1.15	21.7	21	323	163	2233	1
24576	2	0.25	1.15	21.7	18	151	86	1323	1.7

For the C-AMG coarsening algorithms, the introduction of aggressive coarsening together with smaller interpolation stencils results in much lower operator complexities and average stencil widths S_{max}. Non-Galerkin coarse matrix sparsification in *hypre* reduces the operator complexity and solve time. For PA-AMG, the limits on aggregate size improve convergence rates. For our wind turbine simulations with moving meshes, the AMG setup cost is an important factor because, as described above, the matrices are rebuilt every time step and the V-cycle hierarchy must be recomputed. In our comparison runs, different Krylov subspace dimensions are specified with GMRES(50) for MueLu, whereas GMRES(10) is employed with *hypre*. These choices reflect the average number of iterations and reduce communication overhead in *hypre*-BoomerAMG associated with MGS-GMRES. The relative residual tolerance in all cases is 10^{-5}. The complexity C and GMRES iteration counts are correlated with solve time, whereas the average stencil widths S_{max} are highly correlated with setup time. The GMRES iteration counts are lower for *hypre* and reflect faster convergence rates for the C-AMG preconditioner. However, the cost per iteration is higher for *hypre*.

Table 4

Breakdown in seconds of total simulation times listed in Table 2 *(omitting continuity preconditioner setup and solve) for* 10 *time steps of the* $V27$ $R1$ *mesh on Cori. The times in* (a) *are representative of the configurations in Table* 2(a)–(b).

(a) Trilinos continuity & momentum solvers										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
2048	29	18	13	45	116	95	165	75	5	37
4096	19	8	7	30	56	53	89	44	3	18
8192	22	4	4	32	27	31	46	40	2	14
12288	41	3	3	44	18	22	30	31	2	14

(b) *hypre*-BoomerAMG continuity solver & Trilinos momentum solver										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
2048	3	26	26	43	115	94	163	69	5	34
4096	1	12	16	29	56	53	89	44	3	18
8192	1	6	10	30	27	31	45	40	2	9
12288	1	4	7	38	18	22	29	31	2	6

Table 5

Breakdown in seconds of total simulation times listed in Table 3 *(omitting continuity setup and solve) for* 10 *time steps of the* $V27$ $R2$ *mesh on Cori. The times in* (a) *are representative of the configurations in Table* 3(a)–(b).

(a) Trilinos continuity & momentum solvers										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
12288	124	39	8	514	140	221	293	464	12	52
24576	252	10	13	279	71	87	209	363	21	25

(b) *hypre*-BoomerAMG continuity solver & Trilinos momentum solver										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
12288	3	58	25	514	140	221	293	467	12	47
24576	3	16	28	281	71	87	151	365	24	25

Tables 4 and 5 show the breakdown of the overall simulation time into several categories, and Figure 3 gives a plot of the wall-clock run times. Results indicate two other dominant costs for the simulation that are particular to the wind turbine problem. The first major cost is that associated with updating the mesh to account for turbine rotation. The second dominant cost is that due to reinitialization (reallocating space for matrices), global matrix assembly, and finalization (parallel communication of matrix row indices or graphs) of the linear systems at each time step as a result of mesh motion. These costs are being addressed in our ongoing work. The GMRES iteration counts are lower for *hypre* and reflect faster convergence rates for the C-AMG preconditioner. However, the cost per iteration remains constant or increases slightly

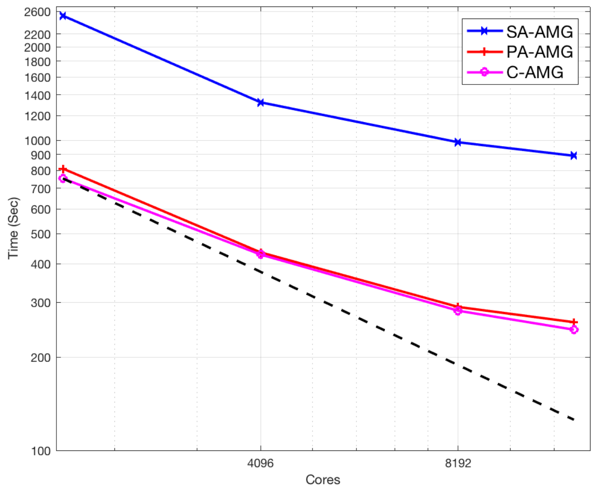

FIG. 3. *Total simulation times as a function of cores (MPI ranks) from Table* 3 *for* 10 *time steps of the $V27$ $R1$ mesh on Cori. Dashed line shows ideal scaling.*

with core count. We note that the MueLu solve times are slightly lower than *hypre* at higher core counts. Setup times are comparable, but now the momentum solve has increased further. The momentum solver cost will become the dominant nonsetup cost at higher CFL numbers.

We now assess SA-AMG and PA-AMG performance for the V27 R1 simulation on the Mira IBM BG/Q supercomputer at Argonne National Laboratory. As before, the simulation was started from a restart and run an additional 10 time steps. Results are reported in Tables 6 and 7. Again, two sweeps of ℓ_1 Gauss–Seidel for presmoothing and for postsmoothing are used in PA-AMG. The continuity solver times for SA-AMG are 10 times larger than PA-AMG, as was observed for Cori. Furthermore, the overall simulation time is now $3\times$ larger than with PA-AMG. The solve and setup for SA-AMG take over 50% of the total simulation time, whereas these represent less than 15% for the PA-AMG simulation on Mira.

As noted previously, it may be possible to improve SA-AMG performance by judiciously modifying the Chebyshev smoother parameters.

4.2. NREL 5-MW turbine simulation results. In this section we analyze Nalu-Wind strong scaling results on the NREL 5-MW G1 and G2 turbine meshes using the Trilinos- and *hypre*-based solver stacks. As before, all results are using an initial restart file, in order to eliminate transient behaviors. We first consider results for simulations using the G1 mesh. Timings for the G1 simulations are provided in Tables 8 and 9. The SA-AMG algorithm with Chebyshev polynomial smoothing was found to be more effective in this case than for the V27 mesh. In particular, the pressure GMRES solver converges with this preconditioner, and the iteration counts are significantly lower than the V27 pressure solves. Here the strength threshold drop tolerance is set to $\theta = 0.02$. For the smoother, two sweeps of an ℓ_1 Gauss–Seidel relaxation are applied. The same rebalancing options are employed as in the V27 simulations. The complexity $C = 1.11$ is slightly smaller for the NREL 5-MW problem than the V27 case. We note that V-cycle complexity C is similar for the

Table 6

Continuity solver settings; iterations; and solve, preconditioner setup, total simulation times (in seconds), and speed-up Sp for 10 *time steps of $V27$ $R1$ simulation on Mira.*

(a) Trilinos continuity & momentum solvers SA-AMG, Chebyshev									
Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	4	0.03	1.25	113	138.0	6576	442	10773	1
8192	4	0.03	1.25	113	146.0	4545	361	7870	1.4
16384	4	0.03	1.25	113	138.0	2897	323	5875	1.8

(b) Trilinos continuity & momentum solvers PA-AMG, ℓ_1 Gauss–Seidel									
Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	2	0.03	1.14	18.1	25.9	762	232	4687	1
8192	2	0.03	1.14	18.1	26.5	449	190	3608	1.3
16384	2	0.03	1.14	18.1	26.3	269	190	3117	1.5

Table 7

Breakdown in seconds of total simulation times listed in Table 6 *(omitting continuity preconditioner setup and solve) for* 10 *time steps of $V27$ $R1$ simulation on Mira. The times are representative of the configurations in Tables* 6(a) *and* (b).

Trilinos continuity & momentum solvers										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
4096	237	79	68	291	579	497	760	646	140	165
8192	411	39	38	440	284	273	384	608	120	75
16384	609	19	24	638	140	156	194	361	120	37

smoothed SA-AMG ($C = 1.12$) and PA-AMG ($C = 1.11$) coarsening algorithms. This was not the case for the V27 problem with $C = 1.25$ for SA-AMG and $C = 1.12$ for PA-AMG. The stencil width S_{max} is also lower for the NREL 5-MW problem ($S_{max} = 19.6$ versus $S_{max} = 113$), leading to reduced setup and solve times.

For the simulations employing *hypre*-BoomerAMG, the momentum solver is GMRES(20). The momentum preconditioner is one sweep of the *hypre*-BoomerAMG l_1 symmetric Gauss–Seidel smoother. The pressure solver is GMRES(20) with the full C-AMG V-cycle, now with one sweep of a symmetric Gauss–Seidel smoother. The *hypre* settings are modified slightly from the V27 case as follows. The smoother is now CF-relaxation. In this scheme, on the fine grid and the down cycle the coarse (C) points are relaxed, followed by the fine (F) points. During the up cycle the F-points are relaxed first, followed by the C-points. The strength threshold was changed to $\theta_C = 0.5$. An interpolation truncation level 0.10 is specified together with a maximum interpolation stencil width of two matrix elements per row. In addition, the non-Galerkin coarse grid sparsification employs the drop tolerances $\gamma = [\,0.0,\ 0.01,\ 0.03\,]$ to maintain the number of nonzeros in the coarse matrices below a threshold.

We first observe that the total NREL 5-MW G1 simulation time using the Trilinos-based solver stack is approximately 20% faster than when using the *hypre*-based solver stack. From Table 9, it can be seen that the difference is primarily due to faster momentum matrix assembly, load, and solve times in Trilinos, rather than any differences in the continuity solve. In fact, the BoomerAMG-based continuity solve itself is approximately 29% faster than the Trilinos-based continuity solve. However, the pressure solve accounts for only 10% of the overall simulation time for the *hypre* solver stack and 18% of the overall simulation time for the Trilinos solver stack. Figure 4 gives a plot of the wall clock run times for the two solver stacks for 10-time-step runs on the Cori machine at NERSC on up to 512 nodes or 16,384 Haswell cores.

Table 8
Continuity solver settings with associated number of solver iterations, preconditioner setup times, solve times, total simulation times (in seconds), and speed-up Sp for 10 *time steps of the NREL* 5*-MW G*1 *simulation on Cori.*

(a) Trilinos continuity & momentum solvers									
PA-AMG, ℓ_1 Gauss–Seidel									
Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	2	0.02	1.07	18.1	66.7	526	83	1819	1
8192	2	0.02	1.07	18.1	47.1	226	51	1038	1.8
16384	2	0.02	1.07	18.1	50.1	152	47	742	2.5
SA-AMG, Chebyshev									
Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	4	0.02	1.15		19.9	311	173	1643	1
8192	4	0.02	1.15		21.6	182	112	1010	1.6
16384	4	0.02	1.15		21.8	111	84	710	2.3
(b) Trilinos continuity & momentum solvers									
SA-AMG, Chebyshev, aggressive rebalancing									
Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	2	0.02	1.16		14.9	139	146	1448	1
8192	2	0.02	1.16		14.4	66	94	881	1.6
16384	2	0.02	1.16		16.2	44	63	611	2.4
(c) *hypre*-BoomerAMG continuity & momentum solver									
C-AMG, symmetric Gauss–Seidel									
Cores	sw	θ_C	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	1	0.50	1.12	23.4	15.0	154	80	1853	1
8192	1	0.50	1.12	23.4	15.0	70	46	1074	1.8
16384	1	0.50	1.12	23.4	15.1	33	36	705	2.6

Table 9
Breakdown in seconds of total simulation times listed in Table 8 *(omitting continuity preconditioner setup and solve) for* 10 *time steps of NREL* 5 *MW G*1 *mesh on Cori. The times are representative of the configurations in Table* 8(a)–(c).

(a) Trilinos continuity & momentum solvers										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
4096	52	28	8	83	119	23	485	290	8	46
8192	45	14	6	62	60	18	234	241	5	24
16384	50	7	4	66	30	13	125	187	2	12
(b) *hypre*-BoomerAMG continuity & momentum solvers										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
4096	3	53	25	10	377	225	553	268	23	52
8192	2	24	19	5	192	172	271	213	5	22
16384	1	12	13	3	105	125	143	186	4	12

We now report solver performance results for the NREL 5-MW wind turbine G2 mesh in Tables 10 and 11. These times are for 10 time step runs after a restart on the Cori machine at NERSC on 1024 nodes or 32,768 Haswell cores. The Trilinos solver stack with PA-AMG preconditioner is compared with the *hypre* momentum and continuity solvers. The authors explored using SA-AMG preconditioning but found that the resulting solver convergence was poor. (This will be subject of future exploration.) For the current results, PA-AMG is employed with aggressive rebalancing and two sweeps of the ℓ_1 Gauss–Seidel smoother. In the case of the continuity equation, the *hypre* pressure solver is 38% faster than Trilinos. The Trilinos momentum solve is roughly 10% faster than the *hypre* solve. The overall run time for the Trilinos-based simulation is roughly 5% slower compared to the *hypre* run.

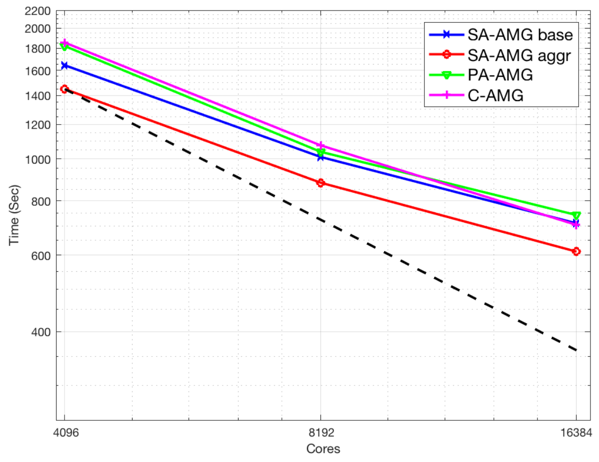

Fig. 4. *Total simulation times as a function of cores (MPI ranks) from Table* 8 *for* 10 *time steps of the NREL* 5*-MW G*1 *mesh on Cori. Dashed line shows ideal scaling.*

Table 10

Continuity solver settings with associated number of solver iterations, preconditioner setup times, solve times, total simulation times (in seconds), and speed-up Sp for 10 *time steps of NREL* 5*-MW G*2 *simulation on Cori.*

(a) Trilinos continuity & momentum solvers PA-AMG, ℓ_1 Gauss–Seidel									
Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
32768	2	0.02	1.07	18.4	40.1	493.6	172.5	3648.5	1
(b) *hypre*-BoomerAMG continuity & momentum solvers C-AMG, symmetric Gauss–Seidel									
Cores	sw	θ_C	C	S_{max}	Iterations	Solve	Setup	Total	Sp
32768	1	0.50	1.13	23.2	26.2	355.5	135.9	3486	1

From the timing breakdowns, we observe that the continuity setup and solve now account for less than 20% of the overall run times. Momentum matrix setup and solve are now dominant costs for both solver stacks. For the Trilinos stack, the continuity matrix initialization, assembly, and load times are also a significant cost.

4.3. Segregated momentum solver approach. Preliminary studies were performed on the NREL 5-MW G1 mesh using a segregated momentum solver. In this approach, a preconditioned GMRES iteration is applied separately for each of the velocity components (u, v, w). The momentum preconditioner is one sweep of the *hypre*-BoomerAMG ℓ_1 symmetric Gauss–Seidel smoother. The pressure solver is GMRES(20) with the full C-AMG V-cycle with the same settings as in the standard approach, which we now refer to as *monolithic*. A comparison of the segregated approach versus the monolithic approach is given in Tables 12 and 13. (The monolithic times are the same as those given in Tables 8 and 9.) The continuity solver times for the segregated *hypre* stack are very similar to the continuity solver times for the monolithic *hypre* preconditioner. However, the overall segregated *hypre* run time is now approximately 39% faster than with the monolithic *hypre* solver stack at 16,384 cores. As can be seen from Table 13, the improvement in overall run time is primarily due to reductions in the assemble, init, and load-complete times for the

Table 11

Breakdown in seconds of total simulation times from Table 10 *(omitting continuity preconditioner setup and solve) for* 10 *time steps of NREL* 5*-MW G*2 *simulation on Cori.*

(a) Trilinos continuity & momentum solvers

Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
32768	406	30.6	12.5	471	131.4	40.3	1353	444	10	51

(b) *hypre*-BoomerAMG continuity solver & momentum solver

Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
32768	3.2	55.1	39.6	11.2	406.7	384.0	1479	432	11	51

Table 12

Continuity solver settings with associated number of solver iterations, preconditioner setup times, solve times, total simulation times (in seconds), and speed-up Sp for 10 *time steps of the NREL* 5*-MW G*1 *simulation on Cori.*

(a) *hypre*-BoomerAMG continuity & segregated momentum solvers
C-AMG, symmetric Gauss–Seidel

Cores	sw	θ	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	1	0.50	1.12	23.4	15.0	151	76	1130	1
8192	1	0.50	1.12	23.4	15.0	67	44	642	1.7
16384	1	0.50	1.12	23.4	15.1	34	36	427	2.6

(b) *hypre*-BoomerAMG continuity & "monolithic" momentum solver
C-AMG, symmetric Gauss–Seidel

Cores	sw	θ_C	C	S_{max}	Iterations	Solve	Setup	Total	Sp
4096	1	0.50	1.12	23.4	15.0	154	80	1853	1
8192	1	0.50	1.12	23.4	15.0	70	46	1074	1.8
16384	1	0.50	1.12	23.4	15.1	33	36	705	2.6

Table 13

Breakdown in seconds of total simulation times listed in Table 12 *(omitting continuity preconditioner setup and solve) for* 10 *time steps of NREL* 5 *MW G*1 *mesh on Cori. The times are representative of the configurations in Table* 12(a).

(a) *hypre*-BoomerAMG continuity & segregated momentum solvers

Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
4096	2	49	25	2	134	34	293	271	10	45
8192	2	24	19	1	67	25	117	214	5	22
16384	1	12	13	1	32	17	59	185	3	11

(b) *hypre*-BoomerAMG continuity & "monolithic" momentum solvers

Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
4096	3	53	25	10	377	225	553	268	23	52
8192	2	24	19	5	192	172	271	213	5	22
16384	1	13	13	3	105	125	144	186	4	12

segregated solver, attributed to the smaller size of the momentum sparse matrix graph representations. Figure 5 gives a plot of the wall clock run times for the monolithic solvers. These times are for 10 time step runs after a restart on the Cori machine at NERSC on up to 512 nodes or 16,384 Haswell cores.

The performance of the *hypre*-BoomerAMG segregated momentum and continuity solvers for the NREL 5-MW wind turbine G2 mesh is reported in Tables 14 and 15. These times are for 10 time step runs after a restart on the Cori machine at NERSC on 1024 nodes or 32,768 Haswell cores. The *hypre* segregated solve is roughly 42%

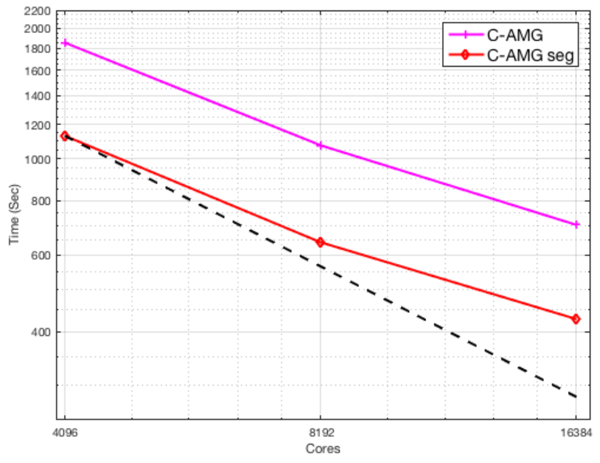

Fig. 5. *Total simulation times as a function of cores (MPI ranks) from Table* 12 *for* 10 *time steps of the NREL* 5*-MW G*1 *mesh on Cori. Dashed line shows ideal scaling.*

Table 14

Continuity solver settings with associated number of solver iterations, preconditioner setup times, solve times, total simulation times (in seconds), and speed-up Sp for 10 *time steps of NREL* 5*-MW G*2 *simulation on Cori.*

(a) *hypre*-BoomerAMG continuity & segregated momentum solvers C-AMG, symmetric Gauss–Seidel									
Cores	*sw*	θ_C	*C*	S_{max}	Iterations	Solve	Setup	Total	*Sp*
32768	1	0.50	1.13	23.2	26.1	349.1	135.1	2193	1
(b) *hypre*-BoomerAMG continuity & “monolithic” momentum solvers C-AMG, symmetric Gauss–Seidel									
Cores	*sw*	θ_C	*C*	S_{max}	Iterations	Solve	Setup	Total	*Sp*
32768	1	0.50	1.13	23.2	26.2	355.5	135.9	3486	1

Table 15

Breakdown in seconds of total simulation times from Table 14 *(omitting continuity preconditioner setup and solve) for* 10 *time steps of NREL* 5*-MW G*2 *simulation on Cori.*

(a) *hypre*-BoomerAMG continuity solver & segregated momentum solver										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
32768	2.8	60.1	42.1	2.8	149.1	52.0	850	432	11	51
(b) *hypre*-BoomerAMG continuity solver & “monolithic” momentum solver										
Cores	Continuity			Momentum				Mesh	I/O	misc
	init	assm	Load	init	assm	Load	Solve			
32768	3.2	55.1	39.6	11.2	406.7	384.0	1479	432	11	51

faster than the *hypre* monolithic momentum solve. The overall run time for the *hypre*-based segregated simulations is almost 63% faster compared to the *hypre*-BoomerAMG monolithic run. The mesh motion now becomes another dominant cost at 432 seconds or roughly 20% of the *hypre* segregated simulation time.

A much longer integration at lower resolution (V27 R0; cf. Figure 2) has also been performed on Cori to demonstrate the fidelity of the model in representing the flow field surrounding the V27 turbine. The target CFL number is 1. The CFL < 1

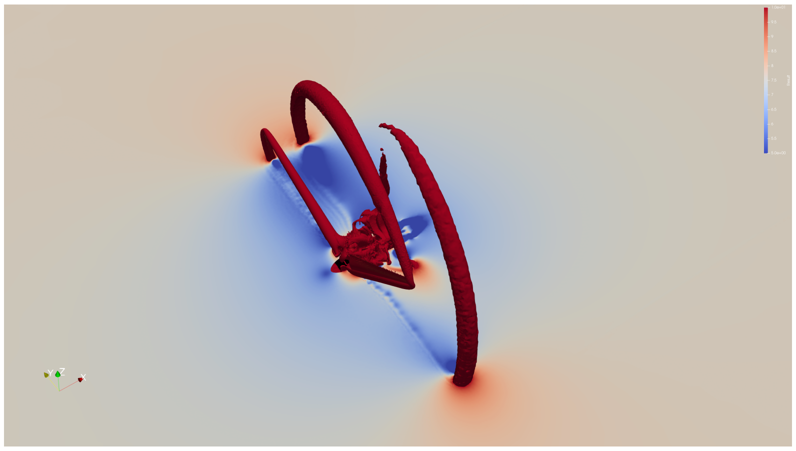

Fig. 6. *Velocity magnitude of the flow field surrounding $V27$ wind turbine at* 0.75 *sec; isosurfaces are for* 10 *m/s velocity magnitude.*

in the vast majority of grid cells. There are very few cells with CFL ≈ 20, and these are all near the tip of the blades on the leading and trailing edges. The high CFL number in these regions is not anticipated to significantly effect the solution. The initial time step is $\Delta t = 5 \times 10^{-6}$ seconds (variable and increasing). In contrast to the previous scaling studies, the BDF-2 time integration was employed. The model was integrated to $t = 0.75$ seconds, representing roughly one half revolution of the turbine. The integration required 7000 time steps and 24 hours of wall clock time. The pressure and velocity are plotted in Figure 6 with a velocity isosurface of 10 m/s magnitude.

5. Conclusion.

In this paper, we performed strong scaling studies with the Nalu-Wind CFD model for Vestas V27 and NREL 5-MW wind turbine simulations in order to compare two fundamental AMG algorithms, namely, classical Ruge–Stüben (C-AMG) and aggregation (smoothed SA-AMG and plain PA-AMG). The simulations were run using Nalu, an open-source low Mach CFD code. Simulations using two different mesh resolutions were run on up to 32,768 Haswell cores on the Cori supercomputer at NERSC for 10 time steps.

Our results demonstrate comparable solve and setup times for both the Trilinos MueLu and *hypre*-BoomerAMG solver stacks when employed as preconditioners for the pressure GMRES solver in the Nalu model. In the case of MueLu, the use of unsmoothed prolongation PA-AMG combined with minimum and maximum aggregate sizes resulted in lower V-cycle complexity C and stencil width $S_{avg}(A_k)$. These are correlated with lower solve and setup times. Consequently, the communication overhead and local operations are reduced. Parallel performance for MueLu was further improved by utilizing a new metric for load-balancing in the V-cycle hierarchy construction.

In the case of *hypre*-BoomerAMG, solver performance at high core counts was improved by the application of sparsification. These are relatively recent additions to the *hypre*-BoomerAMG solver options [12], [3]. Furthermore, GMRES(20) was employed to reduce communication overhead. The complexity C and stencil width S for *hypre* are almost identical to MueLu for the chosen solver parameters applied to the V27 turbine problem. The GMRES iteration counts for *hypre* were lower than with MueLu, but the solve and setup times were comparable. Thus, the cost per iteration of *hypre* is slightly higher with a faster convergence rate.

For the NREL 5-MW wind turbine, the *hypre* solver stack with the monolithic GMRES momentum solver was compared to a segregated *hypre* momentum solver with a separate GMRES iteration for each velocity component and the *hypre*-BoomerAMG l_1 symmetric Gauss–Seidel smoother as the preconditioner. The segregated solver reduces the matrix initialization costs by employing three smaller matrices for the momentum solves. The corresponding sparse matrix graph representations of these matrices are smaller and the associated reinitialization costs are thus reduced. The segregated solver simulation exhibits improved strong scaling characteristics out to a limit of roughly 30K DOFs per MPI rank, despite a slight increase in the momentum solver iteration counts. Furthermore, the overall segregated solver simulation times are 39% less than the monolithic *hypre* solver stack applied to this problem. Improvements to the *hypre*-BoomerAMG settings included one sweep of a symmetric Gauss–Seidel smoother with CF ordering and sparsification leading to a reduced C.

Originally the momentum and continuity solver costs were 75% of the model run time. The optimizations in this paper have reduced these costs to 30% or less. For the *hypre* monolithic and Trilinos solver stack, the scaling of the sparse matrix setup process is a remaining challenge. In the *hypre* solver stack, the challenge is in the matrix assembly and load-complete phases. In the Trilinos stack, the challenge is in the sparse matrix graph initialization. These matrix setup costs are exacerbated by the repeated matrix reinitialization associated with mesh motion. For the segregated momentum solver, the dominant cost becomes the mesh motion algorithm. Future efforts will include implementing segregated momentum solves in the Trilinos solver stack and exploring ways to reduce reinitialization and mesh motion costs. With an increasing CFL number required to achieve more tractable wall clock times for long time integrations, the momentum solver cost becomes important. Further efforts on different preconditioners for momentum will be required. Efforts to mitigate the communication costs in GMRES Krylov solvers are also certainly warranted.

6. Acknowledgments. The authors would like to thank Chris Bruner for providing the V27 meshes and Stefan Domino, the author of Nalu, for his technical contributions. They would also like to thank Ulrike Meier-Yang and the CASC-LLNL *hypre* group for suggestions to optimize *hypre*-BoomerAMG. They would also like to thank Kasia Swirydowicz for a thorough review and helpful suggestions. Nalu version v1.2.0, *hypre* release 2.14.0, and an October/2018 Trilinos developer snapshots were used in our studies.

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under contract DE-AC36-08GO28308. This work was also authored in part by Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This report followed the Sandia National Laboratories formal review and approval process. As such, the technical report is suitable for unlimited release.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-05CH11231. This research also used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357.

The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

REFERENCES

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

[2] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, *Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems*, Scientific Programming, 20 (2012), pp. 241–255.

[3] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, SIAM J. Sci. Comput., 38 (2016), pp. 332–357.

[4] W. Chang, F. Giraldo, and B. Perot, *Analysis of an exact fractional step method*, J. Comput. Phys., 180 (2002), pp. 183–199.

[5] H. De Sterck, R. Falgout, J. Nolting, and U. M. Yang, *Distance-two interpolation for parallel algebraic multigrid*, Numer. Linear Algebra Appl., 15 (2008), pp. 115–139.

[6] 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.

[7] S. Domino, *Toward verification of formal time accuracy for a family of approximate projection methods using the method of manufactured solutions*, in Proceedings of Summer Workshop, Center for Turbulence Research, Stanford, 2006, pp. 387–396.

[8] S. Domino, *A comparison between low-order and higher-order low-Mach discretization approaches*, in Studying Turbulence Using Numerical Simulation Databases — XV, P. Moin and J. Urzay, eds., Stanford Center for Turbulence Research, Stanford, CA, 2014, pp. 387–396.

[9] S. Domino, *Sierra Low Mach Module: Nalu Theory Manual, Version* 1.0, Technical report SAND2015-3107W, Sandia National Laboratories, Albuquerque, NM, 2015.

[10] S. Domino, *Design-order, non-conformal low-Mach fluid algorithms using a hybrid CVFEM/DG approach*, J. Comput. Phys., 359 (2018), pp. 331–351.

[11] H. Elman, V. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro, *A taxonomy and comparison of parallel block multilevel preconditioners for the incompressible Navier-Stokes equations*, J. Comput. Phys., 227 (2008), pp. 1790–1808.

[12] R. D. Falgout and J. B. Schroder, *Non-Galerkin coarse grids for algebraic multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. 309–334.

[13] R. D. Falgout and U. M. Yang, *hypre: A library of high performance preconditioners*, in Computational Science ICCS 2002, Lecture Notes in Comput. Sci. 2331, P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, eds., Springer, Berlin, Heidelberg, 2002, pp. 632–641.

[14] M. W. Gee, J. J. Hu, and R. S. Tuminaro, *A new smoothed aggregation multigrid method for anisotropic problems*, Numer. Linear Algebra Appl., 16 (2009), pp. 19–37.

[15] S. W. Hammond, M. A. Sprague, D. Womble, and M. Barone, *A2e High Fidelity Modeling: Strategic Planning Meetings*, Technical report NREL/TP-2C00-64697, National Renewable Energy Laboratory, 2015, https://www.nrel.gov/docs/fy16osti/64697.pdf.

[16] V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2000), pp. 155–177.

[17] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Software, 31 (2005), pp. 397–423.

[18] J. J. Hu, S. J. Thomas, C. R. Dohrmann, S. Ananthan, S. P. Domino, A. B. Williams, and M. A. Sprague, *Decrease Time-to-Solution Through Improved Linear-System Setup and Solve*, Technical Report SAND2018-3204R, Sandia National Laboratories, Albuquerque, NM, 2018, doi:https://doi.org/10.2172/1431236.

[19] J. Jonkman, S. Butterfield, W. Musial, and G. Scott, *Definition of a 5-MW Reference Wind Turbine for Offshore System Development*, Technical report NREL/TP-500-38060, National Renewable Energy Laboratory, Golden, CO, 2009.

[20] F. Nicoud and F. Ducros, *Subgrid-scale stress modelling based on the square of the velocity gradient tensor*, Flow Turbulence Combustion, 62 (1999), pp. 183–200.

[21] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki, *SuperLU Users Guide*, Lawrence Berkeley National Laboratory, Berkeley, CA, 2011.

[22] P. Lin, M. Bettencourt, S. Domino, T. Fisher, M. Hoemmen, J. J. Hu, E. Phipps, A. Prokopenko, S. Rajamanickam, C. Siefert, and S. Kennon, *Towards extreme-scale simulations for low Mach fluids with second-generation Trilinos*, Parallel Process. Lett., 24 (2014).

[23] P. T. Lin, J. N. Shadid, J. J. Hu, R. P. Pawlowski, and E. C. Cyr, *Performance of fully-coupled algebraic multigrid preconditioners for large-scale VMS resistive MHD*, J. Comput. Appl. Math., 344 (2018), pp. 782–793, doi:https://doi.org/10.1016/j.cam.2017.09.028.

[24] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[25] S. V. Patankar and D. B. Spalding, *A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows*, Int. J. Heat Mass Transfer, 15 (1972), pp. 1787–1806.

[26] J. B. Perot, *An analysis of the fractional step method*, J. Comput. Phys., 108 (1993), pp. 51–58.

[27] A. Prokopenko, J. J. Hu, T. A. Wiesner, C. M. Siefert, and R. S. Tuminaro, *MueLu Users Guide* 1.0, Sandia Report SAND2014-18874, Sandia National Laboratories, Albuquerque, NM, 2014.

[28] A. Quarteroni, F. Saleri, and A. Veneziani, *Factorization methods for the numerical approximation of Navier-Stokes equations*, Comput. Methods Appl. Mech. Engrg., 188 (2000), pp. 505–526.

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

[30] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[31] M. A. Sprague, S. Boldyrev, P. Fischer, R. Grout, W. Gustafson, Jr., and R. Moser, *Turbulent Flow Simulation at the Exascale: Opportunities and Challenges Workshop*, Technical report NREL/TP-2C00-67648, National Renewable Energy Laboratory, 2017, https://www.nrel.gov/docs/fy17osti/67648.pdf.

[32] K. Stüben, *A review of algebraic multigrid*, Comput. Appl. Math., 128 (2001), pp. 281–309.

[33] S. Turek, *On discrete projection methods for the incompressible Navier-Stokes equations: An algorithmical approach*, Comput. Methods Appl. Mech. Engrg., 143 (1997), pp. 271–288.

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

[35] U. M. Yang, *Parallel algebraic multigrid methods — high performance preconditioners*, in Numerical Solution of Partial Differential Equations on Parallel Computers, A. M. Bruaset and A. Tveito, eds., Lect. Notes Comput. Sci. Eng. 51. Springer, Berlin, 2006, pp. 209–236.

[36] U. M. Yang, *On long range interpolation operators for aggressive coarsening*, Numer. Linear Algebra Appl., 17 (2010), pp. 453–472.