SAND2018-6759R

EXASCALE COMPUTING PROJECT

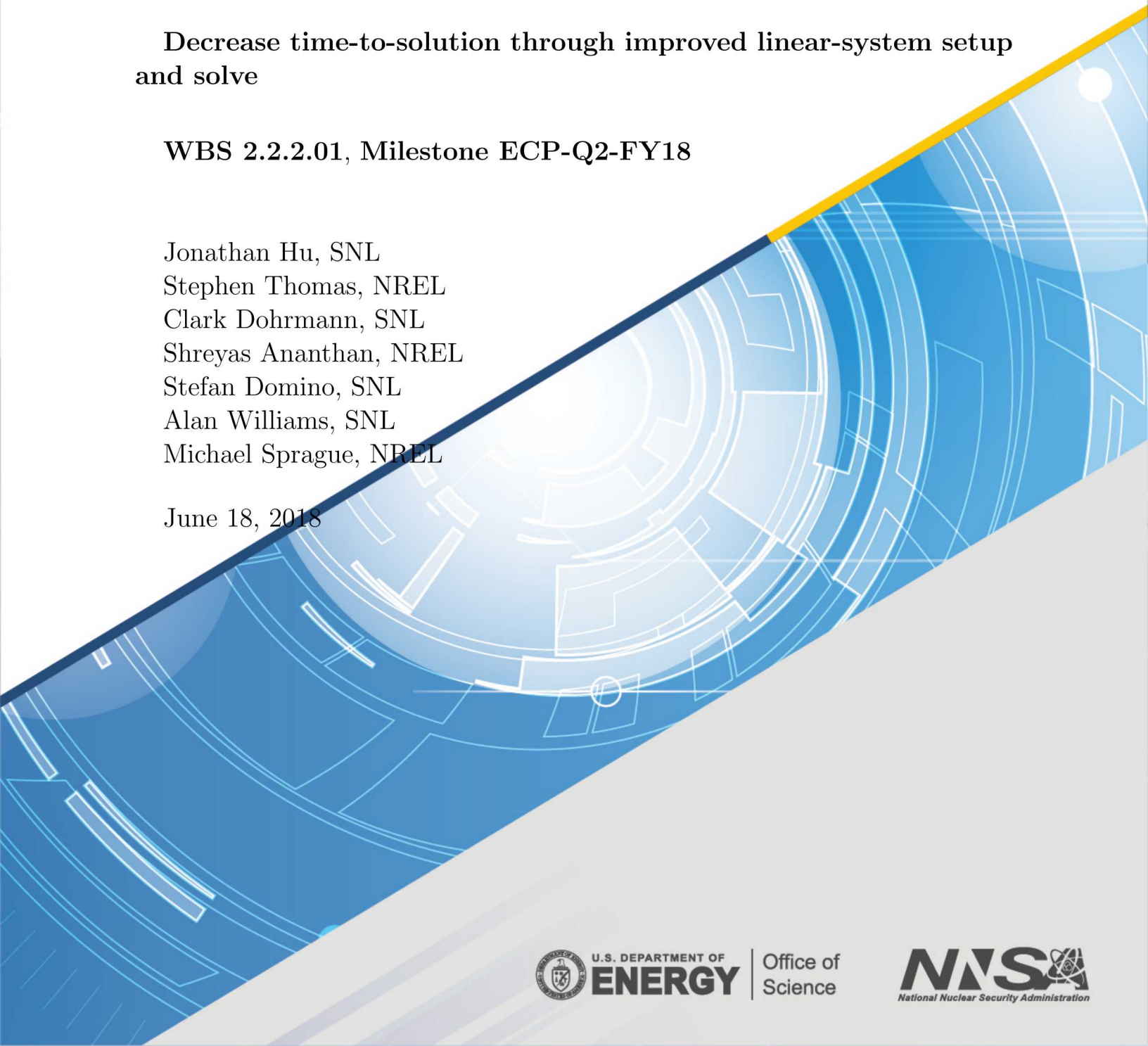
ECP-U-2018-xxx

Decrease time-to-solution through improved linear-system setup and solve

WBS 2.2.2.01, Milestone **ECP-Q2-FY18**

Jonathan Hu, SNL
Stephen Thomas, NREL
Clark Dohrmann, SNL
Shreyas Ananthan, NREL
Stefan Domino, SNL
Alan Williams, SNL
Michael Sprague, NREL

June 18, 2018



U.S. DEPARTMENT OF ENERGY | Office of Science

NNSA National Nuclear Security Administration

ECP-U-2018-xxx

ECP Milestone Report
Decrease time-to-solution through improved linear-system setup and solve
WBS 2.2.2.01, Milestone ECP-Q2-FY18

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

June 18, 2018

ECP Milestone Report
Decrease time-to-solution through improved linear-system setup and solve
WBS 2.2.2.01, Milestone ECP-Q2-FY18

APPROVALS

Submitted by:

18 June 2018

Jonathan Hu, SNL
ECP-Q2-FY18

Date

Concurrence:

18 June 2018

Michael A. Sprague, ExaWind PI
NREL

Date

Approval:

Thomas Evans
ORNL

Date

REVISION LOG

Version	Creation Date	Description	Approval Date
1.0	2018-03-26	Original	
1.1	2018-05-XX	Revision based on comments by T. Evans	

EXECUTIVE SUMMARY

The goal of the ExaWind project is to enable predictive simulations of wind farms composed of many MW-scale turbines situated in complex terrain. Predictive simulations will require computational fluid dynamics (CFD) simulations for which the mesh resolves the geometry of the turbines, and captures the rotation and large deflections of blades. Whereas such simulations for a single turbine are arguably petascale class, multi-turbine wind farm simulations will require exascale-class resources.

The primary code in the ExaWind project is Nalu, which is an unstructured-grid solver for the acoustically-incompressible Navier-Stokes equations, and mass continuity is maintained through pressure projection. The model consists of the mass-continuity Poisson-type equation for pressure and a momentum equation for the velocity. For such modeling approaches, simulation times are dominated by linear-system setup and solution for the continuity and momentum systems. For the ExaWind challenge problem, the moving meshes greatly affect overall solver costs as re-initialization of matrices and re-computation of preconditioners is required at every time step

We describe in this report our efforts to decrease the setup and solution time for the mass-continuity Poisson system with respect to the benchmark timing results reported in FY18 Q1. In particular, we investigate improving and evaluating two types of algebraic multigrid (AMG) preconditioners: Classical Ruge-Stüben AMG (C-AMG) and smoothed-aggregation AMG (SA-AMG), which are implemented in the Hypre and Trilinos/MueLu software stacks, respectively. Preconditioner performance was optimized through existing capabilities and settings. In the case of MueLu, the introduction of unsmoothed aggregation combined with minimum and maximum aggregate sizes resulted in lower V-cycle complexity and stencil width. Additionally, we examined reducing the number of L_1 Gauss-Seidel smoother sweeps from four to two. In the case of Hypre-BoomerAMG, we examined the inclusion of non-Galerkin coarse grids or sparsification. Performance improvements were tested and demonstrated on a single KW-scale turbine simulation, for which the blades, nacelle, and tower were resolved. The rotating blades are accomodated with a sliding-mesh interface described in our FY18-Q1 milestone report [12]. The computational cost of moving meshes is significant, which requires re-initialization of matrices and re-computation of preconditioners at every time step. The model has 229M pressure degrees-of-freedom, and performance-testing examined strong scaling up to 12K cores on NERSC-Cori. Additionally, we describe deployment of BDDC (Balancing Domain Decomposition by Constraints) as a new preconditioner capability for Nalu. Preliminary testing with BDDC focused on an atmospheric boundary layer simulation on a uniform grid and a heat-conduction problem with high-aspect-ratio elements.

Testing demonstrated that Trilinos Belos/MueLu linear solver cost for setup and solve decreased significantly at all core counts. For example, at 12K cores, MueLu setup and solve decreased 36% from the values reported in FY18 Q1. For Hypre-BoomerAMG, setup and solve remained about the same at the lower core counts, but at 12K cores, setup and solve time decreased by about 29%. While Hypre/BoomerAMG setup and solve times are lower than those of MueLu/Belos at smaller core counts, MueLu/Belos setup and solve times are lower at 12K core. The times between the two solver stacks are much more comparable than those shown in FY18 Q1. With the improvements to MueLu and Hypre at the core counts and the model examined here, pressure-system setup and solve times are now on par with those momentum equations. The most expensive aspect in the Nalu simulation using the Belos/SGS solver is now the `init` operation for matrix initialization; reducing `init` time is one focus of FY18 Q3 efforts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1. INTRODUCTION

The goal of the ExaWind project is to enable predictive simulations of wind farms composed of many MW-scale turbines situated in complex terrain. Predictive simulations will require computational fluid dynamics (CFD) simulations for which the mesh resolves the geometry of the turbines, and captures the rotation and large deflections of blades. Whereas such simulations for a single turbine are arguably petascale class, multi-turbine wind farm simulations will require exascale-class resources.

Fluid dynamics in wind farms are essentially incompressible, with Mach numbers less than 0.3 (with maximum flow velocity near the blades). Our chosen model is the acoustically-incompressible Navier-Stokes equations, where mass-continuity is maintained through pressure projection. The model consists of the mass-continuity Poisson-type equation for pressure and a momentum equation for the velocity. For such modeling approaches, simulation times are dominated by linear-system setup and solution for the continuity and momentum systems.

Our CFD solver is the Nalu code, which employs the Sierra Toolkit as described in [10]. The governing equations are discretized in space on generalized topological meshes with either a low-order control volume finite element method (CVFEM) or an edge-based vertex-centered (EBVC) scheme. A high-order CVFEM implementation is available for hexahedral elements. For time-stepping an approximate pressure-projection scheme is applied together with implicit first- or second-order backwards-differentiation time integration. Splitting errors are controlled by fixed-point iteration and, when second-order BDF2 time discretization is employed, the overall scheme demonstrated second-order accuracy. Various sub-grid-scale turbulence models (e.g., wall-adapting local eddy-viscosity (WALE)) are available in order to represent the turbulent energy cascade and dissipation range, though results reported here are limited to the WALE model. Advection stabilization is also applied in order to mitigate spurious oscillations.

For the ExaWind wind turbine simulation capability, both sliding-mesh and overset methods are implemented to capture the rotating blades. The sliding-mesh approach is employed in work described in this report. In a sliding-mesh configuration, a prescribed mesh interface is defined to handle the solid body movement/rotation. Recently, a second-order hybrid CVFEM/discontinuous-Galerkin (DG) scheme has been implemented for wind turbine fluid flow simulations [11]. In this scheme, the DG discretization is applied at the sliding-mesh interface. The computational cost of moving meshes is significant, which includes re-initialization of matrices and re-computation of preconditioners at every time step. These costs are important when evaluating different linear system solvers. In particular, it may be advantageous to obtain a lower set-up time in exchange for a higher solve time for the preconditioner in order to reduce the overall solution time. These costs will be considered in our evaluation of algebraic multigrid (AMG) preconditioners for the continuity equation.

Multigrid methods are effective scalable solvers that are well suited for high performance computer architectures [20]. When employed as a stand-alone solver, or as a preconditioner for a Krylov iteration such as GMRES [25], multigrid methods have the potential to solve a linear system with n unknowns in a minimum number of $\mathcal{O}(n)$ operations. In contrast to geometric multigrid, which would require constructing a sequence of coarser meshes, algebraic multigrid is applied directly to the linear system $Ax = b$, resulting in a hierarchy of coarser matrices, and thus enabling the solution of unstructured problems arising from a finite-volume discretization. In this report, we turn our attention to the performance of the iterative solvers. The nonsymmetric GMRES Krylov iteration is employed to solve both the momentum and mass-continuity equations in the Nalu code for the Navier-Stokes equations. The momentum system is indefinite and a Gauss-Seidel relaxation scheme is employed as the preconditioner. In this report, we focus on improving and comparing two approaches to AMG as employed as a preconditioner for the mass-continuity equation GMRES Krylov solver: Classical Ruge-Stüben AMG (C-AMG) and smoothed-aggregation AMG (SA-AMG). SA-AMG [23] is provided by the Trilinos-MueLu solver framework, whereas C-AMG is provided by the Hypre BoomerAMG library from CASC-LLNL [15]. Our test problem is a KW-scale turbine, for which the mesh resolves the rotor, nacelle, and tower. Simulations and linear-solver times were reported in the ExaWind FY18 Q1 milestone report [12]. In this report we compare new timing results with those reported in FY18 Q1. In addition to our work with Hypre and MueLu solver stacks, we describe preliminary work deploying a Balancing Domain Decomposition by Constraints (BDDC) preconditioner [8, 19]. BDDC is an attractive preconditioner due to its ability to to perform work concurrently at the different levels of the multilevel hierarchy using disjoint MPI subcommunicators, which could be a critical asset in exascale-class wind turbine

simulations.

2. MILESTONE DESCRIPTION

Current simulations are dominated by linear-system solver costs for the continuity (pressure-Poisson) and momentum systems, including initialization, preconditioner setup, and iterative solution. Our goal in this milestone is to demonstrate reduced overall time-to-solution with respect to baseline performance data established in preceding ExaWind milestones (e.g., V27 turbine simulations, McAlister-blade wind tunnel simulations, and atmospheric-boundary-layer simulations). This epic will focus on "flat" MPI simulations, which are critical for the success of blade-resolved single-turbine simulations (FY18 Q4 Milestone) that will be run on the Mira supercomputer under an INCITE allocation. Our efforts to decrease time to solution will primarily focus on two solver packages, Hypre and MueLu/Belos (part of Trilinos), but there will be initial efforts in exploring the effectiveness of the BDDC (balancing domain decomposition by constraints) approach. Our specific activities are as follows:

- Improve time to solution with the Belos/MueLu solver stack:
  - Optimize smoother selection through cost-benefit analysis of effectiveness vs. setup;
  - Reduce solver apply times through approaches such as alternate cycling strategies and load-balancing strategies;
  - Explore potentially non-optimal multigrid methods with less expensive setup costs;
- Improve performance of Hypre solver stack:
  - Evaluate the optimal settings for BoomerAMG through aggressive coarsening strategies published in literature;
  - Evaluate recently published (2016) non-Galerkin 'sparsification strategies for Hypre to further reduce complexity;
  - Implement ICGS-GMRES and evaluate potential reduction in global communication (stretch goal);
  - Evaluate single level, and multi-level approaches for the momentum equations solve.
  - Determine the optimal preconditioner+solver settings through cost-benefit analysis;
- Explore the effectiveness of BDDC solver on a wind turbine simulation problem.

Completion Criteria: Technical report describing the milestone accomplishment, as well as a memo and a highlight slide summarizing those accomplishments.

3. ALGEBRAIC MULTIGRID

The goal of algebraic multigrid (AMG) is to accelerate the solution of a linear system $Ax = b$ of interest through corrections calculated on a *hierarchy* of coarser matrix systems. While AMG was originally used as a solver, it is common practice now to use it as a preconditioner to a Krylov method.

The setup of phase of AMG is nontrivial, as the proper *grid transfer* matrices to move information effectively between the various coarse-grid systems must be calculated. Once these transfer matrices are determined, the coarse-matrix representations can be recursively computed from A through sparse matrix-matrix multiplication. In the setup phase, the coarse-grid variables, interpolation operators P_l, restriction operators R_l ($R_l = P_l^T$ for symmetric problems), and the coarse-grid matrices $A_{l+1} = R_l A_l P_l$ are determined for $l = 0, 1, \ldots m$ levels, where $A_0 = A$.

In the AMG solve phase, a few steps of a smoother, e.g., Gauss-Seidel, polynomial, or incomplete factorizations, are applied to the finest-level linear system with a zero initial guess. This is referred to as *pre-smoothing*. A residual is calculated and restricted to the next coarser level, where 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

grid 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 *post-smoothing* 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 1 for the complete algorithm.

Algorithm 1 Multigrid single cycle algorithm ($\mu = 1$ yields V-cycle, $\mu = 2$ yields W-cycle) for solving $Au = f$. The hierarchy has with N levels. For reference consult Gee et. al. [14]

```
//Solve Au = f.
Set u = 0.
Set μ = 1 for V-cycle, μ = 2 for W-cycle.
call MULTILEVEL(A, u, f, 0, μ).

function MULTILEVEL(A_k, b, u, k, μ)
    // Solve A_k u = b (k is current grid level)
    // Pre-smoothing step
    u = S_k^1(A_k, b, u)
    if (k ≠ N − 1) then
        // P_k is the interpolant for A_k
        // R_k is the restrictor for A_k
        r̂_{k+1} = R_k(b − A_k u)
        A_{k+1} = R_k A_k P_k
        v = 0
        for i = 1 ... μ do
            MULTILEVEL(Â_{k+1}, r̂_{k+1}, v, k + 1, μ)
        end for
        u = u + P_k v
        // Post-smoothing step
        u = S_k^2(A_k, b, u)
    end if
end function
```

There are two common 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 the sum,

$$C = \sum_{l=0}^{m} nnz(A_l)/nnz(A)\,,$$

where $nnz(B)$ is the number of nonzeros of a matrix B. An AMG solver is scalable when the number of iterations to converge is $\mathcal{O}(1)$ and the operator complexity is $\mathcal{O}(n)$, for n unknowns. We note that this notion of complexity does not account for smoother complexity, e.g., if the smoother was an incomplete factorization.

The complexity measure indicates the amount of memory required. To reduce memory utilization, the complexity C should remain small. The complexity affects the number of operations per V-cycle in the solver. Small operator complexities lead to small V-cycle times. The average stencil size $s(A_l)$ is the average number of non-zero elements per row of A_l. Even when the stencil sizes 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 grid 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[1]. There are two main variants of AMG, smoothed-aggregation AMG (SA-AMG) and classical Ruge-Stüben AMG (C-AMG), which will be discussed in Sections 4 and 5, respectively.

4. SMOOTHED AGGREGATION ALGEBRAIC MULTIGRID

As in other AMG approaches, the main challenge for smoothed-aggregation (SA) AMG is in algebraically forming an appropriate grid transfer (interpolation) matrix P. SA-AMG coarsens a matrix A by grouping unknowns into *aggregates* that correspond to the unknowns used on the next level [27]. 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,

$$N_i(\theta) = \{\, j : |a_{ij}| \geq \theta \sqrt{a_{ii}\, a_{jj}} \,\}, \tag{1}$$

where θ is a problem-dependent scalar-valued 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 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 of the matrix. For canonical problems these modes are known. For other types of problems, these modes are application supplied. By default, most SA-AMG codes default to using the constant vector. The specified near-nullspace modes are rewritten as a matrix $\widetilde{P}$ so that they have local support over the aggregates $N_i(\theta)$. $\widetilde{P}$ is called the *tentative prolongator*.

The tentative prolongator is improved by applying a single step of Jacobi smoothing:

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

where $D = \mathrm{diag}(A)$. The matrix A^F reflects connections that may have been dropped via 1. Entries a^F_{ij} of A^F are given by

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

In the Exawind project, smoothed-aggregation (SA) AMG is provided by the MueLu library [23], part of the Sandia Trilinos project [16].

5. RUGE-STÜBEN ALGEBRAIC MULTIGRID

Classic Ruge-Stüben AMG (C-AMG) is setup in two passes. The first pass will make a selection of coarse nodes based on the number of strong connections that each node has. The second pass is a refinement pass. It checks to make sure there are enough coarse nodes that information is not lost. The classical Ruge-Stüben algorithm defines strongly connected nodes as follows. Point j is a neighbor of i if and only if there is a non-zero a_{ij} . Point j strongly influences i if and only if

$$|a_{ij}| \geq \theta \max_{k \neq i} |\, a_{ik}\,|,$$

where θ is the strength of connection threshold. 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_l and F_l be the coarse and fine points selected at level l, and let n_l be the number of grid points at level l ($n_0 = n$). Then, $n_l = |C_l| + |F_l|$, $n_{l+1} = |C_l|$, A_l is a $n_l \times n_l$ matrix, and P_l is a $n_l \times n_{l+1}$ matrix.

[1]The Trilinos-Belos solver framework implements the iterated classical Gram-Schmidt ICGS-GMRES algorithm and requires two passes of Gram-Schmidt to recover the stability of the modified Gram-Schmidt (MGS) algorithm [4]. However, the ICGS-GMRES variant requires less communication than the MGS-GMRES provided by Hypre, which requires separate inner products for the Krylov basis.

Here, the coarsening is performed row-wise 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 a 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 relative decrease of the number of unknowns on the next coarse level).

There has been extensive research on different variants of AMG since the development of the first AMG algorithms [24] with details presented in Stüben [26]. The original Ruge-Stüben interpolation based approach is now referred to as classical AMG (C-AMG). One of the drawbacks with the classical 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 a sub-domain and the smoother is local. Consequently, efforts were made to coarsen more aggressively to reduce operator complexities [20]. More aggressive coarsening leads to often considerably reduced convergence, because it violates conditions required for classical interpolation. Convergence can be improved again by combining more aggressive coarsening with long distance interpolation [21]. Non-Galerkin coarse grids, for which the coarse-level operator does not satisfy the relationship $A_{l+1} = P_l^T A_l P_l$ for each level l, were introduced by Falgout and Schroder [13]. These employ sparsity patterns and drop tolerances γ applied to specific levels [3] to reduce the complexity C through sparsification. In the Exawind project, classical C-AMG is implemented by the Hypre BoomerAMG library from CASC-LLNL [15].

6. BALANCING DOMAIN DECOMPOSITION BY CONSTRAINTS

Balancing Domain Decomposition by Constraints (BDDC) [8, 19] is a popular domain decomposition preconditioner, which has been applied to a variety of problems including the Poisson equation, linear elasticity, Stokes flow, Reissner-Mindlin plates, porous-media flow, problems in the function spaces $H(curl)$ and $H(div)$, and several others (see, e.g., [22] for a more complete list and references). A recent multilevel implementation of BDDC was shown to have excellent weak scalability for 3D Poisson problems on up to 458K IBM BG/Q cores [1]. A key to the success of that implementation was the ability to perform work concurrently at the different levels of the multilevel hierarchy using disjoint MPI subcommunicators. This important feature of BDDC could turn out to be key to the success of exascale wind turbine simulations.

BDDC works with a decomposition of the original problem into smaller subdomains. Computational work is done in parallel over different MPI ranks, and one or more coarse levels are used for numerical scalability of the approach (i.e., number of iterations is bounded as problem size grows). Compared with multigrid approaches such as those in MueLu or Hypre, coarsening between levels can be done much more aggressively (e.g., tens of times in each direction rather than 3 or 2 typical of multigrid approaches). Consequently, the number of coarse levels for BDDC is often much smaller when compared with multigrid for a fixed problem size. For example, the results in [1] showed remarkable scalability on up to the 458K cores using only two coarse levels. Consequently, the number of levels over which stencil sizes may grow is reduced.

The driving goal for BDDC this fiscal year is to demonstrate its viability as a solver for the continuity (pressure-Poisson) equations. To this end, work has progressed on two fronts. First, an interface between BDDC and Nalu was developed. Although not yet complete for problems with sliding meshes or overset grids, we do not anticipate any insurmountable problems moving forward. For example, there is already supporting theory and numerical results for discontinuous-Galerkin formulations [7] similar to the one used by Nalu for sliding meshes. On the second front, the Trilinos BDDC implementation was refactored to address issues related to excessive matrix initialization times identified in the project Q1 report. The original approach to constructing coarse levels for BDDC relied heavily on `Tpetra::CrsGraph` and `CrsMatrix` classes, but in retrospect these classes aren't a very good fit since BDDC works with individual subdomain matrices rather than a globally assembled one. In addition, ongoing work in FY18 Q2 by others suggests that efficiencies could be gained by taking advantage of available information in Nalu (e.g., which processors share nodes in the original mesh decomposition).

Some features of BDDC we hope will make it an attractive option for exascale simulations are listed below.

1. Concurrent work across levels: As noted earlier, the key to the excellent scaling in [1] at large scales

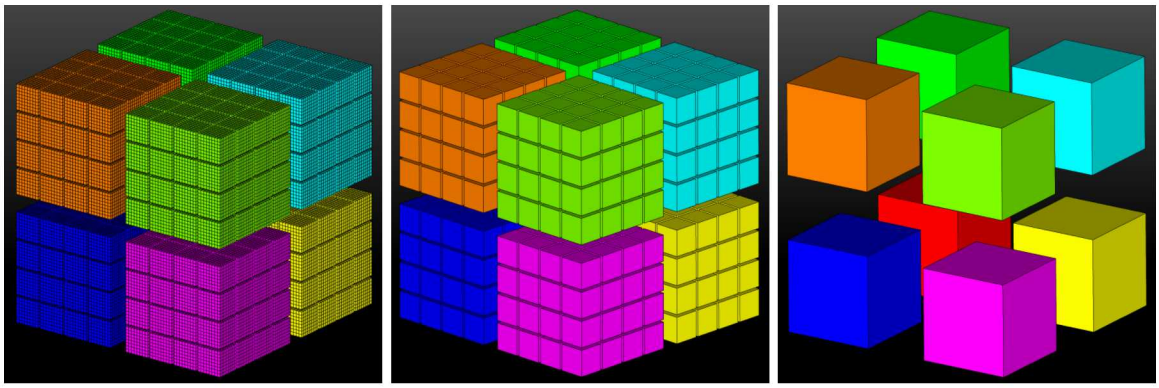


Figure 1: Decomposition for BDDC of 8 subregions into 64 smaller subdomains each (a), coarsening of each subdomain into coarse elements (b), and coarsening of each coarse element into coarser elements (c). Note that each colored block in the figures is associated with a single computational node.

was the ability to do work concurrently at the different coarse levels. This is possible because BDDC is an *additive* rather than a *multiplicative* preconditioner. In contrast to the V-cycle often used by multigrid, computations can commence on the next coarser level prior to completion of all the work at the current level. This opens up the possibility for reducing idle times for cores at all levels.

2. Reduced MPI communications during setup: To help explain this idea, consider a machine that has 8000 computational nodes and 64 processors per node. A unit cube domain is decomposed into 8000 smaller cubic subregions (one per node, 20 in each direction), and each of these subregions is decomposed further into 64 smaller cubic subdomains (an example decomposition for 8 nodes is shown in Figure 1a). The first coarse level reduces each subdomain to a single coarse element (Figure 1b), while the second coarse level reduces each of the coarse elements to coarser elements (Figure 1c). During this part of the initialization phase there is no need for any parallel communication since the coarsening is local to each node. Assembly of the elements at the coarsest level does require MPI communications, but for this simple example they are modest. Specifically, since each of the coarsest elements (see Figure 1c) has just 8 degrees of freedom for the Poisson equation, communication of stiffness matrices at the coarsest level only involves $8 \times 8 = 64$ doubles per computational node.

3. Reduced number of MPI ranks: Assigning a single MPI process to each computational node rather than to each subdomain can reduce the number of MPI ranks significantly. For the example just considered, the reduction is a factor of 64. The potential advantage of using fewer MPI processes by employing an MPI+X approach, e.g., MPI+OpenMP threads, is discussed in [18].

4. Threading opportunities: Since many of the computations for BDDC can be done independently at the subdomain level, it is straightforward for them to be threaded. Threading of certain kernels such as sparse direct solvers and dense linear algebra operations are also possible.

In the ExaWind project, the BDDC preconditioner/solver is provided by the Sandia Trilinos project [16] as part of the ShyLU package.

7. MULTIGRID RESULTS

Many past efforts have focused on linear solver *weak scaling* behavior, where the linear solver work per core is fixed [17]. The current effort is focused on linear solver *strong scaling*, in which the global problem size is fixed, and the number of compute cores is varied. In this section, we present improvements to AMG strong scaling performance for Trilinos/MueLu and Hypre/BoomerAMG for a V27 mesh.

The V27 mesh under consideration has 541,009,944 elements and $229,267,585$ nodes. In particular, the mesh consists of two regions, each with a mix of hexahedral, wedge, pyramid, and tetrahedral elements. The

(a) Executable and settings from FY18 Q1 Report							
					time (seconds)		
cores	sw	C	Iterations	θ	Solve	Set-up	Wall-clock
2048	4	1.13	24.4	0.02	246.1	47.7	1052.5
4096	4	1.13	24.9	0.02	137.4	29.4	603.1
8192	4	1.13	26.2	0.02	82.9	27.6	410.4
12288	4	1.13	26.4	0.02	62.9	40.5	386.4
(b) Executable and settings for newest Nalu executable, same settings as FY18 Q1							
					time (seconds)		
cores	sw	C	Iterations	θ	Solve	Set-up	Wall-clock
2048	4	1.13	25.2	0.02	257.3	38.9	1048.5
4096	4	1.13	25.6	0.02	146.1	23.3	600.8
8192	4	1.13	27.1	0.02	91.6	16.9	405.7
12288	4	1.13	27.2	0.02	68.5	17.3	372.3
(c) Executable and settings for newest Nalu executable, newest settings							
					time (seconds)		
cores	sw	C	Iterations	θ	Solve	Set-up	Wall-clock
2048	2	1.15	26.0	0.03	169.2	47.7	966.3
4096	2	1.15	25.4	0.03	93.3	30.6	557.5
8192	2	1.15	25.8	0.03	54.9	24.2	392.5
12288	2	1.15	25.8	0.03	42.7	23.3	347.4

Table 1: Simulation times reported for 10 time steps of the V27 41 R1 500M element mesh. Baseline Belos/MueLu solver performance in Nalu, FY2018, Quarter 1 as well as performance with the latest Nalu executable and newest MueLu settings.

blocks of tetrahedral elements are a result of mesh refinement. The time integrator is first order with initial time step $\Delta t = 5 \times 10^{-6}$.

In order to avoid transient simulation startup effects, a restart file was generated after 20 timesteps for all core counts. All linear solver timings in this section were then generated from a simulation restart that was run an additional 10 timesteps, with the exception of a final experiment that was run with an additional 100 timesteps. We note that as physics changes, e.g., establishment of wake vortices, demands on the linear solver may change. Accounting for such changes is beyond the scope of the current discussion.

All simulations in this section were run on the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC) in the Phase I partition. Each node in this partition contains two 16-core Xeon E5-2698 v3 2.3 GHz Haswell CPUs. The interconnect is a Cray Aries network with Dragonfly topology.

7.1 TRILINOS/MUELU DISCUSSION

We first provide timing results using the same Nalu executable and input decks as from the Exawind FY2018 Q1 milestone report [12] in order to establish a baseline for demonstrating improvements made in FY2018 Q2. The FY2018 Q1 executable was built with Trilinos master version SHA1 `85dddaa`. The Nalu version was not captured. Table 1 shows timings for the continuity linear solver setup and apply for 10 time steps.

The continuity solve was GMRES preconditioned by unsmoothed $\omega = 0$ aggregation-based algebraic multigrid (AMG). Each preconditioner application was a single V(4, 4) cycle[2] with $l1$ Gauss-Seidel[2] smoothing. Setup refers to construction of the multigrid preconditioner, and apply (or solve) refers to the application of the preconditioner during Krylov iterations. As can be seen from the results, the multigrid setup times stagnate from 4096 to 8192 cores, and actually increase from 8192 to 12288 cores. The solve times decrease

[2]V cycle with four pre-smoothing and four post-smoothing sweeps

monotonically as the core count increases. However, note that the setup times remain smaller than the solve times. (In our experience in multigrid weak scaling, setup is often as expensive as solve.)

We first note that there have been two changes to the MueLu codebase that are not reflected in the data in Table 1(a). The first optimization eliminated the unnecessary construction of an `Import` object (used in transforming the data layout of maps, vectors, and matrices, for example) in the matrix filtering phase of the SA-AMG setup. The second optimization was in the multigrid solve phase, and added caching of coarse level residual and solution vectors, thus eliminating redundant multivector allocations. The effect of these changes are given in Table 1(b). The newest Nalu executable was run with the same inputs as in the FY2018 Q1 report. The AMG setup times in Table 1(b), compared to those reported in the Q1 report (shown here in Table 1(a)), improved from 22% at 2048 cores to 57% at 12288 cores, whereas the solve times slowed from 4% at 2048 cores to 10% at 12288 cores. The solve time results were unexpected and will need further investigation.

The grid transfers used in Tables 1(a) and (b) do not employ prolongator smoothing, i.e., they are *unsmoothed* transfers. (See Section 4 for more discussion on this.) Prolongator smoothing improves solver convergence and iteration count scaling as the problem size increases [28]. For strong scaling, however, we have observed that while the unsmoothed grid transfer leads to higher GMRES iteration counts, the overall run time is faster than with a smoothed prolongator. This is likely due to primarily to the lower overall AMG complexity C for the hierarchy built using unsmoothed prolongators.

As an alternative to prolongator smoothing, we first examined the use of W-cycles (see Algorithm 1) in the multigrid preconditioner to improve GMRES convergence. For the 8192 core case, employing a W-cycle indeed reduced the total number of GMRES iterations by approximately 44% However, the time for the solve phase increased by approximately 162%. We also examined whether starting the W-cycle on a coarser grid could improve convergence *and* runtime. For the 8129 core case, delaying the cycle start until level 3 reduced the total number of GMRES iterations by 11%, but still increased the solve time by 69%.

Initial detailed timing profiling revealed high communication times in applications of the grid transfers during the multigrid solve phase. MueLu has long employed a repartitioning strategy in order to maintain both good load balancing and sufficient work per processor on coarse levels. In this strategy, when certain imbalance criteria are met on a coarse level i, the coarse grid matrix is redistributed across a subset of the initial MPI processes. The criteria are: 1) whether $i > M$, where M is the level number where rebalancing may first occur, 2) minimum number of degrees of freedom (DOFs) per core threshold, and 3) imbalance between number of DOFs per core.

This strategy can optionally rebalance the grid transfers between level $i-1$ to level i, although this is not done by default. The tradeoff is that if the transfers are not rebalanced, additional communication is required during the solve phase. For weak scaling simulations, where setup costs are high, it is typically more cost-effective to avoid rebalancing the transfers. For the current V27 simulation, however, as the solve phase is actually more expensive than the setup phase, we found that it was more effective to explicitly rebalance the restriction matrix from level $i-1$ to level i. While this increases the setup time slightly, the expense is offset by lower solve phase times.

A second change was enabling a new imbalance criterion, a target that specifies how many rows in a rebalanced matrix A_i each MPI process should have. Previously, the minimum DOFs per core threshold was also the distribution of the resulting rebalanced matrix. For Nalu, this value was 1000. By separating the threshold and target values, we were able to increase the work per core after rebalancing. We found that a target of 10000 improved the linear solver time by approximately 15%. This is likely due to several factors. First, the larger target value produced a multigrid hierarchy with one less level, and the coarsening rate between levels was markedly higher starting from level 3 to 4. This implies less communication is necessary in a V-cycle. Second, the number of active MPI processes for the higher target was a factor of 10 less on all levels $3, 4, \ldots$, thus requiring fewer MPI resources. Third, the higher target means more work per processor, i.e., a better volume to surface ratio. The aggregation procedure is a local procedure, and aggregates far from processor boundaries tend to be more compact.

A third significant change was reducing the number of smoothing sweeps from four to two forward Gauss-Seidel[3] sweeps. While using fewer sweeps adversely affects the overall Krylov convergence rate (requiring approximately 6.5 more iterations per continuity linear solve), the overall solve time is faster. This can be

[3]Gauss-Seidel in Trilinos is implemented as a block Jacobi algorithm across processors, with Gauss-Seidel done on a per-process basis. Hence, the Trilinos Gauss-Seidel implementation is equivalent to Hypre's "hybrid" Gauss-Seidel.

explained by understanding that each Gauss-Seidel sweep is equivalent (in flops) to a matrix-vector product. Between multiple sweeps, unknowns are updated by global communication so that all processors have updated unknown values. Thus, reducing the sweeps from four to two is equivalent to eliminating four matrix products *per GMRES iteration*.

Detailed timings due to the improvements described in this section are given in Table 1(c) and Figures 2-5. These results are from an executable using Nalu master SHA1 `4dcfab46f0` and Trilinos develop `SHA1 430bf2ab67`.

7.2 HYPRE-BOOMERAMG DISCUSSION

We first present results produced using the same Nalu executable as from the Exawind 2018 Q1 milestone report [12] in order to establish a baseline for comparison. See Table 2. This was using Hypre version `2.12.0`. The Nalu version was not captured. In the subsequent discussion, we will be presenting results using Nalu master SHA1 `4dcfab46f0`, and Hypre version `2.12.0`.

(a) Executable and settings from FY18 Q1 Report

						time (seconds)		
cores	sw	C	S_{avg}	θ	Iterations	Solve	Set-up	Wall-clock
2048	2	1.154	21.4	0.25	17.2	133.1	23.8	865
4096	2	1.154	21.4	0.25	16.7	79.0	16.5	482
8192	2	1.154	21.4	0.25	17.0	58.1	24.6	355
12288	2	1.154	21.4	0.25	16.5	51.7	53.4	351

(b) Executable and settings for newest Nalu executable, newest settings

						time (seconds)		
cores	sw	C	S_{avg}	θ	Iterations	Solve	Set-up	Wall-clock
2048	2	1.149	21.4	0.25	18.5	133.0	23.4	865
4096	2	1.149	21.4	0.25	17.2	75.0	16.7	483
8192	2	1.149	21.4	0.25	17.7	57.0	21.8	331
12288	2	1.149	21.4	0.25	17.0	43.9	30.7	312

Table 2: Simulation times reported for 10 time steps of the V27 41 R1 500M element mesh. Baseline Hypre/BoomerAMG solver performance in Nalu, FY2018, Quarter 1 as well as performance with the latest Nalu executable and newest Hypre settings.

Coarsening for the Hypre-BoomerAMG is based on the parallel modified independent set (PMIS) algorithm of Hans de Sterck [5] allowing for a parallel set-up phase. A transposed prolongation operator is retained for triple-matrix RAP products. The strength of connection threshold is set to $\theta = 0.25$. Aggressive coarsening is applied on the first two V-cycle levels with multi-pass interpolation and a stencil width of two elements per row. The remaining levels employ extended+i interpolation to construct the prolongation; 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 [21]. Interpolation truncation level $\tau = 0.25$ is specified together with a maximum interpolation stencil width of two matrix elements per row. The smoother is two sweeps of a hybrid Gauss-Seidel relaxation scheme.

The coarsening rate for the V27 problem is roughly 4× 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 GMRES iterations are required compared to standard coarsening. The V27 41 R1 mesh was run on up to 12288 cores on Cori for 10 time steps starting from a restart file generated after the initial 20 steps starting from $t = 0$. Thus, any initial transients in the solution are avoided for an unbiased comparison. The baseline timings based on the Q1 milestone report are presented in Table 2. In order to improve the Hypre timings the non-Galerkin coarse mesh sparsification is applied as described in [3]. The truncation is applied on the first three levels with drop tolerances $\gamma = [\,0.0,\ 0.01,\ 0.01\,]$. Non-Galerkin coarse mesh sparsification clearly reduces the operator complexity and solve time. For unsmoothed prolongation, the limits on aggregate size also reduce C and S_{avg}. For our wind-turbine simulations with moving meshes,

the AMG set-up cost is an important factor because the matrices are re-initialized every time-step and the V-cycle hierarchy must be re-computed. In our comparison runs, different Krylov subspace dimensions are specified with GMRES(50) for MueLu, and 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. Strong scaling results for Q2 with these options are reported in Table 2.

The set-up times reflect the mesh motion and repeated computation of the V-cycle hierarchy every time step. The complexity C and GMRES iteration counts are indicative of the solve time. Whereas, the average stencil widths S_{avg} are highly correlated with the set-up time. The overall time to solution is dominated by high matrix assembly costs for large problems and high core counts. These costs are being addressed in our future work. The GMRES iteration counts are lower for Hypre and reflect faster convergence rates for the C-AMG preconditioner. However, the cost per iteration counts remain constant or increase slightly with core count.

In order to provide a more detailed timing breakdown and comparison with our Q1 milestone results, the model was run for 100 time steps from the restart on 12288 cores at NERSC-Cori. The timing breakdown format reported in Q1 employed and the results are presented in Table 3. We note that the MueLu solve times (`solve` label) are now lower than Hypre at this higher core count. Preconditioner set-up times (`precond setup`) are comparable, but now the momentum solve has increased further. Note that the MueLu `precond setup` time of 266.4 seconds is actually higher in Table 3 than in the Q1 report, where it is 204.1 seconds. The increase is attributable to both the larger rebalancing criterion of $10,000$ DOFs per process and the explicit rebalancing of the multigrid restriction operator, as discussed in Section 7.1. By doing more work in the `precond setup` phase, the linear solver time `solve` is reduced from 580.7 seconds (as reported in Q1) to 442.3 seconds. The net improvement in linear solve plus setup from Q1 to Q2 is 76.1 seconds.

The momentum solver cost will become the dominant non-setup cost at higher Courant numbers. The Trilinos-Tpetra sparse matrix local/global graph initialization `init` is still by far the dominant computational cost. It is also notable that the moving mesh `non-conformal BC` time is around 10% of the total.

A much longer integration at lower resolution has also been performed to demonstrate the fidelity of the model in representing the flow field surrounding the V27 turbine . In this case the mesh contains 166M elements and the run was performed on NERSC Cori using 4096 cores. The mesh is displayed in Figure 6 on the surface of the turbine blades and nacelle. The target Courant number is $C = 20$ with an initial time step of $\Delta t = 5.0e-6$. The high Courant number was chosen to assess both accuracy and computational efficicency. In addition, second order accurate BDF-2 time integration was employed. The model was integrated to $t = 0.75$ sec representing roughly one half revolution of the turbine. The integration required 7000 time steps and 24 hours of wall-clock time. The velocity is plotted in Figure 7 with an isosurface of constant $v = 10$ m/s.

7.3 BDDC DISCUSSION

BDDC is a relative newcomer to the ExaWind project, and one of our goals this fiscal year is to determine its viability as a potential solver for future exascale wind farm simulations. As a first step, we performed some small-scale numerical experiments to help gain basic insights that will guide development and testing of BDDC in the coming months. Larger-scale simulations with the BDDC solver are planned for later this summer, and we expect to find good scalability with respect to increasing problem size based on existing theory and the experience of others (see, e.g., [19], [1], or any of the references in [22]).

The first problem considered is an atmospheric boundary layer (ABL) test problem in the Nalu regression test suite. The ABL regression test is at the easy end of the spectrum for the continuity equations; the elements are all perfect cubes and the assembled coefficient matrix is symmetric and positive definite. This is an ideal situation for a multilevel solver. In particular, multigrid solvers have been shown to be ideally suited to such problems. The second problem involves a model with very high aspect ratio elements. We note that simulations for models containing high aspect ratio elements are relevant to this project.

The first insight gained for nice problems like the ABL regards an upper bound estimate on how many elements per subdomain can be used for BDDC in order for it to be competitive with AMG. The default option for BDDC is to use direct solvers at the subdomain level, but less expensive options are available [9]. It is well known that sparse direct solvers do not scale well for problems in three dimensions. Notably, the number of operations to calculate the factorization of a matrix with N rows is at best of order N^2 for discretizations of partial-differential equations. Thus, there is clearly an upper limit on how many elements

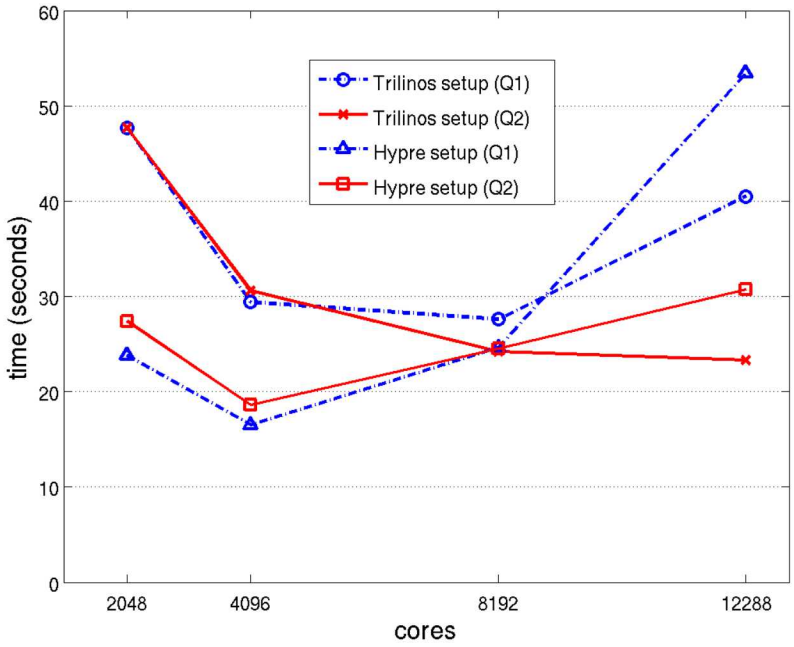


Figure 2: Strong scaling of the linear solver `setup` phase for the continuity equations. Current results (Q2) are compared against previously reported results (Q1). Times reported for 10 time steps of the V27 41 R1 500M element mesh.

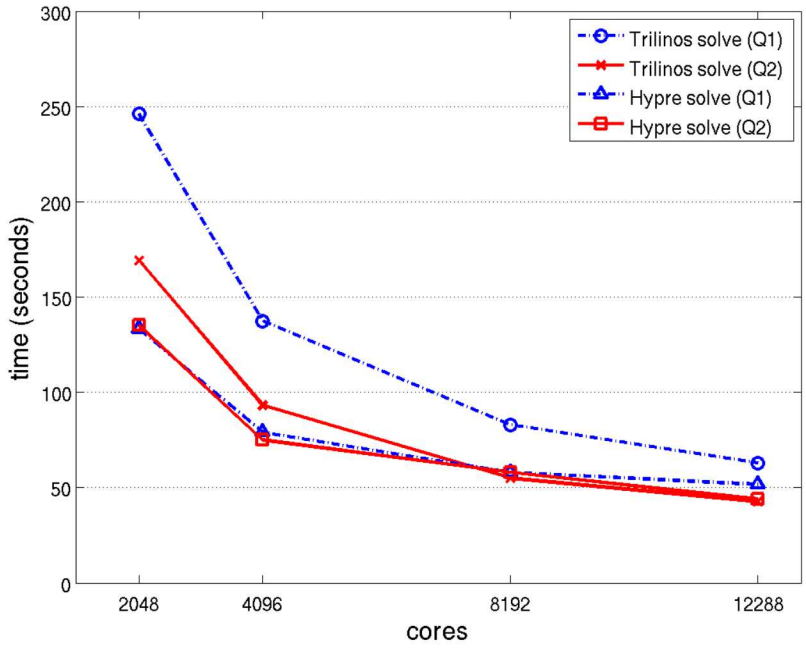


Figure 3: Strong scaling of the linear solver `solve` phase for the continuity equations. Current results (Q2) are compared against previously reported results (Q1). Times reported for 10 time steps of the V27 41 R1 500M element mesh.

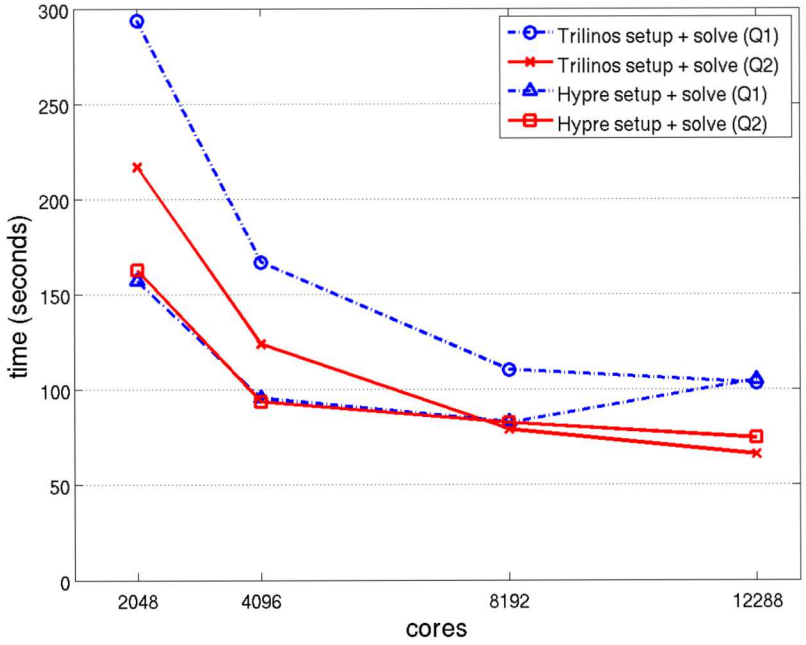


Figure 4: Strong scaling of the linear solver `setup` and `solve` phases for the continuity equations. Current results (Q2) are compared against previously reported results (Q1). Times reported for 10 time steps of the V27 41 R1 500M element mesh.

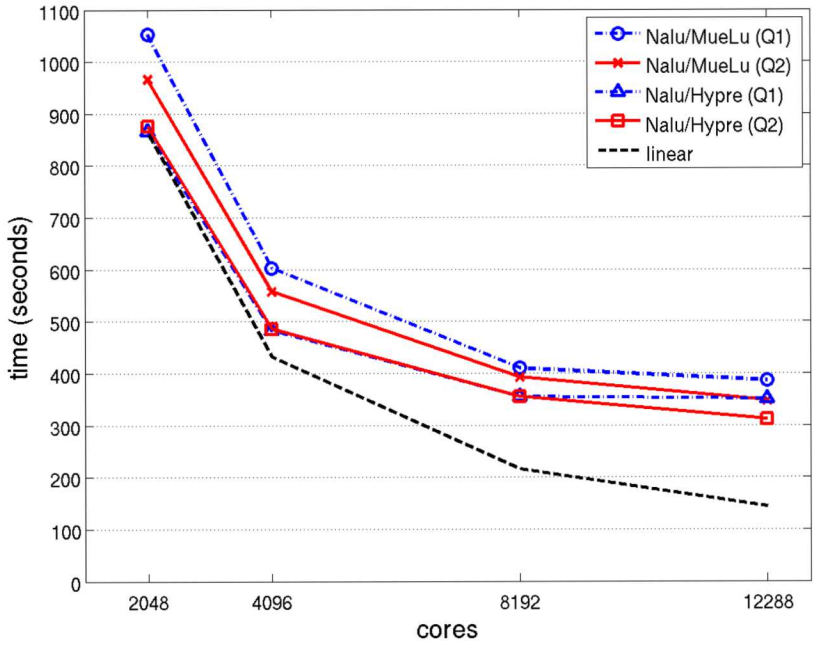


Figure 5: Total simulation times for 10 time steps of the V27 41 R1 500M element mesh versus increasing core counts on Cori's Haswell partition. Current Q2 results are compared against previously reported Q1 results.

myLowMach Solver	Time (sec.)	% of total	Time (sec.)	% of total
misc	0.7	0.0	6.6	0.0
Momentum Solver	Belos/SGS		Belos/SGS	
init	789.5	27.1	711.3	21.7
assemble	189.7	6.5	189.1	5.7
load complete	245.6	8.4	212.2	6.5
solve	356.9	12.3	356.1	10.8
precond setup	0.9	0.0	0.9	0.0
misc	23.6	0.8	24.0	0.7
total time	1606.2	55.1	1493.6	45.4
linear iterations	10		10	
Continuity Solver	Hypre/BoomerAMG		Belos/MueLu	
init	11.6	0.4	654.8	20.0
assemble	48.2	1.6	33.7	1.0
load complete	75.9	2.6	33.4	1.0
solve	537.9	18.5	442.3	13.5
precond setup	278.7	9.5	266.4	8.1
misc	30.2	1.0	40.3	1.2
total time	982.5	33.6	1471.0	44.8
linear iterations	17		25	
IO Time	3.0	0.0	3.5	0.0
Non-conformal BC	309.0	10.0	307.1	9.3
Skin Mesh	0.1	0.0	0.1	0.0
Other	8.0	0.0	8.0	0.0
totals	2910	100	3278	100

Table 3: Comparison of Nalu V27 41 R1 500M element simulation time for 100 timesteps between Belos/SGS+Hypre and Belos/SGS+MueLu on 12288 cores of Cori, with breakdown by solver component.

per subdomain is practical when using direct solvers.

For the ABL problem, we considered three different mesh sizes with m elements in the two horizontal directions and $m/5$ in the vertical direction, with m ranging from 25 to 75. These simulations are certainly quite modest in size, but perfectly suited to gaining insight. The problems were first run on eight processors using MueLu with solver options specified in the xml file for the test. Specifically, Chebyshev smoothing of degree two was used. The same problems were then run using the BDDC solver with SuperLU [6] as the sparse direct solver and GMRES as the Krylov method. The solver parameters for either MueLu or BDDC could have been optimized for better performance, but that was not the purpose here.

The average number of elements per MPI rank is shown in Table 4 for the three different meshes. We note for BDDC that this is the same as the average number of elements per subdomain since only a single subdomain was used per MPI rank. Notice the steep growth in preconditioner setup times for BDDC as the average number of elements per subdomain increases. This is to be expected because of the quadratic complexity of the factorization phase of the sparse direct solver. In contrast, the growth in setup times for MueLu is much more modest and reflective of linear complexity with problem size.

It is clear from the Table 4 that BDDC based on the use of direct solvers is not competitive with AMG for this problem if the average number of elements per subdomain becomes too large. As a rough rule of thumb, one could say this number should not exceed around 5000. This number could likely be increased significantly by replacing the direct solvers with "smoothers" such as polynomial preconditioners, incomplete factorizations, or Gauss-Seidel relaxation, but we did not explore such options here. The good news is that BDDC can be run using multiple smaller subdomains per MPI rank, but this will require some changes to the current solver interface between Nalu and BDDC.

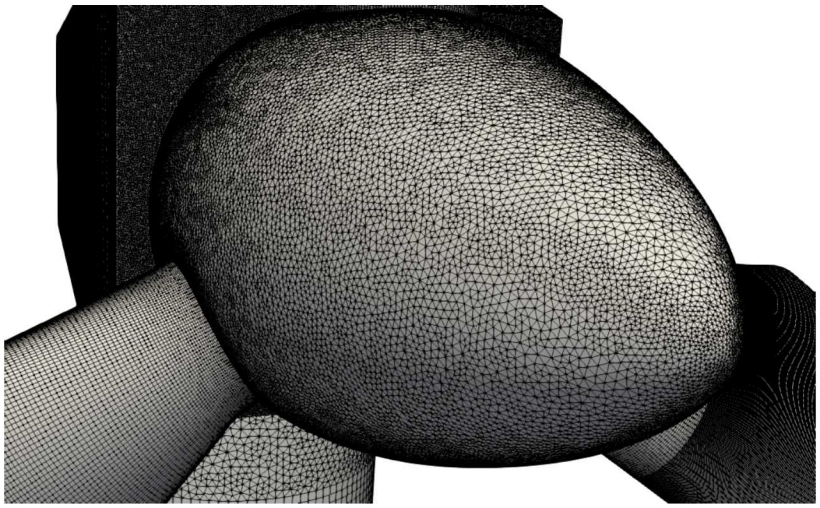


Figure 6: Lower resolution V27 41a R0 166 M elements, 45M DOFs. Depiction of mesh on the surface. Close-up of the cone section and nacelle; surface mesh lines are in black.

The second problem concerns high aspect ratio elements and makes use of a structured mesh of hexahedral elements with 60 elements in the two horizontal directions and 10 in the vertical direction. As a surrogate for the pressure Poisson equation, we consider steady state heat conduction with a specified temperature of zero on the left side and one on the right. The dimension in the vertical direction is reduced from a starting value of 10 (aspect ratio of unity) to smaller values to increase element aspect ratios. Both control volume finite element method (CVFEM) and edge-based vertex-centered (EBVC) discretization approaches are considered.

The results in Table 5 include numbers of iterations and condition number estimates of the preconditioned operator obtained from the conjugate gradient iterations. Notice for both discretization types that the number of iterations and condition number estimates are fairly insensitive to large changes in aspect ratio. Interestingly, the coarse space dimension for the CVFEM discretizations is three, while that for EBVC is seven. Also, the number of iterations for EBVC is always at least twice that for CVFEM. We expect that this difference is related to how edge ownership is assigned by Nalu and will be investigated further. Although this is a simple first test, these results provide some encouragement for BDDC being applied to more realistic problems.

Results from this first study are encouraging, but they point to specific areas requiring more attention. First, the solver interface between Nalu and BDDC needs to be modified to allow for multiple subdomains per MPI rank. For edge-based discretizations, we observed disconnected subdomain components in some cases consisting of just two nodes when using the recursive coordinate bisection (rcb) automatic decomposition option. A lot of experience for BDDC is based on subdomain decompositions obtained from graph partitioning approaches and not rcb. A better understanding of the available decomposition tools, remedies for disconnected components, and how best to assign edges to subdomains in EBVC schemes will likely benefit the BDDC solver in the future.

8. CONCLUSION

The Vestas V27 wind turbine with 500M element mesh was employed for our strong scaling studies reported here. In addition, the purpose of our simulations was to compare the two fundamental algebraic multigrid (AMG) algorithms, namely classical Ruge–Stüben (C-AMG) and smoothed aggregation (SA-AMG). In addition we introduced the balancing domain decomposition by constraints (BDDC) preconditioner as a possible alternative approach. In order to establish a baseline for comparison with our ECP Q2 milestone, the V27 41 R1 mesh was run on up to 12288 cores on NERSC-Cori for 10 time steps starting from a restart file generated after the initial 20 steps starting from $t = 0$. Thus, any initial transients in the solve are avoided for an unbiased comparison. However, it is important to note that the simulations reported here have not

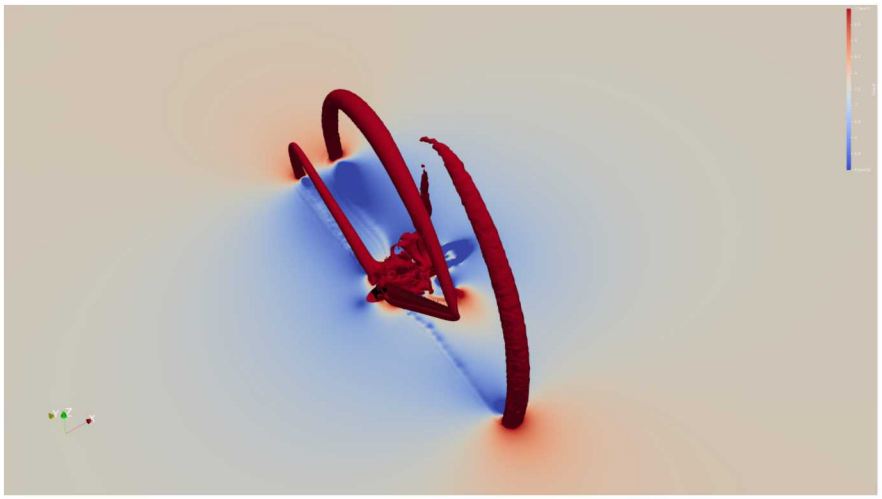


Figure 7: Lower resolution simulation with V27 41a 166M element mesh. Isosurface with volume rendering of constant velocity 10 m/s. Velocity v of flow field surrounding V27 wind turbine at 0.75 sec.

Table 4: BDDC and MueLu results for the ABL test problem for three different meshes. The symbol n_s denotes the average number of elements per subdomain. Preconditioner setup times, solve times, average number of iterations (iter), and total Nalu run times (time) are reported; timing results are in seconds. The test was run for ten time steps and the continuity equations were solved to a relative residual tolerance of 10^{-6}.

		MueLu				BDDC			
mesh	n_s	setup	solve	iter	time	setup	solve	iter	time
25×25×5	391	0.043	0.51	8.8	1.44	0.016	0.13	4.2	1.03
50×50×10	3125	0.051	0.78	9.1	3.54	0.13	0.69	5.8	3.52
75×75×15	10547	0.073	1.67	10.2	9.61	1.31	3.86	6.5	12.9

Table 5: BDDC number of iterations (iter) and condition number estimates (cond) of the preconditioned operator for a problem with varying element aspect ratios for two element formulations. The linear systems were solved to a relative residual tolerance of 10^{-6}.

	CVFEM		EBVC	
aspect ratio	iter	cond	iter	cond
1	7	8.1	23	20.8
10	7	7.8	18	14.0
100	7	7.8	18	14.2
1000	7	7.8	16	14.2

reached a statistically steady flow state.

The results presented in this report clearly demonstrate improvements over the Q1 results in the combined solve plus preconditioner set-up times for the Trilinos MueLu/Belos solver stack and Hypre-BoomerAMG solver stacks when employed as preconditioners for the pressure-Poisson GMRES solver in the Nalu model. While the Trilinos/MueLu preconditioner setup time has increased, this increase is more than offset by a decrease in linear solve time. In the case of MueLu, the introduction of unsmoothed aggregation combined with minimum and maximum aggregate sizes resulted in lower V-cycle complexity C and stencil width $S_{avg}(A_l)$. These are correlated with lower solve and set-up times. In addition, a further decrease in solve time was achieved by reducing the number of L_1 Gauss-Seidel smoother sweeps from four to two. 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, our Q2 results demonstrated improvements over Q1 with the inclusion of non-Galerkin coarse grids or sparsification. These are relatively recent additions to the Hypre-BoomerAMG solver options [13], [3]. Furthermore, we employed GMRES(10) in order to reduce the communication overhead in the MGS-GMRES algorithm. The complexity C and stencil width S for Hypre are almost identical to MueLu for the chosen solver parameters applied to the V27 41 turbine problem. The GMRES iteration counts for Hypre were lower than with MueLu, but the solve and set-up times were comparable. Thus, the cost per iteration of Hypre is slightly higher with a faster convergence rate.

A longer simulation integrating out to 100 time steps was performed in order to identify the largest computational costs in the model. These results are reported in Table 3. By far the highest cost is associated with the Trilinos-Tpetra sparse matrix initialization 'init' for global/local graphs and compressed sparse row memory allocations. The Hypre-BoomerAMG solver stack requires less time to set-up the CSR matrices. However, the equivalent 'init' cost in Hypre is spread across the (init + assemble + load complete) and a fair comparison between the solver stacks should include all of these costs as well. These costs are exacerbated by the repeated matrix re-initialization associated with the moving mesh surrounding the wind turbine blades and nacelle.

Our next steps and future efforts will explore ways to reduce these re-initialization costs. In addition, as we transition our efforts to next-generation platforms (NGP), the focus will include the further application of threading to reduce the solve and set-up costs in AMG. We recommend a more complete analysis of the effect of high-aspect ratio meshes on the solver convergence and performance. With an increasing Courant number required to achieve more tractable wall-clock times for long time integrations, the momentum solver cost becomes important. Further efforts in different preconditioners such as ILU(0) or multi-level AMG for momentum are recommended. We are also exploring monolithic solvers and their associated preconditioners. Further efforts to mitigate the communication costs in GMRES Krylov solvers are certainly warranted and the potential for reducing communication cost with BDDC has clearly demonstrated merit.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the 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—to support the nation's exascale computing imperative.

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

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 (SAND2018-XXXX). As such, the technical report is suitable for unlimited release.

NREL is a national laboratory of the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, operated by the Alliance for Sustainable Energy, LLC under Contract No. DE-AC36-08GO28308.

We would like to thank Dr. Kenny Gruchalla (NREL) for providing the rendering of the flow field in Figure 7.

REFERENCES

[1] S. Badia, A. F. Martín, and J. Principe, *Multilevel balancing domain decomposition at extreme scales*, SIAM Journal on Scientific Computing, 38 (2016), pp. C22–C52.

[2] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, *Multigrid smoothers for ultraparallel computing*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2864–2887.

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

[4] A. Björck, *Numerics of Gram-Schmidt orthogonalization*, Linear Algebra and Its Applications, 197 (1994), pp. 297–316.

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

[6] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 720–755.

[7] L. T. Diosady and D. L. Darmofal, *A unified analysis of balancing domain decomposition by constraints for discontinuous Galerkin discretizations*, SIAM Journal on Numerical Analysis, 50 (2012), pp. 1695–1712.

[8] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, SIAM Journal on Scientific Computing, 25 (2003), pp. 246–258.

[9] ——, *An approximate BDDC preconditioner*, Numerical Linear Algebra with Applications, 14 (2007), pp. 149–168.

[10] S. Domino, *Sierra low Mach module: Nalu theory manual. version 1.0.*, Tech. Rep. SAND2015-3107W, Sandia National Laboratories, 2015.

[11] ——, *Design-order, non-conformal low-Mach fluid algorithms using a hybrid CVFEM/DG approach*, Journal of Computational Physics, (2017).

[12] S. Domino, S. Thomas, M. Barone, A. Williams, S. Ananthan, R. Knaus, J. Overfelt, M. Sprague, and J. Rood, *ECP milestone report: Deploy production sliding mesh capability with linear solver benchmarking*, Tech. Rep. SAND2018-1807R, Sandia National Laboratories, 2018.

[13] R. D. Falgout and J. B. Schroder, *Non-Galerkin coarse grids for algebraic multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. 309–334.

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

[15] V. E. Henson and U. Meier-Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics, 41 (2000), pp. 155–177.

[16] 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. Softw., 31 (2005), pp. 397–423.

[17] P. Lin, M. Bettencourt, S. Domino, T. Fisher, M. Hoemmen, 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 Processing Letters, 24 (2014), p. 1442005.

[18] P. Lin, J. Shadid, J. Hu, R. Pawlowski, and E. Cyr, *Performance of fully-coupled algebraic multigrid preconditioners for large-scale VMS resistive MHD*, Journal of Computational and Applied Mathematics, (2017).

[19] J. Mandel, C. R. Dohrmann, and R. Tezaur, *An algebraic theory for primal and dual substructuring methods by constraints*, Applied Numerical Mathematics, 54 (2005), pp. 167–193.

[20] U. Meier-Yang, *Parallel algebraic multigrid methods - high performance preconditioners*, in Numerical Solution of Partial Differential Equations on Parallel Computers, A. Bruaset and A. Tveito, eds., Lecture Notes in Computational Science and Engineering, Springer-Verlag, 2006.

[21] ——, *On long range interpolation operators for aggressive coarsening*, Numerical Linear Algebra with Applications, 17 (2010), pp. 453–472.

[22] C. Pechstein and C. R. Dohrmann, *A unified framework for adaptive BDDC*, Electronic Transactions on Numerical Analysis, 46 (2017), pp. 273–336.

[23] A. Prokopenko, J. J. Hu, T. A. Wiesner, C. M. Siefert, and R. S. Tuminaro, *MueLu users guide 1.0*, Tech. Rep. SAND2014-18874, Sandia National Laboratories, 2014.

[24] J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, S. McCormick, ed., Frontiers in Applied Mathematics, SIAM, 1987, pp. 73–130.

[25] Y. Saad and M. H. Schultz, *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[26] K. Stüben, *A review of algebraic multigrid*, Computational and Applied Mathematics, 128 (2001), pp. 281–309.

[27] P. Vanék, J. Mandel, and M. Brezina, *Algebraic multigrid on unstructured meshes*, Tech. Rep. CCM Report 34, University of Colorado at Denver, 1994.

[28] ——, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

A. SOLVER SETTINGS

A.1 MUELU

```
<ParameterList name="MueLu">
  <Parameter      name="verbosity"                    type="string"  value="none"/>
  <Parameter      name="coarse: max size"             type="int"     value="1000"/>

  <Parameter name="transpose: use implicit"           type="bool"    value="false"/>
  <Parameter name="repartition: rebalance P and R"    type="bool"    value="true"/>

  <Parameter      name="smoother: type"               type="string"  value="RELAXATION"/>
  <ParameterList  name="smoother: params">
     <Parameter name="relaxation: type"  type="string"  value="Gauss-Seidel"/>
     <Parameter name="relaxation: use l1" type="bool"   value="true"/>
     <Parameter name="relaxation: sweeps" type="int"    value="2"/>
  </ParameterList>

  <Parameter      name="sa: damping factor"           type="double"  value="0."/>
  <Parameter      name="tentative: calculate qr"      type="bool"    value="false"/>

  <Parameter      name="aggregation: type"            type="string"  value="uncoupled"/>
  <Parameter      name="aggregation: drop tol"        type="double"  value="0.03"/>
  <Parameter      name="aggregation: drop scheme"     type="string"  value="distance laplacian"/>
  <Parameter      name="aggregation: max agg size"    type="int"     value="8"/>
  <Parameter      name="aggregation: min agg size"    type="int"     value="3"/>

  <Parameter      name="repartition: enable"          type="bool"    value="true"/>
  <Parameter      name="repartition: min rows per proc" type="int"   value="1000"/>
  <Parameter      name="repartition: start level"     type="int"     value="2"/>
  <Parameter      name="repartition: target rows per proc" type="int" value="10000"/>
  <Parameter      name="repartition: max imbalance"   type="double"  value="1.327"/>
  <Parameter      name="repartition: partitioner"     type="string"  value="zoltan2"/>
</ParameterList>
```

A.2 BOOMERAMG

```
   bamg_output_level: 1
   bamg_coarsen_type: 8
   bamg_interp_type: 6
   bamg_cycle_type: 1
   bamg_relax_type: 3
   bamg_relax_order: 1
   bamg_num_sweeps: 2
   bamg_keep_transpose: 1
   bamg_max_levels: 9
   bamg_trunc_factor: 0.25
   bamg_agg_num_levels: 2
   bamg_agg_interp_type: 4
   bamg_agg_pmax_elmts: 2
   bamg_pmax_elmts: 2
   bamg_strong_threshold: 0.25
   bamg_non_galerkin_tol: 0.05
   bamg_non_galerkin_level_tols:
     levels: [0, 1, 2 ]
     tolerances: [0.0, 0.01, 0.01 ]
```