SAND2020-11998R

ECP-U-2020-xxx

Demonstration and performance testing of extreme-resolution simulations with static meshes on Summit (CPU & GPU) for a parked-turbine configuration and an actuator-line (mid-fidelity model) wind farm configuration

WBS 2.2.2.01, Milestone ECP-Q4-FY20

Shreyas Ananthan
Alan Williams
James Overfelt
Johnathan Vo
Philip Sakievich
Timothy A. Smith
Jonathan Hu
Luc Berger-Vergiat
Paul Mullowney
Stephen Thomas
Marc T. Henry de Frahan
Jeremy Melvin
Robert Moser
Michael Brazell
Michael A. Sprague

October 30, 2020

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative. A portion of this research was supported by the U.S. Department of Energy Wind Energy Technologies Office.

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 No. DE-AC36-08GO28308.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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 (SAND2020-XXXX R). As such, the technical report is suitable for unlimited release.

A portion of the research was performed using computational resources sponsored by the Department of Energy's Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory. A portion of the research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

ECP-U-2020-xxx

ECP Milestone Report
Demonstration and performance testing of extreme-resolution simulations with static meshes on Summit (CPU & GPU) for a parked-turbine configuration and an actuator-line (mid-fidelity model) wind farm configuration
WBS 2.2.2.01, Milestone ECP-Q4-FY20

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

October 30, 2020

ECP Milestone Report
Demonstration and performance testing of extreme-resolution simulations with static meshes on Summit (CPU & GPU) for a parked-turbine configuration and an actuator-line (mid-fidelity model) wind farm configuration
WBS 2.2.2.01, Milestone ECP-Q4-FY20

APPROVALS

Submitted by:

Michael Sprague

30 October 2020

Michael A. Sprague
ECP-Q4-FY20

Date

Approval:

Thomas Evans
ORNL

Date

REVISION LOG

Version	Creation Date	Description	Approval Date
1.0	2020-08-30	Original	

EXECUTIVE SUMMARY

The goal of the ExaWind project is to enable predictive simulations of wind farms comprised of many megawatt-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 physics codes in the ExaWind simulation environment are Nalu-Wind, an unstructured-grid solver for the acoustically incompressible Navier-Stokes equations, AMR-Wind, a block-structured-grid solver with adaptive mesh refinement capabilities, and OpenFAST, a wind-turbine structural dynamics solver. The Nalu-Wind model consists of the mass-continuity Poisson-type equation for pressure and Helmholtz-type equations for transport of momentum and other scalars. 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 reinitialization of matrices and recomputation of preconditioners is required at every time step. The choice of overset-mesh methodology to model the moving and non-moving parts of the computational domain introduces constraint equations in the elliptic pressure-Poisson solver. The presence of constraints greatly affects the performance of algebraic multigrid preconditioners.

In this milestone, we demonstrate the execution of two computational fluid dynamics (CFD) codes in the ExaWind simulation environment, Nalu-Wind and AMR-Wind, on heterogeneous CPU/GPU compute architectures. The core computational kernels in Nalu-Wind, especially those related to linear system assembly and iterative solves, have been offloaded to accelerators. Significant efforts have been undertaken to minimize the data transfers between host and device memory spaces. The performance of computational kernels executing on GPUs was benchmarked on ORNL Summit system for two wind-energy problems of interest: 1. Strong scaling performance for a blade-resolved simulation of the NREL 5-MW wind turbine in uniform inflow, and 2. Strong and weak scaling performance for multi-turbine wind farm simulation operating in turbulent atmospheric boundary layer (ABL) inflow, where turbines are represented as *actuator* source terms in the governing equations. For the problems considered, we have verified the correctness of the simulations when executing in hybrid mode compared to execution on CPUs alone. Comparisons of performance on a per-node basis, indicate that the strong scaling performance on GPUs has opportunities for significant improvement. Computational cost of pressure-Poisson solves using algebraic multigrid (AMG) preconditioners continues to dominate the overall time to solution. In particular, the cost of preconditioner setup on GPUs shows a dramatic increase compared to CPUs and will be the focus of future work within the ExaWind project.

In FY20, the team added a block-structured background CFD solver, AMR-Wind, that is built on top of the AMReX library. The use of block-structured, Cartesian mesh solvers for modeling the atmospheric boundary layer and the wake structures allows the team to realize significant computational gains through use of geometric multigrid solvers in lieu of the AMG preconditioners used in the unstructured Nalu-Wind solver. Performance comparisons of AMR-Wind and Nalu-Wind for the ABL *precursor* problems indicate that AMR-Wind is five times faster on CPUs when compared to Nalu-Wind for the same problem executing on the same computational resource. Strong and weak scaling studies of AMR-Wind were performed on ORNL summit for mesh sizes of up to 9.1×10^9 degrees of freedom.

Considerable advances were made in the development, verification, and validation of the Active Model-Split (AMS) hybrid-RANS/LES turbulence model. In addition to incorporating the recent developments in literature to the code-base, the team also identified and fixed several issues that were causing the instabilities observed in FY19-Q4 milestone. The model has been validated for simple test problems and is currently being tested on wind-energy relevant problems such as the NACA0015 fixed wind and NREL 5-MW wind turbine simulations.

With successful transition of computational kernels to GPUs, the adoption of a hybrid unstructured-structured solver strategy, and advances to hybrid RANS/LES turbulence modeling capabilities, the team is well poised to meet the ExaWind challenge problem on upcoming exascale systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The ultimate goal of the ExaWind project is to enable scientific discovery through predictive simulations of wind farms comprised of many megawatt-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 (blade-resolved) 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 [15].

The expectation is that by developing the computational capability to conduct blade-resolved simulations of a wind farm, more accurate predictions of wake dynamics and their interaction with other turbines will be achieved. Preparing the ExaWind simulation environment to utilize next-generation resources to allow for these massive simulations has been the main focus of the majority of previous milestones. Even with the use of exascale computational resources, the accuracy of simulations will still be limited by the quality of the underlying physical modeling.

The primary physics codes in the ExaWind simulation environment are Nalu-Wind, an unstructured-grid solver for the acoustically incompressible Navier-Stokes equations, AMR-Wind, a block-structured-grid solver with adaptive mesh refinement capabilities, and OpenFAST, a wind-turbine structural dynamics solver. The Nalu-Wind model consists of the mass-continuity Poisson-type equation for pressure and Helmholtz-type equations for transport of momentum and other scalars. For such modeling approaches, simulation times are dominated by linear-system setup and solution for the continuity and momentum systems.

In this milestone, we present the results of our efforts in transitioning the core computational kernels within Nalu-Wind to execute on GPUs. The new codebase heavily leverages the Kokkos abstraction library as well as the considerable advances in the Trilinos solver libraries (e.g., MueLu, Kokkos-kernels, etc.) as well as the newly developed STK NGP library that provides in-memory, unstructured mesh data structures that operate on GPUs. Strong and weak scaling studies are performed on relevant problems to baseline the performance of computational algorithms when executing on GPUs. The baseline performance will be used as the basis to evaluate algorithmic improvements that will be implemented as part of the project. We also document the significant advances made in the development and validation of the hybrid RANS/LES model. The new model, AMS (Active Model-Split), addresses the several limitations in the state-of-the-art hybrid-RANS/LES models used in wind energy research that can seamlessly handle the transition between the ABL and the blade boundary layer spatial and temporal scales.

2. MILESTONE DESCRIPTION

In this section, we provide the approved milestone description and execution plan followed by a brief description of how the milestone was completed. Details regarding completion are included in the following sections.

2.1 DESCRIPTION

This milestone will perform at-scale simulations for two different cases:

- Blade-resolved, static-mesh, hybrid RANS/LES simulation of a single, parked wind turbine in uniform inflow where the blade and the near-wake regions are resolved adequately to capture the thrust and power for the rotor. The objective of this simulation is to perform as much of the linear system assembly and solve, field updates, etc. on the device (GPUs) and develop best practices for simulating blade-resolved turbine calculations on heterogeneous architectures.

- ABL LES simulation of a multi-turbine configuration where the turbines are resolved as actuator lines or actuator disks. The simulation will address the following objectives: investigate the coupling to OpenFAST (the wind-turbine structural model) when Nalu-Wind is executing on a CPU-GPU configuration; demonstrate the increase in resolution possible with mid-fidelity simulations when running on GPU architectures and/or the speed-ups possible with GPUs for the resolutions used in the state-of-the-art simulations.

Performing the aforementioned demonstration simulations will require several developments that are described briefly below:

- GPU assembly and execution of linear systems – Nalu-Wind's discretization operators, linear system assembly logic, and the linear solvers themselves must be fully executable on GPUs.
- Optimization of MPI+GPU execution on multiple MPI ranks – Performance benchmarking to understand the optimal settings of MPI ranks vs GPUs, trade-offs of dedicated MPI rank per GPU on a node vs. sharing a single GPU with multiple MPI ranks, use of NVIDIA MPS etc.
- Advances in linear solvers – The simulations will exercise the latest solver advances established as part of FY20 Q2 milestone. A particular focus will be investigating the performance of AMG preconditioners (setup, smoothers, etc.) and understand the performance bottlenecks that will arise when used with moving mesh simulations (reinitialization of linear system overhead).

2.2 EXECUTION PLAN

1. Converting Nalu-Wind discretization operators and field updates to NGP – To achieve the demonstration simulations targeted within this milestone, Nalu-Wind's codebase must be fully migrated to use STK NGP Mesh API. This will allow the unstructured mesh, and corresponding field data, to reside on device memory and minimize data movement between the host and device. Additional developments will be necessary in the STK NGP mesh library to allow synchronization of field data across multiple MPI ranks for shared entities.
2. Advances in TAMS hybrid-turbulence model – The suitability of TAMS model with k-omega SST turbulence model as the underlying RANS model will be investigated using validation problems (e.g., McAlister wing experiments), and alternate RANS models will be explored to determine the optimal configuration for use with blade-resolved wind turbine simulations operating in turbulent atmospheric boundary layer inflow.
3. Performance benchmarking of structured off-body solvers – The milestone will also explore the performance benefits of using structured off-body (background) solver to boost the performance of moving mesh simulations. The target simulation will be a neutral atmospheric boundary layer precursor simulation that can be compared with existing capability within Nalu-Wind to determine if the structured background solver is a viable pathway for the ExaWind challenge problem.

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

2.3 OVERVIEW OF MILESTONE COMPLETION

The following is a concise description of how each of the items in §2.2 was satisfied for milestone completion.

1. Computational kernels involving linear system assembly, field updates, and linear solvers were transitioned to execute on GPUs. The algorithmic updates are described in §3. The updated codebase was tested on two wind-energy relevant problems: 1. Blade-resolved simulation of NREL 5-MW wind turbine in uniform inflow using an overset mesh methodology (see §4), and 2. Simulation of a wind farm operating in turbine ABL inflow where the turbines are modeled as *actuator* sources in the governing equations (see §5).
2. The team successfully addressed several outstanding issues in the hybrid-RANS/LES model that was documented in the FY19-Q4 reports. The algorithmic improvements and the bug fixes to the underlying $k - \omega$ SST RANS turbulence model are documented in §6.1. Validation of the new AMS model on problems documented in literature (§6.3) as well as application to wind-energy relevant problems such as the NACA0015 fixed-wind simuation (§6.4) and the NREL 5-MW wind turbine (§6.5) simulations are also documented in §6.
3. Details of the new AMReX-based, block-structured, incompressible background solver are described in §3.4. The strong and weak scaling performance of AMR-Wind executing on ORNL Summit system for the ABL precursor simulations are described in §7.1.

3. EXAWIND SIMULATION ENVIRONMENT ON NEXT-GENERATION PLATFORMS

Porting a significant portion of the compute intensive regions of code onto GPUs and demonstrating execution on heterogeneous CPU/GPU architectures was the primary focus of the development efforts in FY20. In addition to Nalu-Wind, the development team also adopted a new block-structured background mesh solver (AMR-Wind) based on AMReX library. This section details the improvements to the Nalu-Wind codebase, as well as supporting enhancements to the Trilinos and *hypre* libraries. This is followed by a description of the AMR-Wind background solver development efforts, its current status, and future work.

3.1 NALU-WIND

The Nalu-Wind application utilizes the Sierra Toolkit (STK) libraries [3], which are distributed with the Trilinos project [7]. The STK Mesh module is used to store the computational mesh and fields, and also provides data access and traversal on GPU platforms. The Trilinos/Kokkos library is used heavily, both within STK Mesh and directly in Nalu-Wind, to provide execution constructs (looping mechanisms as well as data structures) that allow performance portability across traditional CPUs as well as GPU platforms.

A significant portion of the NGP work in Nalu-Wind has been to ensure that the process of assembling linear systems is done on the GPU as much as possible. The initialization of the graph for the linear system must be done on the host CPU. The computation of coefficients for the linear system, including traversing the mesh and master-element calculations, is performed on the GPU. This aspect has been successful and assembly appears to be performing well.

It has been a challenge to ensure that fields (values associated with mesh nodes, edges, etc.) are correctly synchronized between the CPU and GPU. It is necessary to ensure that, when accessing a field on the GPU, it has not been updated more recently on the CPU, and vice versa. The STK Mesh Fields have API calls for marking a field as modified on one memory space or the other, and for sync'ing a field from one memory space to the other. Unfortunately this allows programming errors where the modify or sync call is omitted in one memory space, leading to the situation where a field is out of date when it is later accessed in the other memory space. The STK team has been working to develop a robust capability to detect this kind of programming error.

For blade-resolved simulations of wind turbine flows, there are additional time per time-step costs arising due to mesh motion – update coordinates, compute mesh velocities and geometric conservation terms, update overset connectivity, and, for coupled overset simulations, updating the linear system graphs. With the recent adoption of the *decoupled* overset-solve approach, demonstrated in FY20-Q3 milestone, the linear system graphs need to be computed only once during initialization and can be reused throughout the simulation. However, the other updates remain a bottleneck and will be addressed in the coming year.

3.2 NALU-WIND *HYPRE* ASSEMBLY

In the FY20 Q2 milestone report, we reported on a prototype implementation for executing *hypre* linear system assembly on GPUs in Nalu-Wind. That prototype implementation was written in CUDA thus it only supported NVIDIA hardware–the original CPU implementation had to be preserved in order to execute on standard hardware.

Since then, significant improvements to the Nalu-Wind assembly process have been made that address portability and performance. With regards to portability, the algorithm has been completely rewritten using the Kokkos API. Native CUDA and CPU kernels have been removed in favor of this more portable approach. This allows us to have a single implementation across CPUs and GPUs. We have implemented a number of performance improvements that have made substantial improvements to different parts of the assembly algorithm. This includes, but is not limited to, memory-usage optimizations, up-front graph computation, as well as fast constrained binary search algorithms for rapid matrix element memory location determination. Many of these optimizations are in the current master branch of Nalu-Wind.

All that being said, significant speed improvements can and will be achieved with continued development. This motivates a brief discussion of the linear-system assembly implementation Currently, the algorithm is structured into 3 stages.

Stage 1: Graph Computation Stage

Stage 2: Device Coefficient Application: Assemble Compressed Sparse Row (CSR) matrix and RHS Vectors

Stage 3: Assemble *hypre* Linear System

The graph-computation stage computes the sparsity pattern of Linear System. It provides the exact amount of storage necessary for the matrix values. The majority of this computation runs on the CPU though at the very end, key data structures are either moved or computed on the GPU for use in the next stage. This algorithm is sequential though it could be made more GPU friendly with a significant refactorization. Thus acceleration of this stage is challenging though feasible.

In the Device Coefficient Application stage, the matrix values are computed. This stage runs entirely through a Kokkos implementation. Atomics are used accumulate the matrix and right-hand-side elements. It runs on CPU or GPU depending on the backend compilation target. This stage runs with good performance, especially on GPUs, thus significant additional optimization work is not necessary, though we have targeted a few potential improvements.

In the final stage, the assembled CSR matrix is then used to build the Nalu-Wind *hypre* linear system. We use the *hypre* API methods

`HYPRE_IJMatrixSetValues`

`HYPRE_IJMatrixAddToValues`

`HYPRE_IJVectorSetValues`

`HYPRE_IJVectorAddToValues`

to build the matrix/RHS in 2 steps. First we apply `HYPRE_IJMatrixSetValues`, `HYPRE_IJVectorSetValues` to set the matrix/RHS of the owned rows on the calling MPI rank. In order to set the off-rank matrix/RHS elements, we then call `HYPRE_IJMatrixAddToValues`, `HYPRE_IJVectorAddToValues` using the shared matrix/RHS elements as input. The beauty of this implementation is that it completes the assembly in 4 *hypre* API calls. It then leverages the internal-messaging structure of *hypre* to properly build the full matrix/RHS.

3.3 SOLVERS

To solve a linear system $A\mathbf{x} = \mathbf{b}$, *hypre* and Trilinos employ the generalized minimum residual (GMRES) iteration with a Gauss-Seidel preconditioner for momentum and algebraic multigrid (AMG) preconditioner for continuity using either Chebyshev or Gauss-Seidel smoothers. The Gauss-Seidel (GS) iteration is based on the matrix splitting $A = L + D + U$, where L and U are the strictly lower and upper triangular parts of the matrix A, respectively. Then, the traditional GS iteration updates the solution based on the following recurrence,

$$\mathbf{x}_{k+1} := \mathbf{x}_k + M^{-1}\mathbf{r}_k, \quad k = 0, 1, 2, \ldots \tag{1}$$

where $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, and $M = (L + D)$ or $M = (U + D)$ for the forward or backward sweeps, respectively. When the spectral radius of the iteration matrix, $\rho(I - M^{-1}A)$, is less than one, the iteration is guaranteed to converge. When the matrix A is symmetric, a symmetric variant of GS (SGS) performs the forward sweep followed by the backward sweep to maintain the symmetry of the matrix operation.

To avoid explicitly forming the matrix inverse M^{-1} in the iteration (1), a sparse-triangular solve is used to apply M^{-1} to the current residual vector $\mathbf{r}_k$. Unfortunately, it is a challenge to implement a scalable parallel sparse triangular solve on a distributed-memory computer, where both Trilinos and *hypre* distribute the matrix and the vectors in a 1–D block row fashion among the MPI processes (the local matrix on the p-th MPI process is rectangular, consisting of the square diagonal block $A^{(p)}$ for the rows owned by the MPI process and the off-diagonal block $E^{(p)}$ for the off-process columns. In particular, on each MPI rank, *hypre* stores the local diagonal block $A^{(p)}$ separately from the off-diagonal block $E^{(p)}$). The standard approach based on level-set scheduling will compute the independent set of the solution elements $\mathbf{x}_k$ in parallel at each level of the scheduling. However, the sparsity structure of the triangular matrix often limits the parallelism that the solver can exploit (e.g., the sparsity structure may lead to a long chain of dependency with a small number of solution elements that can be computed at each level). In addition, at the start of each level, neighboring processes need to exchange the elements of the solution vector $\mathbf{x}_k$ on a processor boundary for updating their local right-hand-side vectors.

To improve the solver scalability, both Trilinos and *hypre* implement a hybrid variant of Gauss-Seidel where the neighboring processes first exchange the elements of the solution vector $\mathbf{x}_k$ on the boundary to compute the local part of the residual vector $\mathbf{r}$, but then each process independently applies the local triangular solve. Hence, M is a block diagonal matrix with each diagonal block corresponding to the triangular part of the local diagonal block $A^{(p)}$ on each process (as in block Jacobi).

Furthermore, in *hypre*, each process may apply the multiple local GS sweeps for each round of the neighborhood communication. With this approach, each local iteration updates only the local part of the vector $\mathbf{x}_{k+1}$ (during the local iteration, the non-local solution elements on the boundary are not kept consistent among the neighboring processes). Hence, the local GS iteration, on the p-th MPI rank, solves the linear system $A^{(p)}\mathbf{x}^{(p)} = \mathbf{b}^{(p)} - E^{(p)}\mathbf{y}^{(p)}$ where $\mathbf{y}^{(p)}$ is the non-local part of the solution vector exchanged before the start of the local GS iteration. In contrast, Trilinos currently computes the global residual vector for each local iteration through neighborhood communication. The only exception is the symmetric GS iteration that applies both the forward and backward iterations after a single round of the neighborhood communication.

In this hybrid GS, each process can apply the local sparse-triangular solve or relaxation algorithms independently, improving the parallel scalability compared to the global sparse triangular solve. This hybrid algorithm is shown to be effective, and scalable, on many problems (e.g., the GMRES iteration counts remain roughly constant with an increasing process count). However, to implement the local iteration, each process must still perform a local sparse-triangular solve. In the present report, we consider two-stage Gauss-Seidel that uses a fixed number of "inner" stationary iterations for approximately solving the triangular system with M,

$$\widehat{\mathbf{x}}_{k+1} := \widehat{\mathbf{x}}_k + \widehat{M}_k^{-1}(\mathbf{b} - A\widehat{\mathbf{x}}_k), \quad k = 0, 1, 2, \ldots \tag{2}$$

where $\widehat{M}_k^{-1}$ represents the approximate triangular solution, i.e., $\widehat{M}_k^{-1} \approx M^{-1}$.

In this study, we used Jacobi-Richardson (or Jacobi) iteration for the inner iteration. In particular, if $\mathbf{g}_k^{(j)}$ denotes the approximate solution from the j-th inner iteration at the k-th outer iteration, then we let our initial solution to be the diagonally-scaled residual vector, i.e.,

$$\mathbf{g}_k^{(0)} = D^{-1}\mathbf{r}_k, \tag{3}$$

and the $(j+1)$st JR iteration computes the approximate solution by the recurrence

$$\begin{aligned} \mathbf{g}_k^{(j+1)} &:= \mathbf{g}_k^{(j)} + D^{-1}(\mathbf{r}_k - (L+D)\mathbf{g}_k^{(j)}) && (4) \\ &= D^{-1}(\mathbf{r}_k - L\mathbf{g}_k^{(j)}). && (5) \end{aligned}$$

Fig 1 displays the pseudo-code of the resulting two-stage algorithm.

Each inner sweep of the two-stage GS performs about the same number of floating-point operations (flops) as the triangular solve, but it is based on the sparse matrix vector multiply (SpMV) which is much easier to parallelize than the triangular solve used for the traditional GS (1). When a small number of inner iterations is needed, the two-stage Gauss-Seidel can outperform the traditional GS, especially on a many-core architecture.

***Implementation in* hypre**

The two stage Gauss Seidel smoother has been implemented in a branch of *hypre* that is current with *hypre* master, though this code has not been merged into the master branch. The *hypre* code version is listed in Table 2. At the time of writing this report, a Pull-Request has been submitted to merge this development branch into the *hypre* master branch. We expect this to be completed in November of 2020.

The *hypre* programming model for leveraging Nvidia GPUs relies on Unified Virtual Memory (UVM), combined with CUDA libraries, such as CUSPARSE, CUBLAS, and Thrust, as well as custom kernels. Currently, *hypre* CUDA capabilities only support 32-bit integer values for the row and column indices. It was necessary to support this in Nalu-Wind during the assembly algorithm. The two-stage Gauss Seidel has been implemented as a custom CUDA kernel which can be accessed as a smoother algorithm in the Boomer

```
for $t = 1, 2, \ldots, n_t$ do
1. // exchange interface elements of current solution
    for $k = 1, 2, \ldots, n_k$ do
1.      // compute new residual vector for forward sweep
        $\mathbf{r}_k^{(p)} := \mathbf{b}^{(p)} - A^{(p)}\mathbf{x}_k$
2.      // perform local inner Jacobi iteration
        $\mathbf{g}_1 := (D^{(p)})^{-1}\mathbf{r}_k$
        for $j = 1, 2, \ldots, n_j$ do
            $\mathbf{g}_{j+1}^{(p)} := (D^{(p)})^{-1}(\mathbf{r}_k^{(p)} - L^{(p)}\mathbf{g}_j^{(p)})$
        end for
3.      // update solution vector
        $\mathbf{x}_k^{(p)} := \mathbf{x}_k^{(p)} + \mathbf{g}_{n_j+1}^{(p)}$

4.      // compute new residual vector for backward sweep
        $\mathbf{r}_k^{(p)} := \mathbf{b}^{(p)} - A^{(p)}\mathbf{x}_k$
5.      // perform local inner Jacobi iteration
        $\mathbf{g}_1 := (D^{(p)})^{-1}\mathbf{r}_k$
        for $j = 1, 2, \ldots, n_j$ do
            $\mathbf{g}_{j+1}^{(p)} := (D^{(p)})^{-1}(\mathbf{r}_k^{(p)} - U^{(p)}\mathbf{g}_j^{(p)})$
        end for
6.      // update solution vector
        $\mathbf{x}_{k+1}^{(p)} := \mathbf{x}_k^{(p)} + \mathbf{g}_{n_j+1}^{(p)}$
    end for
end for
```

Figure 1: Pseudo-code of two-stage hybrid Symmetric Gauss-Seidel iteration, using Jacobi-Richardson as the inner sweep. Trilinos sets $n_k = 1$ and hence, performs the neighborhood communication once per sweep.

AMG implementation. A reference CPU implementation has also been developed. In any given row in a *hypre* sparse matrix, the diagonal is always stored the first element. This enables fast access to the diagonal value, which is useful for a variety of algorithms including two stage Gauss Seidel. However, the remaining columns/values often have a random ordering. The two stage Gauss Seidel uses the lower triangular matrix in a SpMV like operation. This can be efficiently implemented in CUDA using csr-SpMV implementation that ignores the upper triangular component. No additional storage is required.

3.4 AMR-WIND

For blade-resolved simulations of multi-turbine wind farms, a significant majority of the computational cells are used to resolve the background atmospheric boundary layer (ABL) flow and the wake structures behind the turbine rotors, and only a small portion of the computational domain is occupied by the turbine structures themselves. The flexibility offered by unstructured meshes is most useful in resolving the surfaces of the turbine structures. In most practical simulations, the background flow is best modeled using structured grids. However, despite having topologically structured grids, Nalu-Wind linear solvers do not exploit the simpler mesh data structures, which can result in sub-optimal strong and weak scaling performance. Since the ExaWind simulation framework has adopted overset mesh methodology to account for arbitrary turbine motion, it can readily support a hybrid solver strategy, i.e., use an unstructured mesh solver (Nalu-Wind) to model the region close to the turbine structures (blades, nacelles, etc.) that is coupled to a background structured solver (AMR-Wind) that solves the atmospheric boundary layer flow. The advantage of this hybrid solver strategy is that the structured block solver can use geometric multigrid algorithms to offer better performance when solving the pressure Poisson problem. Key to the success of the hybrid solver strategy is the decoupling of linear systems, which was the topic of the FY20 Q3 milestone. In this approch, AMR-Wind and Nalu-Wind use their preferred linear-solver strategy and field values are shared through an outer iteration.

In FY20, the AMR-Wind development efforts were focused on establishing a baseline atmospheric boundary layer (ABL) solver on top of the AMReX library. AMR-Wind solves the incompressible Navier-Stokes equations using a split-operator approach. The code is a wind-focused fork of *incflo*, an AMReX solver that was written from scratch to target GPUs. Turbulence models and LES shear-stress wall boundary conditions have been added to the codebase through funding from DOE WETO high-fidelity modeling (HFM) project. The code has been verified for canonical ABL problems documented in literature.

AMR-Wind has been coupled to Nalu-Wind using an overset approach, through the TIOGA library. The initial setup has been verified for simple problems such as convecting Taylor vortex and laminar flow past a sphere.

4. NALU-WIND BLADE-RESOLVED SIMULATIONS OF NREL 5-MW WIND TURBINE

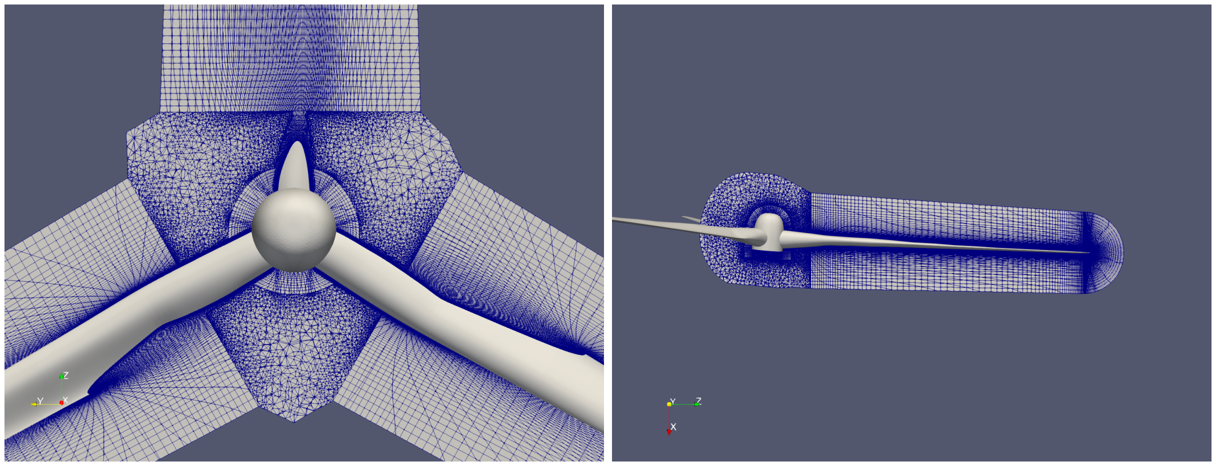
One of the primary objectives of this milestone was to evaluate the strong scaling performance of Nalu-Wind codebase where a significant portion of code execution, especially linear-solver assembly and solve, was performed on GPUs. Reynolds-averaged Navier-Stokes (RANS) simulations, using the $k-\omega$ SST turbulence model, were performed at a uniform inflow of $U = 8$ m/s. A fixed timestep size was used, such that the blade rotates 0.25° at each timestep. The NREL 5-MW turbine [9] is a 126 m diameter reference turbine, designed for use in research of offshore wind. While no such turbine exists, it is widely used in the wind research community and thus is a good baseline turbine geometry to study the capabilities of the code and perform code-to-code comparisons with other simulations published in the literature. For the purposes of this study the turbine geometry was simplified by ignoring the tower and nacelle structures; only the three blades and the hub were modeled.

The simulations performed for this milestone use the same computational meshes that have been extensively verified and reported in FY19-Q2 report. The turbine geometry is resolved using body-conforming meshes that are embedded inside a wake-resolving background mesh. In order to transition smoothly to the hub structure, the structured mesh on the blade surface was constructed outboard of the 20% span – see Fig. 2. The sections inboard used unstructured mesh to transition smoothly to the hub mesh. The near-body mesh was embedded in a wake-capturing mesh that extended half a rotor diameter upstream and about 5 rotor

Table 1: Publicly hosted Git repository URLs and commit identifiers (SHA1) for the codebases used to perform the scaling studies using the Trilinos linear solver stack documented in Sec. 4.

Code	Location	SHA1
Nalu-Wind	https://github.com/exawind/nalu-wind/	ffc049e
Trilinos	https://github.com/trilinos/Trilinos/	6adf2c3

diameters downstream. The wake-capturing mesh was enclosed within a fully unstructured mesh that formed the outer domain. The overall computational domain extended 5 rotor diameters upstream, 10 diameters downstream, and 10 diameters in the lateral directions. The mesh contained a total of 38 million elements (23 million nodes), and the near-body mesh contained 7 million elements for all three blades.



(a) Front view of the NREL 5-MW near-body mesh (b) Side view of the NREL 5-MW near-body mesh

Figure 2: Front and side views of the near-body mesh used to model the NREL 5-MW rotor with Nalu-Wind. The hybrid mesh consists of structured, hyperbolically extruded mesh on the rotor blades and a fully unstructured mesh block around the hub and the hub-blade transition region.

While the milestone description describes a *parked turbine*, simulations were performed for a rotating turbine to evaluate the performance of mesh motion and geometry update executing on GPUs. Simulations of a rotating turbine is closer to the final ExaWind challenge problem and, therefore, exceeds the expectations set for the FY20-Q4 milestone. It must, however, be noted that while the mesh motion portions of the code were executing on GPUs, the overset connectivity (performed by TIOGA library) was still executing on the host. The overhead of performing connectivity updates on host as well as the data exchange at every timestep between host and device memory is benchmarked in the current study and will be a focus of future work.

All simulations in this section used the decoupled, alternating Schwarz approach that was described in FY20-Q3 report. Four outer Picard iterations were used in all cases. Using decoupled overset simulations offers two advantages when executing on GPUs: 1. It removes the constraint rows from linear systems, which was shown to improve the convergence of Poisson systems, and 2. It eliminates the need for STK ghosting of donor elements to the receptor MPI ranks.

In this section, we present results from the strong scaling studies performed using the Trilinos and *hypre* solver stacks for the wind-turbine simulations. All experiments were run on the Summit supercomputer[1] at Oak Ridge National Laboratory. Summit has 4608 compute nodes, each with two IBM Power9 CPUs and six NVIDIA Volta V100 GPUs. Each Power9 CPU has 22 cores, and there are 512 GB of DDR4 memory available to the CPUs.

[1] https://docs.olcf.ornl.gov/systems/summit_user_guide.html#system-overview

4.1 STRONG-SCALING PERFORMANCE OF NALU-WIND USING TRILINOS SOLVERS

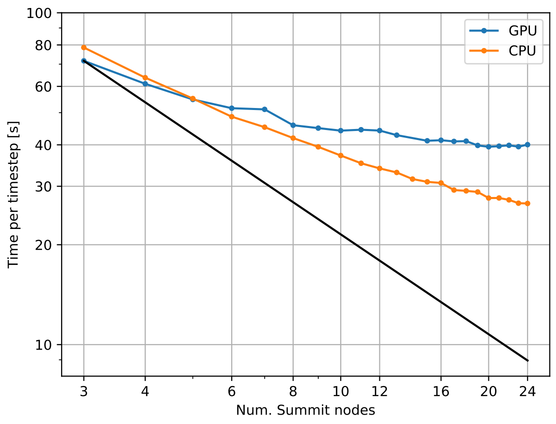
For the experiments in this section, unless otherwise specified, the MueLu AMG solvers used for the continuity solve employ a degree-2 Chebyshev smoothing and a serial direct solver. The linear solvers for other physics are GMRES preconditioned by standard Gauss-Seidel, multi-threaded Gauss-Seidel, or two-stage Gauss-Seidel. A single Summit node has 42 total CPU cores (two 21-core CPUs) and 6 GPUs. When doing performance comparisons, the Trilinos experiments compare performance of all a node's GPU devices to all of the node's CPU cores. When this isn't feasible, e.g., a problem is not large enough to occupy all GPUs or cores, we fall back to using the comparison ratio of a single GPU to approximately 7 CPU cores. Table 1 shows the URLs of the publicly hosted Git repositories and the commit identifiers that were used to perform the scaling studies.

Figure 3 shows the strong scaling performance of Nalu-Wind for the NREL 5-MW blade resolved simulations using the Trilinos solver stack for solving the momentum, continuity, and the scalar transport equations (turbulent kinetic energy (TKE) and specific dissipation rate (SDR)). The simulations were performed for a total of ten timesteps and the timings reported are an average over those ten timesteps. The total time per timestep is dominated by two activities. The first, indicated by *non-linear iterations* is the time per timestep spent in assembling the linear systems and solving those systems for all Picard iterations. In the current simulation, four Picard iterations were performed within each timestep. The second, indicated by *pre-timestep work*, is the time spent in updating the mesh to account for turbine rotation, recomputing the overset mesh connectivity, and reinitializing the linear system data structures at the beginning of each timestep. The results indicate that execution GPUs are faster than on equivalent number of CPU cores (7$\times$) up to eight Summit nodes, beyond which the execution time on CPUs is faster than GPUs. Scaling of the non-linear iterations (i.e., linear system assembly, preconditioner setup, and solve) is quite poor on GPUs. While scaling on CPUs are better than what is observed for GPUs, it is far from ideal scaling (shown as black line in the plots).

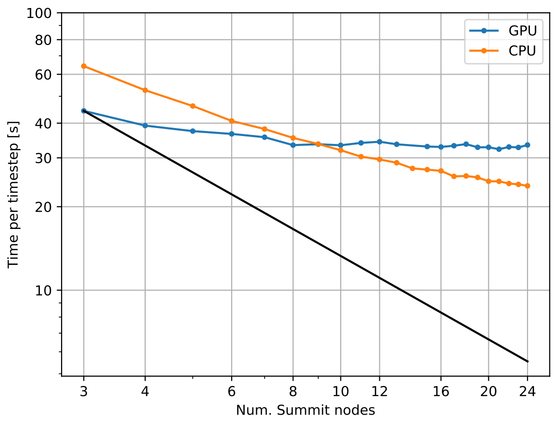
It is also observed that the time spent in pre-timestep activities is considerably higher when executing on GPUs. This is because currently the overset connectivity updates are performed exclusively on CPUs and suffer from additional overhead of data transfer of mesh data from GPU to CPU memory at the beginning of each timestep and the transfer of field from and back to GPU memory after each Picard iteration. Transitioning the overset search algorithms to execute on GPUs is currently underway and will be the main focus of FY21 activities, and is expected to address this bottleneck.

Figure 4 shows the breakdown of the time per timestep for momentum and continuity solves into various components: Initialization of Tpetra matrix and vector data structures, assembling the matrix coefficients and right hand side by Nalu-Wind computational kernels, preconditioner setup (especially multigrid setup), and the time spent in iterative solution of the linear systems. On both CPUs and GPUs, the time spent in continuity solves dominate the overall execution time. For the momentum solves, the linear system reinitialization costs are a significant portion of the overall time as it is assembling a 3$\times$ block diagonal system. While there is an overall decrease in the time spent on momentum solves when executing on GPUs, the strong-scaling behavior of all components need improvement.

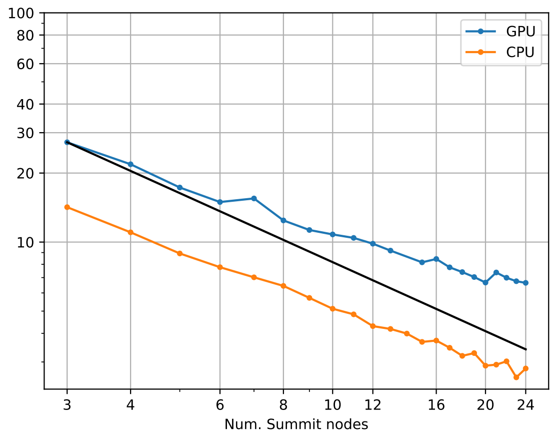
On the other hand, the time spent in solving the continuity system shows a marked decrease when moving to GPUs. However, the gains in solver time are more than offset by the increase in preconditioner setup time on GPUs. In general, the strong scaling behavior of both solve and preconditioner setup is poor and will be the focus of future work within the ExaWind project.



(a) Total time per timestep

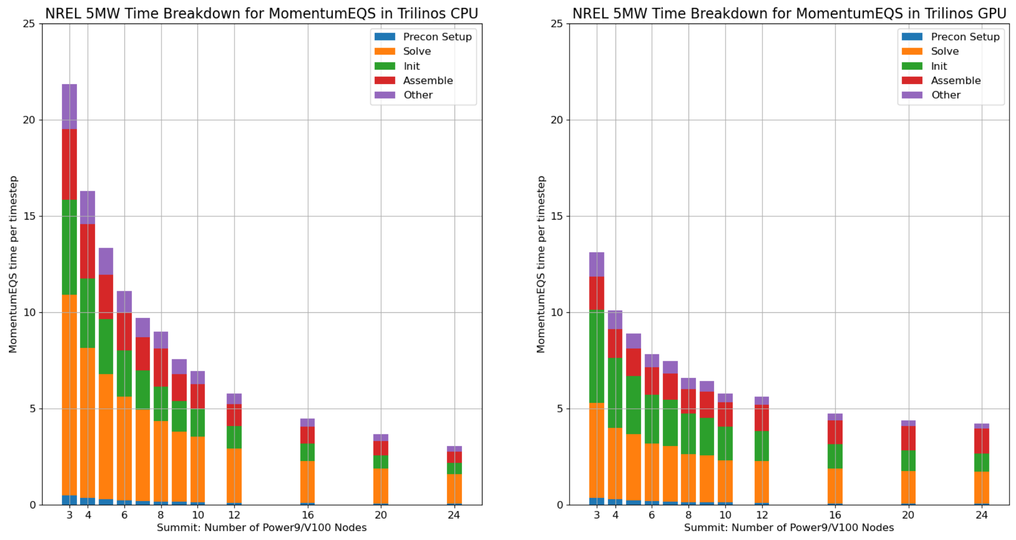


(b) Non-linear iterations

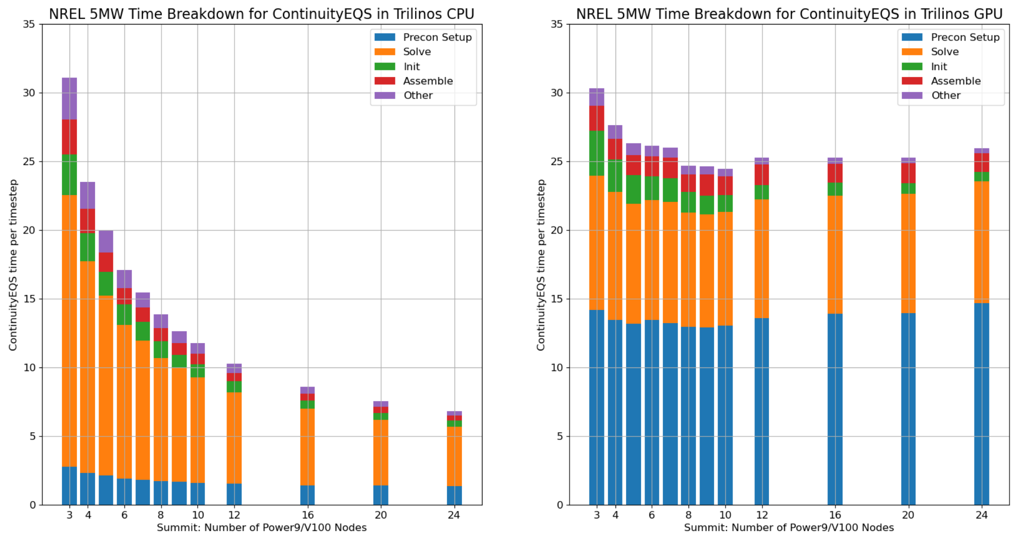


(c) Pre-timestep work

Figure 3: Comparison of the strong scaling performance of the total time per timestep when executing Nalu-Wind with Trilinos solvers on ORNL Summit Power9 CPUs and NVIDIA V100 GPUs compared on a per-node basis (i.e., 6 V100 GPUs are considered equivalent to 42 Power9 CPU cores). Simulation was performed for 10 timesteps and execution times reported in the plot use an average time over those ten timesteps. The two dominant components of the total time per timestep, non-linear iterations and pre-timestep work, are shown in the middle and right figures.



(a) Breakdown of the various linear solver stages during the momentum solve. Left: CPU, right: GPU.



(b) Breakdown of the various linear solver stages during the continuity solve. Left: CPU, right: GPU.

Figure 4: Breakdown of the total time spent in momentum solves for the blade-resolved NREL 5-MW rotor simulation when executing Nalu-Wind on ORNL Summit Power9 CPUs and NVIDIA V100 GPUs using the Trilinos solver stack. Comparisons are performed on a per-node basis, i.e., 6 V100 GPUs are considered equivalent to 42 Power9 CPU cores. The timing breakdown are as follows: *init* – time spent in linear system graph creation, *assemble* – time spent by Nalu-Wind computational kernels in assembling the matrix coefficients and right hand side residual vector, *precon setup* – time setting up the preconditioner, *solve* – time spent in iterative solves, *other* – time spent in other operations, e.g., computing gradients, etc.

Table 2: Publicly hosted Git repository URLs and commit identifiers (SHA1) for the codebases used to perform the scaling studies using the *hypre* linear solver stack documented in Sec. 4.

Code	Location	SHA1
Nalu-Wind Master	`https://github.com/exawin/nalu-wind/`	`7765b5b`
Nalu-Wind Fork	`https://github.com/PaulMullowney/nalu-wind/`	`7765b5b`
Hypre Master	`https://github.com/hypre-space/hypre/`	`636706a`
Hypre Fork	`https://github.com/PaulMullowney/hypre/`	`2acced4`
Trilinos	`https://github.com/trilinos/Trilinos/`	`19b4e9b`

4.2 STRONG SCALING PERFORMANCE OF NALU-WIND USING *HYPRE*

In the previous section we reported on the scaling performance of the Trilinos solver stack to solve all the equations in the blade-resolved simulations. In this section, we report several sets of results.

Case 1: We use *hypre* for the Continuity Solve, Trilinos for the other equations. In this case, we use a monolithic momentum solve in Trilinos.

Case 2: We use *hypre* for the Continuity Solve, Trilinos for the other equations, and we reuse the linear systems built at the beginning of the simulation for each equation system. In this case, we use a monolithic momentum solve in Trilinos.

Case 3: We use *hypre* for Continuity and a segregated momentum solve and Trilinos for the other equations including turbulent scalar transport. Also, we reuse the linear systems built at the beginning of the simulation for each equation system.

Case 4: We use *hypre* for all equations including the segregated momentum equation. Also, we reuse the linear systems built at the beginning of the simulation for each equation system.

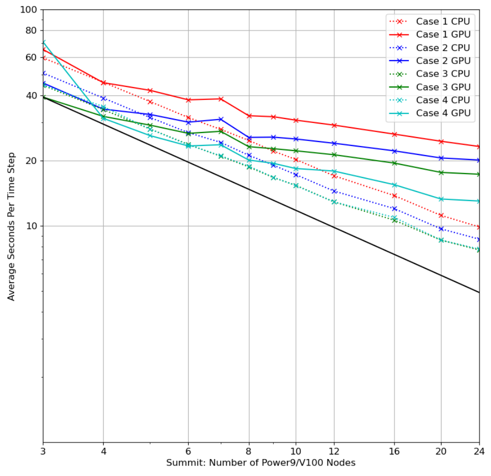
Each of these modifications provides nontrivial performance benefits, especially in the strong scaling limit. All results using *hypre* solver pathway were generated with the code versions listed in Table 2.

We use *hypre* 's GMRES Krylov solver using the Boomer AMG preconditioner. The two stage Gauss Seidel smoother is used with a maximum of 7 AMG levels. We use 2 internal smoother sweeps, PMIS coarsening and extended interpolation. Each of these algorithms has wide stecil widths. This solver configuration has been tested on various linear systems arising in Nalu simulations and has the optimal preconditioner setup and solve time. Assembly, preconditioner setup, and solve are ALL performed on the GPU.

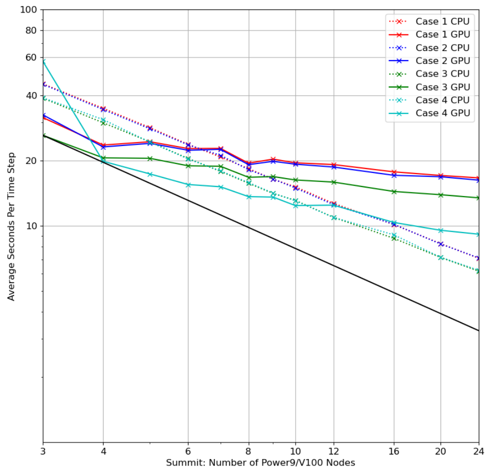
Figure 5 shows results for 10 Time steps of the blade resolved NREL 5-MW turbine under the 4 different simulation setups described above. The left panel panel shows the average total timer per time step, the center panel shows the average non-linear iteration time, and the right panel shows the average pre-timestep work. Non-linear iteration time consists of preconditioner setup and solve where as setup time consists of graph construction and linear system assembly. Several key observations are:

- reusing the graph/linear system provides performance benefit,
- the CPU strong scaling is near optimal all the way out to 1008 CPU cores (∼22K DOFs/core),
- the GPU strong scaling is less optimal and the average cost per iteration is at most 2x slower than the CPU, and
- and solving all systems using the *hypre* pathway yields significant performance benefits. In particular, using a segregated momentum solve provides substantial performance benefits in terms of run time and memory consumed.

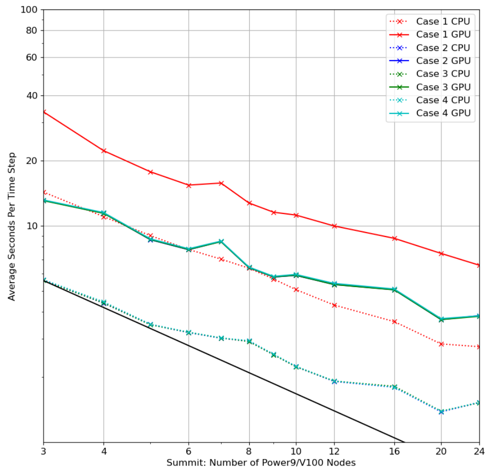
The third observation, in particular the less optimal scaling, is not entirely surprising. GPUs typically require large amounts of work in order to fully leverage their massively parallel architecture. Thus, we expect GPUs to perform better as DOFs/GPU increases as it enables computation (SpMV and other algorithm elements) to be a more dominant factor than communication. As DOFs/GPU decreases, messaging costs, including



(a) Total time per timestep



(b) Non-linear iterations



(c) Pre-timestep work

Figure 5: Comparison of the strong scaling performance of the total time per timestep when executing Nalu-Wind with *hypre* and Trilinos solvers on ORNL Summit Power9 CPUs and NVIDIA V100 GPUs compared on a per-node basis (i.e., 6 V100 GPUs are considered equivalent to 42 Power9 CPU cores). Simulation was performed for 10 timesteps and execution times reported in the plot use an average time over those ten timesteps. The two dominant components of the total time per timestep, non-linear iterations and pre-timestep work, are shown in the middle and right figures. Case 1: Hypre Continuity/Trilinos Other, Case 2: Hypre Continuity/Trilinos Other, Reuse Linear System, Case 3: Hypre for Continuity and Segregated Momentum/Trilinos Other, Reuse Linear System, Case 4: Hypre for all equations including segregated momentum, Reuse Linear System.

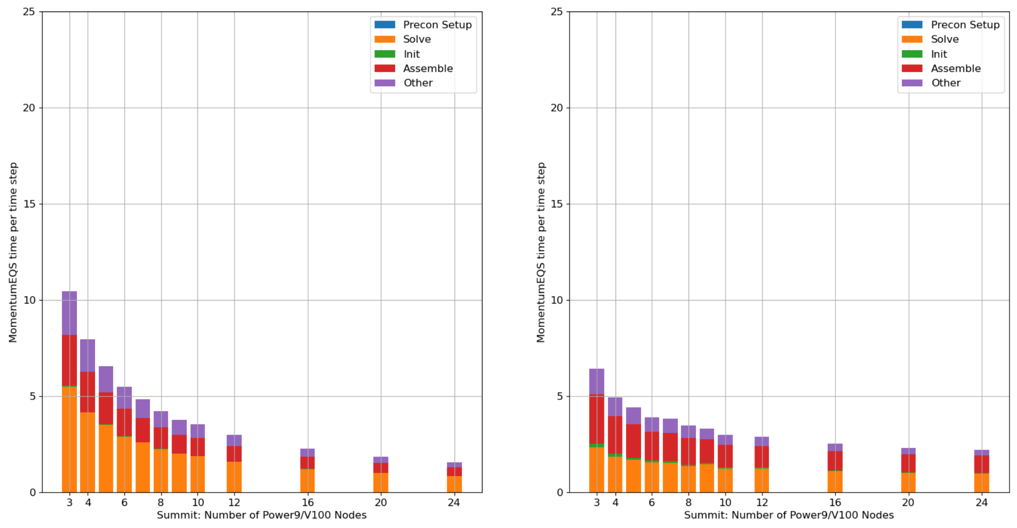
both device and MPI, will have more of an impact. Achieving better strong scaling requires very carefully designed compute kernels to hide these costs and overcome these hurdles.

In the regime where we expect GPUs to be more competitive, i.e. for 3 summit V100 nodes, we have have 1.27M DOFs/GPU. In this regime, non-linear iteration time is faster on the GPU however the setup costs are slower. Overall 7 CPU cores is roughly equal to 1 GPU. This motivates a deeper dive into the solver performance for the various equation systems. PS: I agree we don't need the momentum solve discussion in this section Figure 6 shows the breakdown of the time per timestep costs of the segregated momentum and continuity solves for increasing number of Summit compute nodes in Case 3. The CPU results are given in the left panel of each plot, the GPU results are in the right panel. The color of each bar shows different algorithmic components–assembly and load complete are added together and referred to together as Assemble. The y-axis of each plot is fixed at the same value in order show the relative costs associated with solving each physics equation and their algorithmic subcomponents. From this, we can easily deduce where we need to focus our GPU optimization efforts in the future.

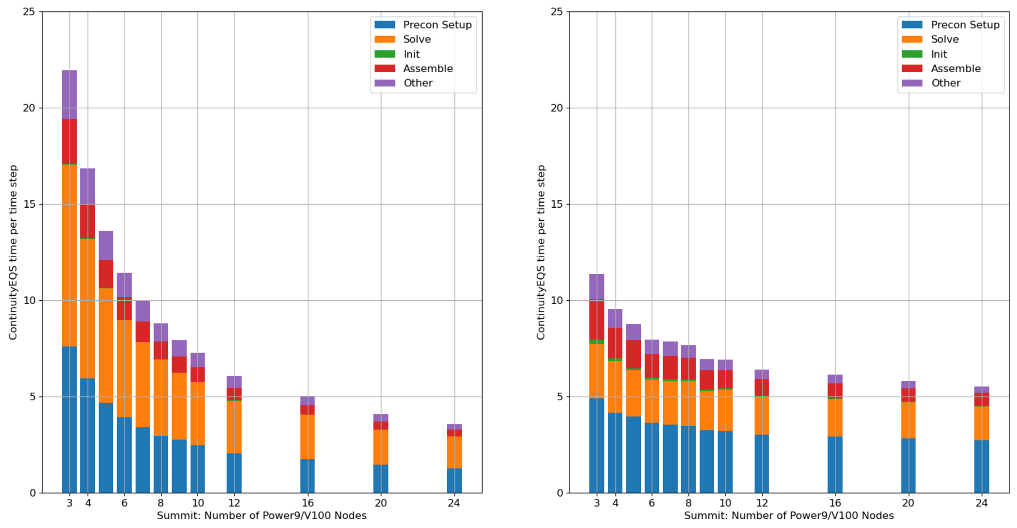
Overall, we see that Continuity is the dominant cost on the GPU. Digging deeper, we see that preconditioner setup is the most costly algorithm for the *hypre* solver stack (right panel of figure 6). We have relied entirely on the *hypre* team to implement this portion of the algorithm on GPUs. Overall the performance of AMG (Preconditioner) setup is good although it does not scale as well as its CPU counterpart. For simulations where the matrix is not changing all that often, one could use the same preconditioner for all Picard iters in a time step. This logic has already been implemented for the *hypre* solver stack in Nalu. For the cold start blade resolved simulation, one has to recompute the preconditioner every time the matrix is updated–the solvers will not converge otherwise. However, after many time steps, it may be possible to reuse the preconditioner and get a significant performance boost.

Solve performance is good on the GPU however it is not scaling particularly well. The low baseline for the solver performance for many DOFs/GPU is due to fast and effective nature of two stage Gauss Seidel smoother–the main workhorse of AMG. The reasons for the poor scaling are not known. This could be a side effect of using UVM memory or it is possible that communication is not being properly overlapped with computation in the SpMV algorithms. We will investigate this in the coming months.

On the other hand, the performance of Assemble is excellent. It is faster than the corresponding CPU version and it scales well. This is due in large part to significant time being spent on this part of the code in the past two quarters.



(a) Breakdown of the various linear solver stages during the segregated momentum solve in Case 3. Left: CPU, right: GPU.



(b) Breakdown of the various linear solver stages during the continuity solve in Case 3. Left: CPU, right: GPU.

Figure 6: Breakdown of the total time spent in segregated momentum and Continuity solves for the blade-resolved NREL 5-MW rotor simulation when executing Nalu-Wind on ORNL Summit Power9 CPUs and NVIDIA V100 GPU using *hypre* library (Case 3). The scalar transport equations are solved using the Trilinos solver stack. Comparisons are performed on a per-node basis, i.e., 6 V100 GPUs are considered equivalent to 42 Power9 CPU cores. The timing breakdown are as follows: *init* – time spent in linear system graph creation, *assemble* – time spent by Nalu-Wind computational kernels in assembling the matrix coefficients and right hand side residual vector, *precon setup* – time setting up the preconditioner, *solve* – time spent in iterative solves, *other* – time spent in other operations, e.g., computing gradients, etc.

5. MULTI-TURBINE ACTUATOR LINE SIMULATION IN ABL

This section is dedicated to evaluating the performance of the mid-fidelity actuator line model (ALM) in an atmospheric boundary layer simulation (ABL) with Nalu-Wind and the Trilinos linear-system solver stack. Actuator lines are becoming more common for modeling the complex multi-turbine interactions seen in wind farms, and so we evaluate Nalu-Wind's performance with a single turbine via strong scaling, and multiple turbines through a weak scaling study that is reflective of the multi-turbine wind farm use case.

5.1 ATMOSPHERIC BOUNDARY LAYER (ABL) MODELING IN NALU-WIND

Atmospheric boundary layers are modeled in Nalu-Wind by adding the three additional source terms to the filtered incompressible momentum equations (equation 6): the Earth's Coriolis force (term **V**), buoyancy effects through the Boussinesq approximation (term **VI**), and a horizontal-forcing term to drive the mean velocity at specific heights (term **VII**).

$$\underbrace{\frac{\partial}{\partial t}\left(\bar{\rho}\,\widetilde{u}_i\right)}_{\mathbf{I}} + \underbrace{\frac{\partial}{\partial x_j}\left(\bar{\rho}\,\widetilde{u}_i\widetilde{u}_j\right)}_{\mathbf{II}} = -\underbrace{\frac{\partial p'}{\partial x_j}\delta_{ij}}_{\mathbf{III}} - \underbrace{\frac{\partial \tau_{ij}}{\partial x_j}}_{\mathbf{IV}} - \underbrace{2\bar{\rho}\,\epsilon_{ijk}\,\Omega_j u_k}_{\mathbf{V}} + \underbrace{\left(\bar{\rho}-\rho_\circ\right) g_i}_{\mathbf{VI}} + \underbrace{S_i^u}_{\mathbf{VII}} + \underbrace{f_i^T}_{\mathbf{VIII}} \qquad (6)$$

These sources coupled with a stress boundary condition derived from the Monin-Obukhov theory have been shown yield ABL conditions have that can be tuned to match specific wind site locations and conditions (see [2, 1]). Further details on the theory can be found in [2].

In Nalu-Wind we typically run ABL simulations for two general configurations: *precursor* simulations and *inflow/outflow* simulations. Precursor simulations use a domain with periodic horizontal boundaries and detailed results for the performance of this case are provided in the FY20-Q2 milestone. The results from this domain run are used to provide initial conditions and inflow boundary conditions for the inflow/outflow simulations. Inflow/outflow simulations use prescribed inflow, with open boundaries and are typically what we use to study turbines and wind farms via actuators or blade-resolved turbine models. In this report we will run a hybrid of these two cases, a periodic box with turbines placed inside the domain. We use this configuration since we are running scaling studies with only 10 timesteps and the cost of generating a precursor for each mesh is not necessary to judge the how the code scales.

5.2 MODELING WIND TURBINES AS ACTUATOR LINES OR DISKS

Actuator methods are a common way to model turbines in ABL simulations. These methods discretize the turbine components (tower, blades, nacelle) as a series of discrete points where forces can be computed based on the local fluid properties. These forces are spread to the fluid domain using a Gaussian function as first proposed by [14].

There are two main types of actuator models for wind turbines, the actuator line model (ALM) and the actuator disk model (ADM), both of which are available in Nalu-Wind. The ALM discretizes the individual turbine blades with points and then the simulation resolves the motion of these blades while the ADM uses a disk to model the swept path of the turbine blades. In this sense the ADM is a time averaged result of the blade motion. The advantage of the ADM is that it can be run with no additional restriction on the timestep and still produce a reasonable approximation for the turbine effects. ALM resolves more of the wake dynamics by better capturing the blade motions and tip vortices but requires greater resolution in time and space. For illustrative purposes an image of an ALM in uniform inflow is provided in figure 7. The actuator in figure 7 utilized 50 points along each blade and 20 points for the tower to generate the resulting force displayed in the figure. For the remainder of this report the discussion will be restricted to the ALM method since it is the more computationally taxing of the two.

Nalu-Wind couples with OpenFAST, an open source "whole-turbine" model suite, to implement it's actuator models. A weak coupling is employed between the two models by the following procedure. First, Nalu-Wind computes the fluid properties and interpolates them to the exact location of the actuator points. Next, these fluid properties are supplied to OpenFAST to compute the forces and displacements of the actuator points. Nalu-Wind then spreads the forces computed by OpenFAST to the fluid domain and updates the actuator point locations for the next momentum equation solve.

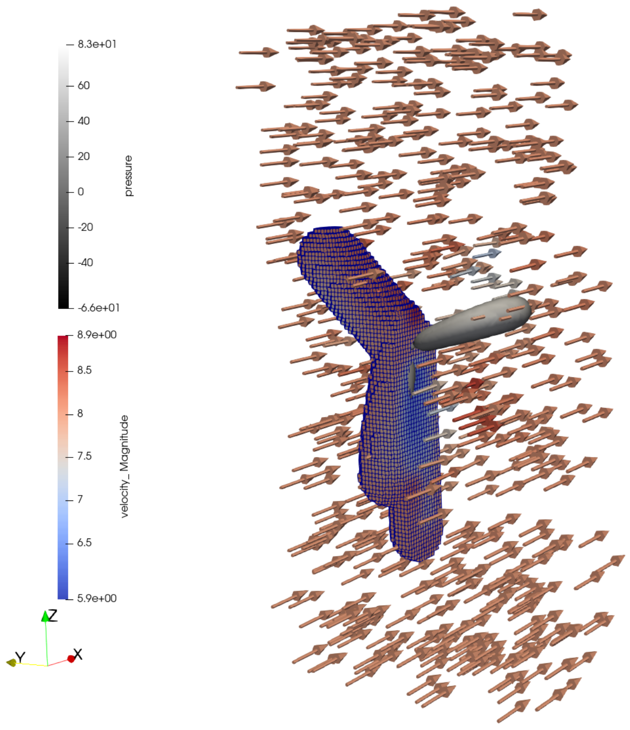


Figure 7: A actuator line model in uniform inflow conditions. Threshholding is applied to half the domain to show the extent the actuator source term propagates into the domain through the Gaussian spreading kernel (elements colored by velocity magnitude). An iso-surface of the actuator source term (magnitude=5) is also provided (colored by the pressure term).

OpenFAST has a wide array of capabilities and models for the turbine physics including structural and aerodynamic loadings, control systems, and the drive train efficiency. The combined results of these models generates the blade displacements, forces and several additional quantities of interest such as generator power, blade bending moments and torque. Leveraging the well established and validated OpenFAST model suite gives Nalu-Wind actuators a large array of capabilities that are providing very promising results in recent verification and uncertainty quantification studies [1, 8].

However, there is a portability penalty to pay for the foreseeable future when using OpenFAST because OpenFAST is principally written in Fortran. A GPU compatible version of OpenFAST would be a very large undertaking and there are no immediate plans to begin that process. As such, all OpenFAST computations in these simulations are restricted to the host. To minimize this penalty OpenFAST computations are distributed across the CPU's with their maximum level of parallelization (1 turbine per process), and with distributed memory techniques to minimize communication and maximize parallel utilization on the Nalu-Wind portions of the coupling procedure.

The most expensive operations for Nalu-Wind's actuator computations are the search for actuator points and the spreading of the actuator force. Both of these operations are performed at every timestep for the ALM method and they are both executed on the host because the search and interpolate operations have not been ported to the GPU yet. Flat, or one-dimensional, data structures are utilized for the actuator point locations and forces that allow every rank/device to access the data for every turbine. For perspective the memory foot-print of these data structures is relatively small with typically only $O(100)$ points per turbine. MPI all-to-all communications are done once per timestep to sync the point locations (pre-search), fluid properties (post-interpolation) and forces (pre-spreading). Once the point locations for all the turbines are known each rank can then do a local only search to find the points it contains. A coarse search is performed and cached to determine the local elements that are touched by the Gaussian kernel for each actuator point. The results of this coarse search are reused for a fine search to determine the exact location of the actuator points for interpolation of fluid properties. After the interpolation process is complete the fluid properties are synced across all ranks and OpenFAST can perform its computations. Then the forces are taken from OpenFAST, synced across all ranks and the coarse search results are re-used again to spread the forces to the fluid domain. The key item here is that using a local-only search allows for minimal communication costs and maximum distributed utilization.

However, it should be noted that this implementation is sensitive to the domain decomposition and distributed utilization is only available on the ranks that have elements intersecting with the Gaussian kernels.

5.3 SCALING STUDIES

5.3.1 *Strong scaling*

For the following strong scaling study, the ALM is used to model an ABL simulation with mesh resolution of 20 m. The ABL 20 m simulation contains 3.3 millions pressure DOFs, and the number of Summit nodes used for each run is increased from 1 to 7. Figures 8 shows the time per time step breakdown for the momentum and continuity equations. Both equations exhibit poor strong scaling behavior for the GPU as the number of nodes increases, similar to the results of the blade resolved simulations that were previously reported. One notable difference between these simulations and the blade resolved cases is that the assembly time, which includes the ALM kernels, dominates the time per time step for the momentum equation. The assembly time consistently drops for the CPU runs, but it remains relatively constant for the GPU runs and even increased a bit for the runs with 5 and 6 nodes. This is likely due to the difference in the domain decomposition between the CPU and GPU runs. On the GPU runs a 1:1 ratio is applied to the number of devices and hosts utilized per node. Therefore a GPU run using a single node will only have 6 host ranks while a CPU run will utilize all 42 processors on the Power 9 node. This leads to a large difference in how the domain is decomposed between ranks and the amount of work that needs to be done on the host. Since the GPU has 7× fewer ranks the domain per rank will be 7× larger than the CPU only runs and the local only search will take longer. Additionally, since search and interpolate are still restricted to host the GPU runs are further handicapped. The strong scaling should improve when Nalu-Wind's search and interpolate functions are ported to GPU's. Additional improvements can also be expected by expanding the number of host ranks that can be paired to a single GPU. The continuity equation is dominated by the solve for the pressure Poisson equation. This is a known scaling limitation for algebraic multigrid preconditioners, and is consistent with the results from the *hypre* runs.

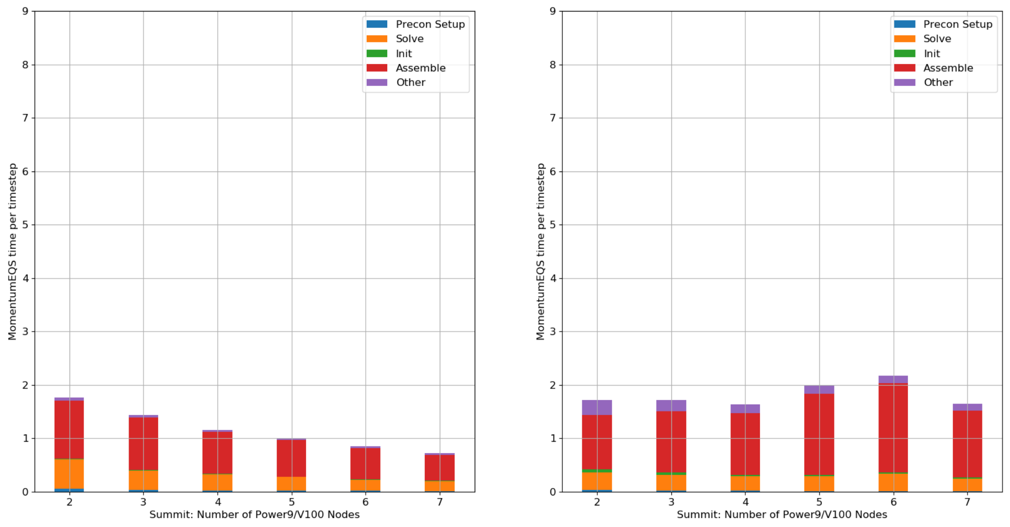
Figure 9 shows the strong scaling behavior of the ALM execution time per time step. While initially showing good scaling for the GPU with fewer than 3 nodes, the execution time increases for 4 and 5 nodes, before continuing to scale with more than 5 nodes. This could also be due to the overhead of having to move actuator data from device to host for the broadcast to all MPI ranks before moving it back to the device in addition to the issues with domain decomposition that were highlighted in the previous paragraph. Unfortunately, there is not much that can be done to reduce the host-device copy requirements as long as OpenFAST remains a host-only code.

5.3.2 Weak scaling

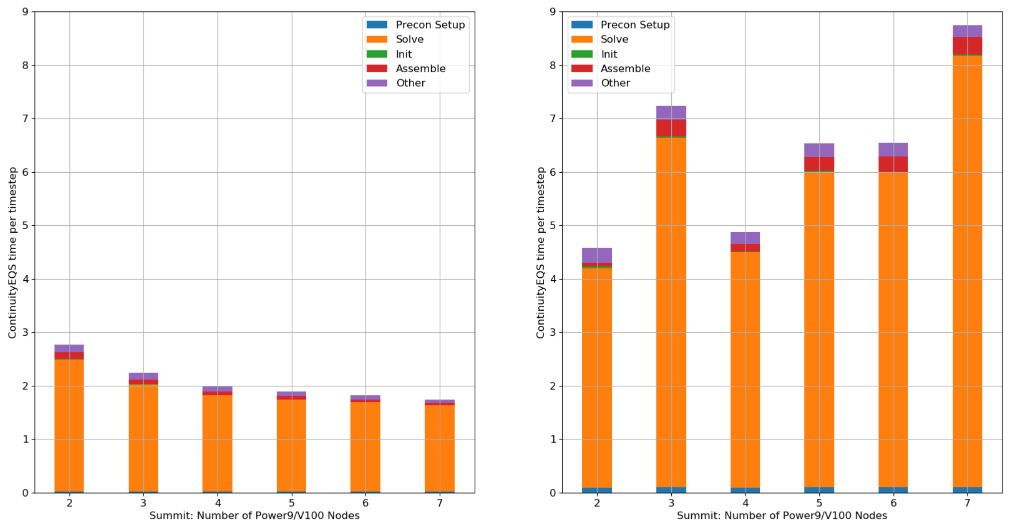
We now consider the weak scaling of the ALM method. Here the problem size per node is fixed as the number of nodes for each run increases. To do this, we begin with a domain of 1 km $\times$ 1 km $\times$ 0.5 km with a single turbine in the center of the domain. The domain is discretized into 100 $\times$ 100 $\times$ 50 elements. The domain is then elongated in the x-direction by essentially copying the domain and appending the copy to the end of the current domain to double the size of the problem. This leads to spatial extents of 2 km $\times$ 1 km $\times$ 0.5 km, then 4 km $\times$ 1 km $\times$ 0.5 km, and finally 8 km $\times$ 1 km $\times$ 0.5 km for the additional runs. For this weak scaling study, the number of elements per GPU (or 7 CPUs) remains constant at 500,000 with the same mesh size. With each elongation of the domain, the number of elements in the mesh is doubled and the number of turbines is also doubled to be 2, 4 and 8 turbines where each turbine is centered in a 1 km square. These meshes are run using 1, 2, 4, and 8 GPUs, respectively. This corresponds to 7, 14, 28, and 56 CPUs. This study is a pure weak scaling study since the problem size is truly the only thing being changed.

Figure 10 shows the timing breakdown for each substep of the momentum and continuity equations, respectively. Ideally, the timings should remain constant for perfect weak scaling because the load on each processor is held constant. For the momentum equation, both the CPU and GPU runs have increasing runtime with processor count. However, the GPU runs have a 2-3X speedup for the assembly and solve time, both of which dominate the runtime for the momentum solve, compared to the corresponding CPU runs. This performance improvement is slightly offset by the fact that the GPU also requires more time for the initialization and other substeps. A similar case is shown for the continuity equation, however, it is the solve that dominates the runtime. While both the CPU and GPU exhibit poor weak scaling, the GPU has a 1.5-2X speedup in solve and assembly time.

Figure 11 shows the total time per time step for the CPU and GPU weak scaling runs. Both curves increase with processor count, with the GPU showing slight improvements in runtime over the CPU. Figure 12 shows the weak scaling behavior for the actuator line execution time. The CPU and GPU actuator line execution times are nearly perfect in weak scaling, but the performance improvements of the GPU for the momentum and continuity equations are countered by the actuator line execution time. This is likely caused by the overhead of moving actuator data between the device and host in order to perform the broadcast to all MPI ranks.



(a) Breakdown of the various linear solver stages during the momentum solve. Left: CPU, right: GPU.



(b) Breakdown of the various linear solver stages during the continuity solve. Left: CPU, right: GPU.

Figure 8: Breakdown of the strong scaling performance of momentum and continuity solves for a single NREL 5MW turbine operating in ABL inflow using the Trilinos solver stack. The turbine blades are modeled using actuator lines. Comparisons are performed on a per-node basis, i.e., 6 V100 GPUs are considered equivalent to 42 Power9 CPU cores. The timing breakdown are as follows: *init* – time spent in linear system graph creation, *assemble* – time spent by Nalu-Wind computational kernels in assembling the matrix coefficients and right hand side residual vector, *precon setup* – time setting up the preconditioner, *solve* – time spent in iterative solves, *other* – time spent in other operations, e.g., computing gradients, etc.

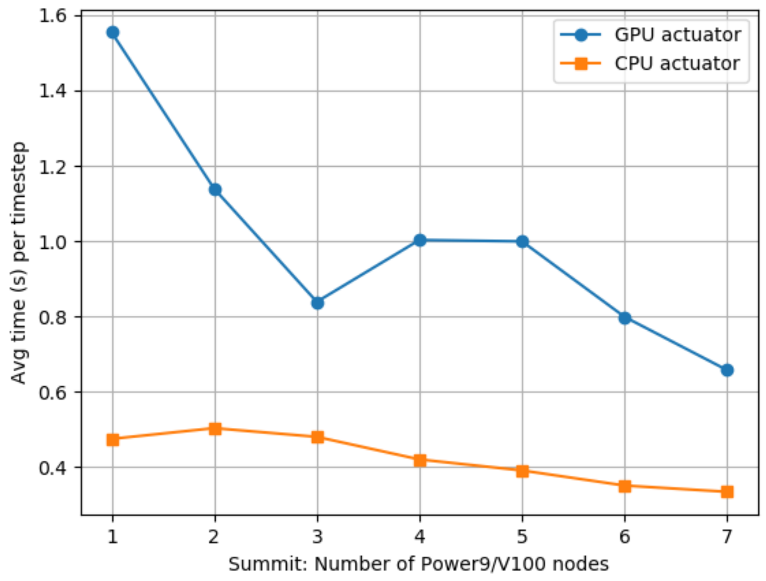
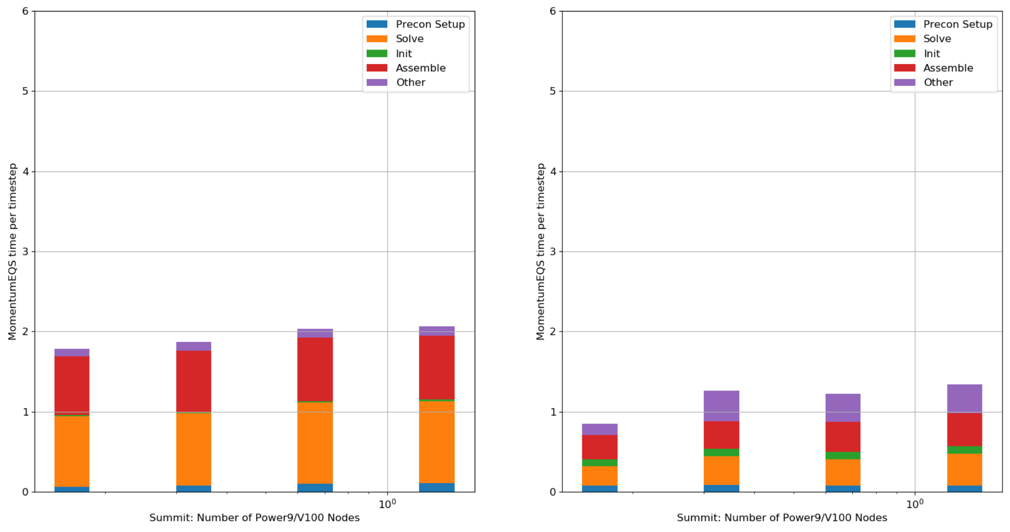
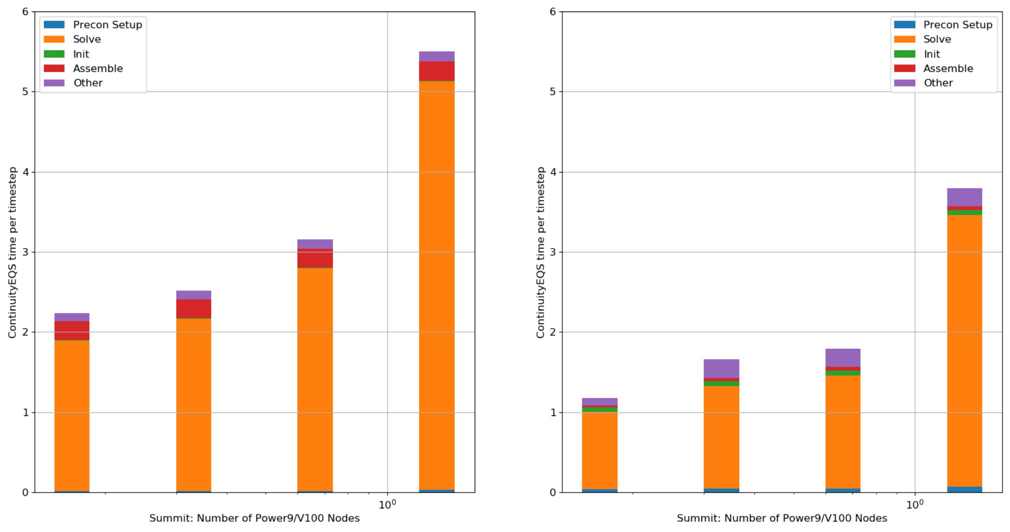


Figure 9: Strong Scaling: Actuator line execution time per time step.



(a) Breakdown of the various linear solver stages during the momentum solve. Left: CPU, right: GPU.



(b) Breakdown of the various linear solver stages during the continuity solve. Left: CPU, right: GPU.

Figure 10: Breakdown of the weak scaling performance of momentum and continuity solves for NREL 5MW turbines operating in ABL inflow using the Trilinos solver stack. The turbine blades are modeled using actuator lines. Comparisons are performed on a per-node basis, i.e., 6 V100 GPUs are considered equivalent to 42 Power9 CPU cores. The timing breakdown are as follows: *init* – time spent in linear system graph creation, *assemble* – time spent by Nalu-Wind computational kernels in assembling the matrix coefficients and right hand side residual vector, *precon setup* – time setting up the preconditioner, *solve* – time spent in iterative solves, *other* – time spent in other operations, e.g., computing gradients, etc.

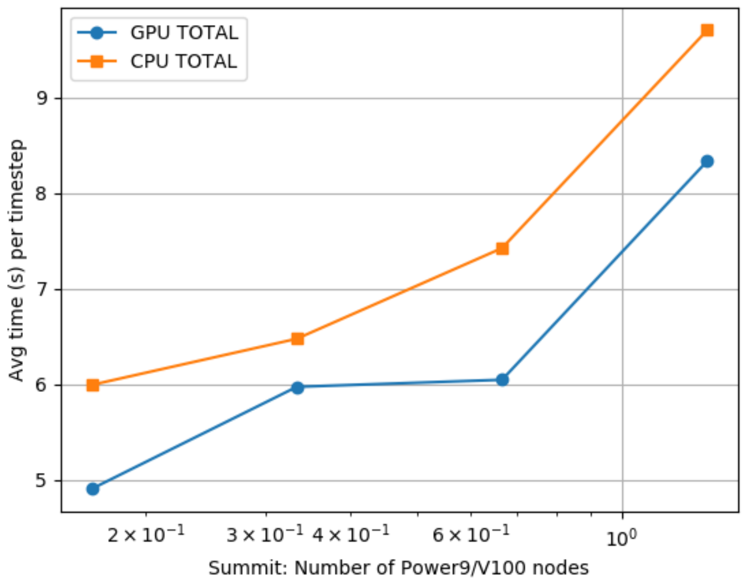


Figure 11: Weak Scaling: NREL5MW actuator line total time per time step.

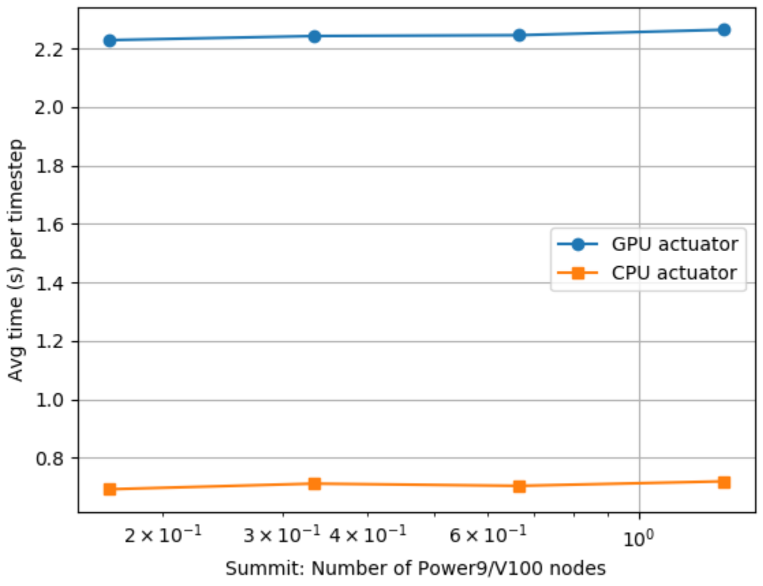


Figure 12: Weak Scaling: Actuator line execution time per time step.

6. HYBRID RANS/LES TURBULENCE MODELING

In this section, we discuss the advances in the AMS (Active Model-Split) [6] turbulence modeling capabilities present in Nalu-Wind, noting that this model was previously referred to as TAMS (Time-averaged Active Model Split). In the FY19-Q4 milestone report, we successfully implemented AMS into the Nalu-Wind codebase and performed a canonical channel flow verification of the implementation. Further validation attempts on wind-relevant problems, such as a NACA-0015 fixed wing and a turbine in atmospheric turbulent inflow, using the AMS model, highlighted drawbacks of the initial implementation. The main challenge that we encountered was a stability time constraint requiring AMS simulations to be run with Courant < 1 everywhere in the domain. It was hypothesized in FY19-Q4 that this stability issue may be related to the explicit treatment in Nalu-Wind of the mean stress in the model-split formulation. We have since updated the implementation with an approximate implicit treatment of the mean diffusion term, as well a set of other updates to the implementation, which have removed the stability constraints present in FY19-Q4 simulations. The AMS model is now capable of running all configurations of Nalu-Wind that are setup for the baseline turbulence models. In the remainder of this section, we detail the updates to the implementation and revisit the validation tests attempted in FY19-Q4, with a specific focus on test cases of relevance to the wind challenge problems: the turbulent periodic hill, the McAlister fixed wing, and the rotating turbine.

6.1 IMPROVEMENTS TO THE NALU-WIND IMPLEMENTATION OF THE SST RANS MODEL

RANS simulations using the SST model presented in previous milestones have consistently shown stability issues, oscillations in residuals, and difficulty producing solutions for some geometries. While these difficulties have not generally impacted the ability to obtain an adequate SST solution, these outstanding issues were negatively affecting AMS simulations by introducing spurious field values in the domain. Namely, the turbulent kinetic energy field for SST simulations was exhibiting large values in several cells right above the wall for certain geometries. This was observed in the periodic hill case, described in Section 6.3, and the McAlister fixed wind simulation, described in Section 6.4.

In typical RANS SST simulations, it is sometimes necessary to clip the turbulent kinetic energy and specific dissipation rate fields if the update to these fields would lead to negative values. This clipping in Nalu-Wind was performed by resetting the turbulent kinetic energy and specific dissipation rate values to those obtained through relations with the molecular viscosity if either becomes negative. This led to unphysical values of turbulent kinetic energy in several grid cells near the wall in certain complex geometries. This clipping was replaced by a simple clipping of turbulent kinetic energy to a minimum value of 10^{-8} if it becomes negative, as done in other leading RANS CFD codes. This led to more robust SST simulations by avoiding spurious turbulent kinetic energy values in the domain. Figures 13 and 14 illustrate the removal of spurious turbulent kinetic energy values near the wall through the use of the updated clipping methodology. We verified, using the simulations of the wall-mounted hump, that this clipping did not affect previous verification and validation efforts.

6.2 IMPROVEMENTS TO THE NALU-WIND IMPLEMENTATION OF THE AMS HYBRID RANS-LES MODEL

For completeness, we revisit the governing equations for the AMS framework. AMS solves a resolved momentum equation with a split modeled stress term,

$$\frac{\partial \rho \overline{u}_i}{\partial t} + \frac{\partial \rho \overline{u}_i \, \overline{u}_j}{\partial x_j} = -\frac{\partial \overline{P}}{\partial x_i} + \mu \frac{\partial^2 \overline{u}_i}{\partial x_j \partial x_j} + \frac{\partial \tau_{ij}^{SGRS}}{\partial x_j} + \frac{\partial \tau_{ij}^{SGET}}{\partial x_j} + F_i. \tag{7}$$

Here, $\overline{u}_i$ is the component of resolved velocity in the i^{th} direction, ρ, a constant density, $\overline{P}$, the resolved pressure, μ, the dynamic molecular viscosity and F_i, an active forcing term designed to generate turbulent fluctuations in the regions where the model determines the grid to be suitable to resolve some turbulent content. The mean subgrid stress is represented by the τ_{ij}^{SGRS} term, while τ_{ij}^{SGET} represents the fluctuating subgrid stress.

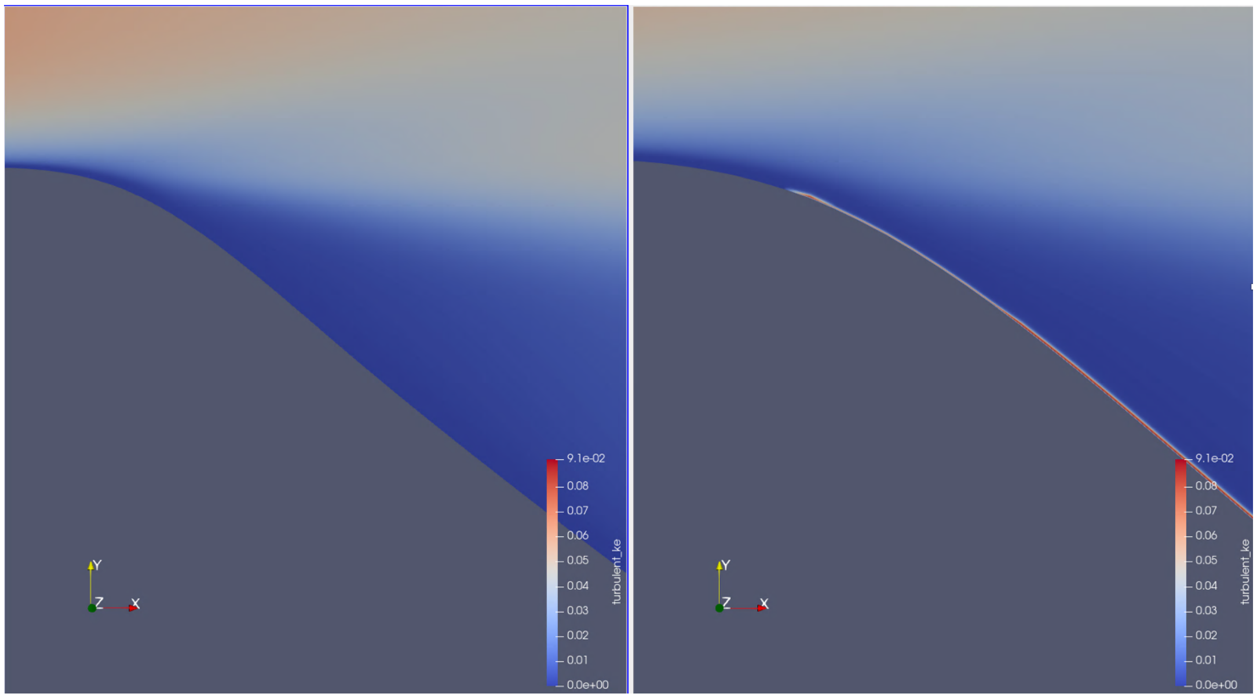


Figure 13: Turbulent kinetic energy for the periodic hill (close-up view on the top of the hill) under different clipping mechanisms: left: clipping to a prescribed minimum value, right: clipping using molecular viscosity relation.

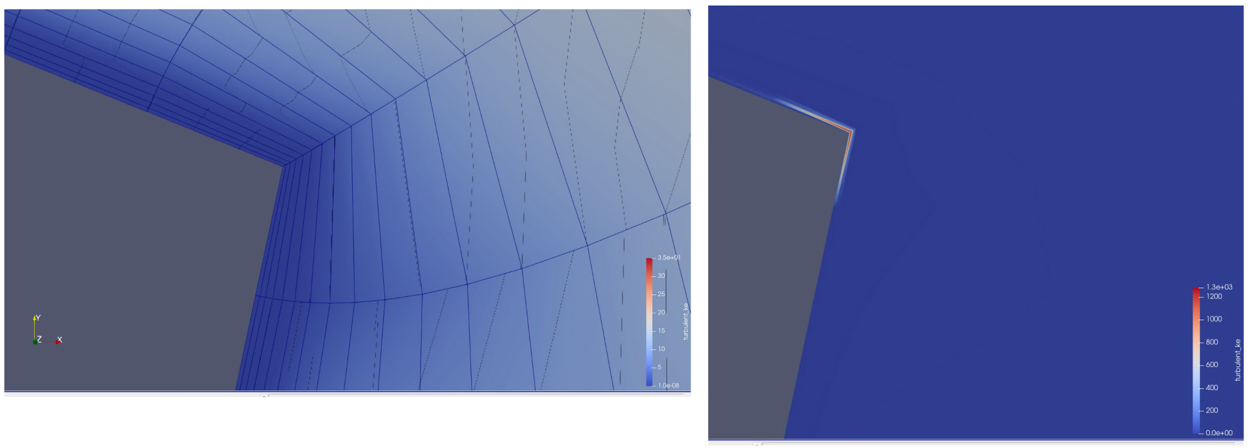


Figure 14: Turbulent kinetic energy for the McAlister fixed wing (close-up view on the trailing edge of the wing) under different clipping mechanisms: left: clipping to a prescribed minimum value, right: clipping using molecular viscosity relation.

To update the Nalu-Wind implementation of the AMS equations to the most recent theory of the underlying framework, described in Haering et al. [6], the following changes were made in the codebase.

Scaling of the τ_{ij}^{SGRS} term The first term in the model split equations, τ_{ij}^{SGRS}, is designed to represent the mean subgrid stress and thus serves as the "RANS-like" part of the model. When the flow field has some level of resolved turbulence, the modeled mean subgrid stress term should scale appropriately. In the previous implementation, $\tau_{ij}^{SGRS} = \alpha\tau_{ij}^{RANS}$, where $\alpha = 1 - k_{\text{res}}/k_{\text{total}}$ was the scaling used on the mean subgrid stress determined from a typical RANS model (the Menter 2003 SST model [11] in Nalu-Wind). In Haering et al. [6], a more accurate scaling is derived based off of eddy viscosity arguments applied to the decomposition of the subgrid stress tensor and empirical data taken from channel flow DNS, leading to the improved form,

$$\begin{aligned} \beta &= 1 - k_{\text{res}}/k_{\text{total}}, \\ \alpha &= k_{\text{sgs}}/k_{\text{total}} \approx \beta^{1.7}, \\ \tau_{ij}^{SGRS} &= \alpha(2-\alpha)\tau_{ij}^{RANS}. \end{aligned} \tag{8}$$

Formulation of the F_i term Haering et al. [6] present a few simplifications to the formulation of the active forcing term, F_i, in the momentum equations. The goal of the active forcing term is to actively introduce turbulent fluctuations in regions of the grid where the AMS model determines the grid is capable of resolving turbulent content, but none is present. In order to do this in a way that slowly introduces resolved structures that evolve into actual turbulence without corrupting the mean, an artificial field based on the structure of a Taylor-Green (TG) vortex is currently used. The length scale and magnitude of the artificial field are designed to match the largest of the locally unresolved fluctuations. The determination of these length scales and magnitudes in Nalu-Wind, as described in FY19-Q4, has been slightly simplified to match the one presented in Haering et al. [6]. Along with the updates to the AMS implementation to match the current theoretical developments, a couple design updates specifically for Nalu-Wind were also deployed.

Coupling of Time-Averaged quantities to Picard Iterations In the AMS framework, the mean subgrid stress term is calculated as it would be in a RANS context, from the expected values. Since the AMS governing equations evolve the resolved instantaneous quantities, a method for approximating the expected values is needed. This is done through a simple causal average equation for a desired flow quantity ϕ:

$$\frac{\partial\langle\phi\rangle}{\partial t} = \frac{1}{T_{RANS}}\left(\phi - \langle\phi\rangle\right), \tag{9}$$

where $\langle\cdot\rangle$ refers to an expected value (approximated through a causal time average) and T_{RANS} is the timescale of the turbulence determined by the underlying RANS quantities, e.g., for the SST model used in Nalu-Wind, $T_{RANS} = 1/(\beta^*\omega)$.

In the prior implementation, in FY19-Q4, the averaging updates were done at the beginning of each timestep, and stayed constant throughout the outer Picard iterations that are taken in Nalu-Wind, as the solution of the instantaneous quantities are driven towards convergence. However, this decoupling between the averaging updates and the Picard iterations caused the second term in the model split equations, τ_{ij}^{SGET}, to activate after the first Picard iteration in regions where we expected only RANS-like behavior and the term did not activate in the first Picard iteration. Since τ_{ij}^{SGET} is designed to represent the fluctuating subgrid stress, it is not designed to be active in regions very close to a wall, where we expect the model to be represented solely by the mean flow.

The reason for this activation was that even in our "converged" SST RANS simulations of complex flows, when we were taking large timesteps (Courant $\gg 1$), we did not have pointwise convergence of the mean quantities and there were small fluctuations of our velocity fields near walls throughout the Picard iterations. In the explicit discretization of the causal averaging equation used in Nalu-Wind,

$$\begin{aligned} t_{wt} &= \min\left(\frac{\Delta t}{T_{RANS}}, 1\right) \\ \langle\phi^{n+1}\rangle &= \langle\phi^n\rangle + t_{wt}\left(\phi^n - \langle\phi^n\rangle\right), \\ \langle\phi^{n+1}\rangle &= t_{wt}\phi^n + (1 - t_{wt})\langle\phi^n\rangle, \end{aligned} \tag{10}$$

when the timestep is larger than the local turbulent timescale, the average quantity is simply updated to the most recent instantaneous quantity. In these near-wall regions, as the local turbulent timescale approached its limiting behavior and dropped below the simulation timestep, these fluctuations, which were simply small changes in the approximate mean solution state, were being captured as fluctuations, as they were being compared to the previous iterations approximate mean state. If the averaging operation had been rerun at the beginning of the Picard iteration, these averages would of been updated to the current approximate mean state and the differencing between the "instantaneous" fields and the mean fields, would not generate these spurious fluctuations.

To resolve this, in the current Nalu-Wind implementation, all approximate expected values calculated in Nalu-Wind are updated at the beginning of each Picard iteration, using the following discretization,

$$\begin{aligned} t^*_{wt} &= \min\left(\frac{\Delta t}{T^*_{RANS}}, 1\right) \\ \langle \phi^* \rangle &= t^*_{wt}\phi^* + (1 - t^*_{wt})\langle \phi^n \rangle, \end{aligned} \tag{11}$$

where a $()*$ quantity represents the intermediate state at the end of the last Picard iteration and $()^n$ represents the state at the end of the previous step.

Implicit Treatment of Mean Stress The mean stress term in the model-split form of the momentum equation, using Menter's 2003 SST as the underlying RANS model, takes the form,

$$\tau^{SGRS}_{ij} = \alpha(2-\alpha)2\mu_{SST}\langle S_{ij}\rangle - \frac{2}{3}\alpha\rho k\delta_{ij}. \tag{12}$$

Since this term is a function of the mean velocity fields and the momentum equation solves for the instantaneous velocity, this term was previously treated explicitly. It was assumed then, that since the mean quantities evolve slowly, the explicit treatment of this term should not have inherent stability issues associated with it. When prior attempts at running problems with large Courant numbers were conducted, however, stability issues arose and the simulations were unable to run.

To treat this term implicitly in Nalu-Wind, the coefficient on the velocity derivatives, $\alpha(2-\alpha)\nu_{SST}$, would be added to the left-hand side (LHS) of the linear system. In the regions of the domain where we expect a purely RANS-like behavior, the instantaneous resolved velocities are equal to the approximate expected values and thus this would be an appropriate term to move to the LHS. In the other limiting case, where the instantaneous field is dominated by the fluctuating velocities, $\alpha \to 0$ and this term would not add any contribution to the LHS, even if it were added. In regions where both the mean and fluctuating velocities play a role, the eddy viscosity contribution from the mean term and from the fluctuating term would both be contributing on the LHS and thus would represent not an exact but approximate representation of the total effective eddy viscosity on the instantaneous velocities.

To address the stability issues in Nalu-Wind when using AMS as the turbulence model for simulations with Courant $\gg 1$, the coefficient in the mean stress term was added to the LHS in the current implementation. This has successfully allowed for AMS to be used as the turbulence model in all simulations that the baseline SST turbulence can be used in. Initial validation studies, discussed in the following sections, suggest that the approximate representation of the total effective eddy viscosity in these hybrid regions does not seem to have a meaningful impact on the accuracy of the solution. Further validation testing will be necessary to definitively determine if this approach to treating the mean stress term implicitly for stability considerations is appropriate.

6.3 PERIODIC HILL MODEL VALIDATION

For initial model validation, we conducted the periodic hill turbulence test case based on the ERCOFTAC UFR 330 test case [4]. This case was used as a model validation simulation in the FY19-Q4 report, but there, it was run in a secondary code, CDP, using the underlying $\overline{v}^2 - f$ RANS model. The Reynolds number (based on the hill height) is 10600 and the hill height is 1 m. The fluid density is set to $1\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}$. The fluid bulk velocity, $u_b = 1\,{}^{\mathrm{m}}/_{\mathrm{s}}$, which leads to a viscosity of ${}^{1}/_{10600}$. The boundary conditions are periodic in the streamwise and the spanwise directions, with the bottom and top boundary conditions as walls.

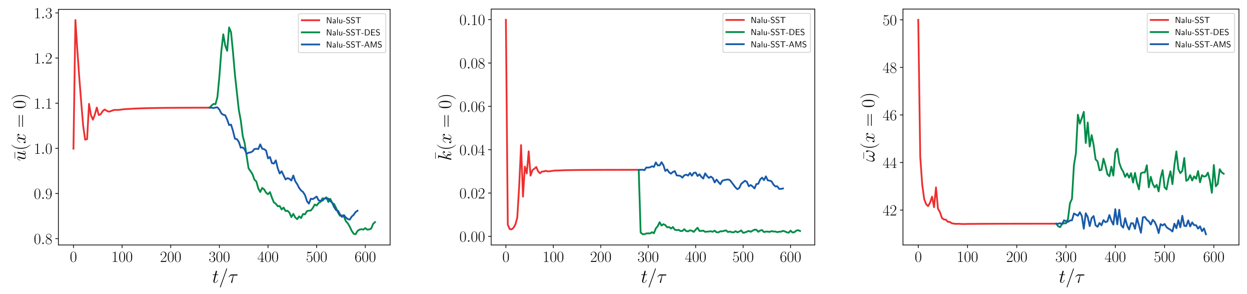


Figure 15: Bulk inlet quantities as a function of time.

In order to drive the flow to a statistically stationary state, a fixed forcing was applied at $y/h > 1$ to approximate a unity bulk velocity at $x/h = 0$. While this was sufficient to approximate the bulk velocity for the RANS case, the hybrid cases (AMS and SST-DES) experienced a reduction in bulk velocity, as can be seen in Fig. 15. A more appropriate forcing for the hybrid cases, would use a dynamic calculation of the body force, to ensure a unity bulk velocity is achieved as done by other researchers [12, 5]. Nevertheless, some useful insights can be still be drawn from the comparisons.

Steady state SST solutions were obtained by running this case to $t = 300\tau$, where $\tau = h/u_b$, using a time step $\Delta t = 4 \times 10^{-2}$. Bulk inlet quantities of velocity, turbulent kinetic energy, and specific dissipation rate were monitored to ensure a steady-state solution, see Fig. 15. The SST solution was then used as the initial condition for both the SST-DES and AMS simulations, which used a reduced time step $\Delta t = 4 \times 10^{-3}$ and was evolved for another 300τ.

Once statistical convergence is reached, time-averaged velocity profiles are computed using the velocity fields from 20 flow-through times, spaced every 1.2 flow-throughs, to achieve statistical convergence in the profiles. In Fig. 16, the mean velocity profiles are compared for the SST, SST-DES, and AMS simulations. It can be observed that the SST simulation significantly over-predicts the velocity in the upper half of the domain and predicts a separated flow throughout the domain, predicting no reattachment, counter to what is observed in the experimental data. This is improved in both hybrid RANS-LES simulations, SST-DES and AMS, which predict a reattachment location around $x = 5h$, a little downstream of the experimental value of $x = 4h$. Overall, the AMS simulation velocities are largely comparable to those from the SST-DES simulations, where both under-predict the experimental velocities due to the improper forcing causing a loss in bulk velocity. We note that similarity to SST-DES simulations is a promising sign here, as researchers have shown good agreement on the periodic hill test case when using SST-DES [12, 5]. This suggests that the correction to the applied forcing will lead to strong validation results for the use of AMS in Nalu-Wind, based off of the SST RANS model.

6.4 AMS SIMULATIONS OF A NACA-0015 FIXED WING

With the updated AMS implementation, we revisit simulations of a NACA-0015 fixed wing. SST and SST-DES simulations were conducted for the FY19-Q2 milestone and attempts at AMS simulations were conducted in FY19-Q4. The prior stability constraint with AMS required us to simulate using Courant < 1, leading to timesteps that were ~ 4 orders of magnitude smaller than the one taken in the pure SST case. That restriction has been completely removed and for the initial tests conducted here, the maximum Courant ~ 5000 in the most restrictive cells in the boundary layer on the wing.

Simulation Description We briefly review the simulation setup, which is based on the McAlister-Takahashi NACA-0015 experiment [10]. We use a full-span, 1 m chord length fixed NACA-0015 wing (untwisted, untapered, unswept) placed at a 12° angle of attack in an unbounded flow (neglecting the tunnel walls in the experiment). The wing has aspect ratio of 3.3 chords and a square wing tip. The Reynolds number is $Re = 1.5 \times 10^6$ (inflow velocity is $u_\infty = 46$ m/s, density is $\rho_\infty = 1.225$ kg/m^3, and dynamic viscosity is $\mu = 3.756 \times 10^{-5}$ kg/(m s)). The domain size is 10 chords upstream, 20 chords downstream, and 30 chords in the span. Symmetry boundary conditions are used in the spanwise direction, inflow at the bottom and inlet,

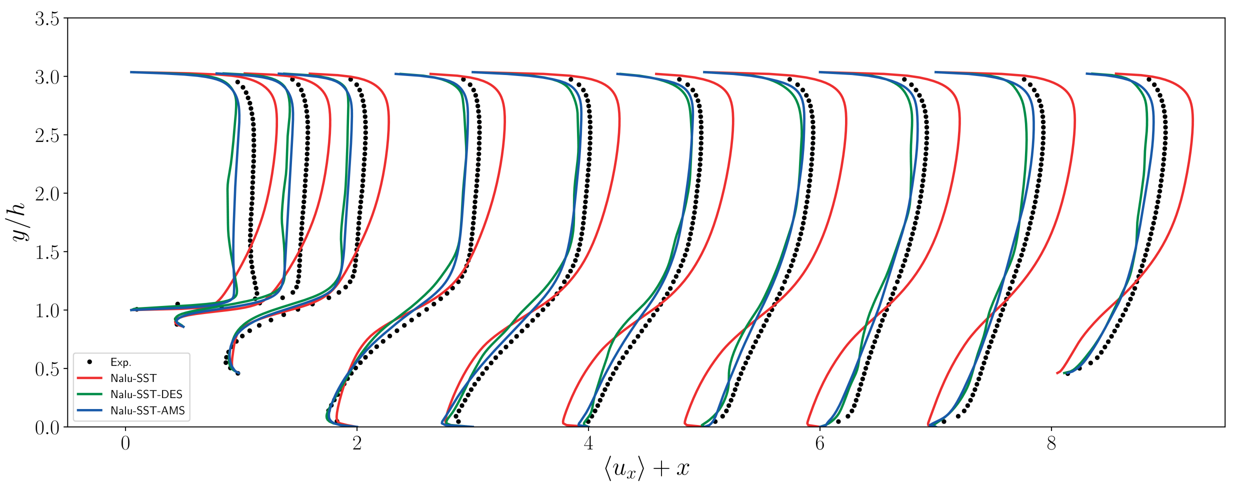


Figure 16: Average u_x profiles for the periodic hill.

outflow at the top and outlet. AMS Simulations are initialized from a converged (≈ 40 chord flow-through times) SST solution on the same domain. The mesh is a hexahedral-dominant mesh with ≈ 9 million nodes, wall-normal spacing along the wing is $10^{-5}c$, ensuring $y+ < 1$, $5 \times 10^{-4}c$ at the leading edge and $3 \times 10^{-3}c$ at the trailing edge. Wedges and pyramids far from the wing region, and an overset O-H grid (or "Butterfly") positioned behind the wing tip are used to capture the wing-tip vortex.

AMS Computational Cost To assess the computational cost incurred by the updated AMS turbulence model implementation, we run the NACA-0015 fixed wing for 100 steps using both the AMS and SST configurations on the same initial condition. The simulations were performed using an Intel compiler build of Nalu-Wind on 576 cores on NREL's Eagle supercomputer, which is built with Intel Skylake processors. Overall, the AMS simulation was approximately 1% slower than the SST simulation, suggesting that no meaningful additional computational cost is incurred through use of the AMS hybrid model.

Results The goal of any hybrid RANS/LES model is to use RANS modeling in regions where resolving turbulence would come with great computational cost, such as in attached boundary layers, and to allow for the simulation to resolve some turbulent fluctuations in regions of dynamic flow conditions, such as wakes, where the mean flow solution of the RANS equations would be insufficient. To qualitatively assess if the AMS model is generating fluctuations in the desired regions, we plot, in Fig. 17, instantaneous snapshots of vorticity magnitude contours at two points in time. The slices are taken in the plane through $z = 3.3$, which is the edge of the wing at the tip vortex and show the velocity magnitude. It is clear that there has been a transition to some resolved turbulent structures in the wake regions of the domain. A breakdown in the tip vortices can be observed in the tip vortex wake as well as some turbulent fluctuations present in the trailing edge of the wing. To emphasize the breakdown in the tip vortices, we plot three x-slices in Fig. 18, showing the evolution of the tip vortices downstream of the fixed wing for the SST simulation (left) and two instantaneous snapshots of the AMS simulation (center and right). This is an encouraging sign, as the AMS model is appropriately transitioning from the RANS solution into one with reasonable resolved turbulent content.

In the vorticity contours in Fig. 17, especially in the right frame, you can observe spurious regions of increased vorticity at the center line where the mesh was reflected to generate the full wing and in the span just off of the wing tips, where the mesh was extruded off the end of the wing. These suggest a mesh interaction issue present in the AMS simulations. The cause of this needs further investigation, but one possible explanation is that while this mesh may be suitable for RANS simulations, more care will need to be taken in constructing meshes that are used for AMS simulations. A second type of mesh effect can be observed by looking closely at the end of the tip vortex wake in the slice of the left frame of Fig. 17. Some spurious noise is observed at the transition from the overset refined tip vortex mesh into the background mesh.

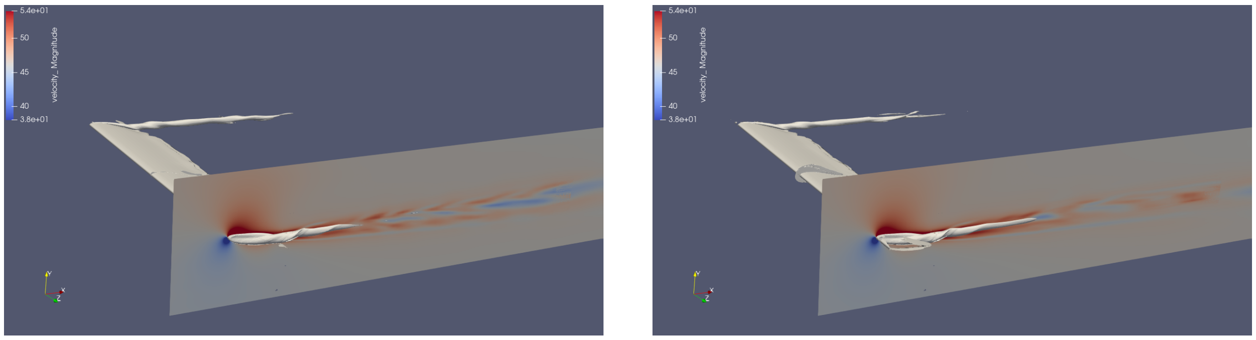


Figure 17: NACA-0015 fixed wing contours of velocity magnitude and a slice of vorticity on the plane through the tip of the wing in the AMS simulation. Reasonable turbulent fluctuations can be observed.

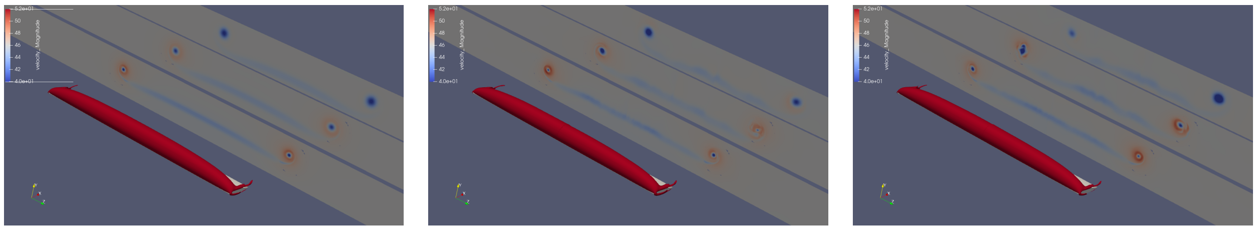


Figure 18: NACA-0015 fixed wing evolution of tip vortices downstream of the fixed wing. Right: SST simulation, Center and Left: Instantaneous snapshots from the AMS simulation. Influences of resolved turbulence can be seen in the AMS snapshots, with the structure breaking down and evolving over time.

Again, this will require further investigation to identify the root cause, but one possibility is the interaction of turbulence with sudden grid changes, which can cause spurious reflections at the mesh interface as discussed in Yalla et al. [16].

To assess the AMS simulation from a quantitative standpoint, in Fig. 19, we compare the pressure coefficient,

$$c_p = \frac{2p}{\rho_\infty u_\infty^2} \tag{13}$$

where p is the pressure at several locations along the wing span, to the SST simulation, experimental data [10] and previously published results of a fixed wing in unconstrained flow using the Spalart-Allmaras RANS model and adaptive mesh refinement to resolve the wake [13].

As can be seen in the set of pressure coefficient plots, no observable change can be inferred between the SST and the AMS simulations. This is to be expected as the model relies on the RANS SST solution in the boundary layers on the wing. The fact that the AMS model does not corrupt the RANS solution on the wing while generating turbulent fluctuations in the wake, suggests that the model is performing correctly, even while taking large timesteps. Both the SST and AMS simulations under-predict the pressure coefficient, which has been previously hypothesized to result from improper modeling of blockage effects due to the tunnel walls (as the simulations conducted here use a symmetry boundary and a double inflow/outflow configuration). Future simulations will be conducted to investigate this specific issue.

6.5 AMS SIMULATIONS OF A ROTATING TURBINE IN UNIFORM INFLOW

As a demonstration for the AMS turbulence model on wind turbines, we revisit simulations of an NREL 5-MW turbine at a uniform inflow of $U = 8$ m/s. The additional requirement for the rotating turbine, is that the AMS model properly handle the mesh motion, which the other simulations did not address. The updated AMS implementation was able to simulate the rotating turbine using the same fixed timestep as

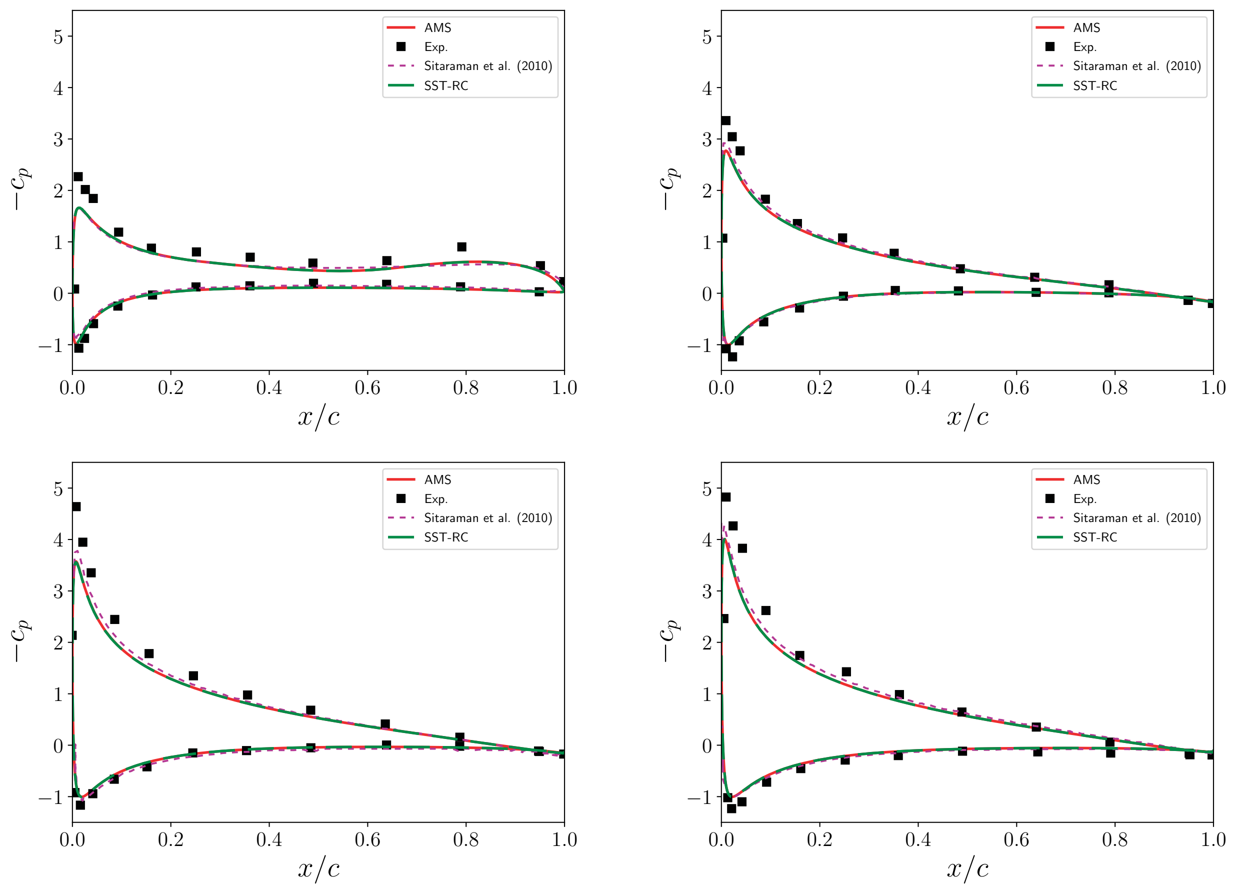


Figure 19: Chordwise variation of the pressure coefficient, c_p, at different locations along the wing span.

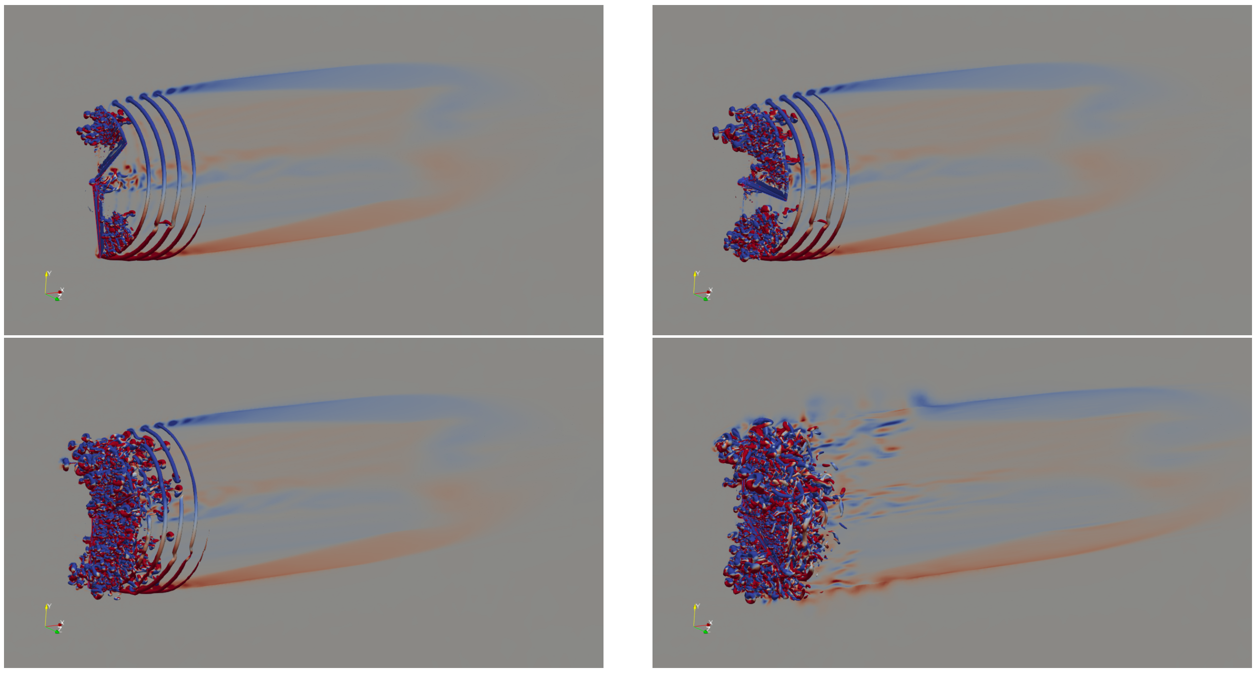


Figure 20: Isocontours of Q-criterion and slices of z-vorticity in snapshots after the AMS turbulence model is turned on. Top Left: After a 45° rotation, Top Right: After a 90° rotation, Bottom Left: After a 180° rotation and Bottom Right: After a 1080° rotation,.

discussed in Sec. 4, such that the blade rotates 0.25° for each timestep. The mesh setup is the same as for the turbine used in Sec. 4, but for this turbulence model demonstration, the simulation was run on 1080 cores on NREL's Eagle supercomputer and thus did not utilize any GPU pathways.

In Fig. 20, we plot isocontours of Q-criterion, with the slice showing the z-component of vorticity during the evolution AMS takes from the initial RANS state. Starting in the top left and moving across the top row and then across the bottom row, the snapshots are taken after $45°, 90°, 180°$ and $1080°$ rotations of the blade, measured from the point the AMS turbulence model is turned on. Clearly, an elaborate set of turbulent fluctuations is generated in the wake behind the turbine, which moves downstream and disturbs the wake structure predicted by the RANS solution. Examining the initial frame in the top left, we can see turbulent fluctuations moving both upstream and downstream from the blade location. It is not clear that this is a physical behavior and could be related to the sudden change in mesh resolution that occurs when the blade-resolved turbine mesh moves along with the blade and the mesh resolution returns to the background in-between blades mesh. This could potentially also be addressed by the work referenced in Sec. 6.4 by Yalla et al. [16]. However, further investigation into the dynamics driving this behavior is needed before that can be assessed.

Overall, the AMS turbulence model is performing up to expectations, capable of handling all turbulent simulations in Nalu-Wind using the desired configurations. A more extensive validation program is underway to resolve the outstanding issues identified in the test cases performed for the FY20-Q4 milestone. In addition, separate work is being carried out to add DDES (Delayed Detached Eddy Simulation) and IDDES (Improved Delayed Detached Eddy Simulation) hybrid model capabilities to Nalu-Wind, which can be then be used in useful A/B comparisons on these and other validation cases, to more accurately assess the performance and potential improvement of AMS over typical hybrid approaches. At this point, the SST model is working sufficiently well, as the underlying RANS model in Nalu-Wind and while other models, such as $\overline{v}^2 - f$, have been used with AMS and shown some superior results, the extensive use of SST in wind-relevant problems and the additional complications and costs associated with a more complex RANS model, such as $\overline{v}^2 - f$, do not warrant its use in Nalu-Wind at this time.

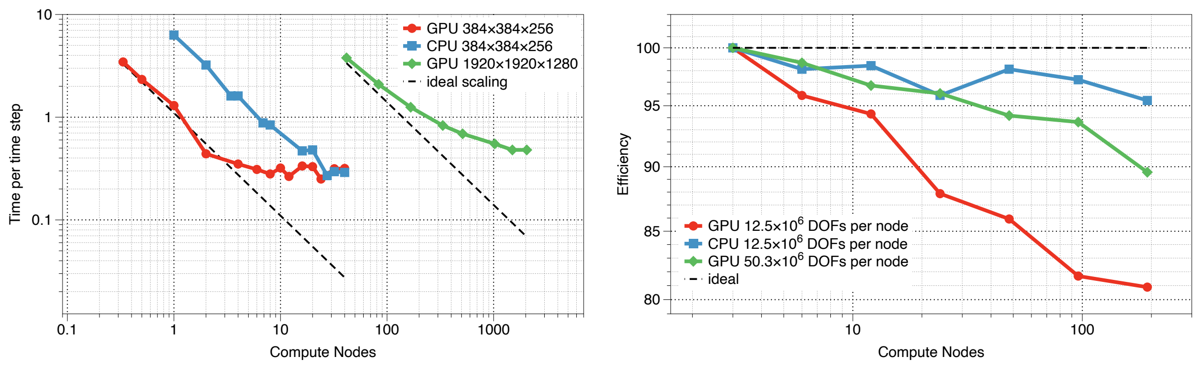


Figure 21: Strong- and weak-scaling performance for AMR-Wind on the ABL problem on ORNL Summit. Left: Strong-scaling performance on two problem sizes (3.7×10^7 and 4.7×10^9 DOFs) performed on GPUs and CPUs. Right: Weak-scaling efficiency for two different workloads 2×10^6 and 8.3×10^6 DOFs/GPU.

7. ADVANCES IN STRUCTURED BACKGROUND SOLVER

As mentioned in §3.4, the ExaWind project adopted an AMReX-based block-structured, incompressible CFD solver for modeling the background atmospheric flowi: AMR-Wind. The choice of a structured background solver was motivated by the observation that a significant portion of the computational domain in wind farm simulations are usually modeled with structured hexahedral cells and do not require the flexibility offered by an unstructured mesh. The adoption of an overset mesh methodology as well as the recent demonstration of the viability *decoupled* overset simulation in FY20-Q3 milestone paves the way for a hybrid-solver strategy wherein the structured background solver is coupled with Nalu-Wind, which solves the flow domain in the vicinity of turbine aerodynamic structures.

However, adoption of a hybrid solver strategy is not without its downsides, as it adds an additional layer of complexity in the development and maintenance of the codebase. Thus, the new pathway must offer significant performance benefits to justify the added complexity within the simulation environment. To this end, in FY20, the objectives were implementing a basic atmospheric boundary layer solver within AMR-Wind and benchmarking the performance of the solver on the ABL precursor problem on both CPUs and GPUs. The results of the strong and weak scaling studies are discussed in §7.1. This is followed by a brief update on the current status of the hybrid Nalu-/AMR-Wind simulations using overset mesh methodology on simple test problems.

7.1 STRONG & WEAK SCALING PERFORMANCE OF AMR-WIND

The ability to model atmospheric boundary layers was implemented in AMR-Wind and uses the same equation sets as in Nalu-Wind (see §5.1). The codebase was used to perform both strong and weak scaling studies of the ABL precursor simulations similar to the studies performed for Nalu-Wind in FY20-Q2 milestone. One notable difference is that the results presented here use the Smagorinsky turbulence model and not the 1-equation $k - sgs$ turbulence model used in Nalu-Wind. The 1-equation $k - sgs$ model has been recently added to the AMR-Wind codebase and has since been verified for canonical problems by comparing with results published in literature as well as Nalu-Wind results.

Figure 21 shows the strong- and weak-scaling performance of AMR-Wind, running on the ORNL Summit system for the ABL problem. AMR-Wind shows very good strong- and weak-scaling trends. The CPU vs. GPU results are compared on a Summit-compute-node basis, i.e., 7 CPU cores are considered equivalent to 1 GPU. Compared to Nalu-Wind results presented in FY20-Q2, Fig. 21 (a) shows that AMR-Wind simulations are $\approx 5\times$ faster than on CPUs. Figure 21 (b) shows the weak-scaling efficiency of the AMR-Wind code on two problem sizes: the first ranging from $\sim 3.7 \times 10^7$ cells to $\sim 2.1 \times 10^9$ cells and the second ranging from $\sim 1.51 \times 10^8$ cells to $\sim 9.6 \times 10^9$ cells. For the weak scaling study both cases start at three Summit nodes (18 GPUs and 126 CPUs) and as the mesh size is doubled the nodes are doubled until 192 nodes (1152

GPUs and 8064 CPUs) are reached. The ABL precursor simulations have also been executed successfully on pre-exascale hardware Iris and Tulip using the Intel OneAPI DPC++ and AMD HIP compilers, respectively.

8. CONCLUSIONS

This milestone successfully demonstrated the transition of the core computational kernels in Nalu-Wind to execute on GPUs. The new GPU-based codebase was tested on wind-relevant problems on the ORNL Summit system. Scaling studies indicate that the solution of elliptic pressure Poisson systems using algebraic multigrid remains the primary bottleneck that dominates the time per timestep. This milestone also documents the significant advances made in the development of the hybrid RANS/LES turbulence model that have improved the robustness and stability of the model implementation within Nalu-Wind.

The major observations in this milestone are summarized below:

1. Over the last two years, the Nalu-Wind codebase underwent a significant overhaul to transition on computationally intensive kernels to execute on GPUs. The new kernels leverage the abstractions provided by the Kokkos library for execution on devices as well as managing memory on both host and device. Successful transition to GPUs was greatly facilitated by parallel development efforts in the Trilinos libraries, in particular the STK NGP library.

2. Nalu-Wind developers adopted a test-driven development philosophy to ensure that a seamless transition of the core algorithms without significant disruption to the users of Nalu-Wind. However, despite careful development and testing, performance benchmarking runs in FY20-Q2 and FY20-Q4 uncovered several implementation bugs that required considerable efforts to fix. All bugs were related to inconsistent updates, access, and synchronization of field data structures between host and device memory. Efforts are underway in STK library to assist application developers in detecting such inconsistent memory access patterns in future.

3. For the blade-resolved simulations with overset meshes, the *hypre* solver stack significantly outperforms the Trilinos Belos/MueLu stack on the CPUs, and maintains a slight advantage over the Trilinos stack on GPUs.

4. Solution of the elliptic pressure Poisson system using algebraic multigrid remains the primary bottleneck on GPUs. Both preconditioner setup costs and solver costs scale poorly on GPUs and will be the focus of future efforts. The poor scaling trends are observed for both *hypre* and the Trilinos solver stacks. The costs remain high despite the use of decoupled overset pressure solution approach, described in detail in FY20-Q3 report, which removes the constraint rows that have been a challenge for algebraic multigrid preconditioners.

5. For moving mesh problems, a significant increase in the time spent in pre-processing steps within each timestep is observed. The time spent is dominated by two major tasks: updating the overset connectivity and reinitialization of the linear systems. With decoupled overset, the need for linear system reinitialization can be eliminated. A feature has been added to Nalu-Wind codebase to allow reuse of linear systems when using decoupled overset mesh simulation strategy for moving mesh applications. Marked reductions in pre-processing time when reusing linear system is observed (see Fig. 5). Overset mesh connectivity updates are still being performed on the host and the transfer of field data to and from device during overset connectivity updates and solution exchange still contribute to a significant overhead when executing on GPUs. Eliminating this overhead by transitioning the overset search algorithms to execute on GPUs is currently underway and will be a major focus of upcoming milestone activities.

6. Significant progress was made in improving the robustness and stability of the AMS hybrid RANS/LES turbulence model implementation within Nalu-Wind. The model is currently being tested on the NACA0015 fixed-wing problem for comparison with available experimental data as well as for blade-resolved simulations of the NREL 5-MW rotor in uniform inflow.

7. The ExaWind project has adopted an AMReX-based, block-structured, incompressible solver, AMR-Wind, as a structured background solver for simulating the atmospheric boundary layer in wind farm simulations. Strong and weak scaling studies were performed on problem sizes ranging from 9.1×10^6 degrees of freedom. Overall, AMR-Wind shows good strong and weak scaling characteristics on both CPUs and GPUs. In production runs, AMR-Wind is also $\approx 5\times$ faster than Nalu-Wind for ABL simulations. The team has also compiled and executed AMR-Wind on both pre-exascale systems, Iris and Tulip, using the Intel OneAPI/DPC++ and AMD HIP compilers respectively. ExaWind developers are working with ANL performance engineers in performance profiling of AMR-Wind to identify and address performance bottlenecks.

REFERENCES

[1] M. L. Blaylock, B. C. Houchens, D. C. Maniaci, P. Sakievich, and R. C. Knaus, *Comparison of field measurements and large eddy simulations of the scaled wind farm technology (swift) site*, in ASME-JSME-KSME 2019 Joint Fluids Engineering Conference, Proceedings of the ASME-JSME-KSME 2019 Joint Fluids Engineering Conference, ASME, 2019.

[2] M. Churchfield, S. Lee, P. Moriarty, L. Martinez, S. Leonardi, G. Vijayakumar, and J. Brasseur, *A large-eddy simulation of wind-plant aerodynamics*, in 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2012, p. 537.

[3] H. Edwards, A. Williams, G. Sjaardema, D. Baur, and W. Cochran, *Sierra toolkit computational mesh conceptual model*, Tech. Rep. SAND2010-1192, Sandia National Laboratories, 2010.

[4] ERCOFTAC, *ERCOFTAC UFR 330 Test Case*. http://qnet-ercoftac.cfms.org.uk/w/index.php/UFR_3-30_Test_Case.

[5] J. Fröhlich and D. von Terzi, *Hybrid les/rans methods for the simulation of turbulent flows*, Progress in Aerospace Sciences, 44 (2008), pp. 349 – 377.

[6] S. Haering, T. Oliver, and R. Moser, *Active model split hybrid RANS/LES*, Journal of Fluid Mechanics, (2020). Submitted.

[7] M. Heroux and et. al., *An overview of the trilinos project*, ACM Trans. Math. Softw., (2005).

[8] A. Hsieh, D. C. Maniaci, T. G. Herges, G. Geraci, D. T. Seidl, M. S. Eldred, M. L. Blaylock, and B. C. Houchens, *Multilevel uncertainty quantification using cfd and openfast simulations of the swift facility*, in AIAA Scitech 2020 Forum, 2020.

[9] J. Jonkman, S. Butterfield, W. Musial, and G. Scott, *Definition of a 5-MW reference wind turbine for offshore system development*, Tech. Rep. NREL/TP-500-38060, National Renewable Energy Laboratory, 2009.

[10] K. W. McAlister and R. K. Takahashi, *NACA 0015 wing pressure and trailing vortex measurements*, Tech. Rep. NASA-A-91056, National Aeronautics and Space Administration, AMES Research Center, Moffett Field, CA, 1991.

[11] F. R. Menter, M. Kuntz, and R. Langtry, *Ten years of industrial experience with the SST turbulence model*, in Turbulence, Heat and Mass Transfer 4, K. Hanjalic, Y. Nagano, and M. Tummers, eds., Begell House, Inc., 2003, pp. 625–632.

[12] Sari´c, S., Jakirli´c, S., Breuer, M., Jaffrézic, B., Deng, G., Chikhaoui, O., Fröhlich, J., von Terzi, D., Manhart, M., and Peller, N., *Evaluation of detached eddy simulations for predicting the flow over periodic hills*, ESAIM: Proc., 16 (2007), pp. 133–145.

[13] J. Sitaraman, M. Floros, A. Wissink, and M. Potsdam, *Parallel domain connectivity algorithm for unsteady flow computations using overlapping and adaptive grids*, Journal of Computational Physics, 229 (2010), pp. 4703–4723.

[14] J. N. Sorensen and W. Z. Shen, *Numerical modeling of wind turbine wakes*, Journal of Fluids Engineering, 124 (2002), pp. 393–399.

[15] M. Sprague, S. Boldyrev, P. Fischer, R. Grout, W. Gustafson Jr., and R. Moser, *Turbulent flow simulation at the Exascale: Opportunities and challenges workshop*, tech. rep., U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, 2017. Published as Tech. Rep. NREL/TP-2C00-67648 by the National Renewable Energy Laboratory.

[16] G. Yalla, T. Oliver, S. Haering, B. Engquist, and R. Moser, *On the effects of resolution inhomogeneity in LES*, Physical Review Fluids, (2020). Submitted.