# ABSTRACT


Title of dissertation: ADVANCING THE MULTI-SOLVER PARADIGM FOR OVERSET CFD TOWARD HETEROGENEOUS ARCHITECTURES

Dylan Jude
Doctor of Philosophy, 2019

Dissertation directed by: Professor James Baeder
Department of Aerospace Engineering



A multi-solver, overset, computational fluid dynamics framework is developed for efficient, large-scale simulation of rotorcraft problems. Two primary features distinguish the developed framework from the current state of the art. First, the framework is designed for heterogeneous compute architectures, making use of both traditional codes run on the Central Processing Unit (CPU) as well as codes run on the Graphics Processing Unit (GPU). Second, a framework-level implementation of the Generalized Minimal Residual linear solver is used to consider all meshes from all solvers in a single linear system. The developed GPU flow solver and framework are validated against conventional implementations, achieving a 5.35× speedup for a single GPU compared to 24 CPU cores. Similarly, the overset linear solver is compared to traditional techniques, demonstrating the same convergence order can be achieved using as few as half the number of iterations.

Applications of the developed methods are organized into two chapters. First, the heterogeneous, overset framework is applied to a notional helicopter configuration based on the ROBIN wind tunnel experiments. A tail rotor and hub are added to create a challenging case representative of a realistic, full-rotorcraft simulation.

Interactional aerodynamics between the different components are reviewed in detail. The second application chapter focuses on performance of the overset linear solver for unsteady applications. The GPU solver is used along with an unstructured code to simulate laminar flow over a sphere as well as laminar coaxial rotors designed for a Mars helicopter. In all results, the overset linear solver out-performs the traditional, de-coupled approach. Conclusions drawn from both the full-rotorcraft and overset linear solver simulations can have a significant impact on improving modeling of complex rotorcraft aerodynamics.

# ADVANCING THE MULTI-SOLVER PARADIGM FOR OVERSET CFD TOWARD HETEROGENEOUS ARCHITECTURES

by

Dylan Philip Nordblom Jude

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor James Baeder, Chair/Advisor
Professor Inderjit Chopra
Professor Anubhav Datta
Professor Stuart Laurence
Professor Kayo Ide

# Dedication

To Rachael

# Acknowledgments

My PhD research from the past five years has not been a sole endeavor and could not have been accomplished without the help and influence of many individuals. First, I would like to thank my PhD advisor, Dr. James Baeder, whose dedication and propensity for novel computational fluid dynamics have guided my research. I am also grateful to my other committee members, Dr. Inderjit Chopra, Dr. Anubhav Datta, Dr. Stuart Laurence, and Dr. Kayo Ide for their expertise and recommendations for contextualizing the applications of my research.

A significant portion of my dissertation research is a direct result of an internship conducted at the Army Aviation Development Directorate at Ames Research Center. Jay Sitaraman, Vinod Lakshminarayan, Andy Wissink, and Roger Strawn have all been tremendous mentors and have motivated expanding the scope of my research to new areas. I have greatly enjoyed working with them and look forward to continuing to work with them in the future.

Many major breakthroughs during my research trace back to help received from two friends and post doctoral researchers in our department: Ananth Sridharan and Bharath Govindarajan. Ananth's recommendations, often through dogged exchanges, always serve as a grounding for realistic, rotorcraft-related applications. Bharath's assistance and support in almost all facets of my research and time at the University of Maryland cannot be understated. He has always been a first-stop for most challenges encountered and has continually steered my research toward higher standards. I will greatly miss our morning coffee trips and necessary football and frisbee breaks.

Both work and extracurricular activities have been greatly enriched by many friends and colleagues over the last five years. YongSu and BumSeok from Dr. Baeder's research group have greatly helped in research collaborations and preparation for deadlines. Dan and Tyler always provide a balance between bringing in home-baked

desserts and organizing intramural sports. The friendship of Dan, Tyler and Bharath have ensured many much-needed breaks playing soccer and football and weekends at Deep Creek.

I am fortunate to have completed over two decades of schooling and university education, all with the continuous support of my family. For the biggest and the smallest moments in life, my mom has always been there to give love and advice that I can count on and trust. For as long as I can remember she has been the one to set me on a path towards my greatest potential. I am forever grateful for her kindness and selflessness. To Laura, my sister, I have immense admiration for her dedication and strength in pursuing her goals. I am so thankful to have had her living nearby for a large part of my time at Maryland. My dad's fascination with science instilled a view of education and learning as a life-long activity. He was a primary driver for my decision to pursue a PhD. His curiosity about the world around him and the conversations that we would have about my interests and research are something that I miss deeply since his passing. I only wish I could share this dissertation with him.

Lastly, I owe a tremendous amount of my success during my PhD to the endless love and support of my wife, Rachael. Despite my schedule of long days and stressful deadlines, she was always there to remind me of the big picture of life and plan getaways to Shenandoah or the Poconos. Her enduring optimism and encouragement has been a primary factor in many of my accomplishments over our last ten years together. Her confidence and zest for life inspires me to be the best that I can be, both academically and personally.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| BC | Boundary Condition |
| CFD | Computational Fluid Dynamics |
| CFL | Courant–Friedrichs–Lewy (number) |
| CPU | Central Processing Unit |
| CSD | Computational Structural Dynamics |
| CUDA | Compute Unified Device Architecture |
| DADI | Diagonalized Alternating Direction Implicit (preconditioner) |
| FLOPS | Floating-point Operations Per Second |
| Garfield | GPU-Accelerated Rotor Flow Field (solver) |
| GMRES | Generalized Minimal Residual |
| GPU | Graphics Processing Unit |
| GS | Gauss–Seidel (preconditioner) |
| HamStr | Hamlitonial Strand (solver) |
| OverTURNS | Overset Transonic Unsteady Rotor Navier–Stokes (solver) |
| O-GMRES | Overset Generalized Minimal Residual |
| LHS | Left-hand-side |
| MARCC | Maryland Advanced Research Computing Center |
| MPI | Message Passing Interface |
| MUSCL | Monotone Upstream-centered Scheme for Conservation Laws |
| PDE | Partial Differential Equation |
| RANS | Reynolds-Averaged Navier–Stokes |
| RHS | Right-hand-side |
| ROBIN | ROtor Body INteraction |
| SMR | Single Main Rotor |
| SA | Spalart–Allmaras |
| Tioga | Topology Independent Overset Grid Assembler |
| WENO | Weighted Essentiall Non-Oscillatory |

# Nomenclature

| | |
|---|---|
| $A$ | Rotor Area |
| $a$ | Speed of sound |
| $c$ | Blade or wing chord |
| $C_T$ | Rotor thrust coefficient $= T/(\rho A V_{tip}^2)$ |
| $C_l$ | Sectional lift coefficient |
| $C_T/\sigma$ | Blade loading |
| $C_P$ | Rotor power coefficient $= P/(\rho A V_{tip}^3)$ |
| $C_p$ | Pressure coefficient |
| $C_{p,air}$ | Specific heat at constant pressure |
| $e$ | Energy (internal + kinetic) |
| $k_{air}$ | Thermal conductivity for air |
| $M_{tip}$ | Rotor tip Mach number $= V_{tip}/a_\infty$ |
| $N_b$ | Number of rotor blades |
| $P$ | Rotor power |
| $Pr$ | Prandtl number $= \mu C_{p,air}/k_{air}$ |
| $p$ | Static pressure |
| $Q_c$ | Q-criterion |
| $\mathbf{Q}$ | Vector of conserved variables |
| $\mathbf{F}_i$, $\mathbf{G}_i$, $\mathbf{H}_i$ | Inviscid Flux Vectors in $x$, $y$, $z$-directions |
| $\mathbf{F}_v$, $\mathbf{G}_v$, $\mathbf{H}_v$ | Viscous Flux Vectors in $x$, $y$, $z$-directions |
| $Re$ | Reynolds Number |
| $R$ | Rotor Radius |
| $S$ | Fluid Strain Tensor |
| $T$ | Rotor Thrust |
| $\Delta t$ | Physical timestep size |
| $\Delta\tau$ | Dual- or Pseudo-timestep size |
| $\tau$ | Viscous stress tensor |
| $u, v, w$ | $x$, $y$, $z$-components of velocity |
| $V$ | Total Velocity |
| $V_{tip}$ | Rotor Tip Speed |
| $V_\infty$ | Total Free-stream Velocity |
| $y+$ | Non-dimensional turbulent wall spacing |
| $\theta_{tw}$ | Linear Twist |
| $\theta_0$ | Blade Collective Pitch Angle |
| $\theta_{1c}$ | Blade Longitudinal Cyclic Angle |
| $\theta_{1s}$ | Blade Lateral Cyclic Angle |
| $\gamma$ | Ratio of specific heats |
| $\Omega$ | Rotor angular velocity |

| | |
|---|---|
| $\mathbf{\Omega}$ | Fluid Rotation Tensor |
| $\omega$ | Fluid Vorticity |
| $\psi$ | Rotor azimuth |
| $\rho$ | Density |
| $\sigma$ | Rotor Solidity $= N_b c/(\pi R)$ |
| $\xi, \eta, \zeta$ | Computational coordinate directions |

All other symbols are defined locally during use.

# Chapter 1: Introduction

## 1.1 Challenges of Rotorcraft Aerodynamic Simulations

Rotorcraft are widely used for their versatility in different flight conditions. The ability to take off and land vertically makes the helicopter a vital aircraft for search and rescue, military, and more recently urban air mobility applications. The aerodynamics of rotorcraft involve interactions between many different components; as rotorcraft designs become more compact and sophisticated, the aerodynamic interactions between components plays a larger role on the performance and safety of the aircraft.

The aerodynamics of rotorcraft flight are very different from those of fixed-wing flight, and a key difference is the influence of the rotor wake. Fig. 1.1 shows an illustration of a single-main-rotor helicopter with the tip vortices, indicating the boundary of the rotor wake, shown as dotted lines. In most flight conditions the wake from each blade passes directly in the path of the next passing blade. The wake of the main rotor interacts with the fuselage and in some flight conditions, the tail rotor as well. Not shown in the simple illustration are other components which also contribute to the complex interactions, including the tail empennage and rotor hub. For rotorcraft with multiple rotors, the potential for aerodynamic interaction is increased.

A history of momentum theory analysis and studies of two and three-dimensional aerodynamic models have allowed innovative and well-trusted helicopter designs used

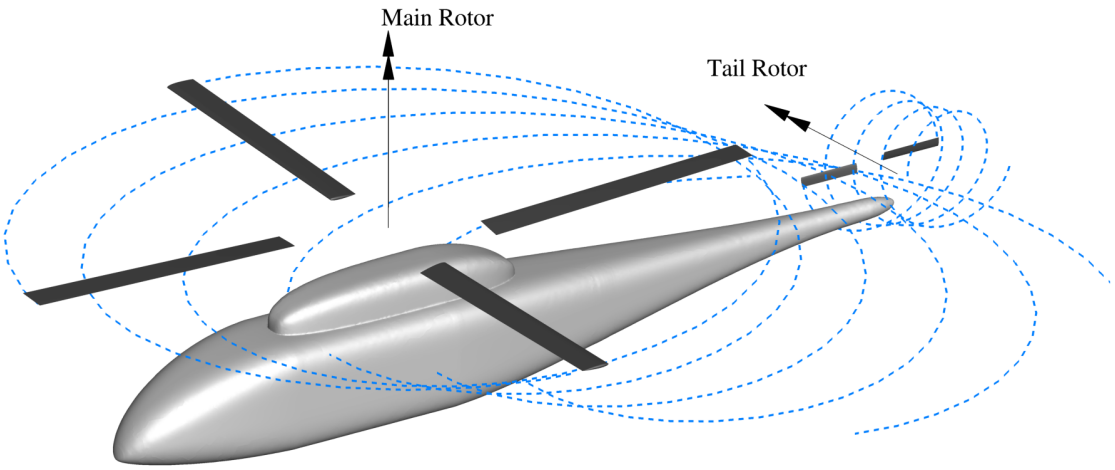



Figure 1.1: Illustration of the wake of a single-main-rotor helicopter. The tip vortices, shown as dotted lines, may interact with the fuselage and tail rotor.

for decades. Limitations of the available design tools, however, also result in flight conditions where the dynamics are not well understood. Maneuvers resulting in unstable or dangerous conditions are identified as best as possible and then avoided by pilots. A new generation of aircraft promises a wealth of novel, compact designs with challenging aerodynamic interactions still not understood or considered in the design process. The increasing power of computational tools for large scale aerodynamic analysis presents a promising aid for analysis of complex designs.

### 1.1.1 Physics of Rotorcraft Aerodynamics

Rotorcraft aerodynamics are characterized by a large envelope of flight conditions very different from those seen by fixed-wing aircraft. The two most elementary flight conditions for a helicopter, hover and forward-flight, present many aerodynamic challenges. The primary lifting forces from rotorcraft come from rotors, which include multiple blades. Unsteady rotation, wake interactions, and aeroelastic effects are all important factors which cannot be ignored for rotorcraft analysis. From the perspective of using Computational Fluid Dynamics (CFD) for the simulation of

rotorcraft, understanding the key aerodynamic challenges and how they translate into numerical or modeling challenges is crucial.

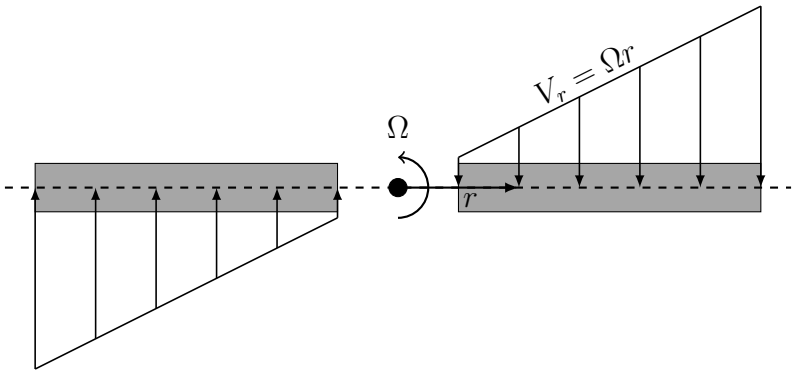



Figure 1.2: Illustration of aerodynamic variation across a rotor blade in hover.

In all rotor flight conditions, the inner and outer portions of the blade operate in very different flow regimes. Due to the rotational velocity of the blade, the outer portion of the blade experiences a much higher flow velocity compared to the blade root. In hover, the effective velocity at a radial station is approximately proportional to the radius $r$ from the center of rotation. Fig. 1.2 shows a schematic of the effective velocity along the blade. The Reynolds number, a ratio of inertial to viscous forces, is defined as

$$Re = \frac{\rho V_r L}{\mu} \tag{1.1}$$

where $\rho$ is the density, $V_r$ is the effective fluid velocity at some point along the blade, $L$ is a reference length, and $\mu$ is dynamic viscosity. The Reynolds number is highest on the outer portion of the blade where the effective velocity is highest. The Mach number, a ratio of the local fluid speed to the speed of sound, is defined as

$$M = \frac{V_r}{a} \tag{1.2}$$

where $a$ is the local speed of sound. Mach number is also higher on the outer portion of the blade. With a high enough tip speed, part of a helicopter blade may even

operate in transonic conditions resulting in a shock. CFD solvers for rotorcraft aerodynamics must be able to handle a large range of Reynolds number and Mach numbers to accurately resolve the flow physics.

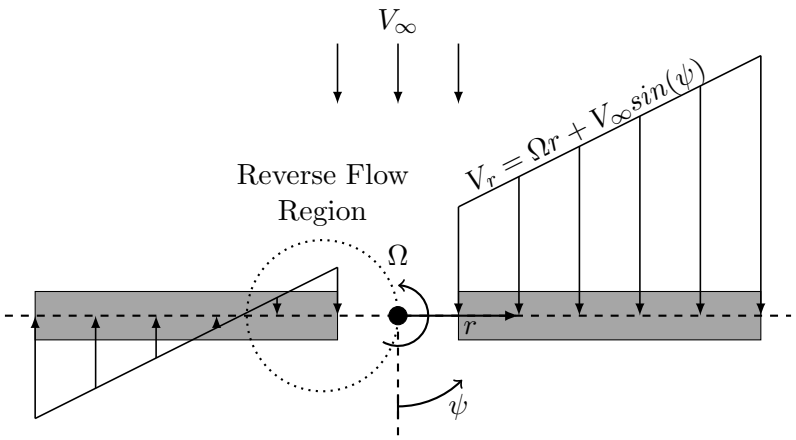



Figure 1.3: Illustration of aerodynamic variation across a rotor blade in forward flight. A reverse-flow region develops on the retreating side.

In forward flight, there is an azimuthal variation in the flow condition experience by the blade. Fig. 1.3 shows how the addition of a forward flight speed adds a variation based on azimuthal angle $\psi$. On the retreating blade side, the interior portion of the blade may operate in a reverse flow region, where the fluid travels "backwards" over the airfoil from the trailing edge to the leading edge. The reverse flow region often results in flow separation and unpredictable loads.

A key difference between fixed and rotary wing aerodynamics is that rotors operate in the influence of their own wake. A fixed wing aircraft trails vortices behind the wing but a rotating blade trails a vortex directly into the path of the next following blade. In the context of rotor simulations, CFD must therefore be able to accurately model the rotor wake, in addition to capturing the near-blade aerodynamics. A rotor wake typically extends far below the rotor in hover, indicating the need for a large simulation domain and proper treatment of boundaries.

Considering realistic flight conditions for a rotor further complicates the aerodynamics by adding a temporal variation to the flow field. In steady flight the dynamics

may be periodic but for maneuvers this is also not necessarily the case. Flight controls for conventional rotorcraft are prescribed using cyclic and collective pitch for the main rotor blades, meaning the pitch of the blade varies with blade azimuth. Either a periodic or entirely unsteady flow condition means that for CFD purposes, an accurate time-marching method is required to analyze temporal variation in the aerodynamics.

Most rotorcraft use multiple rotors to balance the aircraft torque and prevent uncontrollable spin. In a conventional single-main-rotor configuration, for example, the tail rotor provides a thrust to counter the torque generated to power the main rotor. In coaxial, tandem, and other multi-rotor designs, the rotational directions of rotors are reversed to cancel torques. A primary challenge from an aerodynamic perspective comes from the interaction between rotors. Fig. 1.4 shows a simulation result of a helicopter flow-field in "cross-wind" flight, where the vortex trailed from the main rotor enters the tail rotor. Such a flight condition might occur for forward flight in the presence of strong cross-winds. The vortices shed by the retreating side of the rotor combine into a larger vortex, which convects downstream. In cross-wind flight with the tail rotor downstream from the main rotor, the vortices from the main rotor enter the tail rotor plane and can cause unwanted vibrational or acoustic results.

In his book on *Military Helicopter Design Technology*, Prouty [4] discusses the design of the tail rotor and suggests the direction of rotation such that the top of the tail rotor travels aft of the aircraft for "reasons that are not well understood." Through wind tunnel and flight-testing, engineers have rules-of-thumb for such design choices which are known to reduce unpredictable loads, but without much aerodynamic justification. Besides main-tail rotor interactions, the placement of horizontal stabilizers is also a major challenge because of interactions with the main rotor wake. Such design considerations for single-main-rotor aircraft become only

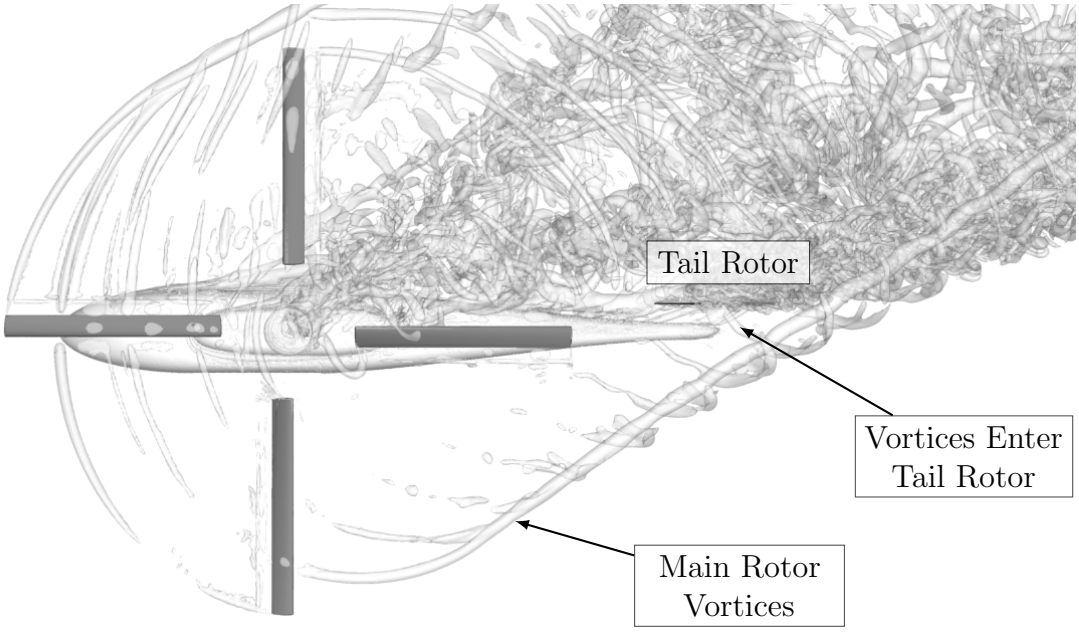



Figure 1.4: Rotor wake for a single-main-rotor in cross-wind flight. Main rotor vortices convect into the path of the tail rotor, possibly resulting in a dangerous flight condition.

more challenging for configurations with multiple, compact rotors interacting with large lifting surfaces. A CFD simulation for a full, complex rotorcraft configuration should therefore be capable of representing all aerodynamic components of interest.

Rotorcraft dynamics is a multi-disciplinary problem with coupling between structural dynamics and aerodynamics. Aeroelastic effects of the rotor blades cannot be ignored. Elastic flap, lag, and torsion all play a vital role in the control system design for the aircraft. Correct blade deformations rely upon correct aerodynamic loads; conversely the aerodynamics require accurate blade deflection.

### 1.1.2 Numerical Challenges of Rotorcraft CFD

The physical challenges associated with moving, deforming, multi-body and wake aerodynamics translate into numerical issues when attempting to simulate a rotorcraft configuration with CFD. The first step of any simulation process is the identification relevant aerodynamic areas and generating a mesh (or meshes) for

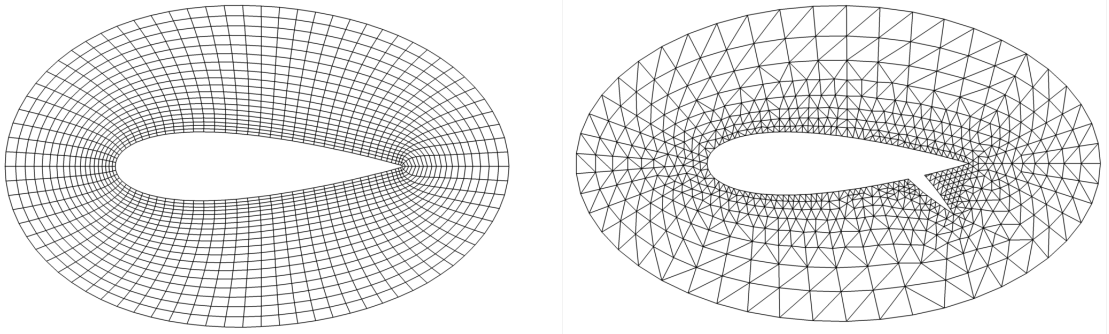
the simulation. The CFD mesh defines the points where a the flow solution can be analyzed. The meshes will undoubtedly include all relevant aerodynamic bodies but also any area where the capture of fluid structures might influence important engineering quantities. For rotorcraft applications a grid system must be generated for bodies like the rotorcraft blades, as well as areas in the "background" where the rotor wake is convected.

Mesh generation for a background grid is generally straightforward since it can be accomplished with a Cartesian mesh system. The goal of a background grid is generally to resolve wake structures and this is very efficiently done with equispaced Cartesian grids. The primary challenge in the generation of background grid is in knowing how fine the grid should be. In some areas perhaps a very small cell size might be needed to capture fluid structures with low dissipation whereas far away from the area of interest larger cells may suffice. The desire for different cell sizes in a domain may indicate the need for grid stretching or perhaps multiple, combined grids.

For a body of simple geometry, a grid may be generated which is curved, but maps to a structured, Cartesian domain. One example might be a cylindrical mesh, where one coordinate maps to the radial direction and another to the angular direction. Airfoils in 2D or blades and wings in 3D are typically of simple geometry and can be meshed using the structured, curvilinear approach. An example of a 2D structured, curvilinear airfoil mesh is shown in Fig. 1.5(a). One coordinate is in the airfoil-wrap direction (with a periodic boundary), and the second coordinate is in the airfoil-normal direction. The flow solution on the structured meshes can then be solved with compatible CFD software.

For a body of complex geometry, a mesh that maps to a Cartesian domain may not be possible, in which case an unstructured mesh might be generated instead. An unstructured mesh in 2D might contain a mixture of triangular and quadrilateral

elements and allow refinement in areas of aerodynamic interest. An example of a 2D unstructured mesh around an airfoil with an odd shape is shown in Fig. 1.5(b). Such a geometry would be difficult to mesh with a structured approach. While an unstructured mesh allows for significant meshing flexibility, depending on the CFD solver the benefits typically come at a cost of added computational time and/or reduced accuracy.



(a) Structured, Curvilinear (b) Unstructured

Figure 1.5: Examples of a structured, curvilinear mesh and an unstructured mesh for two airfoils.

The pros and cons of three general mesh categories present a core challenge in the use of CFD. Structured or Cartesian meshes are ideal for the background area far away from complex surfaces; unstructured grids are convenient to generate near complex surfaces, but inefficient elsewhere. Section 1.2 reviews some of the creative solutions developed to solve these issues.

### 1.1.3 CFD Time and Cost Challenges

Once the domain and flow condition is specified, a CFD solver has to run on a computer, or cluster of computers, to approximate the flow solution. CFD codes for large applications are typically made to run in parallel across multiple processors on a supercomputer. A supercomputer is a cluster of compute resources, for example cores of a Central Processing Unit (CPU). Based on the domain size,

availability of resources, and urgency of the simulation, a CFD run for a complex rotorcraft application could take on the order of days or weeks. For example, the simulation of a 9-second maneuver for the UH-60 Blackhawk helicopter performed by the Army Aviation Development Directorate took 5 days on 3600 CPU cores of a supercomputer [5].

A simulation taking days or weeks may not be feasible for some engineering analysis. For example in the process of rotorcraft design, tens or hundreds of potential designs might want to be tested but the simulation time of even one day is prohibitively high. The alternative of using more computational resources, however, means that the simulation could be significantly more expensive.

Avoiding high computation time for simulations is most easily done by reducing the numerical size or complexity of the problem and making assumptions about the physics for part of the domain. For example, the blade aerodynamics of a full, single-main-rotor configuration in hover might be simplified to the analysis of the isolated main rotor. The fuselage, tail rotor, and other components can either be ignored or included in the form of a lower-order model. In the case where more complicated physics are of interest, however, for example with rotor-rotor interactions or rotor-fuselage interaction, fewer assumptions can be made and a full, high-fidelity simulation cannot be avoided.

Fortunately for engineers, advancements in computer technology has enabled larger simulations to run faster and on fewer, more affordable resources. The benefit comes, however, at the cost of having to optimize or re-write code to take advantage of new compute architectures.

## 1.2 Literature Survey of Previous Work

Incomplete understanding of complex rotorcraft flight conditions motivates the need for improved high-fidelity, large-scale aerodynamic analysis tools. The

focus of this work the development of a computational framework for large-scale, efficient CFD. As such, a literature survey first presents an overview of CFD and other computational tools for interactional aerodynamic analysis. The survey then briefly reviews popular computer architectures and how the evolution of computers has facilitated physics-based simulations.

### 1.2.1 Computational Aerodynamics

The first implementations of CFD for transonic potential flow theory date back to the early 1970s with the work of Magnus and Yoshihara [6] and shortly after by Murman and Cole [7]. Using a mapped Cartesian grid, the transonic small disturbance equations correctly predicted airfoil characteristics for flow-fields with moderate shocks. The transonic potential theory, however, lacked accurate representation of vorticity and prediction of unique shock solutions. By 1981, Jameson et. al. [8] presented an efficient and accurate implementation of the 2D Euler equations for airfoil analysis. The 2D Euler equations applied to a transonic cylinder and NACA0012 were solved using a finite-volume scheme on curvilinear meshes. A scalar dissipation scheme was used with coefficients defined based on local pressure variation.

By 1980 implicit formulations of the viscous Navier–Stokes equations were also developed, for example by Pulliam and Steger [9], however the lack of computational power for realistic cases limited the viscous terms to thin-layer approximations. Significant effort was placed in formulating computationally efficient methods for the 3D compressible Navier–Stokes equations. Notably a diagonal alternating-direction implicit form of the governing equations was proposed by Pulliam and Chausee in 1981 [10] and an efficient L-U factorized form by Obayashi and Kuwahara in 1984 [11].

The development of turbulence models closely followed the development of

Navier–Stokes codes through the 1970s and 1980s. The works of Cebeci and Smith [12], Baldwin and Lomax [13], and Johnson and King [14] present algebraic turbulence models, which depend on the existence of a well-defined boundary layer. Later, one-equation (e.g. Spalart–Allmaras [15]) and two-equation (e.g. $k-\epsilon$ [16], $k-\omega$ SST [17]) transport models better captured separated flow physics and became widely used for rotorcraft and fixed-wing applications. The use of a turbulence model adds further computational complexity to the flow solver, since it adds another partial differential equation (PDE) to the system of equations solved at each timestep.

As mentioned in the previous section, one of the largest challenges for rotorcraft CFD is modeling the wake of a rotor. The wake from each blade must be adequately captured to accurately predict the rotor inflow and in turn, near-blade aerodynamics. The finite-volume and finite-difference methods commonly used in CFD have numerical dissipation, which results in accelerated, non-physical vortex diffusion. Using smaller cell sizes in the wake region helps reduce dissipation but at the cost of greatly increasing the degrees of freedom and cost of computation.

One method of reducing wake dissipation is with the use of a Lagrangian free-wake or other lower-order wake model. A free-wake model uses vorticity transport properties in a Lagrangian frame to represent the location and strength of discrete vortex filaments. With the assumption that the wake region far from the blade is governed by inviscid flow, the wake model represents the vortices trailed from blades without numerical dissipation. Rotorcraft applications of free-vortex methods are presented in detail in the work of Bagai and Leishman [18]. Free-wake models have been used with 2D CFD tables for interactional aerodynamic analysis. Yin and Ahmed [19] used free-vortex methods to analyze the acoustics of main-rotor and tail-rotor interactions. For improved, 3D representation of blade physics, the free-wake solution is typically coupled to an Euler or Navier–Stokes CFD solution using the Biot–Savart law at either the blade surface [20] or at a boundary surface

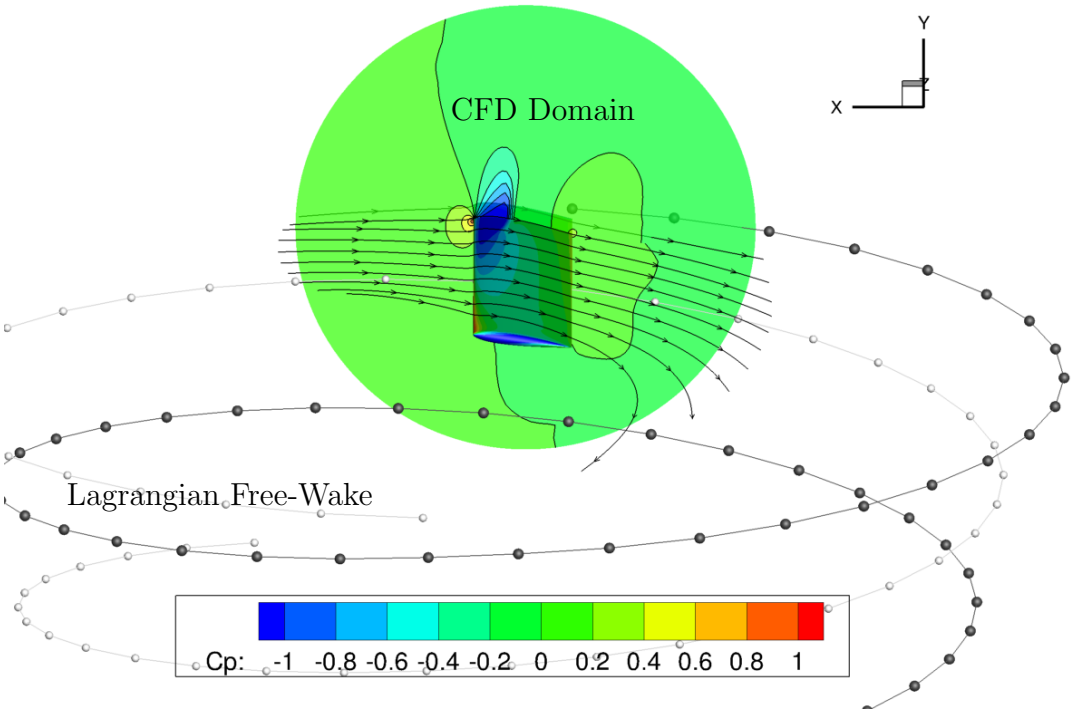



Figure 1.6: Hybrid Navier–Stokes + free-wake model of a rotor in hover. The CFD domain is near the blade and Lagrangian free-wake models the wake below the rotor.

near the blade [21]. An example of the latter implementation is shown in Fig. 1.6 for a hovering rotor. For the shown case, the Navier–Stokes CFD domain resolves the flow solution to a distance of $\sim$ 1.5 chords away and the free-wake markers model the influence of the other rotor blade and wake below the rotor. While the Navier–Stokes to free-wake coupling procedure may work for hover or periodic simulations, application of the Biot-Savart and Bernoulli equations for the influence of the vortices on the Navier–Stokes domain do not extend easily to complex, unsteady flight conditions.

When directly modeling the wake with Navier–Stokes CFD cannot be avoided, high-order numerical schemes can be used to reduce dissipation. High-order schemes can be achieved with relative ease on Cartesian grids by using a larger computational stencil. Furthermore with the use of Cartesian mapping, body-fitted structured grids can also make use of high order stencils. High-order central stencils may be used in areas of low gradients but especially near shocks, methods like the 3$^{\text{rd}}$ order

Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) [22] or $5^{\text{th}}$ order Weighted Essentially Non Oscillatory (WENO) [23] schemes use limiters for added stability. Central, MUSCL, and WENO schemes all require a stencil in a coordinate direction, which for structured or Cartesian meshes is clearly defined. In unstructured meshes it is more difficult to apply line-based stencils without the presence of clear coordinate directions. Most unstructured, finite-volume formulations are therefore limited to second-order accuracy.

The numerical methods mentioned thus far have been primarily finite-difference or finite-volume formulations. Historically, the development of CFD has favored finite-volume-type schemes because they inherently follow conservation laws. Finite-element methods, however, have also gained popularity especially in recent years. The finite-element method was successfully used by Brooks and Hughes [24] in the early 1980s using a Streamwise-Upwind Petrov-Galerkin approach. Lately Discontinuous Galerkin (DG) [25] and Hybridizable DG (HDG) [26] have shown tremendous promise in allowing high-order methods on unstructured grids. Finite-element codes can attain high-orders of accuracy by adding quadrature points in cells but the added degrees of freedom increase the numerical cost of the simulation. Furthermore for implicit time-marching schemes, having multiple solution points and shape functions per cell results in a very complex implicit system that is not easily solved using approximate factorization or Gauss–Seidel-type methods.

For rotorcraft flows, and other simulations involving motion, the problem emerges of how to mesh a domain when different grid-types are desired in different areas. Near rotor blades or other objects of simple geometry, it may be possible to use a curvilinear, Cartesian-mapped grid. For more complex geometries it may only be feasible to use unstructured meshes near the body. In the background, however, it is beneficial from an accuracy and computational cost perspective to use a simple Cartesian grid. Many researchers have had success using periodicity to simplify the

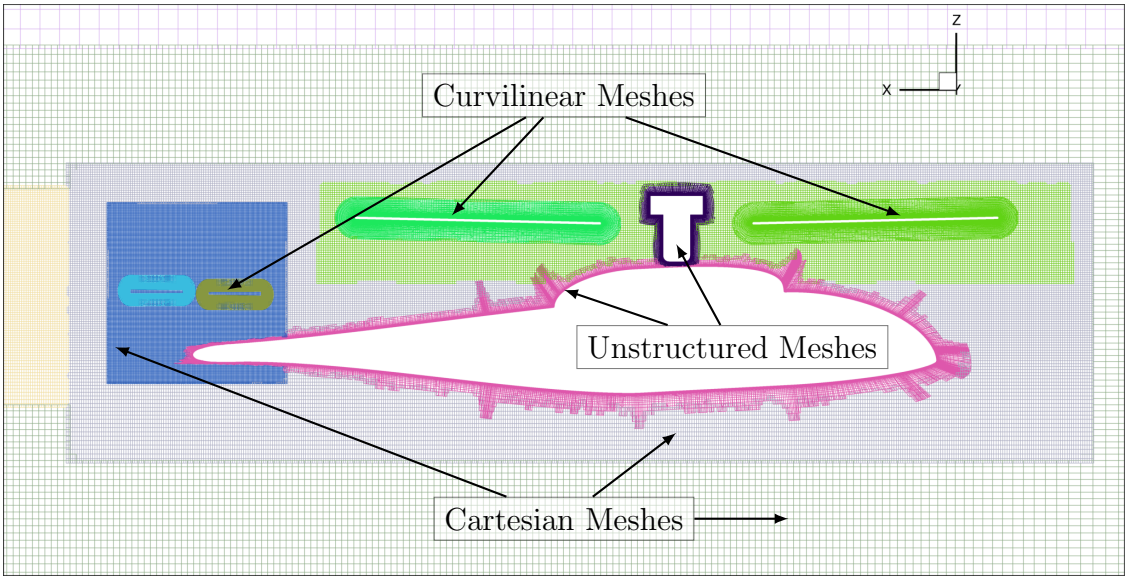



Figure 1.7: Overset CFD mesh system: unstructured and structured meshes used together.

problem in hover. Notable numerical studies for a rotor in hover include those from Caradonna and Isom [27] with potential flow equations, Roberts and Murman [28] and Chen and McCroskey [29] for Euler equations, and Srinivasan et. al. [30] for the Navier–Stokes equations.

Modern approaches for rotorcraft applications typically use overset (chimera) methods. Benek et. al. [31] first introduced the overset method in 1983 and referred to the method as "chimera" because it allowed the use of different mesh topologies together to form a single domain. At the boundary of each individual mesh, interpolation is used to stitch together a continuous solution. Duque and Srinivasan [32] applied the overset method to rotor hover applications in 1992, followed by Ahmed and Strawn in 1999 [33], and by Pomin and Wagner [34] for a rotor in forward flight. An example of an overset mesh system for a helicopter using different types of grids is shown in Fig. 1.7. Cartesian grids are used for the background region of the aircraft to resolve the wake. Blades of the main and tail rotor are of simple geometry and are meshed using a curvi-linear approach. The fuselage and hub, however, would be very difficult to mesh using structured grids and therefore use an unstructured approach.

Overset methods have been successfully used in a number of CFD solvers and frameworks. For the purposes of this work, a "solver" refers to stand-alone CFD software whereas a "framework" refers to a collection of CFD solvers and other numerical tools. The NASA Overflow [35] and FUN3d [36] codes are examples of two CFD solvers that can be used independently for single-grid or overset problems. The HPCMP CREATE AV$^{TM}$ Helios framework [37, 38], from the Department of Defense CREATE-AV program, interfaces with Overflow, FUN3D, and other CFD codes. In addition to CFD codes, frameworks can also use separate modules for grid-motion, domain connectivity, and fluid-structure interaction. The multi-solver, multi-mesh paradigm has been successfully applied to many complex aerodynamic applications including fixed-wing aircraft [39], rotary-wing aircraft [37, 38], and wind-turbine farms [40].

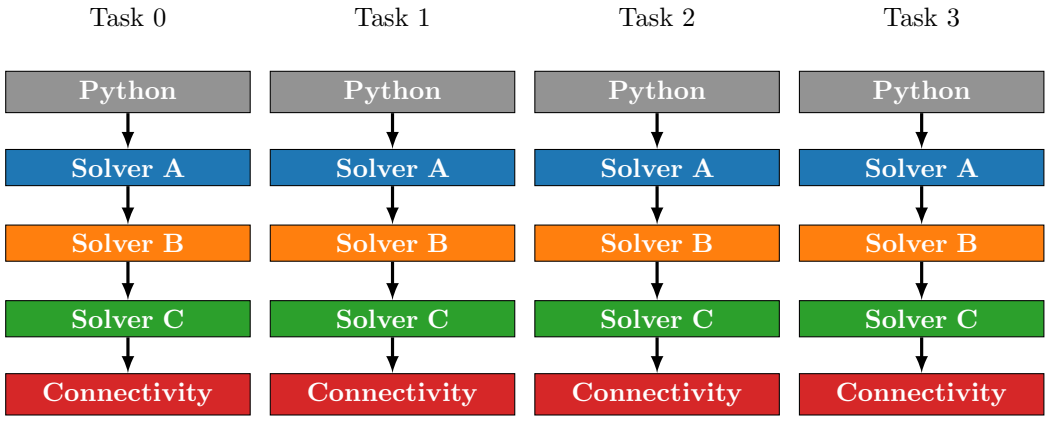



Figure 1.8: Example distribution of tasks suitable for a CPU-only CFD framework. Each solver uses all available MPI tasks.

Parallelization for the Helios framework is accomplished with domain decomposition and communication using the Message Passing Interface (MPI). For a framework with multiple CFD solvers, however, there are different ways of distributing resources. MPI uses "tasks" for parallelization and a task is typically run on a CPU core, but different CPU cores may have access to different hardware. The Helios framework was designed for CPU-based codes, and distributes all work over

all tasks. If there are three CFD solvers used, Solver A, Solver B, and Solver C, and a total of 12 MPI tasks, each solver uses all 12 tasks for domain decomposition. An flow chart of this approach in Fig. 1.8 shows the progression of routines on each task for a single simulation iteration. As long as each solver performs load balancing across tasks, the overall framework should be load balanced across all CPU cores.

For highly unsteady flow, fluid structures must be accurately captured and convected between overset meshes. Treatment of fringe points at mesh boundaries can have a large effect on convergence for complex problems. The traditional approach, for example used in Overflow and Helios, is to "freeze" the interpolated fringe points during sub-iterations. Subsequently, the overset boundary conditions in linear solver iterations are treated effectively as Dirichlet. The traditional methods are equivalent to a Schwarz alternating procedure [41], commonly used in numerical treatment of domain decomposition boundaries [42, 43]. The Schwarz method is convenient because the linear solver iterations typically reside within a CFD code that does not have access to overset connectivity routines. Some single-solver implementations have been reported in literature, where the interpolation constraints on the fringe points were included as part of an implicit linear system leading to improved convergence. In particular, Galbraith et. al. [44] used implicit overset boundaries in a discontinuous Galerkin code, showing the implicit treatment of interpolated points results in improved accuracy for a vortex convection case. Brazell et. al. [45] also used implicit interpolated terms in their implementation of a discontinuous Galerkin code to improve accuracy and convergence. The Overture overset code, from the work of Henshaw [46], uses overset interpolation equations as constraints for a multi-grid method applied to elliptic equations.

The works of both Galbraith and Brazell used a flexible Generalized Minimal Residual Method (GMRES) for the linear solver of the Newton or quasi-Newton linearization of the Navier–Stokes equations. GMRES is just one of many methods

that can be used to solve a linear system. Traditional methods in commonly CFD use Gauss–Seidel sweeps or approximate factorization methods to iteratively approximate the inversion of the linear system. Although GMRES has generally been popular in unstructured solvers like FUN3D (NASA) [47] or mStrand (CREATE AV$^{TM}$) [48], the linear solver can also help with the convergence of structured grid based codes using large, high-order stencils in areas of large gradients. The GMRES method by itself can take as many iterations as there are unknowns to achieve convergence, which is clearly intractable. Therefore, a suitable preconditioning scheme is always necessary within GMRES to achieve improved convergence. In this context, it is worth noting that both Galbraith and Brazell used the exact same preconditioners for all of their overset meshes. In a multi-solver context, this is not often feasible since the preconditioners that are implemented within each code depends on the specific needs of that solver.

### 1.2.2 Computational Architectures

Advancement of CFD algorithms and methods since the second half of the 20$^{th}$ century were made possible with the evolution of the computer. The work of Magnus and Yoshihara [6] in 1970 was performed on a CDC6400 mainframe computer; a computational domain containing 3700 points took 3.5 hours to simulate. Shortly after Murman and Cole improved the efficiency of the algorithm, achieving computation times of 30 minutes for a 3034-point mesh on an IBM 360/44 mainframe computer. The same pattern of algorithm development and algorithm optimization for computational hardware continues today.

One of the most notable advancements in scientific computing technology was the use of vectorizable or parallel operations. A primary conclusion from the work of Jameson et. al. in 1981 [8] was not only the ability to solve the Euler equations, but the ability to do so using vector operations on a Cray 1 computer.

Cray FORTRAN vectorized sections of the code as part of the compilation process and as a result, 500 cycles of a 4096-point domain could be run in 24 seconds. Besides compiler vectorization, OpenMP directives in codes were introduced to take advantage of multiple cores on a given Central Processing Unit (CPU) to make loops over a domain run in parallel. The Message Passing Interface [49] allows clusters of CPUs to communicate efficiently and has been one of the most popular methods of parallelization in scientific computing codes.

The availability of multi-core compute architectures has motivated the study of parallelism in popular numerical routines. In the work of Decker et. al. [50] the ARC3D code was made parallel using MPI and routines classified into three different categories: parallel-by-point, parallel-by-line, and parallel-by-plane. Each category describes the granularity with which an algorithm can be parallelized. Although the terminology was applied to compute architectures of the 1990s, the same terminology applies to the modern, massively parallel architectures available today.

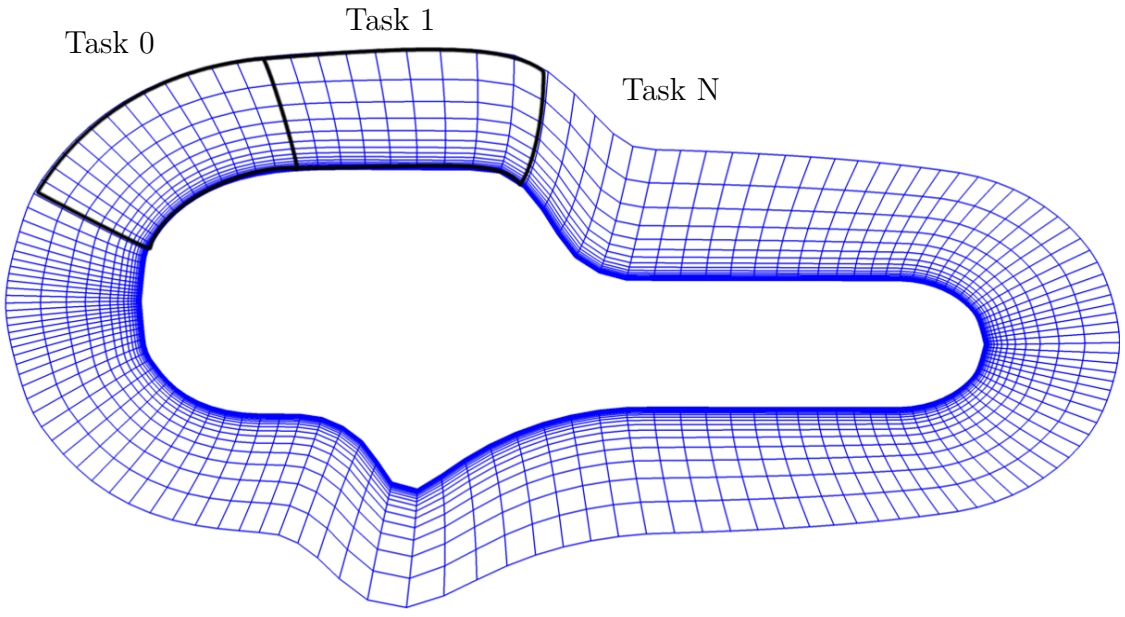



Figure 1.9: Domain partitioning.

Perhaps the most straight-forward implementation of parallelism is the with the use of domain decomposition. For a domain containing $N$ points, the problem

divided into $N/2$ points on two processors should run in half the time. Using MPI and a multi-core CPU or a cluster of CPUs, CFD domains can be sub-divided and distributed. Large clusters of computers, called supercomputers, were developed for large-scale simulations. Fig. 1.9 shows a cross-section of a coarse Hart II fuselage mesh, which can be divided into $N$ partitions and run on the same number of MPI Tasks. Partitioning can be performed for both structured or unstructured grids, although the presence of coordinate directions in the case of structured grids simplifies the process significantly.

The historical trend of CPU performance has generally followed Moore's law, which states that the number of transistors in an integrated circuit doubles every two years. The exponential growth of CPU computational power continued into the early 2000s, after which chip makers made up for stagnating processor speeds with increased core counts [2, 51]. Fig. 1.10 shows a history of micro processor data with trends for the number of transistors, single-thread performance, frequency, power, and logical core count [2]. Up until 2005, the majority of processors used a single logical core for computing. The slowing of single-threaded performance corresponds directly to the frequency stagnation around 3GHz and the need for additional cores to maintain the trend of increasing overall performance. The transistor count for processors has continued to rise as companies like Intel, NVIDIA, and AMD are able etch smaller and smaller transistors in silicon.

The Graphics Processing Unit (GPU) is one example of a many-core architecture that became widely used for video game applications. The GPU, however, is also capable of general purpose computing, as demonstrated in 2001 by Larsen and McAllister [52] for a matrix-matrix multiplication operation. NVIDIA, one of the largest manufacturers of GPUs, released the Compute Unified Device Architecture (CUDA), a C-like interface for general purpose GPU computing. Since then, many research groups have used GPUs for accelerated scientific computing. Directive-based

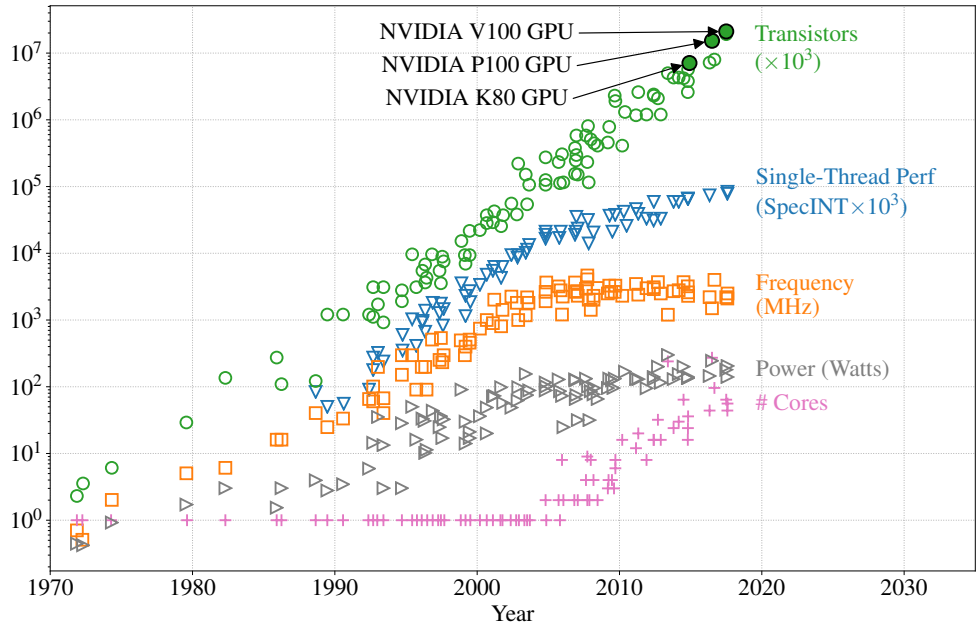



Figure 1.10: 45 years of micro-processor data with the transistor counts of three NVIDIA GPUs highlighted. Original data up to the year 2010 collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. New data collected for 2010-2015 by K. Rupp [2].

approaches have also been introduced such as OpenACC, which eliminate the need to write low-level code. Since the GPU is a processor just like the CPU, the performance can be compared to CPU performance. Three GPUs are highlighted in Fig. 1.10: the K80, P100, and V100 GPUs from NVIDIA. All three GPUs are at the top of the spectrum for transistor counts compared to CPUs and the trend shows continued increase.

The evolution of processors to include more transistors has resulted in a gradual shift in the hardware used by the world's most powerful supercomputers. Table 1.1 shows the 5 most powerful supercomputers in the world, based on data collected by TOP500 [1]. Sunway TaihuLight has by far the most number of CPU cores, at over 10 million. Both Summit and Sierra clusters, however, out-perform Sunway TaihuLight because of the presence of GPUs. Both Summit and Sierra use 6 and 4 NVIDIA V100 GPUs per node respectively.

| # | Name | Location | CPU Cores | $R_{MAX}$ (TFlop/s) | GPUs |
|---|---|---|---|---|---|
| 1 | Summit | USA | 2,397,824 | 143,500.0 | yes |
| 2 | Sierra | USA | 1,572,480 | 94,640.0 | yes |
| 3 | Sunway TaihuLight | China | 10,646,600 | 93,014.6 | no |
| 4 | Tianhe-2A | China | 4,981,760 | 61,444.5 | no |
| 5 | Piz Daint | Switzerland | 387,872 | 21,230.0 | yes |

Table 1.1: Top Supercomputers by Performance [1]. $R_{MAX}$ refers to maximum achieved LINPACK performance in tera-floating point operations per second.

Performance for GPU computing can be reported as a metric in absolute units, for example Floating Point Operations per second (FLOPS), or in relative units as a speedup compared to baseline hardware. Typically a speedup is defined compared to a CPU, where the metric is the time on the CPU divided by time on the GPU. To further complicate matters, speedups can be reported verses a single CPU core or the full multi-core CPU and performance varies significantly based on the generation of hardware used. As an order of magnitude comparison, a modern, high-end workstation CPU may have two dozen cores and performance measured in hundreds of Giga-FLOPS. A NVIDIA Tesla (compute-line) GPU has thousands of "CUDA cores" and performance measured in Tera-FLOPS.

Preliminary work from various groups using GPUs for CFD with NVIDIA Fermi (2010) and Kepler (2013) architectures show mixed performance results. Pickering et. al. [53] used OpenACC to accelerate an incompressible, explicit-timestepping, Navier–Stokes solver obtaining up to a 2.5× speedup for a Kepler-generation GPU over a 16-core Xeon processor. Soni et. al. [54] developed a full GPU-accelerated code in CUDA for an overset, compressible Navier–Stokes solver on structured and unstructured meshes. Single-precision and explicit time marching were used, resulting in up to a 27× speedup with the Fermi architecture over a single CPU core. The work of Jespersen [55] used CUDA to accelerate certain routines of the NASA Overflow CFD code, obtaining a $2.5 - 3\times$ speedup with the Fermi GPU architecture over a single CPU core.

With Pascal (2016) and Volta (2018) GPU architectures, larger, prominent CFD codes have started to shift towards using CUDA for acceleration. The NASA FUN3D code, an unstructured, implicit, compressible Navier–Stokes solver, was recently accelerated using CUDA and benchmarked on Pascal and Volta generation GPUs [56]. Compared to a dual 14-core Xeon node, the Pascal P100 and Volta V100 GPUs show a 3× and 6× speedup respectively. In a per-node comparison on the Summit supercomputer (Oak Ridge National Laboratory), 6 V100 GPUs showed a $23 - 27\times$ speedup over the dual-socket IBM Power9 CPUs. The FUN3D team was able to attain such high speedups by optimizing all algorithms and routines with CUDA.

## 1.3 Objectives

The main objectives of this dissertation are organized such that the resulting procedures and knowledge gained from this work may contribute to improved analysis of interactional aerodynamics.

- *Develop a GPU-accelerated, implicit, time-accurate Navier–Stokes solver for overset applications.* An existing GPU-accelerated solver for single, stationary grids is heavily modified for compatibility with an overset CFD framework and rotorcraft applications. The implicit, finite-volume methodology is implemented and analyzed using different spatial schemes, preconditioners and GMRES as a linear solver.

- *Design and implement a framework for the overset multi-solver paradigm amenable to heterogeneous architectures like the GPU.* A Python framework to organize function calls and data sharing is implemented and all CFD, grid motion, and domain connectivity modules are wrapped for access from Python. The framework distributes compute resources based on CFD code type (CPU

or GPU) and availability of resources on a cluster.

- *Identify and derive a robust, flexible algorithm for the overset multi-solver paradigm that considers fringe points implicitly.* Improve upon the traditional overset time-stepping approach by using a global GMRES solver applied to all grids. Implicit handling of interpolation regions is done with the goal of improving convergence for highly unsteady cases. The methodology is a framework-level implementation which can be easily adopted by other overset frameworks.

- *Verify and validate the individual components of the overset framework.* The CFD flow solver, grid motion module, and grid-connectivity codes are independently tested for verification of behavior and/or validation against experimental results. The overset GMRES algorithm is also validated for simple cases like the Poisson equation and inviscid flow.

- *Apply the overset framework to a full, notional rotorcraft configuration.* A notional configuration consisting of a ROBIN fuselage, hub, main rotor and tail rotor is run with the heterogeneous CFD framework. Aerodynamics and forces are analyzed for main-tail rotor interactions if forward and cross-wind flight.

- *Apply the implicit overset algorithm to a coaxial rotor configuration.* The heterogeneous framework along with the developed overset GMRES algorithm is applied to a laminar sphere and coaxial rotors for a Mars helicopter. Both low-Reynolds number case have significant unsteady flow structures passing between meshes, causing convergence challenges with traditional overset methods.

## 1.4 Organization of the Dissertation

The current chapter has reviewed challenges and existing approaches for analysis of complex rotorcraft applications using CFD. The problems and existing approaches were discussed from three perspectives: first from a physics standpoint, second from a numerical standpoint, and finally from a cost and performance standpoint. The remainder of the dissertation is organized into chapters for the methodology, validation, key results, and conclusions, in that order. The key results are divided into two chapters to distinguish between different applications.

The methodology chapter reviews the development or usage of each component in the overset framework as well as the framework itself. A detailed description of the governing fluid equations and numerical discretization is followed by implementation details for the GPU architecture. The mesh-motion and grid-connectivity modules are presented in context of heterogeneous, overset CFD. The framework itself is then described, including a framework-level implementation of a GMRES linear solver for convergence acceleration. Finally, a load-balancing strategy for CPU and GPU resources is described.

The verification and validation chapter presents the results of simple cases to validate sub-sets of the whole framework. Specifically, the developed GPU flow solver is analyzed for convergence and accuracy using single meshes, without any oversetting or grid-motion. The connectivity routines and GPU flow solver are validated together for a steady, transonic wing case. The grid-motion module is validated against a trusted NASA CFD solver for a UH-60 rotor flight condition. The developed overset GMRES algorithm is verified for a Poisson algorithm and validation is also performed for an unsteady, inviscid wedge case. Performance analysis of a steady, overset case run on different GPUs is compared to a similar NASA code to obtain speedup approximations and scalability metrics.

The first set of results pertains to a full configuration. The case from the ROBIN wind tunnel experiments, which analyze rotor-fuselage interaction, is augmented with a notional hub and tail rotor. The full configuration is analyzed in a forward-flight condition where the wake of the main rotor enters the tail rotor. Frequency analysis of the tail rotor loads with and without interactional aerodynamics are compared. Analysis of the full configuration and main-tail interactions is also performed for a cross-wind case.

The second set of results is concerned with the convergence of simulations. The heterogeneous framework is applied to a low-Reynolds number sphere and a set of laminar coaxial rotors designed for flight on Mars. Both cases can be challenging to converge using overset meshes and the effect of the overset GMRES algorithm on convergence is analyzed in detail.

The final chapter discusses conclusions from each of the outlined thesis objectives. Specifically the cumulative findings from the validation cases and results sections are reviewed in the context of advancing the state of the art. Contributions of the present work are described along with recommendations for future research directions.

# Chapter 2: Methodology

This chapter describes the mathematical and numerical methodology for the presented dissertation research. The governing equations and numerical implementation for the CFD solvers, grid motion, and domain connectivity are presented in detail. Design choices for the organization of the heterogeneous framework are discussed in context of performance and load-balancing between parallel tasks. Finally, a framework-level implementation of an overset GMRES algorithm is derived for an implicit global system.

## 2.1 Fluid Dynamics

### 2.1.1 Navier–Stokes Equations

The Navier–Stokes equations are a set of partial differential equations that govern the physical dynamics of a fluid. The Navier–Stokes equations are a mathematical description of three conservation relations:

1. Conservation of mass
2. Conservation of momentum (in $x, y$, and $z$ directions)
3. Conservation of energy

In strong conservation form, the Navier–Stokes equations are written as [9]:

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x} + \frac{\partial \mathbf{G}_i}{\partial y} + \frac{\partial \mathbf{H}_i}{\partial z} = \frac{\partial \mathbf{F}_v}{\partial x} + \frac{\partial \mathbf{G}_v}{\partial y} + \frac{\partial \mathbf{H}_v}{\partial z} + \mathbf{S} \tag{2.1}$$

where $\mathbf{Q}$ is the vector of conserved variables

$$\mathbf{Q} = \left\{ \begin{array}{c} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{array} \right\} \tag{2.2}$$

with $\rho$, $u$, $v$, $w$ and $e$ denoting the local flow density, three components of velocity and total energy, respectively. The inviscid fluxes and $\mathbf{F}_i, \mathbf{G}_i$, and $\mathbf{H}_i$ are

$$\mathbf{F}_i = \left\{ \begin{array}{c} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(e+p) \end{array} \right\}, \quad \mathbf{G}_i = \left\{ \begin{array}{c} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(e+p) \end{array} \right\}, \quad \mathbf{H}_i = \left\{ \begin{array}{c} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(e+p) \end{array} \right\} \tag{2.3}$$

where $p$ is the local pressure. The viscous fluxes $\mathbf{F}_v, \mathbf{G}_v$, and $\mathbf{H}_v$ are

$$\mathbf{F}_v = \left\{ \begin{array}{c} 0 \\ \tau_{\mathrm{xx}} \\ \tau_{\mathrm{yx}} \\ \tau_{\mathrm{zx}} \\ u\tau_{\mathrm{xx}} + v\tau_{\mathrm{xy}} + w\tau_{\mathrm{xz}} - q_x \end{array} \right\} \tag{2.4}$$

$$\mathbf{G}_v = \begin{Bmatrix} 0 \\ \tau_{\mathrm{xy}} \\ \tau_{\mathrm{yy}} \\ \tau_{\mathrm{zy}} \\ u\tau_{\mathrm{yx}} + v\tau_{\mathrm{yy}} + w\tau_{\mathrm{yz}} - q_y \end{Bmatrix} \tag{2.5}$$

$$\mathbf{H}_v = \begin{Bmatrix} 0 \\ \tau_{\mathrm{xz}} \\ \tau_{\mathrm{yz}} \\ \tau_{\mathrm{zz}} \\ u\tau_{\mathrm{zx}} + v\tau_{\mathrm{zy}} + w\tau_{\mathrm{zz}} - q_z \end{Bmatrix} \tag{2.6}$$

where $\tau$ is the stress tensor and $q$ is the rate of thermal conduction. The source term $\mathbf{S}$ is zero for all applications considered by this work. The pressure is obtained from the perfect gas law

$$p = (\gamma - 1)\left[e - \frac{1}{2}\rho(u^2 + v^2 + w^2)\right\} \tag{2.7}$$

where $\gamma$ is the ratio of the heat capacity at constant pressure ($C_p$) to the heat capacity at constant volume ($C_v$). For air $\gamma$ is assumed constant at 1.4. The rates of thermal conduction $q_x, q_y$ and $q_z$ are obtained from Fourrier's law

$$q_i = -k\frac{\partial T}{\partial x_i} \tag{2.8}$$

where $x_i$ is the direction, $T$ is the temperature, and $k$ is the thermal conductivity. The temperature is defined as a function of pressure and density using the perfect gas law

$$T = \frac{p}{\rho R} \tag{2.9}$$

where $R$ is the specific gas constant for air.

The stress tensor $\tau$ is represented as

$$\tau_{ij} = \mu \left[ \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right] + \lambda \left[ \frac{\partial u_k}{\partial x_k} \delta_{ij} \right] \tag{2.10}$$

where $\mu$ and $\lambda = -2\mu/3$ are the first and second coefficients of viscosity using Stokes' hypothesis. $\delta_{ij}$ is the Kronecker delta function. The fluid is assumed to be *Newtonian*, meaning the strain rate varies linearly with the applied shear. An approximation to the viscosity coefficient is obtained as a function of temperature using Sutherland's formula

$$\mu = C_1 \frac{T^{3/2}}{T + C_2} \tag{2.11}$$

where $C_1$ and $C_2$ are constants for air.

### 2.1.2 Non-dimensionalization of the Navier–Stokes Equations

Non-dimensional quantities such as Reynolds number and Mach number are generally used to describe a flow condition in a generic way that allows comparison without the need for similar unit scales. In CFD, it also normalizes quantities to near unity, ensuring accurate floating point representations. Non-dimensionalization of the Navier–Stokes equations is performed for all quantities as follows:

$$\begin{aligned} t^* &= \frac{t a_\infty}{L}, \quad (x^*, y^*, z^*) = \frac{(x, y, z)}{L}, \quad (u^*, v^*, w^*) = \frac{(u, v, w)}{a_\infty} \\ \rho^* &= \frac{\rho}{\rho_\infty}, \quad T^* = \frac{T}{T_\infty}, \quad p^* = \frac{p}{\rho a_\infty^2}, \quad e^* = \frac{e}{\rho a_\infty^2}, \quad \mu^* = \frac{\mu}{\mu_\infty} \end{aligned} \tag{2.12}$$

where $(\ )_\infty$ denotes a free-stream quantity and $L$ is a reference length, typically that of an airfoil chord. The non-dimensional parameters used to define the flow condition

are therefore:

$$\text{Reynolds number:} \quad Re_\infty = \frac{\rho V_\infty L}{\mu_\infty} \tag{2.13}$$

$$\text{Mach number:} \quad M_\infty = \frac{V_\infty}{a_\infty} \tag{2.14}$$

$$\text{Prandtl number:} \quad Pr_\infty = \frac{\mu_\infty C_{p,air}}{k_{air}} \tag{2.15}$$

where $C_p$ is the specific heat at constant pressure. A constant Prandtl number of 0.72 is used for all cases considered in this work. The free-stream velocity is defined as the total velocity from the three components:

$$V_\infty = \sqrt{u_\infty^2 + v_\infty^2 + w_\infty^2} \tag{2.16}$$

When non-dimensional $(\,)^*$ quantities are substituted into the governing equation, Eq. (2.1), the formulation is identical to the original formulation but with a few non-dimensional parameters appearing. For the remainder of this work the $(\,)^*$ notation will be dropped and all flow variables are non-dimensional unless otherwise specified. In non-dimensional form the formula for the stress tensor $\tau$ becomes:

$$\tau_{ij} = \frac{\mu M_\infty}{Re_\infty}\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] + \lambda\left[\frac{\partial u_k}{\partial x_k}\delta_{ij}\right] \tag{2.17}$$

and the non-dimensional thermal heat conduction term becomes:

$$q_i = \frac{\mu M_\infty}{Re_\infty Pr(\gamma - 1)}\frac{\partial T}{\partial x_i} \tag{2.18}$$

### 2.1.3 Reynolds-Averaged Navier–Stokes

The Navier–Stokes equations govern the flow of both laminar and turbulent regimes. With high Reynolds numbers, however, the onset of turbulence presents a challenge since all turbulent length scales must be accurately resolved. The use of

Reynolds-averaging significantly reduces the computational cost of simulations by allowing reasonable cell sizes. The Reynolds-averaged formulation is accomplished using a statistical averaging of the governing equations. Each variable $\phi$ is represented as a sum of averaged and fluctuating components

$$\phi = \overline{\phi} + \phi' \tag{2.19}$$

where $\phi'$ is the fluctuation and $\overline{\phi}$ is the average defined by

$$\overline{\phi} = \frac{1}{\chi}\frac{1}{\Delta t}\int_{t_0}^{t_0+\Delta t} \chi \cdot \phi(t)dt \tag{2.20}$$

where $\chi = \rho$ for a mass-weighted average. The Reynolds-Averaged Navier–Stokes (RANS) equations are obtained from substituting each flow variable into the original governing equation, Eq. (2.1), using the fluctuation assumption from Eq. (2.19). Using mathematical identities for averaged and fluctuating quantities, many terms cancel and the only resulting difference in the equations is the presence of the Reynolds stress tensor and heat-flux contribution:

$$(\tau_{ij})_{turb} = -\rho\overline{u_i' u_j'}, \quad (q_i)_{turb} = -\rho\overline{u_i' e'} \tag{2.21}$$

using the Boussinesq eddy viscosity hypothesis, the effective stress tensor is

$$(\tau_{ij})_{total} = (\mu_{lam} + \mu_{turb})\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\left(\frac{\partial u_k}{\partial x_k}\right)\delta_{ij}\right] \tag{2.22}$$

and $\mu_{turb}$ is found using a turbulence model.

### 2.1.4 Spalart–Allmaras Turbulence Model

The Spalart–Allmaras turbulence model [15] is used to approximate the value of $\mu_{turb}$ using transport properties. The model uses emperical constants based on

experimental results for turbulence over a flat plate. The laminar kinematic viscosity, $\nu = \mu/\rho$, is used along with the model variable $\tilde{\nu}$

$$\frac{D\tilde{\nu}}{Dt} = c_{b1}\tilde{S}\tilde{\nu} - c_{w1}f_w\left[\frac{\tilde{\nu}}{\tilde{d}}\right]^2 + \frac{1}{\sigma}\left[\nabla\cdot((\nu+\tilde{\nu})\nabla\tilde{\nu}) + c_{b2}(\nabla\tilde{\nu})^2\right] \tag{2.23}$$

where the turbulent eddy kinematic viscosity is

$$\nu_t = \tilde{\nu}f_{v1}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu} \tag{2.24}$$

and other variables are

$$\begin{gathered}\tilde{S} = \omega + \frac{\tilde{\nu}}{\kappa^2\tilde{d}^2}, \quad f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}}, \quad f_w = g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right]^{1/6} \\ g = r + c_{w2}(r^6 - r), \quad r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2\tilde{d}^2}\end{gathered} \tag{2.25}$$

and constants are defined as

$$\begin{aligned} c_{b1} &= 0.135 & c_{w2} &= 0.3 \\ c_{b2} &= 0.622 & c_{\nu 1} &= 7.1 \\ \sigma &= 2/3 & \kappa &= 0.41 \\ c_{w3} &= 2 & c_{w1} &= \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma} \end{aligned}$$

## 2.2 GPU-Accelerated Flow Solver

The GPU-Accelerated Rotor flow FIELD (GARFIELD) code solves a discretization of the RANS equations with SA turbulence model on structured, curvilinear

grids. GARFIELD uses a mix of finite-volume and finite-difference methods, with all fluid quantities stored at the center of grid cells. The use of structured, curvilinear grids limits applications to relatively simple geometries because the grid coordinates must map to an equi-spaced computational domain. Background mesh generation, with or without stretching is trivial. Body-fitted meshes for applications considered in this work use elliptic grid generation for cross sections, extrusion or interpolation between cross section, and smoothing in the span-wise direction. Once a structured grid is generated, a temporal and spatial discretization of the governing equations has to be applied.

### 2.2.1 Hybrid Finite-Difference, Finite-Volume

A finite-difference discretization of the governing Navier–Stokes equation,Eq. (2.1), is accomplished by performing a coordinate transformation from the physical domain, $(x, y, z)$, to the computational domain, $(\xi, \eta, \zeta)$. The $3 \times 3$ Jacobian matrix $(\overline{\overline{J}})$ and determinant $(J)$ are defined as

$$\overline{\overline{J}} = \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)}, \quad J = det\left(\overline{\overline{J}}\right) \tag{2.26}$$

Through the chain rule, the governing equations re-written in computational space are

$$\frac{\partial \hat{\mathbf{Q}}}{\partial t} + \frac{\partial \hat{\mathbf{F}}_i}{\partial \xi} + \frac{\partial \hat{\mathbf{G}}_i}{\partial \eta} + \frac{\partial \hat{\mathbf{H}}_i}{\partial \zeta} = \frac{\partial \hat{\mathbf{F}}_v}{\partial \xi} + \frac{\partial \hat{\mathbf{G}}_v}{\partial \eta} + \frac{\partial \hat{\mathbf{H}}_v}{\partial \zeta} + \mathbf{S} \tag{2.27}$$

where the conservative vector is scaled by the Jacobian

$$\hat{\mathbf{Q}} = \frac{1}{J}\begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{Bmatrix} \tag{2.28}$$

and the inviscid fluxes are transformed to

$$\hat{\mathbf{F}}_i = \frac{1}{J}\begin{Bmatrix} \rho V_\xi \\ \rho u V_\xi + \xi_x p \\ \rho v V_\xi + \xi_y p \\ \rho w V_\xi + \xi_z p \\ V_\xi(e+p) + V_{\xi,t} p \end{Bmatrix}, \quad \hat{\mathbf{F}}_v = \frac{1}{J}\begin{Bmatrix} 0 \\ \xi_x\tau_{xx} + \xi_y\tau_{xy} + \xi_z\tau_{xz} \\ \xi_x\tau_{yx} + \xi_y\tau_{yy} + \xi_z\tau_{yz} \\ \xi_x\tau_{zx} + \xi_y\tau_{zy} + \xi_z\tau_{zz} \\ \xi_x\Theta_x + \xi_y\Theta_y + \xi_z\Theta_z \end{Bmatrix} \tag{2.29}$$

where

$$\begin{aligned} \Theta_x &= u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + k\frac{\partial T}{\partial x} \\ \Theta_y &= u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + k\frac{\partial T}{\partial y} \\ \Theta_z &= u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + k\frac{\partial T}{\partial z}. \end{aligned}$$

The formulations for $\hat{\mathbf{H}}_{i,v}$ and $\hat{\mathbf{G}}_{i,v}$ are the same as for $\hat{\mathbf{F}}_{i,v}$ but with $\xi$ replaced by $\eta$ and $\zeta$ respectively. The contra-variant velocity in coordinate direction $\xi$ is defined as

$$V_\xi = \xi_x u + \xi_y v + \xi_z w - V_{\xi,t}, \quad V_{\xi,t} = \xi_x\frac{\partial x}{\partial t} + \xi_y\frac{\partial y}{\partial t} + \xi_z\frac{\partial z}{\partial t} \tag{2.30}$$

which includes influence of grid motion. Again relations for the two other coordinate directions are found by replacing $\xi$ with $\eta$ or $\zeta$. Note that grid motion appears in the convective fluxes but does not affect the viscous fluxes.

In computational space, coordinates $(\xi, \eta, \zeta)$ are indexed using $(j, k, l)$ for three dimensions. GARFIELD stores grid coordinates at grid nodes but the conservative solution vector, $\mathbf{Q}$, at the cell center. The entries of the Jacobian tensor, referred to as metrics, can be interpreted geometrically as face normal vectors. For example the face area at location $(j, k)$ in direction $\xi$ is

$$\mathbf{S}_{\xi,jk} = \frac{1}{J} \begin{bmatrix} \xi_x \\ \xi_y \\ \xi_z \end{bmatrix} \tag{2.31}$$

Furthermore the Jacobian $J$ is equivalent to the inverse cell volume on orthogonal grids. The geometric analogy is especially relevant because fluxes are evaluated at faces between cells and not at cell centers. As a result, the finite-difference implementation is equivalent to a finite-volume implementation. Fig. 2.1 shows cell $(j, k)$ for a 2D grid. Cartesian coordinate directions $x$ and $y$ do not line up with grid coordinate directions $\xi$ and $\eta$. The fluxes are evaluated at cell faces, which in a finite-volume approach is where the face geometry and vector is well defined.

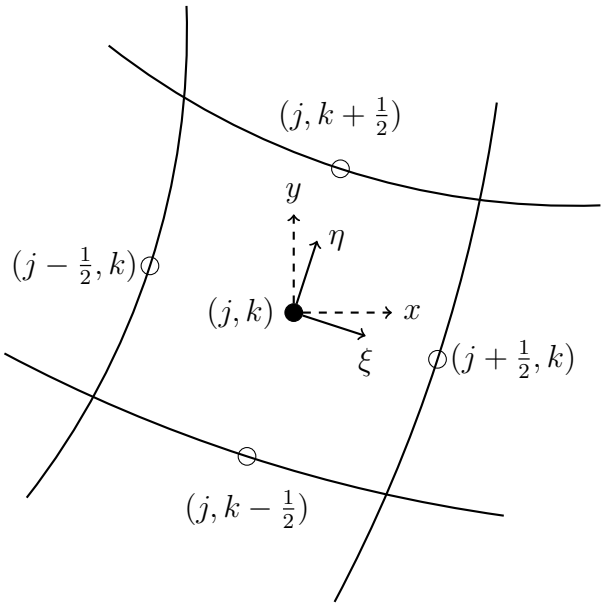



Figure 2.1: Cell-centered, curvilinear mesh for cell $(j, k)$ with computational coordinate directions.

The partial derivatives in space from Eq. (2.27) are discretized using a second order stencil. For example in the $\xi$ direction,

$$\frac{\partial \hat{\mathbf{F}}}{\partial \xi} = \hat{\mathbf{F}}_{j+1/2,k} - \hat{\mathbf{F}}_{j-1/2,k} \tag{2.32}$$

where $\hat{\mathbf{F}}$ is either the inviscid or viscous flux vector.

All discretizations discussed thus far have been in space. Discretization of Eq. (2.27) in time can be accomplished using either an implicit or explicit method. The Euler explicit method is numerically simple but imposes a large restriction on the timestep for the simulation. For rotorcraft flows, where fast-rotating blades result in challenging physical timestep sizes, the Euler implicit method is a commonly used approach. In discussing temporal discretizations, it is convenient to group all spatial terms in a single, residual term. The spatial residual, $\mathbf{R}$, is defined from Eq. (2.27) such that

$$\frac{\partial \hat{\mathbf{Q}}}{\partial t} + \mathbf{R} = 0 \tag{2.33}$$

meaning $\mathbf{R}$ contains all spatial operators. Next a second order backwards Euler implicit time marching scheme is imposed

$$\frac{3\hat{\mathbf{Q}}^{n+1} - 4\hat{\mathbf{Q}}^{n} + \hat{\mathbf{Q}}^{n-1}}{2\Delta t} + \mathbf{R}^{n+1} = 0 \tag{2.34}$$

The term $\mathbf{R}^{n+1}$ is non-linear and can be linearized using the first few terms of a Taylor series. A dual-time marching term, denoted with index $p$, is also required to get rid of linearization error. The dual-time marching term corresponds to a subiteration, or non-linear iteration, and uses symbol $\tau$. The solution at subiteration index $p+1$ serves as an approximation to the solution at physical temporal index $n+1$. The resulting discretization is

$$\frac{\hat{\mathbf{Q}}^{p+1} - \hat{\mathbf{Q}}^{p}}{\Delta\tau} + \frac{3\hat{\mathbf{Q}}^{p+1} - 4\hat{\mathbf{Q}}^{n} + \hat{\mathbf{Q}}^{n-1}}{2\Delta t} + \frac{\partial \mathbf{R}^{p}}{\partial \hat{\mathbf{Q}}}\Delta\hat{\mathbf{Q}} = -\mathbf{R}^{p} \tag{2.35}$$

where

$$\Delta\hat{\mathbf{Q}} = \hat{\mathbf{Q}}^{p+1} - \hat{\mathbf{Q}}^{p} \tag{2.36}$$

which can be further modified into the linear system

$$\left[\frac{\mathbf{I}}{\Delta\tau} + \frac{\mathbf{I}}{2/3\Delta t} + \frac{\partial \mathbf{R}^{p}}{\partial \hat{\mathbf{Q}}}\right]\Delta\hat{\mathbf{Q}} = -\mathbf{R}^{p} - \frac{3\hat{\mathbf{Q}}^{p} - 4\hat{\mathbf{Q}}^{n} + \hat{\mathbf{Q}}^{n-1}}{2\Delta t}. \tag{2.37}$$

where $\mathbf{I}$ is the identity matrix. The temporal terms with $t$ and $\tau$ on the left-hand-side can be lumped into a single time term $\hat{t}$

$$\left[\frac{\mathbf{I}}{\Delta\hat{t}} + \frac{\partial \mathbf{R}^{p}}{\partial \hat{\mathbf{Q}}}\right]\Delta\hat{\mathbf{Q}} = -\mathbf{R}^{p} - \frac{3\hat{\mathbf{Q}}^{p} - 4\hat{\mathbf{Q}}^{n} + \hat{\mathbf{Q}}^{n-1}}{2\Delta t}. \tag{2.38}$$

where

$$\Delta\hat{t} = \frac{\Delta\tau}{1 + \frac{3\Delta\tau}{2\Delta t}} \tag{2.39}$$

which can be scaled by the Jacobian (inverse volume) and re-written as

$$\left[\frac{\mathbf{I}}{\Delta\hat{t}} + J \cdot \frac{\partial \mathbf{R}^{p}}{\partial \mathbf{Q}}\right]\Delta\mathbf{Q} = -J \cdot \mathbf{R}^{p} - \frac{3\mathbf{Q}^{p} - 4\mathbf{Q}^{n} + \mathbf{Q}^{n-1}}{2\Delta t}. \tag{2.40}$$

The volume-scaled formulation from Eq. (2.40) is preferred over Eq. (2.38) for two reasons. First, it replaces computational vector $\hat{\mathbf{Q}}$ with the more familiar, original conservative vector $\mathbf{Q}$. Second, the right-hand-side of Eq. (2.40) is the residual for unsteady simulations used to measure convergence. Without volume scaling the convergence trend is biased by the influence of large cells, which typically converge well because they are far from surfaces.

The right-hand-side (RHS) of Eq. (2.40) is the *explicit* side, because all data is readily available. The left-hand-side (LHS) of Eq. (2.40) is the *implicit* side, which forms a matrix that needs to be inverted to find the solution, $\Delta\mathbf{Q}$. The entire RHS, including the temporal terms, is referred to as the residual for time-accurate computation. The L2-norm of the residual is a measure of subiteration convergence, since as the solution at $p$ approaches $p+1=n+1$, Eq. (2.34) is recovered. The inversion of the LHS matrix is rarely exactly computed and inverted, since this process is typically very computationally expensive. Instead there are three general procedures that are used in GARFIELD to obtain the solution $\Delta\mathbf{Q}$:

1. Approximate Factorization preconditioner: Diagonalized Alternating Direction Implicit (DADI), applied iteratively.

2. Red-Black Gauss–Seidel preconditioner: Point Red-Black, Line Red-Black, or mixed, applied iteratively.

3. Generalized Minimal Residual Method (GMRES): A minimization-problem approach to solving linear systems.

### 2.2.2 Inviscid Fluxes

The inviscid flux contribution to the residual from the hybrid finite-volume, finite-difference approach use the formulation

$$\frac{\partial\hat{\mathbf{F}}}{\partial\xi}=\hat{\mathbf{F}}_{j+1/2,k}-\hat{\mathbf{F}}_{j-1/2,k} \tag{2.41}$$

for the $\xi$-direction. The flux quantity can be either interpreted in computational space using the second-order finite-difference relation, or in geometric space where the $j+1/2$ index corresponds to a $\xi$ face of the cell with normal $\mathbf{S}_\xi$.

Computing the flux at the face is performed in two steps. The fluxes are functions of the conservative or primitive variables. The first step is to reconstruct

the value of the flow variables at the face in a way that allows for discontinuities (for example in the case of shocks). After reconstruction, values on the left and right sides of the interface are combined into a single flux using the Roe Riemann solver [57].

Reconstruction in GARFIELD is performed for using primitive variables $\rho$,$u$,$v$,$w$, and $p$. Two different reconstruction schemes are available for use. The Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) [22] uses a 3-cell stencil to for a 3$^{\text{rd}}$ order accurate reconstruction. The MUSCL reconstruction stencil for the right and left values of face $j-1/2$ is shown in Fig. 2.2.

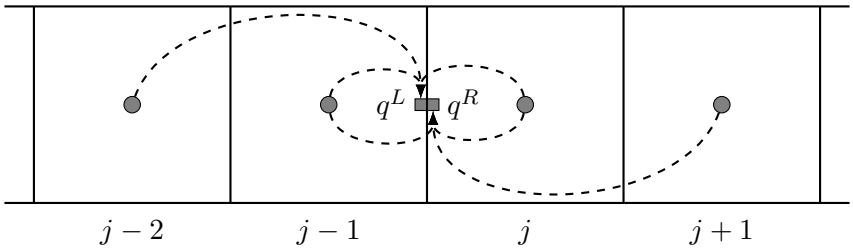



Figure 2.2: MUSCL 3-point stencil for left and right primitive variables $q_L$ and $q_R$.

A mathematical form of the scheme is

$$q^L_{j-1/2} = \overline{q} + \phi_j \left[ \frac{1}{3} \left( \overline{q}_{j+1} - \overline{q}_j \right) + \frac{1}{6} \left( \overline{q}_j - \overline{q}_{j-1} \right) \right] \tag{2.42}$$

$$q^R_{j+1/2} = \overline{q} + \phi_j \left[ \frac{1}{3} \left( \overline{q}_{j+1} - \overline{q}_j \right) + \frac{1}{6} \left( \overline{q}_j - \overline{q}_{j-1} \right) \right] \tag{2.43}$$

where the Koren's limiter [22] $\phi$ is given by

$$\phi_j = \frac{3\Delta\overline{q}_j \nabla\overline{q}_j + \epsilon}{2(\Delta\overline{q}_j - \nabla\overline{q}_j)^2 + 3\Delta\overline{q}_j \nabla\overline{q}_j^2 + \epsilon} \tag{2.44}$$

and $\Delta$ and $\nabla$ are forward and backward differences respectively. For any given face, the stencil requires two points on both sides of that face.

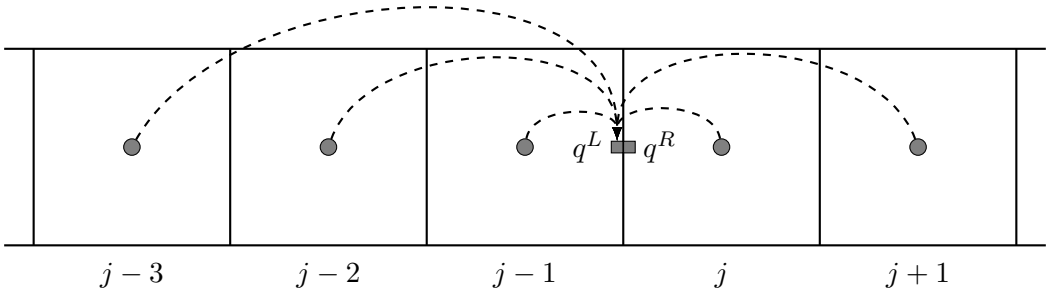



Figure 2.3: WENO 5-point stencil for left and right primitive variables $q_L$ and $q_R$.

The $5^{\text{th}}$ order Weighted Essentially Non Oscillatory (WENO) scheme [23, 58] uses a larger stencil, as shown in Fig. 2.3. For clarity, the figure shown only the stencil for the left state, however just as with the MUSCL scheme, the right state stencil is a mirror image of the left stencil. The scheme is implemented with two different options for weights: either from Jiang and Shu [23] or from Borges et. al. [58].

Once left and right primitive variables are found for a face, the Roe-flux Riemann solver [57] is used to construct a flux with appropriate dissipation to ensure numerical stability. The Roe-flux calculations can be summarized as follows

$$\hat{\mathbf{F}}_{j-1/2} = \frac{\hat{\mathbf{F}}(q^L) + \hat{\mathbf{F}}(q^R)}{2} - ||\mathbf{A}_{\text{Roe}}(q^L, q^R)||(q^R - q^L) \tag{2.45}$$

where $||\mathbf{A}_{\text{Roe}}(q^L, q^R)||$ is the flux Jacobian matrix, $\partial\hat{\mathbf{F}}/\partial\hat{\mathbf{Q}}$ constructed using so called Roe-averaged values, and the absolute value of all eigenvalues. A detailed description of the methodology is given in the work of Blazek [59]. The implementation in Garfield also uses Harten's entropy correction [60] to correct behavior near sonic points:

$$|\Lambda_c| = \begin{cases} |\Lambda_c| & \text{if } |\Lambda_c| > \epsilon \\ \frac{\Lambda_c^2 + \epsilon^2}{2\epsilon} & \text{if } |\Lambda_c| \leq \epsilon \end{cases} \tag{2.46}$$

where $\Lambda_c$ is an eigenvalue of the flux Jacobian.

### 2.2.3 Viscous Fluxes

The viscous fluxes are also found at the faces of each cell and require both the flow quantities and gradients at the center of the face. The quantities are found using a simple average in the face direction. For the $x$-direction velocity, for example:

$$u_{j+1/2} = \frac{u_j + u_{j+1}}{2}. \tag{2.47}$$

Finding the gradient of quantities is more challenging. Gradients are required for the flow velocities, $u, v, w$, and the temperature $T$. Using the grid metrics, finite differences in the computational domain can be translated into gradients in the physical domain. Using the $x$-direction velocity as an example:

$$\begin{aligned} \frac{\partial u}{\partial x} &= \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x} + \frac{\partial u}{\partial \zeta}\frac{\partial \zeta}{\partial x} \\ &= \frac{\partial u}{\partial \xi}\xi_x + \frac{\partial u}{\partial \eta}\eta_x + \frac{\partial u}{\partial \zeta}\zeta_x \end{aligned} \tag{2.48}$$

where stencils for $\frac{\partial u}{\partial \xi}, \frac{\partial u}{\partial \eta}, \frac{\partial u}{\partial \zeta}$ need to be obtained. Using the $\xi$-direction face as an example, the stencil is simply

$$\frac{\partial u}{\partial \xi}\Big)_{j+1/2} = u_{j+1} - u_j \tag{2.49}$$

the other directions require averaging components from neighboring cells. Fig. 2.4 shows the cells used in the computational stencil for cross terms.

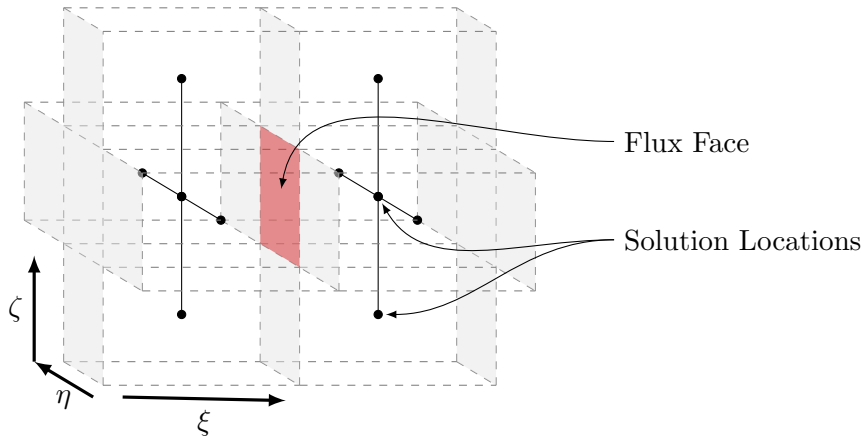



Figure 2.4: Viscous flux stencil for finite-difference relations.

### 2.2.4 DADI Preconditioner

The Diagonalized Alternating Direction Implicit (DADI) preconditioner [10] uses an approximate factorization technique to approximate the solution of the linear system from Eq. (2.40). Consider only the inviscid terms from the left-hand-side of the system:

$$\begin{aligned}\frac{\partial \mathbf{R}^p}{\partial \mathbf{Q}} &= \frac{\partial}{\partial \mathbf{Q}}\frac{\partial \hat{\mathbf{F}}_i}{\partial \xi} + \frac{\partial}{\partial \mathbf{Q}}\frac{\partial \hat{\mathbf{G}}_i}{\partial \eta} + \frac{\partial}{\partial \mathbf{Q}}\frac{\partial \hat{\mathbf{H}}_i}{\partial \zeta} \\ &= \frac{\partial}{\partial \xi}\frac{\partial \hat{\mathbf{F}}_i}{\partial \mathbf{Q}} + \frac{\partial}{\partial \eta}\frac{\partial \hat{\mathbf{G}}_i}{\partial \mathbf{Q}} + \frac{\partial}{\partial \zeta}\frac{\partial \hat{\mathbf{H}}_i}{\partial \mathbf{Q}} \\ &= \frac{\partial}{\partial \xi}\mathbf{A} + \frac{\partial}{\partial \eta}\mathbf{B} + \frac{\partial}{\partial \zeta}\mathbf{C} \\ &= \partial_\xi \mathbf{A} + \partial_\eta \mathbf{B} + \partial_\zeta \mathbf{C}\end{aligned} \tag{2.50}$$

where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are the flux Jacobians in the $j$-,$k$-,and $l$-directions respectively. The full left hand side of the linear system can be written as

$$\left[\frac{\mathbf{I}}{\Delta \hat{t}} + \partial_\xi \mathbf{A} + \partial_\eta \mathbf{B} + \partial_\zeta \mathbf{C}\right] \Delta \mathbf{Q} \tag{2.51}$$

After multiplying through by the timestep, Eq. (2.51) can be approximately

factored into

$$\left[\mathbf{I} + \Delta\hat{t}\partial_\xi \mathbf{A}\right]\left[\mathbf{I} + \Delta\hat{t}\partial_\eta \mathbf{B}\right]\left[\mathbf{I} + \Delta\hat{t}\partial_\zeta \mathbf{C}\right]\Delta\mathbf{Q} \tag{2.52}$$

Each flux Jacobian can be factored into eigenvectors and eigenvalues. Assuming the eigenvectors do not vary significantly in space, the eigenvectors can be factored out of the bracketed quantities entirely:

$$T_\xi\left[\mathbf{I} + \Delta\hat{t}\partial_\xi \Lambda_\xi\right]T_\xi^{-1}T_\eta\left[\mathbf{I} + \Delta\hat{t}\partial_\eta \Lambda_\eta\right]T_\eta^{-1}T_\zeta\left[\mathbf{I} + \Delta\hat{t}\partial_\zeta \Lambda_\zeta\right]T_\zeta^{-1}\Delta\mathbf{Q} \tag{2.53}$$

Because analytical forms of the eigenvectors are known, they can easily be inverted. The bracketed quantities in Eq. (2.53) are scalar, and form a tri-diagonal system if the finite difference operator $\partial$ uses an upwind scheme. Each set of tri-diagonal systems can be inverted analytically using the Thomas or Sherman–Morrison (if periodic) algorithms.

The DADI method as described does not include any influence of viscous terms. Influence of viscosity can be added as an eigenvalue contribution to the system. The viscous eigenvalues are

$$\Lambda_{v,\xi} = \gamma\left(\frac{\nu_L}{Pr_L} + \frac{\nu_T}{Pr_T}\right)\frac{\xi_x^2 + \xi_y^2 + \xi_z^2}{J} \tag{2.54}$$

for the $\xi$ direction where $()_L$ and $()_T$ denote laminar and turbulent quantities respectively. Eigenvalues in the other two coordinate directions are found by replacing $\xi$ with $\eta$ or $\zeta$.

### 2.2.5 Red-Black Gauss–Seidel Preconditioner

Gauss–Seidel-type preconditioners take a different approach to approximating the inversion of the left-hand-side. The same first step is taken as with DADI

$$\left[\frac{\mathbf{I}}{\Delta\hat{t}} + \partial_\xi \mathbf{A} + \partial_\eta \mathbf{B} + \partial_\zeta \mathbf{C}\right] \Delta\mathbf{Q} = RHS \tag{2.55}$$

but the flux Jacobians are kept intact as $5 \times 5$ block matrices. Furthermore the flux Jacobians are the full Jacobians of the Roe-fluxes but using only a first-order reconstruction. The matrix on the left-hand-side can be broken down into lower, diagonal, and upper quantities:

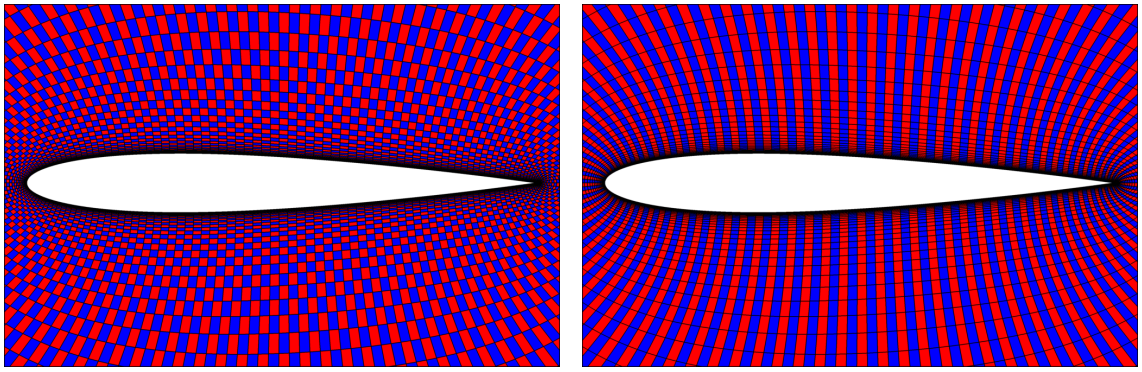
$$\begin{aligned}[\mathbf{I} + \Delta\hat{t}(&\mathbf{L_A} + \mathbf{D_A} + \mathbf{U_A} + \\ &\mathbf{L_B} + \mathbf{D_B} + \mathbf{U_B} + \\ &\mathbf{L_C} + \mathbf{D_C} + \mathbf{U_C})]\Delta\mathbf{Q} = RHS\end{aligned} \tag{2.56}$$

where **A**, **B** and **C** subscripts denote the different coordinate directions and **L**, **D** and **U** denote the lower, diagonal, and upper blocks of the matrix. A coloring method is chosen to paint cells and each color is solved independently of the other colors. A red-black coloring method uses two colors and a point-red-black results in the coloring shown in Fig. 2.5(a) (blue is used in the place of black to distinguish from the document text color). Point-red-black methods are popular for unstructured CFD solvers since they do not rely upon line structures in the grid [47]. Since the blue cells only depend on red cells, an approximation to $\Delta\mathbf{Q}$ can be used to move all of the red values to the right-hand side of the equation:

$$\begin{aligned}[\mathbf{I} + \Delta\hat{t}(\mathbf{D_A} + \mathbf{D_B} + \mathbf{D_C})]\Delta\mathbf{Q} = \\ RHS - \Delta t(\mathbf{L_A} + \mathbf{U_A} + \mathbf{L_B} + \mathbf{U_B} + \mathbf{L_C} + \mathbf{U_C})\Delta\mathbf{Q}\end{aligned} \tag{2.57}$$

Subsequently the left-hand-side is easily invertible and a new approximation to

$\Delta\mathbf{Q}$ is obtained. Alternating back and forth between colors results in a converged approximation to $\Delta\mathbf{Q}$.



(a) Point Coloring (b) Line Coloring

Figure 2.5: Illustration of red-black coloring.

Line red-black coloring is shown in Fig. 2.5(b) where each wall-normal line is painted the same color. The resulting equations now include a block tri-diagonal system in the direction of the lines:

$$[\mathbf{I} + \Delta t(\mathbf{L_A} + \mathbf{D_A} + \mathbf{U_A} + \mathbf{D_B} + \mathbf{D_C})]\Delta\mathbf{Q} = RHS - \Delta t(\mathbf{L_B} + \mathbf{U_B} + \mathbf{L_C} + \mathbf{U_C})\Delta\mathbf{Q} \quad (2.58)$$

Solving a block tri-diagonal system can be done with a block-implementation of the Thomas algorithm. While the tri-diagonal solve is more computationally expensive than the simple diagonal inversion from the point method, the line-implicit nature of the algorithm results in improved stability.

Garfield has the option of using the point Gauss–Seidel algorithm, line algorithm, or a mixture of both. In the case of a mixed approach for body meshes, the wall-normal direction (the stiffest numerically) uses a line method and the two wall-parallel coordinates use the point method.

The Red-Black Gauss–Seidel method described has not included viscous effects.

A similar approach to the one used for the DADI algorithm is applied to the Red-Black algorithm. Because the Red-Black method does not decompose matrices into eigenvectors, however, an approximation is made by adding the discretized eigenvalues to the diagonals of the block matrices.

### 2.2.6 GMRES Linear Solver

The Generalized Minimal Residual (GMRES) linear solver implemented in Garfield is identical to the framework-wide implementation described in Sec. 2.8.

### 2.2.7 Boundary Conditions

As a structured, cell-centered code, Garfield stores the grid coordinates at nodes and solution at cell centers. Ghost cells are extended from the physical domain to prescribe boundary conditions. Ghost cells at physical boundaries are prescribed such that when they are used in a spatial stencil, the resulting solution at the boundary face matches the desired physical boundary. Fig. 2.6 illustrates interior points as filled circles, ghost points as hollow circles, and the location of boundaries. Two ghost points are used in this example, as required by the MUSCL scheme.

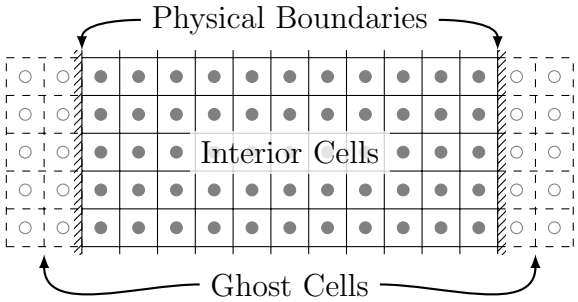



Figure 2.6: Full domain consisting of interior cells, ghost, cells, and boundaries.

Domain decomposition allows a single domain to be distributed onto different, concurrent MPI tasks. The full domain from Fig. 2.6 can be divided onto two MPI tasks as shown in Fig. 2.7. Additional MPI boundaries are introduced and ghost

points get their values from corresponding field points in the neighboring meshes.

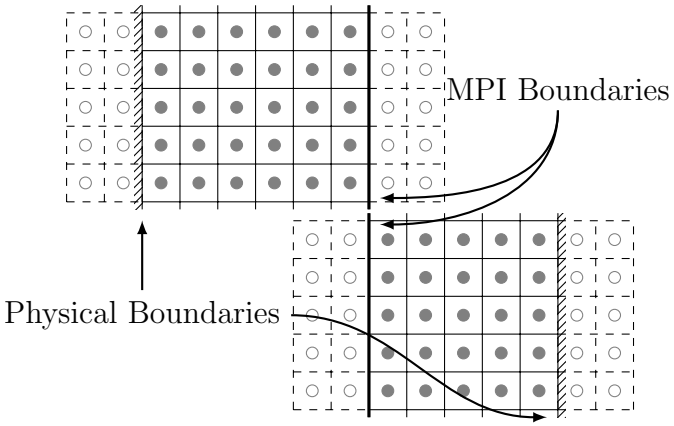



Figure 2.7: Partitioned domain on two MPI tasks.

Besides an MPI-boundary, there are other geometric boundary conditions where ghost points are assigned values from elsewhere in the mesh. One is a *periodic* boundary, which appears most commonly in the wrap-around direction of an airfoil mesh. A periodic boundary for a 2D cross-section of an O-mesh is shown in Fig. 2.8. Ghost points are added across the periodic boundary and exchanged each sub-iteration. Periodic boundaries also results in the use of the Sherman–Morrison algorithm in the DADI routine to solver a periodic tri-diagonal system of equations.

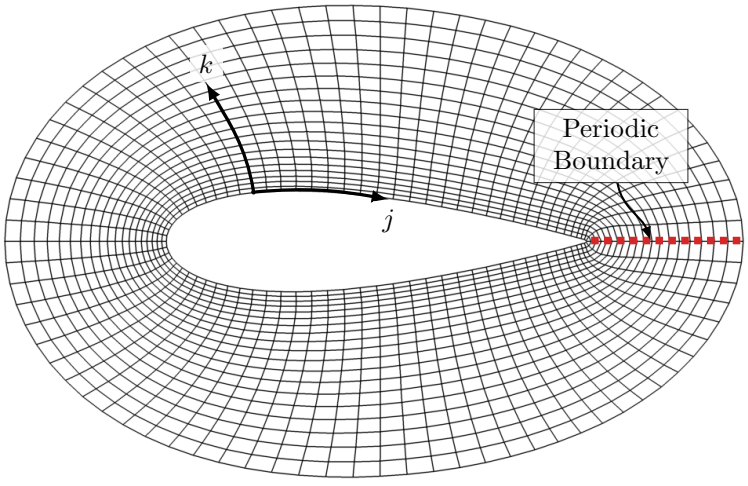



Figure 2.8: O-Mesh curvilinear coordinates result in a periodic boundary in the wrap-around direction.

A *wake* boundary can appear in 2D or 3D C-type meshes as well as at the

root and tip of blade meshes. An example of coarse blade mesh is shown in Fig. 2.9. As the span-wise ($l$-direction) sections of the mesh approach the tip of the blade, the sections start to fan towards the end of the blade and eventually collapse to a plane. This behavior occurs on both the root and tip of the blade. Similar to the treatment of periodic boundaries, the existence of physical domain cells across the boundary means that ghost points can be given real coordinates and values from interior points in the mesh.

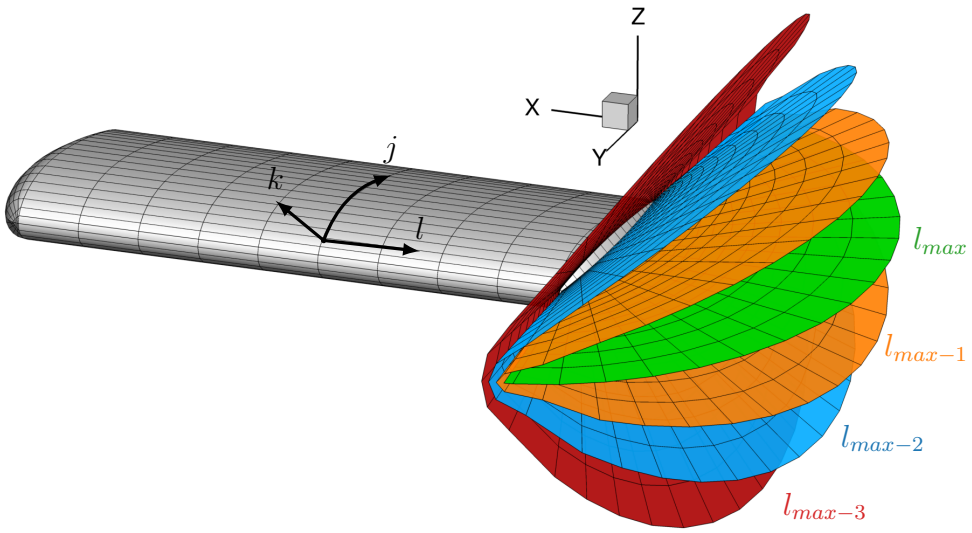



Figure 2.9: Wake boundary condition at root and tip of blade or wing meshes.

Physical boundaries such as walls are still treated with ghost points however the ghost points do not have meaningful coordinates. For an inviscid wall, the velocities of ghost points are mirrored across the wall boundary, as shown in Fig. 2.10. For a viscous wall, the velocities in ghost points are reversed to enforce a no-slip condition, as shown in Fig. 2.11. Values of density and energy are prescribed such that the wall boundary is adiabatic.

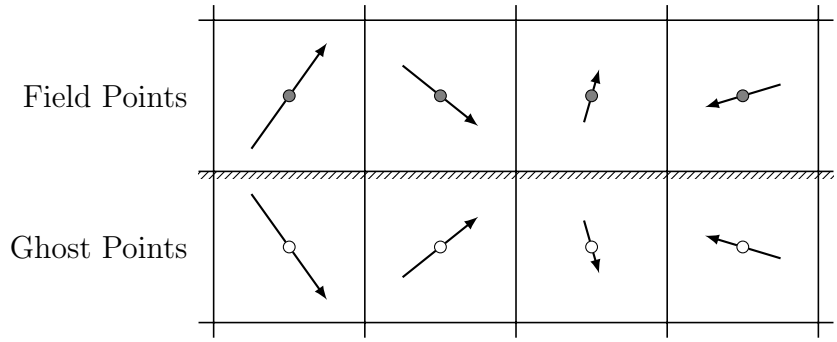



Figure 2.10: Inviscid wall boundary condition: mirrored velocities.

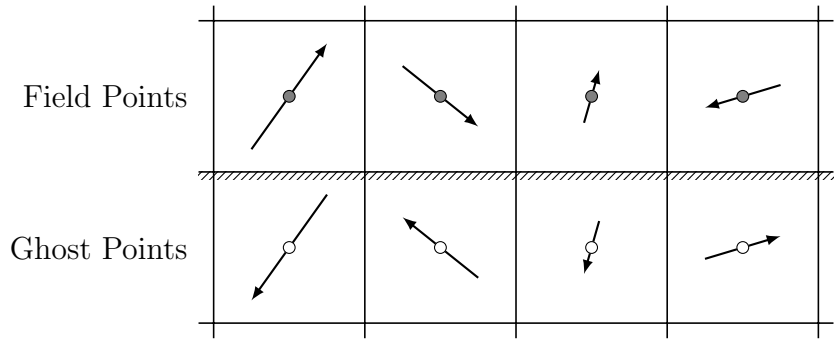



Figure 2.11: Viscous wall boundary condition: negative velocities.

## 2.3 Algorithm Implementation

### 2.3.1 Numerical Implementation on the GPU

The flow solver and other computationally expensive routines in Garfield are all implemented on the GPU using data stored in GPU memory. Data transfer between the CPU and GPU is generally limited by a PCIe bus, where transfer rates are 32GB/s. In contrast, memory bandwidth on the GPU is an order of magnitude higher, up to 900 GB/s for the Tesla V100 device. A core design strategy for a good GPU code is therefore to keep all data on the GPU and reduce data transfers to the CPU as much as possible.

Each routine in Garfield is parallelized using CUDA. Once the grids and domain are initialized, function calls launch CUDA kernels, which are handled in parallel by the GPUs many "stream multiprocessors." A flowchart of the main flow

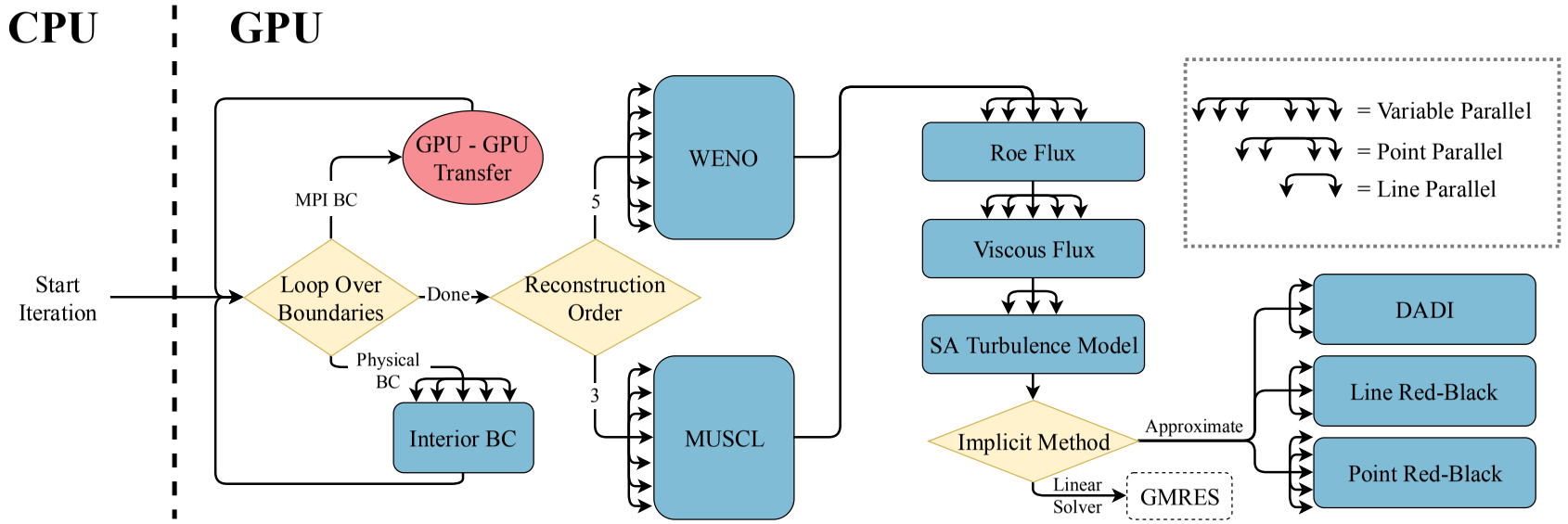



Figure 2.12: Flowchart showing GARFIELD flow solver routines. Different levels of parallelism (point, variable, and line) are represented using different numbers of arrows.

solver routines in GARFIELD is shown in Fig. 2.12. For flux and boundary routines, a kernel is launched for each cell. For the $j$ direction flux, an illustration of kernel $(j, k, l)$ is shown in Fig. 2.13, where the specified kernel is responsible for the flux on the highlighted face. The gray cells show the locations of possibly concurrent kernels. Using one kernel per cell is labeled as "point parallel" in Fig. 2.12. The reconstruction routines (WENO or MUSCL), from Sec. 2.2.2, are slightly different in that a kernel is launched per direction, per face, per variable, since the mathematics in preparation for the left and right states of the Riemann solver are the same for all five flow variables. The additional parallelism is labeled as "variable parallel" in Fig. 2.12.

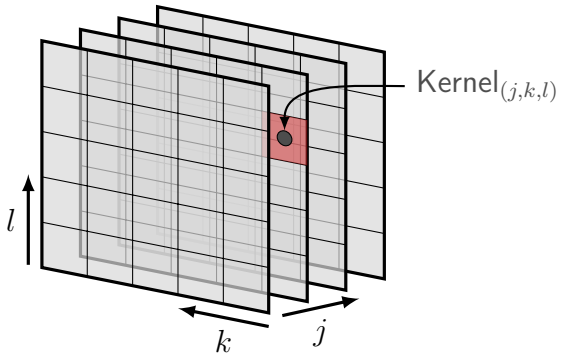



Figure 2.13: One CUDA Kernel used per $(j, k, l)$ cell ("Point-Parallel").

On the left-hand-side of the discretized linear system, Eq. (2.27), the implicit routines of the DADI or line Gauss–Seidel algorithms are not fully parallelizeable. For DADI, inversion of eigenvectors and preparation of eigenvalues for the tri-diagonal system is parallel, and therefore one kernel is launched per cell. The inversion of the tri-diagonal system, however, is done with the Thomas or Sherman–Morrison algorithm and is inherently serial. To maximize parallelism on the GPU, a kernel is launched for each of the five eigenvalues in each cell of a $k$-$l$ plane for the $j$-direction tri-diagonal system. The $(\Lambda_i, k, l)^{\text{th}}$ kernel, shown in Fig. 2.14 then loops over a $j$ line of the domain. The same procedure is subsequently used for $k$ and $l$ directions. In the flowchart in Fig. 2.12, the kernels for routines involving tri-diagonal systems

are labeled as "line-parallel." For the line Gauss–Seidel method, the full block-tri-diagonal system is solved with one thread per line, since there are no factored eigenvalues.

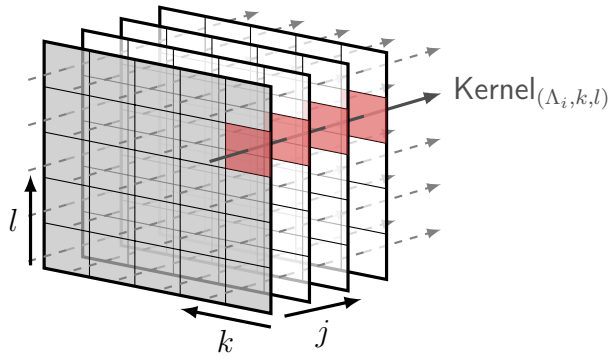



Figure 2.14: In J-direction, one CUDA Kernel used per cell eigenvalue in a $k-l$ plane.

A kernel is a function that runs on the GPU in parallel in a multi-dimensional grid of threads and blocks. For example, 16 kernels could be launched with the use of 2 blocks and 8 threads-per-block. In general, the threads and blocks are three-dimensional vectors, which conveniently map to the structured 3D computational CFD domain. The optimal number of threads-per-block is hardware specific, though in Garfield the number 128 has been found to work well with modern GPUs. The 128 threads can be organized, for example, using 3D indexing as 16 in the $j$-direction, 8 in the $k$-direction, and 2 in the $l$-direction. Then the number blocks required to run kernels over the full set of $jtot \times ktot \times ltot$ points is found such that each point can be assigned 1 thread. For example, calling the viscous flux routine could be done as shown in Listing 2.1.

Listing 2.1: Use of 3D blocks and threads mapping the 3D computational domain.

```
1 dim3 blocks, thr(16,8,2);
2 blocks.x = (jtot-1)/thr.x+1;
3 blocks.y = (ktot-1)/thr.y+1;
```

```
blocks.z = (ltot-1)/thr.z+1;
viscous_flux<<<blocks,thr>>>(...);
```

Inside the viscous flux kernel, the function can determine which index that kernel is responsible for based on thread and block index variables, as shown in Listing 2.2.

Listing 2.2: Each kernel determines which point it is in charge of using block and thread indices.

```
__global__ void viscous_flux(...){
  int j = blockDim.x * blockIdx.x + threadIdx.x;
  int k = blockDim.y * blockIdx.y + threadIdx.y;
  int l = blockDim.z * blockIdx.z + threadIdx.z;
  // ...
}
```

Similar to the CPU, the GPU has a memory/cache hierarchy. There is *global memory*, which is the most commonly used form on the GPU, although it is relatively slow to access. In the kernels there is a *register* cache, which is not large but has the fastest read/write time. In between is *shared* memory, which operates as a manually managed cache shared by all threads on a single block. For stencil-type operations where neighboring values are required, using the shared memory cache significantly improves performance by reducing global memory operations. As an example, Fig. 2.15 shows a 2D version of the viscous flux stencil from Fig. 2.4. The flux face of interest, colored in red, requires neighboring cell info which may not exist within the same thread block. The $(4 \times 4)$ threads perform $(6 \times 6)$ loads into shared memory; once the shared memory is loaded it can be used by all threads repeatedly and very efficiently.

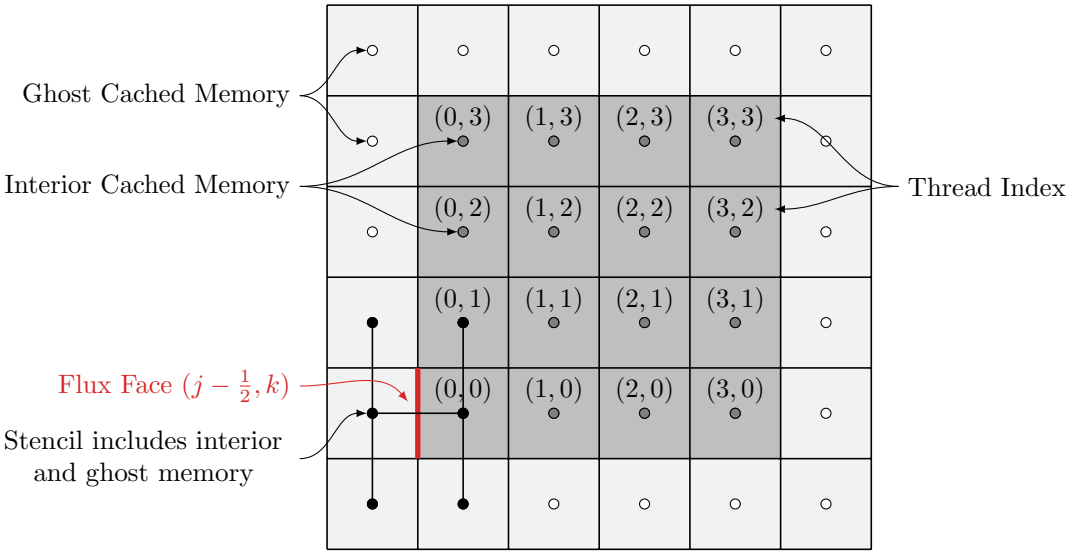



Figure 2.15: Set of $(4 \times 4)$ kernel threads using $(6 \times 6)$ shared memory cache for efficient application of a stencil.

### 2.3.2 Framework Wrapping

Framework details are presented in Sec. 2.7; for GARFIELD to work in the Python framework, GARFIELD needs to have functions accessible to Python. As primarily a C++ code, GARFIELD uses the native Python-C API to create a Python-readable binary. This approach requires more verbosity in programming compared to alternatives like SWIG or BoostPython but the primary advantage is full compatibility on all compute clusters.

Core functions accessible from the Python framework with a brief description are summarized in Table 2.1. The last four, `mvp`, `precondition`, `vdp`, and `getrhs`, are for use with GMRES, described in Sec. 2.8. A more detailed explanation and definition of *iBlanking* is included in Sec. 2.6.

Routines which either return vectors or are passed vectors, do so using pointers. Vectors of CPU data are wrapped using the popular NumPy numerical python library. Vectors of GPU data can also be passed with the use of Python Capsules. Capsules are meant specifically for data passed between two C-like codes. So although the

data cannot be accessed directly from Python, when passed to another module like for grid motion or connectivity, those modules can access the pointers inside.

| Function Name | Description |
|---|---|
| `startStep` | Called at the start of a physical timestep to save previous solutions (for finite-differences in time). |
| `updateIblank` | Indicate to Garfield the map of overset interpolation nodes has been updated. |
| `updateGrid` | Indicate to Garfield the grid nodes have moved, indicating the need to calculate grid velocity. |
| `getUnstructData` | Obtain a Python dictionary storing pointers to the grid, solution, and other pertinent information required for overset connectivity. |
| `writeSolution` | Write a solution file (CGNS binary format). |
| `computeForces` | Compute and return integrated forces on walls in a specified direction. |
| `spanLoads` | Compute and return a span-wise distribution of loads, typically for use in a structural dynamics code. |
| `runSubSteps` | Take a number of non-linear sub-iterations (usually 1). |
| `mvp` | Perform a Matrix-Vector-Product (MVP) operation on the provided vector and the left-hand-side matrix of the linear system, Eq. (2.40). |
| `precondition` | Perform a precondition operation on the provided vector, either with DADI or the specified flavor of Red-Black Gauss–Seidel methods. |
| `vdp` | Perform a vector dot product on two vectors. |
| `getrhs` | Return the right-hand-side of Eq. (2.40). |

Table 2.1: Python-visible functions in Garfield.

## 2.4 Other CFD Solvers

A multi-solver CFD framework is not complete without at least a second solver to run in conjunction with Garfield. Three other codes are used in this framework, and each is a RANS solvers with the same governing equations presented for Garfield. This section will briefly review each solver, without going into implementation details.

### 2.4.1 OverTURNS

The Overset Transonic Unsteady Rotor Navier–Stokes (OverTURNS) solver has been used for the past few decades for rotorcraft simulation. From the 1990s [30, 30] an early version of TURNS (without oversetting) was used for a free-wake model coupled to an Euler/Navier–Stokes code. More recently the solver has been used extensively for the overset analysis of a hovering S-76 rotor [61] and micro air vehicle applications [62].

OverTURNS is a structured, finite-difference code with a very similar discretization to that used in Garfield, described in Sec. 2.2. The main difference is that OverTURNS runs on the CPU and stores the solution at the grid nodes, not at the cell centers. OverTURNS still uses a hybrid finite-difference, finite-volume approach; the control volume is constructed around each grid node.

OverTURNS has all overset functionality built into the FORTRAN code, however it has also been wrapped in Python (using F2PY) for use in an overset framework. The same Python functions from Table 2.1 are accessible for OverTURNS.

### 2.4.2 HamStr

The Hamiltonian Strand (HamStr) solver is another in-house development effort at the University of Maryland [63]. The finite-volume code for unstructured, hexahedral meshes uses Hamiltonian paths through each cell to make use of line-based reconstruction and preconditioning techniques. Triangular surface meshes are transformed into all-quad meshes through sub-division, as shown in Fig. 2.16. The opposite faces of each quad are connected using a line which can be used for reconstruction (ie. WENO or MUSCL), and line-based preconditioners (ie. Line Gauss–Seidel or DADI). The surface-normal direction is generated using a strand-approach, where the mesh is marched outward from the surface of a body. The

strand-direction also forms a line used for spatial and implicit schemes.

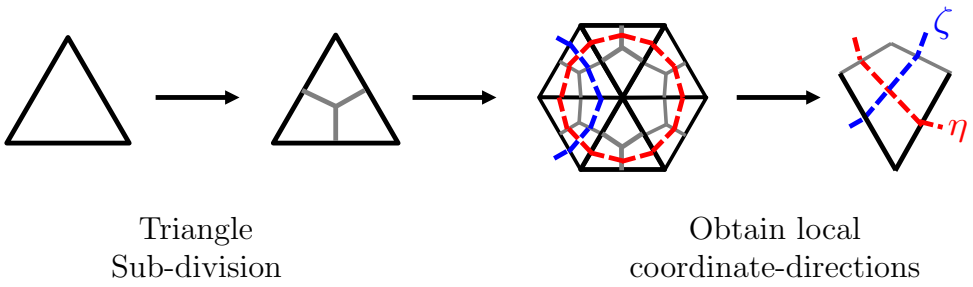



Figure 2.16: General procedure in 2D for obtaining Hamiltonian paths used as local coordinate directions.

HamStr is run entirely on the CPU and is wrapped for use with Python using SWIG. All the functions from Table 2.1 are made accessible to Python for HamStr.

### 2.4.3 mStrand

mStrand is a multi-strand solver [48] present in the HPCMP CREATE$^{TM}$-AV Helios framework, developed at the Army Aviation Development Directorate. Access to mStrand was solely through a pre-compiled library on Department of Defense Supercomputing Resource Centers. Because Helios is a Python framework, routines in mStrand are accessible from Python. All of the functionality listed in Table 2.1 is available in mStrand and an additional Python layer was added to properly map function names and parameters.

As a strand solver, mStrand uses unstructured grids in the wall-surface direction which are marched outward in the strand direction. A unique feature of mStrand is the ability to assign multiple nodes at the same coincident point on sharp convex corners. The use of multiple strands emanating from the same node helps with grid resolution and cell quality for complex geometries.

### 2.4.4 Overflow

NASA's Overflow solver [35] is used in this work as a trusted and well optimized CPU code for comparison against Garfield. Both Overflow and Garfield are structured, implicit, compressible, Reynolds-Averaged Navier–Stokes solvers and modifications to Overflow have been made to ensure consistent implicit algorithms are used between both codes. Engineering results from Overflow serve as a trusted baseline for accuracy comparisons. Timing results from Overflow serve as a baseline for estimating performance speedup provided by the GPU.

The primary algorithmic differences between Garfield and Overflow is the location of the solution in the grid. Garfield stores the solution at cell centers and uses a ghost-point strategy to apply boundary conditions. Overflow stores solutions at grid nodes and can directly apply a restriction to points on a physical boundary. The main implications of cell vs node-centered differences apply to the wake boundary at the tip of a blade or wing mesh. At a wake boundary, the structured mesh folds on itself, as previously shown in Fig. 2.9. With a cell-centered approach, Garfield prescribes ghost cells across the fold and while cells might be skewed, the full stencil is maintained. With a vertex-centered approach, Overflow drops to a lower order model and performs averaging of points above and below the boundary.

All explicit routines for flux computation of the mean-flow equations are equivalent between Overflow and Garfield. The turbulence model convection, dissipation, and source terms are also equivalent between solvers with the exception with the calculation of vorticity. Overflow uses a finite-difference method and grid metrics to compute vorticity whereas Garfield uses a linear-least-squares algorithm. The finite-difference approach is very efficient and works well for good quality meshes however the linear-least-squares method is found to improve accuracy

in regions of highly skewed cells.

OVERFLOW version 2.2*l* is used, which does not have exactly the same DADI algorithm as implemented in GARFIELD. Instead, OVERFLOW has two methods which can be easily combined to get the DADI method. The first is the Diagonally Dominant Diagonalized Alternating Direction Implicit (D3ADI) [64], which results in an upwinded, scalar tri-diagonal in the three coordinate directions. The second is a three-factor Diagonalized ADI method but with central dissipation, resulting in a penta-diagonal system. The upwinded, tri-diagonal routines from D3ADI are used but without the diagonal dominance along with the eigenvector inversion routines from the penta-diagonal diagonalized ADI scheme.

## 2.5 Grid Motion

Grid motion is implemented as an independent module for an overset CFD framework. The principles of grid motion simplify to transformation matrix multiplications, where the transform corresponds to translation or rotation. For applications with rigid grid motion, all points in a mesh undergo the same transform. With elastic deformation, the deflection at each point in a mesh may be different. In representing the elastic deformation of rotorcraft blades, it is important to accurately represent the motion of the blade surface without causing errors in the mesh. This section describes the mathematical implementation of an algebraic grid motion and elastic deformation module.

### 2.5.1 Transformation Matrices

Given a point in space with coordinate $(x_0, y_0, z_0)$, the point can be translated by $(\Delta x, \Delta y, \Delta z)$ with the transformation:

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & \Delta x \\ 0 & 0 & 0 & \Delta y \\ 0 & 0 & 0 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.59}$$

Such that the new point has coordinates

$$[x, y, z, 1] = \mathbf{T} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \tag{2.60}$$

Similarly, rotation matrices can be defined which describe rotation of a point around the $x$-axis by $\theta_x$, $y$-axis by $\theta_y$, and $z$-axis by $\theta_z$:

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) & 0 \\ 0 & sin(\theta_x) & cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_y = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{R}_z = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.61}$$

Combining translation and rotation operations using multiplication, the transformation matrix for rotation by $\theta_x, \theta_y, \theta_z$ about point $(x_0, y_0, z_0)$ and translated by $(\Delta x, \Delta y, \Delta z)$ is

$$\mathbf{M} = \mathbf{T_\Delta T_0 R_z R_y R_x T_0}^{-1} \tag{2.62}$$

where $\mathbf{T}_0$ is translation by $(x_0, y_0, z_0)$ and $\mathbf{T}_\Delta$ is translation by $(\Delta x, \Delta y, \Delta z)$. The sequence, from right to left, is to translate such that the origin is at the point of interest, rotate about $x$, $y$, and $z$, translate back, and perform the final $\Delta$ translation. The multiplication of these matrices can be done analytically.

### 2.5.2 Elastic Deflection

Elastic deflection information typically comes from a Computational Structural Dynamics (CSD) code. For rotorcraft blade deflection from 1D beam analysis, it is common to obtain deflections about points defined in a line along the blade. An example for a swept-tip rotor blade is shown in Fig. 2.17. The line where discrete deflections are defined may shift based on the blade geometry. The general methodology is to find the nearest point on the deflection line to each CFD mesh point and deform the CFD point by applying the interpolated deflection. If the line of deflection points has any curves or kinks, however, there may be a discontinuity in the deflections applied to certain locations of the CFD mesh. In Fig. 2.17, two neighboring CFD points may get very different interpolated deflection values.

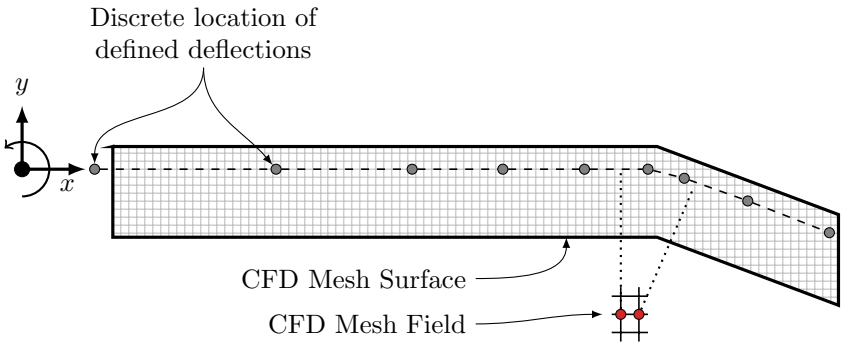



Figure 2.17: Control points along blade where deflections are defined. Distances to CFD coordinates may be discontinuous.

To avoid deflection discontinuities in the CFD domain, the deflection points are

projected onto a straight line, using the distance from the first deflection point as the parameterizing variable. The modified formulation is no longer exact on the blade surface, however assuming any blade sweep occurs near the blade tip, angles and corresponding error are both small. Once the closest point on the projected line is determined, linear interpolation between control points is used to find the deflection. Deflections are still performed about the original defined deflections points, not the projected approximation. An important detail in the implementation of CFD motion is that no assumptions are made about the topology of the CFD mesh. The procedure is the same for curvilinear, Cartesian, or unstructured meshes.

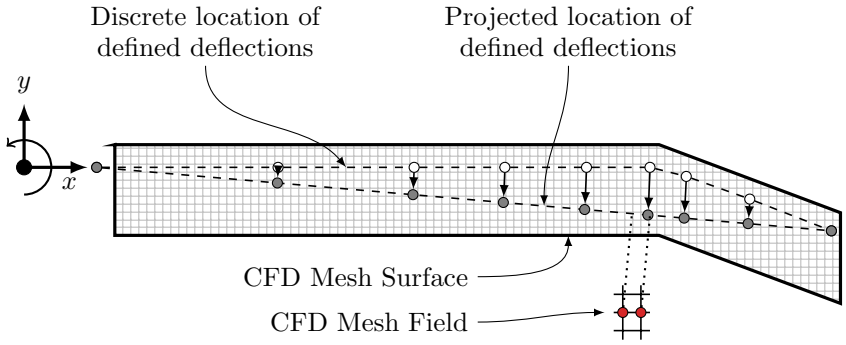



Figure 2.18: Use of projected control points results in continuous distances to CFD coordinates.

Six quantities are provided at each discrete deflection point: $\Delta x$, $\Delta y$, $\Delta z$, $\theta_x$, $\theta_y$, $\theta_z$. The deflections are performed about the coordinate of the specified deflection point in the undeformed frame. At each step, the blade must therefore be transformed back to its original position (along the $x$-axis) before deflections can be applied. The procedure for deforming a point in a blade mesh and rotating it about the $z$-axis is:

$$\mathbf{M}_n = \mathbf{R}_z \mathbf{D}_n \tag{2.63}$$

$$\begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix} = \mathbf{M}_n \mathbf{M}_{n-1}^{-1} \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ z_{n-1} \\ 1 \end{bmatrix} \tag{2.64}$$

where $\mathbf{M}_n$ is the forward transformation from the undeformed frame (with the blade along the $x$-axis) consisting of a deflection $\mathbf{D}$ and rotation $\mathbf{R}_z$. $\mathbf{D}$ changes for every grid point and therefore $\mathbf{M}$ is also different at each point. The inverse transform $\mathbf{M}_{n-1}^{-1}$ un-does the movement from the previous step.

Rotation is assumed about the $z$-axis however rotation about an arbitrary axis is possible with the use of a frame transform. The frame transform is another matrix defining another rotation and translation, allowing rotors to be place arbitrarily in space. With the frame transform, the effective transformation matrix $\mathbf{M}$ is

$$\mathbf{M}_n = \mathbf{F}_r \mathbf{R}_z \mathbf{D}_n \tag{2.65}$$

where $\mathbf{F}_r$ is the rotor frame transformation.

### 2.5.3 Grid Velocity

The grid motion module defines the location of grids at discrete times and this information must be translated into grid velocity within each CFD solver. For rigid grid motion, the shapes and volumes of each cell remains constant and therefore finite-difference relations can be used to obtain a grid velocity:

$$\frac{\partial x}{\partial t} = \frac{x_n - x_{n-1}}{\Delta t} \tag{2.66}$$

where $n$ is the current timestep and $n-1$ is the previous timestep. A second order backwards differencing procedure can also be used. When cells are skewed

or volumes change during elastic grid motion, however, simple finite-differences do not conserve mass, momentum, and energy in the governing equations. Instead, a Geometric Conservation Law (GCL) interpretation of grid motion should be used. For finite-volume methods, the GCL implementation is done by finding the velocity of the cell faces, instead of the cell centers. Since the face velocities appear in the discretization of fluxes, any changes in cell volume from grid motion are accounted for as a flux. A common method of finding the face velocity is to find the volume swept by the face over time $\Delta t$, and to divide that volume by the face area multiplied by $\Delta t$. Different treatments of CGL conditions are described in the work of Blazek [59].

### 2.5.4 Framework Wrapping

The motion module is written in C++ and wrapped for use with Python using the native Python-C API. This is the same approach taken for Garfield, presented in Sec. 2.3.2. Core functions accessible from the Python framework with a brief description are summarized in Table2.2.

| **Function Name** | **Description** |
|---|---|
| `register_data` | Pass pointers of grid coordinates from CFD. |
| `step` | Step the grid to the next location in space. |
| `write_forces` | Write sectional blade forces for use with CFD-CSD coupling applications. |

Table 2.2: Python-visible functions in the motion module.

## 2.6 Domain Connectivity

The open source Topology Independent Overset Grid Assembler (Tioga) is used to connect and interpolate between overset domains. Tioga has been used extensively by other research groups [45, 65] for both finite-volume and high-order finite-element cases. For all overset cases presented in this work Tioga performs two tasks:

1. Connect grids

    - Classify points as *hole*,*fringe*, or *field*.
    - For fringe points, find surrounding donor points in a neighboring mesh and compute interpolation weights.

2. Exchange Data

    - Given a vector, interpolate the data at each fringe location and send the values to the corresponding mesh.

Classification of points, often called "hole-cutting", is done using an integer *iBlank* array. A *hole* point is denoted as a point in one mesh that resides within a solid-wall boundary of another mesh, tagged using *iBlank*=0. A *fringe* point is denoted as a point that receives an interpolated value in another mesh, tagged using *iBlank*=-1. A *fringe* point is either near the boundary of a mesh or in a region where another mesh has finer resolution. A *field* point is a regular point where the solution is solved on that mesh, tagged using *iBlank*=1.

### 2.6.1 Connectivity and Mesh Grouping

TIOGA is topology agnostic and works with structured or unstructured grids. Given a coordinate that is a desired interpolation location, the search for donors is performed by sorting the grid coordinates in an Alternating Digital Tree (ADT). Once the ADT is created, the search time is fast. Creating the search tree is expensive, however, and must be done every time a grid moves.

Grids that are not moving should not be reconnected every timestep, since this would be a waste of computation. TIOGA does not distinguish between moving or non-moving grids but the distinction can be made at the framework level by grouping the grids. Fig. 2.19 shows an example with two rotating blades and two background

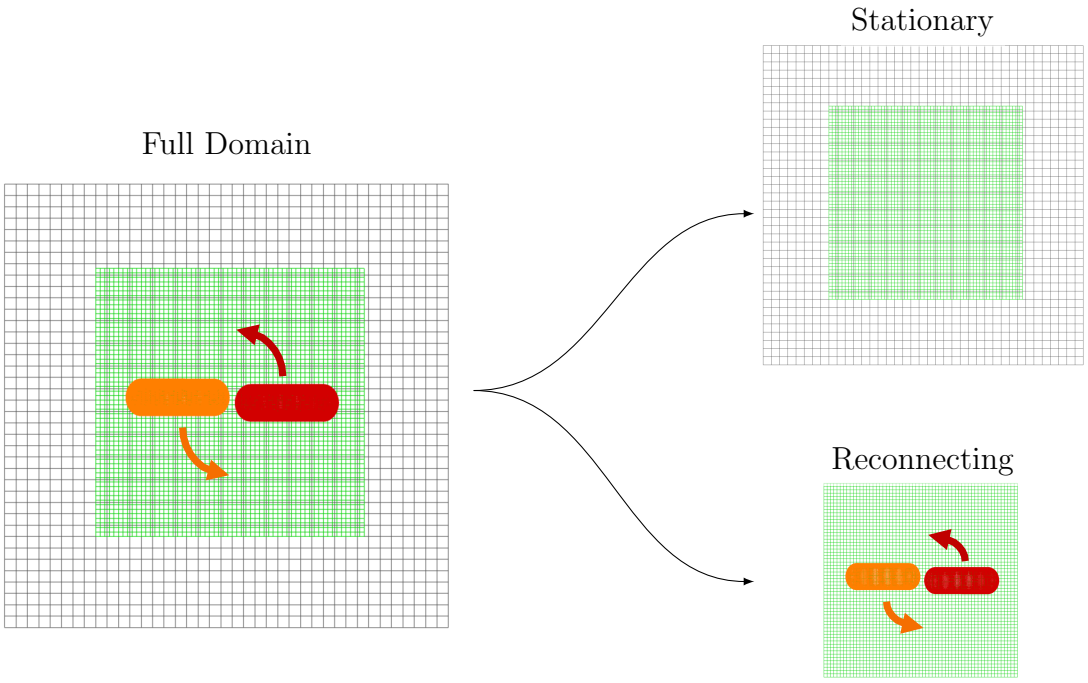



Figure 2.19: Grouping of stationary meshes and reconnecting separately so that only the "reconnecting" group reconnects every timestep.

grids. The two blades and finest nested mesh need to be reconnected every timestep, and are categorized as "reconnecting". The two background meshes only need to be connected once at the beginning of the simulation. The finer nested mesh in green appears in both groups and will be assigned two *iBlanking* values. The two maps can be combined into a single *iBlanking* map with the simple operator shown in Eq. (2.67).

$$iB = \min(iB_{\text{reconnecting}}, iB_{\text{stationary}}) \tag{2.67}$$

Garfield is a cell-centered code however Tioga requires grid coordinates and solutions be stored at the same location. To remedy this Garfield constructs a dual-mesh of grid nodes aligned with the cell centers of the original mesh where solutions are stored. For structured grids the dual-mesh is trivial to construct and only has to be done once even for unsteady cases.

### 2.6.2 Interpolation

Once the *iBlank* map is set and all interpolation donors have been assigned, data can be passed for interpolation and exchanged between meshes. TIOGA uses second order transfinite linear interpolation. All interpolated values are then efficiently packaged and sent with MPI to their respective fringe cells. Fig. 2.20 illustrates in 2D the interpolation procedure between a Cartesian background mesh (orange) and a fringe point in a cylindrical mesh. Solid symbols represent field points (*iBlank=1*) and hollow symbols represent fringe points (*iBlank=-1*).

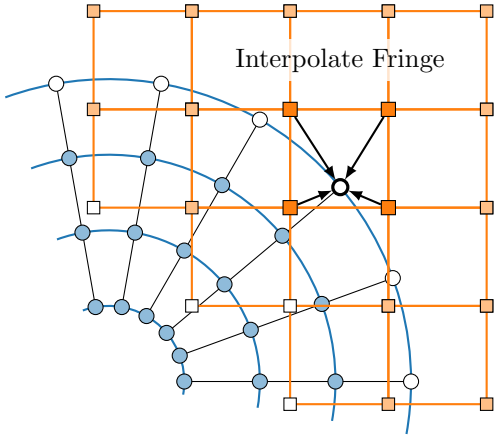
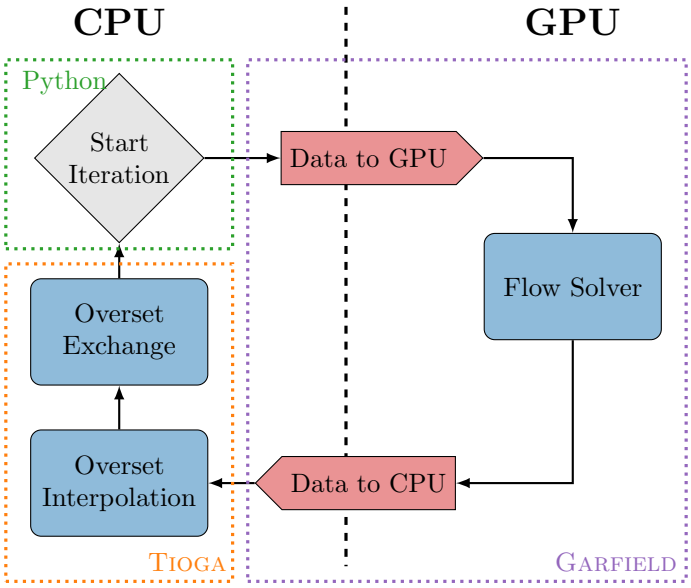


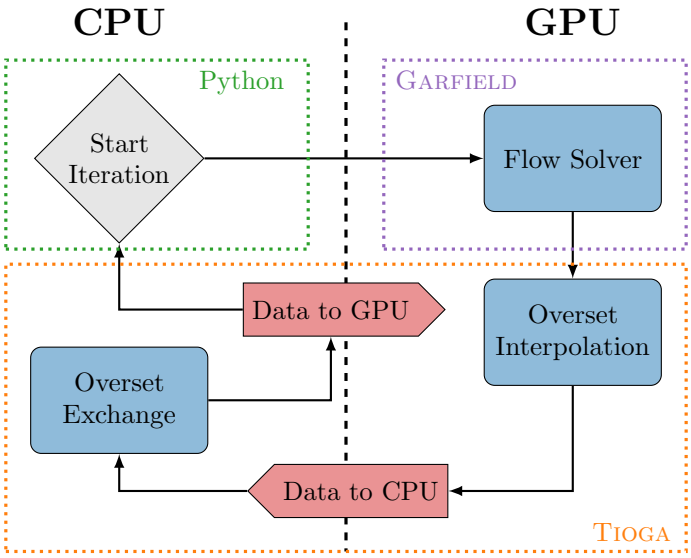

Figure 2.20: Illustration of the interpolation step for two overset grids with fringe points as hollow symbols, field points as solid symbols.

TIOGA is a CPU-based code and for many of the connectivity routines, altering the code to use GPU memory would require significant effort. The interpolation routines, however, are more straightforward. Furthermore while reconnecting grids occurs a maximum of once per timestep, exchanging data between grids can occur dozens of times per timestep for time-accurate cases.

A flowchart of a baseline implementation with all routines on the CPU is shown in Fig. 2.21(a). Because GARFIELD does not have information about which cells are required for interpolation, the entire domain must be transferred back to the CPU. This is very costly, since CPU-GPU data transfer is limited by a relatively low-




(a) Baseline TIOGA




(b) TIOGA w/ Interpolation on GPU

Figure 2.21: Flowchart of routines showing location of data: on the CPU or on the GPU. Performing interpolation on the GPU drastically reduces data transfer to the CPU.

bandwidth hardware bus. Furthermore, interpolation is a highly parallel algorithm but is performed on the CPU, a serial device.

Fig. 2.21(b) shows the updated flowchart for a framework where TIOGA handles GPU data from GARFIELD and performs the interpolation on the GPU. In this case GARFIELD passes only pointers of GPU memory to TIOGA, avoiding transfers

back to the CPU. After interpolation, only a small fraction of the initial data set is transferred back to the CPU and exchanged between MPI tasks. TIOGA does not make use of CUDA-aware MPI (for GPU-GPU transfer) for two reasons. First, since TIOGA is written as a CPU code, the MPI packing and transfer routines are not written for the GPU and would take significant effort to write in CUDA. Second, not all clusters used for this work support GPU-GPU communication and the data therefore still travels through CPU memory buffers.

### 2.6.3 Framework Wrapping

TIOGA is primarily written in C++ and wrapped for use with Python using the native Python-C API. This is the same approach taken for GARFIELD, presented in Sec. 2.3.2. Core functions accessible from the Python framework with a brief description are summarized in Table 2.3.

| **Function Name** | **Description** |
|---|---|
| `register_data` | Pass pointers of grid data from CFD. |
| `connect` | Regenerate *iBlank* map and search for donors for each fringe point. |
| `update` | Given a vector of data, interpolate the quantities, send to the relevant mesh, and update the relevant fringes. |

Table 2.3: Python-visible functions for TIOGA.

## 2.7 Framework Description

The CFD solvers, grid motion module, and domain connectivity all fit together in a single framework, as illustrated in Fig. 2.22. Python is a convenient scripting language used to tie together each of the framework components. Python is very well supported across the majority of clusters as a flexible and user-friendly programming interface.

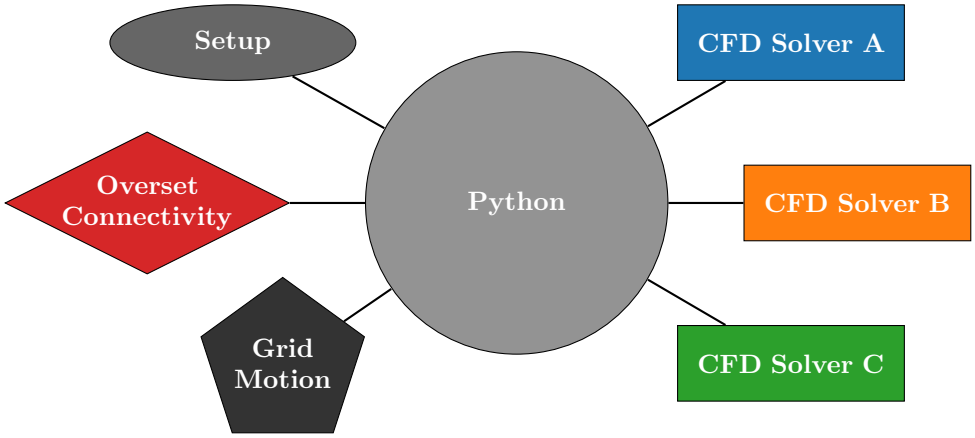



Figure 2.22: Modules for CFD framework.

Data can be shared between modules using Python dictionaries. The `gridData` dictionary from a CFD solver, for example, includes all information about the grid coordinates and topology. A few of the lines are shown in Listing 2.3. Note that for CPU data, Python has convenient access to data for debugging.

Listing 2.3: Example dictionary of grid data shared between modules of the framework.

```
gridData = {
 'grid-coordinates' = array([0.1, 2.3, ..., 3.4]),
 'hexaConn'         = array([0,1,...,123]),
 'iblanking'        = array([1,1,...,1])
 # ...
}
```

### 2.7.1 Load Balancing

One of the main tasks performed by the framework is to distribute cluster resources between solvers. A feasible cluster layout with 2 GPUs and 16 CPU cores

per node is shown in Fig. 2.23. The framework is started with an allotted number of MPI tasks. If all solvers are CPU-based, the simplest solution is to run each solver on all tasks. This approach, previously presented in Sec. 1.2.1 describing the procedure from the Helios framework, is shown in Fig. 2.24.

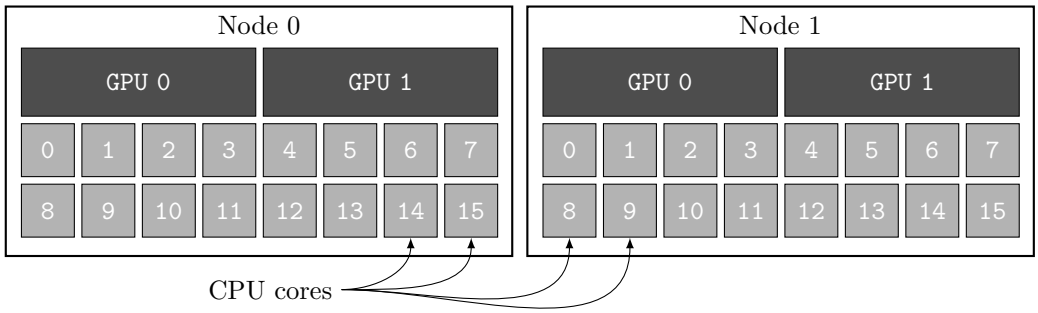



Figure 2.23: Cluster node topology with 2 GPUs and 16 CPU cores per node.

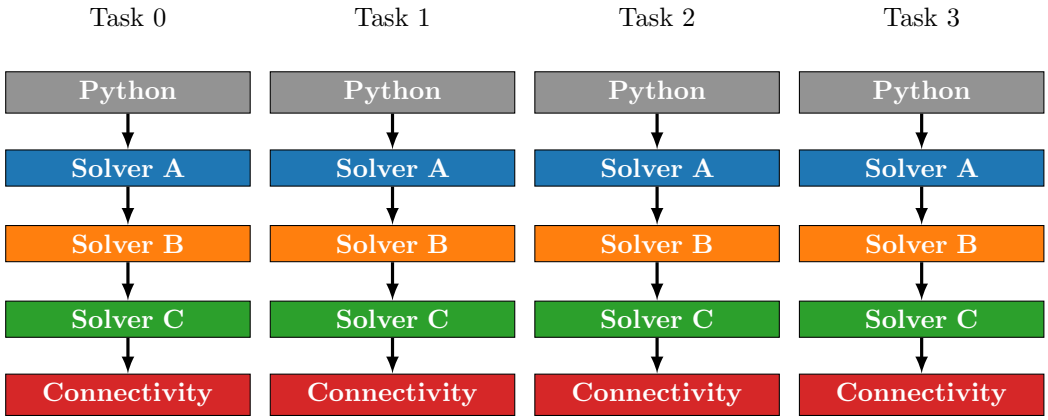



Figure 2.24: Example distribution of tasks suitable for a CPU-only CFD framework. Each solver uses all available MPI tasks.

When CPU and GPU solvers are used together, it is better to separate solvers by task based on the available hardware resources. Fig. 2.25 shows an example using three solvers distributed between four MPI tasks. Solver A is split between two tasks, whereas Solver B and C each have 1 task. Typically dozens or hundreds of tasks may be launched depending on the size of the simulation, meaning no single solver would typically be assigned only a single core.

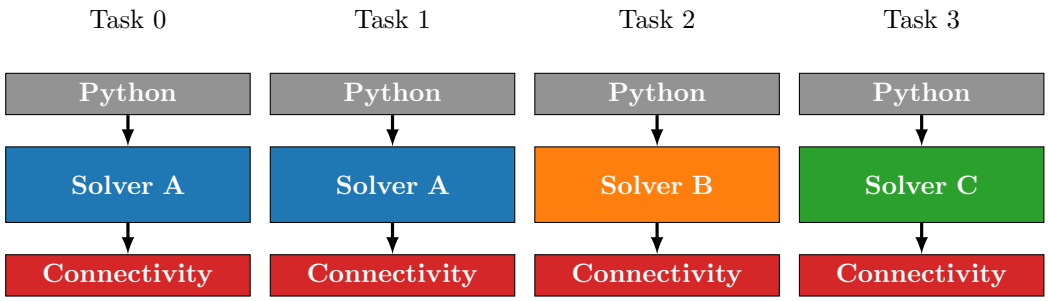



Figure 2.25: Example distribution of tasks suitable for a CPU-GPU framework. Different tasks assigned different solvers.

The example node topology from Fig. 2.23 can be used to illustrate a scenario with three solvers, where Solver A is GPU-accelerated and the other two use only CPU cores. A GPU code, like GARFIELD, is assigned 6 cores: 4 on the first node sharing 2 GPUs and 2 on the second node sharing 2 GPUs. If the other two solvers are CPU-based, the rest of the cores are distributed sequentially. The distribution of cores for GPU solvers is based on the number of GPUs. In the example, there are 2 GPUs per node so the framework loops over all nodes, assigning 2 tasks per node until the total number of tasks is reached. This is why Node 0 has 4 GPU tasks running and Node 1 has 2.

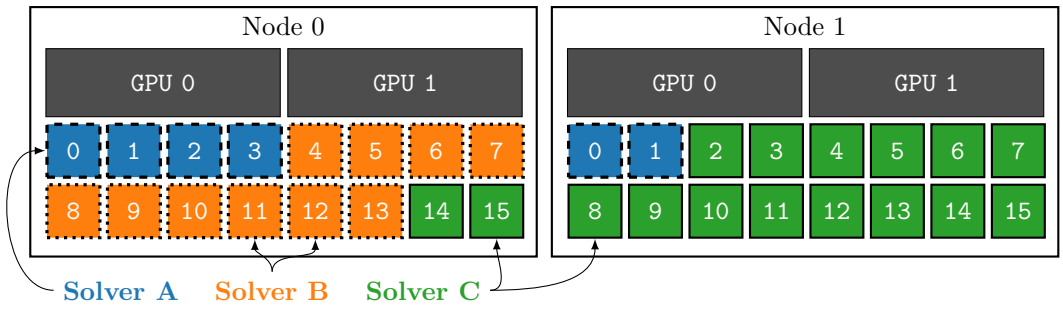



Figure 2.26: Example distribution of resources for a 3-Solver framework where Solver A is GPU-accelerated.

## 2.8 GMRES and Overset GMRES

Once a set of PDEs are discretized and linearized, as in Eq. (2.40), the resulting system of linear algebraic equations can be represented as a general linear system

of form $Ax = b$. The Generalized Minimal Residual (GMRES) algorithm is one of many different methods that can be used to solve such a system. Compared to more traditional, iterative methods like Gauss–Seidel or approximate factorization, GMRES is a minimization algorithm on the quantity $||b - Ax||$. This minimization is accomplished with the use of $m$ Krylov-subspace vectors $V_m$, where each vector is the size of the solution vector $x$. A more detailed overview of GMRES is presented in detail in the work of Yousef Saad [66]. For brevity this section will focus on applications of GMRES to CFD.

GMRES with $m$ vectors applied to the Navier–Stokes equations to approximate the change in the solution $\mathbf{\Delta Q}$ will require $m \times sizeof(\mathbf{\Delta Q})$ in memory. High memory usage is one of the disadvantages of GMRES compared to Gauss–Seidel or approximate factorization methods. However, in the era of exa-scale computing, domains are split onto enough processors that memory often no longer becomes a primary bottleneck.

Algorithm 1: Right-Preconditioned GMRES Algorithm.

1: Compute $r_o = b - Ax_o$, $\beta := ||r_o||_2$, and $v_1 := r_o/\beta$
2: **for** $j = 1, 2, ..., m$ **do**
3: Compute $r_j :=$ PRECONDITION$(v_j)$ ▷ Preconditioner
4: Compute $w_j := Ar_j$ ▷ Matrix-Vector Product
5: **for** $i = 1, ..., j$ **do**
6: $h_{ij} := (w_j, v_i)$ ▷ Vector Dot Product
7: $w_j := w_j - h_{ij}v_i$
8: **end for**
9: $h_{j+1,j} = ||w_j||_2$
10: **if** $h_{j+1,j} == 0$ **then**
11: $m := j$
12: **goto** 18
13: **end if**
14: $v_{j+1} = w_j/h_{j+1,j}$
15: **end for**

Linear Solver (Krylov) Iterations

16: Define the $(m + 1) \times (m)$ Hessenberg matrix $\tilde{H} = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$ and $V_m = [v_1 ... v_m]$
17: Compute $y_m$, the minimum of $||\beta e_1 - \tilde{H}_m y||_2$ and $r_m = V_m y_m$
18: Compute $x_m = x_o +$ PRECONDITION$(r_m)$

The GMRES algorithm with right-preconditioning is shown in Alg. 1. Preconditioning is the process of altering the system of equations to a different system that has the same solution but is less stiff and hence easier to solve. Preconditioning is required to obtain good performance from the GMRES algorithm and generally right preconditioning is preferred over left because it preserves the magnitude of the residual within linear iterations. In Alg. 1, semantically $r$ is the result of preconditioning the vector $v_j$. Mathematically the preconditioner is often expressed as a matrix $\mathbf{M}$, which approximates the matrix $\mathbf{A}$ but is easily invertible such that for the system $\mathbf{M}r = v_j$, $r = \mathbf{M}^{-1}v_j$.

As recommended by Saad [66], the Hessenberg matrix from Alg. 1 is made upper-triangular with Givens rotations in each Krylov iteration, resulting in an easily solvable minimization problem and approximation of the residual in each linear solver iteration. For time-accurate simulations the residual can therefore be tracked for each temporal step, non-linear sub-iteration, and linear solver iteration (Krylov iteration).

The three most expensive steps from Alg. 1 are highlighted: the preconditioner, matrix-vector product, and vector dot product. This section uses a 1-dimensional heat-equation problem to describe these steps for a simple overset system. Consider the heat (Poisson) equation between two walls where the left wall is kept at temperature $T_L$ and the right wall is kept at temperature $T_R$ with a thermal diffusivity constant of 1. A discretization on two overset meshes using a total of 8 points with $\Delta x = 1$ is shown in Fig. 2.27.

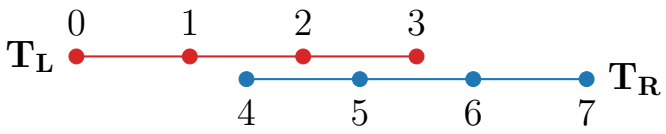



Figure 2.27: 1D heat problem between left and right walls at temperatures $T_L$ and $T_R$ respectively.

The discrete linear system for the full overset system is shown in Eq. (2.68) with each element colored by the grid color, red or blue. Points 0 and 7 are boundary points fixed at the exact temperature solution. Points 1,2,5 and 6 are interior points and therefore have the familiar central discretization stencil $[1, -2, 1]$. Point 3 receives an interpolated quantity from points 5 and 6; point 4 receives an interpolated quantity from points 1 and 2. Note the interpolation coefficients (0.5 for simplicity) are on the left-hand side of the linear system and therefore treated implicitly.

$$\begin{bmatrix} 1 & & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & & 1 & -0.5 & -0.5 & & \\ & -0.5 & -0.5 & & 1 & & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & 1 \\ & & & & & & & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} T_L \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ T_R \end{bmatrix} \tag{2.68}$$

The red and blue meshes are coupled by the interpolation weights. The linear systems can be de-coupled with a Schwartz iteration procedure such that the solution at time $n$ is dependent on an approximate solution at time $n-1$. The de-coupled system, shown in Eq. (2.69) is the traditional, Schwarz-alternating, approach that most overset solvers take because it means each grid is treated separately.

$$\begin{bmatrix} 1 & & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & 1 \\ & & & & & & & 1 \end{bmatrix} \begin{bmatrix} x_0^n \\ x_1^n \\ x_2^n \\ x_3^n \\ x_4^n \\ x_5^n \\ x_6^n \\ x_7^n \end{bmatrix} = \begin{bmatrix} T_L \\ 0 \\ 0 \\ 0.5(x_5^{n-1} + x_6^{n-1}) \\ 0.5(x_1^{n-1} + x_2^{n-1}) \\ 0 \\ 0 \\ T_R \end{bmatrix} \tag{2.69}$$

A GMRES algorithm which treats all interpolated boundaries implicitly cannot rely on the decoupled system from Eq. (2.69) and should instead solve the coupled system Eq. (2.68). For convenient tagging of field and fringe points an iBlanking procedure is used to set $iB = 1$ at field points and $iB = -1$ at interpolated fringe points. A simple Gauss–Seidel preconditioner can then be applied in two loops: one over the field points and another over the fringe points. Pseudo-code for a Gauss–Seidel preconditioner applied to the example 8-point problem is shown in Alg. 2.

Algorithm 2: 1D Example: Gauss–Seidel Preconditioning $r = \mathbf{M}^{-1}v$.

**function** PRECONDITION($v$)
  Let $r[0, \ldots, 7] = 0$ be a new array
  **for** $s = 1, \ldots, N_{sweep}$ **do**
    **for** $i = 1, 2, 5, 6$ **do** ▷ All field points
      $r_i := (v_i - r_{i-1} - r_{i+1})/(-2)$
    **end for**
  **end for**
  $r_3 = 0.5r_5 + 0.5r_6$ ▷ Iterpolation at fringe points
  $r_4 = 0.5r_1 + 0.5r_2$
  **return** $r$
**end function**

Algorithm 3: 1D Example: Matrix-Vector Product $w = \mathbf{A}r$.

```
1: function MVP(r)
2:     Let w[0, ..., 7] = 0 be a new array
3:     for i = 1, 2, 5, 6 do                    ▷ All field points
4:         w_i := r_{i-1} − 2r_i + r_{i+1}
5:     end for
6:     return w
7: end function
```

Note the preconditioning of the interior points ignores the fringe points and the preconditioning of fringe points is merely an interpolation of the other preconditioned points. From Alg.1, the preconditioning step is followed by a matrix-vector product. By preconditioning the fringe points as the interpolant of preconditioned field points, the matrix-vector product at the fringe points is guaranteed to be zero as long as no interpolated point is used as a donor in the interpolation of another point. Pseudo-code for a matrix-vector product routine for the 8-point 1D problem is shown if Alg. 3. While the matrix-vector product uses the interpolated values at fringe points, the matrix-vector product at fringe points themselves are zero and need not be filled.

The last expensive step of the GMRES algorithm is the vector dot product. Since the procedure thus far has guaranteed that values in blanked indices are zero, the vector dot product of the full vector $w_j$ from line 4 of Alg.1 is the sum of the vector dot products of each individual mesh. As long as each solver ignores blanked values from vectors in the precondition and matrix-vector product steps, the only influence of coupling between meshes comes from interpolation after preconditioning and conducting a global sum for the vector dot product. The O-GMRES algorithm including communication between meshes is shown in Alg.4. The highlighted lines show changes from the original algorithm.

For the Euler or Navier–Stokes equations the residual, i.e. the RHS of Eq. (2.38) is proportional to the volume of the cell and therefore the Krylov vectors $v_j$ would

also be proportional to volume and cannot be directly compared between meshes. The volume-scaled system of equations Eq 2.40 should instead be used to guarantee that Krylov vectors are compatable between meshes and the converged solution has smooth contours across overset boundaries. The O-GMRES algorithm is written in Python as a partitioned algorithm that does not build the full global solution vector on any processor. Nor does O-GMRES require additional memory over the traditional GMRES algorithm. The only requirement is that each solver expose routines for preconditioning and matrix-vector product. Python handles pointers to large buffers of memory but all numerical operations are handled with function calls to fast, compiled languages. Pseudo-code with the traditional de-coupled GMRES algorithm is shown in Lst. 2.4. The improved implementation is shown in Lst. 2.5, where now the GMRES algorithm is written as part of the framework and the connectivity information is built in to the Krylov iterations.

Algorithm 4: Right-Preconditioned O-GMRES Algorithm.

1: Compute $r_o = b - Ax_o$, $\beta := ||r_o||_2$, and $v_1 := r_o/\beta$
2: **for** $j = 1, 2, ..., m$ **do**
3: Compute $r_j :=$ PRECONDITION$(v_j)$ ▷ Preconditioner
4: OVERSET_INTERPOLATE$(r_j)$
5: Compute $w_j := Ar_j$ ▷ Matrix-Vector Product
6: **for** $i = 1, ..., j$ **do**
7: $h_{ij} := (w_j, v_i)$ ▷ Vector Dot Product
8: OVERSET_SUM$(h_{ij})$
9: $w_j := w_j - h_{ij}v_i$
10: **end for**
11: $h_{j+1,j} = ||w_j||_2$
12: **if** $h_{j+1,j} == 0$ **then**
13: $m := j$
14: **goto** 18
15: **end if**
16: $v_{j+1} = w_j/h_{j+1,j}$
17: **end for**
18: Define the $(m+1) \times (m)$ Hessenberg matrix $\tilde{H} = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$
19: Compute $y_m$, the minimum of $||\beta e_1 - \tilde{H}_m y||_2$ and $r_m = V_m y_m$
20: Compute $x_m = x_o +$ PRECONDITION$(r_m)$
21: OVERSET_INTERPOLATE$(x_m)$

Listing 2.4: Traditional Framework.

```
for t in timesteps:
  for s in sub_iterations:
    connectivity.exchange(q)
    # CFD solver GMRES (frozen fringe points)
    CFDSolver.do_GMRES()
    # ...
```

Listing 2.5: O-GMRES Framework.

```
for t in timesteps:
  for s in sub_iterations:
    # ...
    for j in krylov_iterations:
      # ...
      connectivity.exchange(r_j)
      # ...
```

In all codes where GMRES is used, the matrix-vector product is computed using the approximation

$$\left[\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right]\mathbf{X} \approx \frac{\mathbf{R}(\mathbf{Q}+\epsilon\mathbf{X}) - \mathbf{R}(\mathbf{Q})}{\epsilon} \tag{2.70}$$

where $\epsilon$ is a small scalar chosen based on the magnitude of $\mathbf{Q}$. Eq. (2.70) uses the property of Fréchet derivatives and has been shown to work well for CFD applications [47, 67]. Using the finite-difference approximation introduces error in the calculation of the matrix-vector product however this error is small compared to the error of the first-order linearization between Eq. (2.34) and Eq. (2.40).

# Chapter 3: Verification and Validation

This chapter presents verification and validation of four primary components of the full overset framework: Garfield flow solver, overset connectivity, elastic grid-motion, and the Overset-GMRES algorithm. Garfield in particular, as a novel, GPU-accelerated code, is analyzed for accuracy, algorithm convergence, and performance. The components are analyzed either individually or in sub-sets of the full framework using simple cases.

The first simple application considers an Onera M6 wing used to validate the accuracy of Garfield against results from experiment and the NASA Overflow CFD solver. The Onera M6 case uses two overset grids and also serves as a simple application to verify proper treatment of overset boundaries with the Tioga connectivity code.

The Onera M6 case is then subsequently used in a scalability study to analyze timing, performance, and cost. Again NASA's Overflow solver is used as a baseline for timing comparisons to estimate speedups and cost-savings for parallel GPU and CPU codes.

The third section verifies the proper treatment of elastic grid-motion by looking at a UH-60 rotor in high-speed forward flight. Grid motion must be properly handled in two components of the framework. First, the grid-motion module must accurately and efficiently deform each blade. Second, Garfield must correctly apply the grid-velocities in a conservative way to the governing, discretized equations. NASA's Overflow code is used for comparison of sectional airloads given the same set of

elastic deflections.

The fourth chapter section focuses on single-grid convergence for GARFIELD with the newly implemented implicit algorithms. Convergence with both point- and line-variations of the Red-Black Gauss–Seidel method are compared to the original DADI method for an inviscid wing. The GMRES method with DADI as preconditioner in GARFIELD is compared to DADI alone for a laminar cylinder to verify improved convergence characteristics.

The final verification and validation section continues to analyze convergence but from the framework level for overset cases. The developed Overset-GMRES (partitioned) algorithm implementation is verified against a single-processor implementation for the 2D Poisson equations. Single-solver inviscid flow over a wedge is also used for convergence analysis with and without the implicit treatment of interpolation boundaries.

## 3.1 Onera-M6 Case

The Onera-M6 wing experiments of Schmitt et. al. from 1979 [68] provide a popular validation data set for CFD. The swept aircraft wing was mounted in a wind tunnel and run at transonic conditions with a Mach number of $M_\infty = 0.8395$ and angle of attack $\alpha = 3.06°$. The Reynolds number of the case is $Re = 11.72 \times 10^6$, based on the free-stream velocity and mean chord. The solution is characterized by the $\lambda$-shaped shock appearing on the upper surface of the wing.

For the purposes of this study, the wing is mirrored in the span-wise direction to double the problem size and avoid modeling the wind tunnel wall. The problem size is intentionally doubled to create a large domain suitable for a scalability study, presented in Sec. 3.2. The wing geometry, provided in the experimental report, is meshed using an O-O topology. A simple O-O mesh is shown in Fig. 2.9 from a previous chapter. The $j$ coordinate is the airfoil wrap-around direction resulting in a

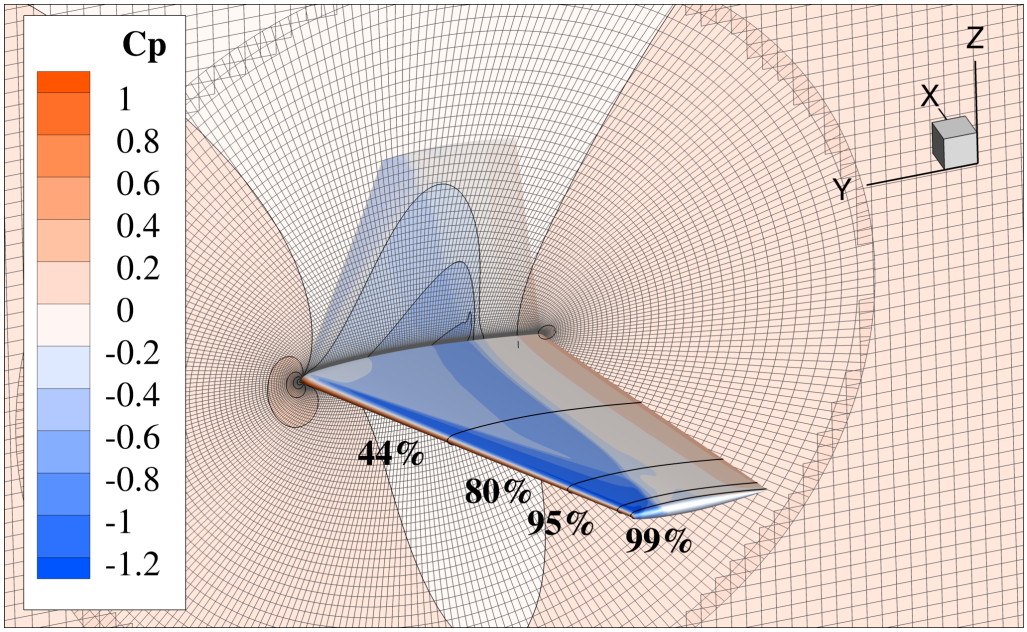



Figure 3.1: Overset mesh system and pressure coefficient solution contours.

periodic boundary at the start and end. The $k$-coordinate is in the airfoil normal direction, resulting in a no-slip boundary at one end and an overset boundary far from the wing. The $l$-boundaries are "wake" boundaries where the mesh collapses to a plane. The non-dimensional viscous spacing at the wall corresponds to $y^+ = 1.38$ and the near-body mesh extends to a distance $2c$ from the wing, where $c$ is the mean chord of the wing. The number of points in the wrap, normal, and span-wise directions is 291, 110, and 179 respectively for a total of 5.73 million points.

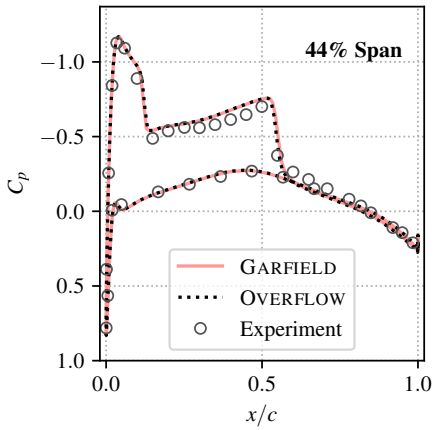
The off-body domain is a stretched Cartesian mesh containing 289 points in the stream-wise direction and 241 in the other two directions for a total of 16.8 million points. The cell spacing varies from $0.08c$ near the wing to $1c$ in the far-field, where $c$ is the mean chord of the wing. A hyperbolic tangent function is used to stretch the grid in all three directions. A view of the mesh system is shown in Fig. 3.1 with the pressure solution on the wing. The resulting background mesh has width and height of $50c \times 50c$ and a streamwise-length of $60c$.

The four span-wise-locations shown in Fig. 3.1 are selected locations where pressure-tap experimental data is available for comparison. Validation is also per-

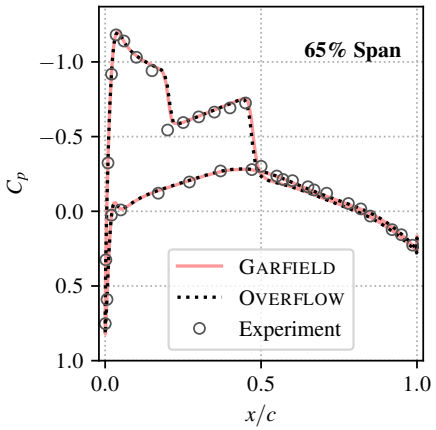
formed against the NASA Overflow solver using the same meshes. Overflow, presented in more detail in Sec. 2.4.4, is also a structured, curvilinear RANS solver but for the CPU. The source has been modified to ensure the same explicit and implicit algorithms are used between Garfield and Overflow.

The pressure coefficient at six locations from experiment, Garfield, and Overflow are shown in Fig. 3.2. The span percentage is defined such that 0% is at the center of the CFD wing mesh (near the wind-tunnel wall) and 100% is at the wing tip. Garfield (solid) and Overflow (dotted) lines of pressure coefficient are almost indistinguishable from each other. The most notable difference is at the 99% span location near $x/c = 0.95$ and Overflow over-predicts (higher negative $C_P$) the pressure compared to both Garfield and experiment. The low-pressure bump observed in the experimental results at the tip of the blade near the trailing edge of the blade is from the blade tip vortex. Garfield likely predicts a better solution compared to Overflow in this region because at the wake boundary near the blade tip Overflow reduces to a lower order spatial scheme whereas Garfield maintains the full reconstruction stencil.

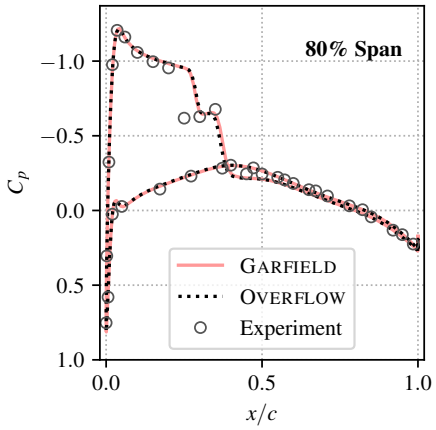
The only large discrepancy between the CFD and experimental results away from the wing tip occurs at the 80% span location. There is a double shock at the 80% span-wise location but the shocks are much closer together compared to the 40% or 60% locations. Garfield and Overflow do a reasonable job at predicting the location of the second shock but both codes over-predict the location of the first shock by about 5% chord. The discrepancy could be a result of a mesh resolution or turbulence model limitations.



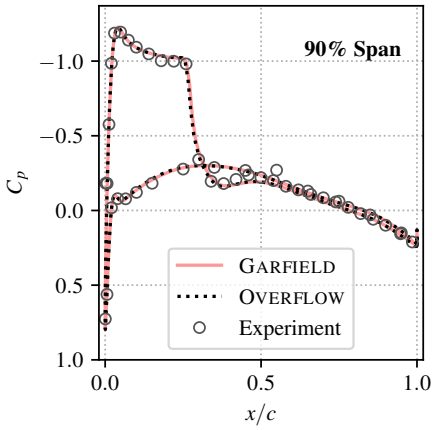

(a) 44% Span



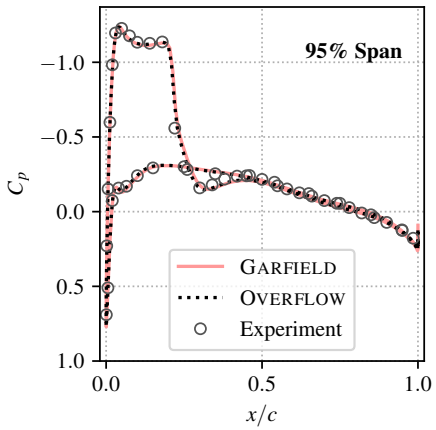

(b) 65% Span



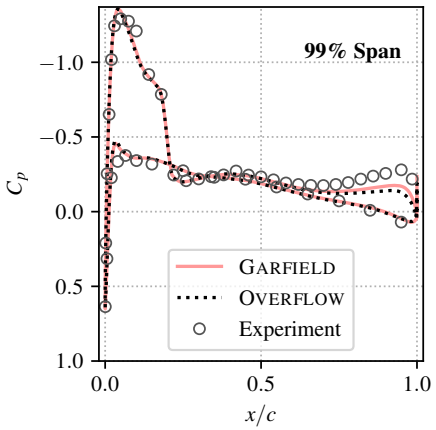

(c) 80% Span




(d) 90% Span




(e) 95% Span




(f) 99% Span

Figure 3.2: Pressure coefficient solution at six wing sections comparing experimental, GARFIELD, and OVERFLOW results.

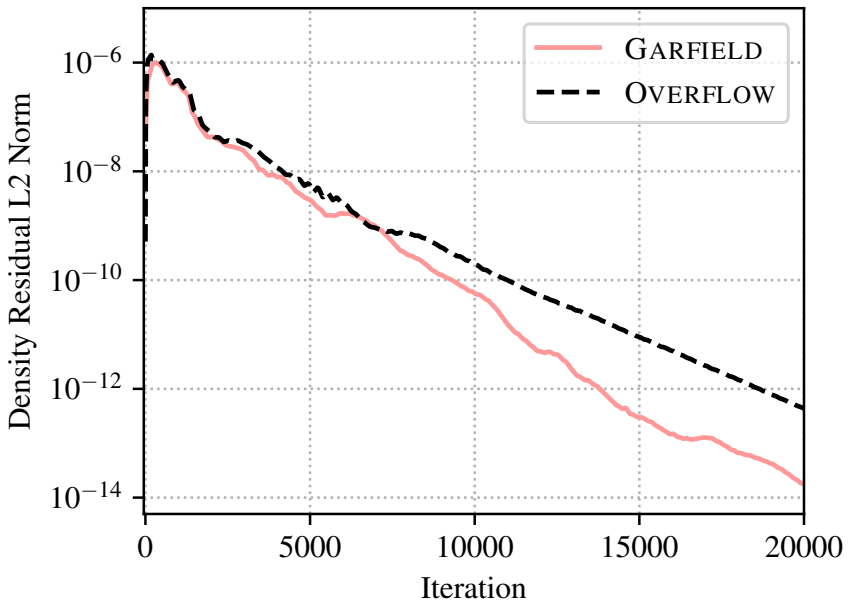



Figure 3.3: Convergence of Overflow and Garfield.

Convergence per-iteration for Garfield and Overflow is compared in Fig. 3.3. Convergence over time is also a useful metric for comparison; timing results are presented for the same wing case in Sec. 3.2.2. For the first 7,000 iterations the convergence trends of both codes are very similar, as is expected for codes using the same spatial and temporal schemes. After approximately 7,000 iterations, however, Garfield appears to converge at a faster rate than Overflow. Using $10^{-12}$ as convergence criteria, Overflow requires 18,647 iterations compared to 13,777 with Garfield, indicating roughly a 25% difference. The convergence difference may be attributed to the difference in boundary condition for the wake cut at the wing tips. The use of ghost points in Garfield means that data is included as part of the implicit scheme "across" the wake, whereas in Overflow the wake boundaries are held fixed during the implicit DADI algorithm.

## 3.2 Performance, Scalability, and Timing

The performance of an overset framework refers to how efficiently components of the framework operate for a representative problem. Scalability refers to how efficiently the domain be divided to run on additional computing hardware. For CPU codes, a scalability study involves distribution of the domain onto different numbers of CPU cores. For a GPU code, different numbers of GPUs are used instead. Finally timing analysis gives an indication of speedup against a baseline implementation. Three different clusters are used for performance, scaling, and timing studies: the Maryland Advanced Research Computing Center (MARCC) cluster, the Mustang cluster from the Department of Defense Supercomputing Resource Center, and the West-1B region of Google Cloud Platform. NASA's Overflow code, described in Sec. 2.4.4, is the baseline for timing comparison. Overflow case is run on 24 Xeon cores, one CPU socket of a Mustang node, and therefore the definition of a speedup is

$$\text{Speedup} = \frac{T_{24 \text{ CPU Cores}}}{T_{GPU}} \tag{3.1}$$

where $T$ is the time taken for a fixed number of iterations.

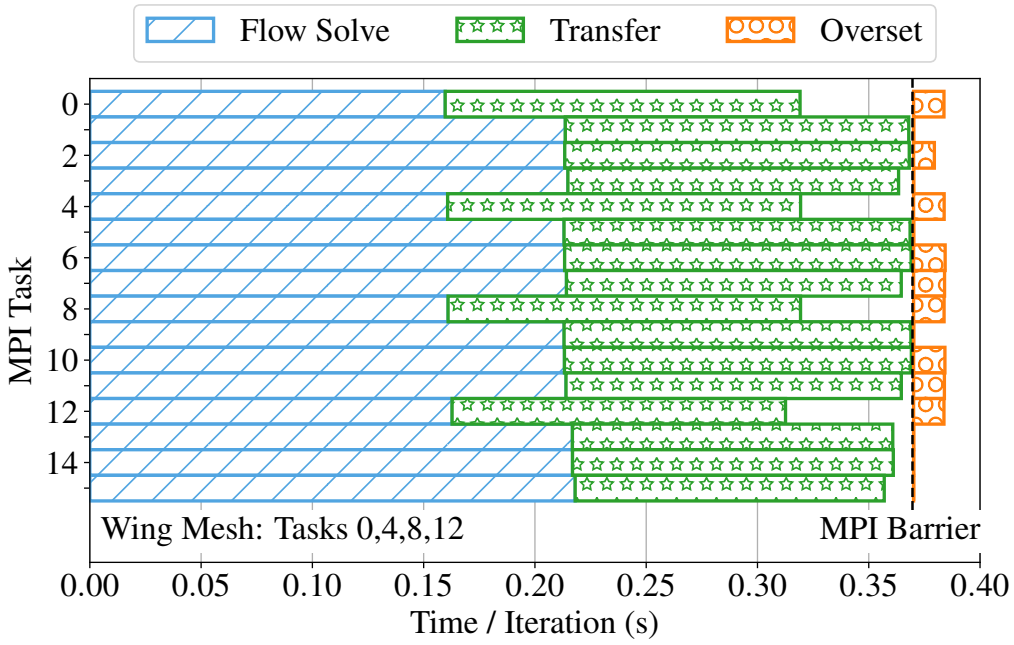
### 3.2.1 Performance

The convergence and pressure results for Garfield from the Onera results presented in Sec. 3.1 were run on the Mustang cluster using four GPUs. The Mustang cluster has one P100 GPU per node, therefore requiring the use of four nodes. Each MPI task from Garfield runs on one CPU core and a single GPU can be shared by multiple MPI tasks. The wing mesh has 5.73 million points and the background has 16.8 million points, indicating the resource distribution should roughly be a ratio of 1:3. For a scenario with 16 tasks over 4 Mustang nodes, this section analyzes the

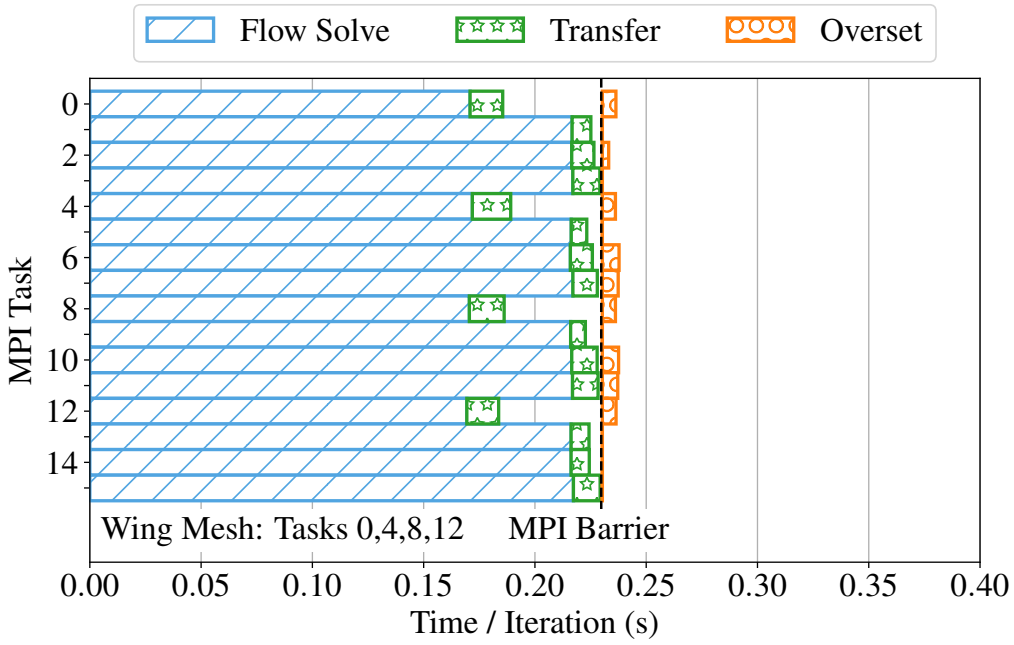
load balancing between tasks and identify proper design choices for GPU solvers in an overset framework.

The amount of memory available on GPUs is typically significantly less than is available on the CPU. A primary motivation for the use of domain decomposition is to avoid hitting the GPU memory limit. Garfield attempts to evenly divide domains across unique GPUs. With 16 tasks and a ratio of 1:3, the blade mesh is assigned 4 MPI tasks and each task resides on a core of a different node to use the GPU on that node. The background mesh uses the remaining 12 tasks with 3 per node, each sharing the GPU on that node. Timing for the flow solver is shown with blue lines in Fig. 3.4. Tasks 0, 4, 8, and 12, which contain the wing mesh, run slightly faster than the 12 other tasks running the background mesh. This is a result of the distribution of points not being exactly the same between tasks. The wing mesh is divided only in the span-wise direction whereas the background Cartesian mesh is divided in all three coordinate directions. So although the $5.73/4 = 1.43$ million points for the wing mesh is larger than $16.8/12 = 1.4$ million in background partitions, the background meshes have added ghost cells in all directions to overlap with neighboring partitions and the added cells contribute to the flow solver calculation time.

Preliminary overset results with Garfield focuses only on optimizing the flow solver without consideration of data transfer for overset routines. Results for a preliminary, naïve run are shown in Fig. 3.4(a). Time is shown as a bar in the $x$-direction for each of the 16 MPI tasks. After flow solver calculations, Garfield transfers the full domain of GPU data back to CPU memory buffers shared with Tioga. After the overset routines are finished, Garfield then transfers the full domain back to the GPU. Because Garfield has no knowledge of donor or interpolation data, the full domain is copied back and forth. The transfer time, shown as green stars in Fig. 3.4(a), takes on average 75.6% the time of the flow solver. The transfer time is also hard to mask with asynchronous memory copying




(a) Without GPU memory in TIOGA




(b) With GPU memory in TIOGA

Figure 3.4: Overset run with the use of GPU memory in TIOGA reduces CPU/GPU data transfers.

because different asynchronous streams would have to be managed through Python in Garfield and Tioga. The time required for overset connectivity, shown in orange circles, is 3.7% of the average flow solver time. An MPI barrier is intentionally placed before overset communication in order to show a clear time-line and cost comparison. The time required for interpolation and exchange of data is significantly smaller than the time taken by the flow solver and transfer routines.

With routines added in Tioga to handle interpolation on the GPU, Garfield only needs to transfer data to a buffer of GPU memory shared with Tioga. Data transfer speeds between the CPU and GPU is limited by the bandwidth of the PCIe bus, typically 32 GB/s. The bandwidth for GPU-GPU memory transfer is much higher: 240 GB/s for the K80 GPU, 720 GB/s for the P100 or 900 GB/s for the V100. Fig. 3.4(b) shows updated results with the use of GPU memory in Tioga. The time taken for the flow solver did not change from the case shown in Fig. 3.4(a) but the transfer time dropped from 75.6% of the flow solver time to 5.5%. The time taken for interpolation and data exchange also reduced from 3.7% to 1.7% because the interpolation is now performed on the GPU. Following the interpolation step data is still transferred back to the CPU over a PCIe bus but the amount of data transferred is significantly reduced.

### 3.2.2 Scalability and Timing

A baseline for timing is established using Overflow on a single socket of the Mustang cluster (24 cores of an Intel Xeon Platinum 8168 at 2.7GHz). As shown in the previous section, different components of the overall simulation can account for significant fractions of the total simulation time. Garfield and Overflow are similar in implementation of flow solver routines, but the overset routines from Overflow are not the same as the routines from Tioga. For a fair comparison, therefore, only the flow solver time is used to compare Overflow and Garfield.

Because the flow solver time can vary between MPI tasks the most practical metric for wall-clock time comparison is the total time taken by the slowest processor, averaged over multiple runs. Results for the baseline OVERFLOW case are shown in Table 3.1.

| **CPU** | **#MPI Tasks** | **Time** (s/100 iter) |
|---|---|---|
| Xeon 8168 (24-core) | 24 | 215.50 |

Table 3.1: Baseline CPU case for timing and speedup comparison.

GARFIELD is run on three different compute clusters: Mustang, MARCC, and the West-1B region of the Google Cloud Platform. The Mustang cluster has 24 GPU nodes with one Pascal P100 GPU per node and 16GB per GPU. MARCC has 72 nodes each with two Kepler K80 GPU cards. Since the K80 is a dual GPU, MARCC effectively has 4 discrete Kepler GPUs per node with 12GB each. Some nodes on MARCC contain two P100 GPUs, however the research facility limits usage to one node. MARCC also has a single trial node containing 4 Volta V100 GPUs, each with 32GB of memory. To expand scalability analysis of V100 GPUs, a virtual machine on the Google Cloud Platform is used containing 8 V100 GPUs, each with 16GB of memory. In each run the time taken by the flow solver in GARFIELD is recorded for 100 iterations. Just as with OVERFLOW, the total time taken by the slowest processor, averaged over multiple runs is used.

A summary of all timing results is shown in Table 3.2. The data is organized by cluster and GPU generation. Cases 1-5 show a scalability study from 2-32 K80 GPUs. The scalability study cannot be run on one GPU because the 12GB memory of a single K80 GPU is insufficient to fit the entire 22.5 million point domain. The last column showing speedup indicates that two K80 GPUs performs 41% faster than OVERFLOW. Using 32 GPUs across 8 nodes, GARFIELD runs 17.68× faster than OVERFLOW. The speedup and strong scalability for GARFIELD on the Kepler architectures is shown as the green line with triangle symbols in Fig. 3.5. The slope

| Case | Cluster | GPU Type | Memory | #GPU | #MPI Tasks | **Time** (s/100 iter) | **Speedup** Over 24 Cores |
|---|---|---|---|---|---|---|---|
| 1 | MARCC | K80 | 12GB | 2 | 8 | 153.33 | 1.41 |
| 2 | | | | 4 | 32 | 81.88 | 2.63 |
| 3 | | | | 8 | 32 | 41.96 | 5.14 |
| 4 | | | | 16 | 32 | 22.56 | 9.55 |
| 5 | | | | 32 | 32 | 12.19 | 17.68 |
| 6 | MARCC | P100 | 16GB | 2 | 16 | 43.03 | 5.01 |
| 7 | Mustang | P100 | 16GB | 2 | 16 | 44.28 | 4.87 |
| 8 | | | | 4 | 16 | 23.03 | 9.36 |
| 9 | | | | 8 | 16 | 13.71 | 15.72 |
| 10 | | | | 16 | 16 | 7.38 | 29.22 |
| 11 | MARCC | V100 | 32GB | 1 | 8 | 40.29 | 5.35 |
| 12 | | | | 2 | 8 | 20.81 | 10.35 |
| 13 | | | | 4 | 8 | 13.04 | 16.53 |
| 14 | Google | V100 | 16GB | 2 | 8 | 21.10 | 10.21 |
| 15 | | | | 4 | 8 | 13.16 | 16.38 |
| 16 | | | | 8 | 8 | 7.55 | 28.54 |

Table 3.2: Timing data for GARFIELD on different clusters and GPU architectures. Speedup is shown against OVERFLOW on a 24-core CPU.

of the ideal speedup is derived from the slope from the origin to the first data point with 2 K80 GPUs. Efficiency of the strong scalability is defined in percent as the actual speedup of the code divided by the ideal speedup. Efficiency for K80 study is shown as the green line in Fig. 3.6. GARFIELD scales well to 32 GPUs with 78% efficiency.

Mustang is used for a scalability study of Pascal-generation GPUs. Case 6 with two P100 GPUs is run on MARCC for comparison with a two P100 case from Mustang, case 7. Although the CPUs are different between MARCC and Mustang, because GARFIELD runs entirely on the GPU and both GPUs are the same the variation between case 6 and 7 is less than 3%. Cases 7-10 scale from 2-16 P100 GPUs.

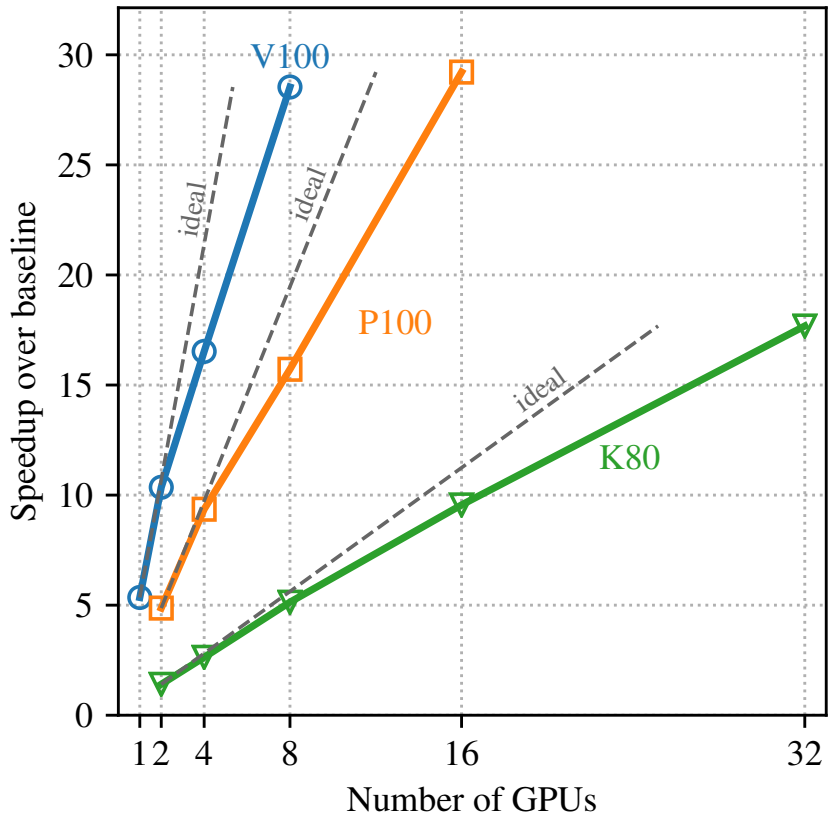



Figure 3.5: Speedup and strong scalability of Kepler (cases 1-5), Pascal (cases 7-10) and Volta (cases 11-13,16) GPUs.

Similar to the K80 GPU, the 16GB memory of a single P100 GPU is insufficient to fit the entire flow domain. On two P100 GPUs GARFIELD runs 4.87× faster than OVERFLOW. The speedup and strong scalability for GARFIELD on the Pascal architecture is shown as the orange line with square symbols in Fig. 3.5. GARFIELD scales from 2-16 GPUs with 75% efficiency, shown in Fig. 3.6.

The V100 GPUs on MARCC have 32GB of memory, which is large enough to fit the entire flow domain of 22.5 million points. Since only 4 GPUs are available, however, the scalability study for the Volta architecture is done with both MARCC and Google platforms. A single V100 GPU attains a 5.35× speedup compared to OVERFLOW. The dual V100 cases on MARCC and Google differ in speed by only 1.4% and the 4-V100 case differ by less than 1%. Up to 8 GPUs GARFIELD scales to 28.54× as fast as OVERFLOW. Cases 11-13 on MARCC are combined with case 16 on Google shown in Fig. 3.5 as a blue line with circular symbols. Previously mentioned Kepler and Pascal cases were run across multiple nodes however on both

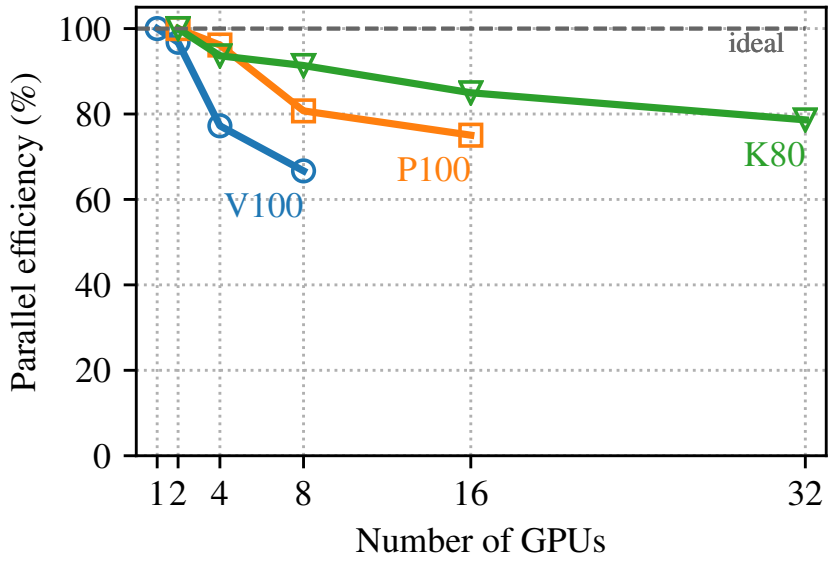



Figure 3.6: Strong scaling efficiency of Kepler (cases 1-5), Pascal (cases 7-10) and Volta (cases 11-13,16) GPUs.

MARCC and Google, the V100s for the scalability study reside on one node. The efficiency of the scalability is shown in Fig. 3.6. Interestingly Fig. 3.5 suggests a 1:2:8 ratio of performance for V100 to P100 to K80 GPUs. 1 V100, 2 P100s and 8 K80s perform almost equivalently and the pattern is approximately maintained for 2:4:16 and 4:8:32.

Looking at a comparison of strong scaling efficiency, Fig. 3.6 shows that the V100 scales less efficiently than the K80 or P100 devices. This result is partially misleading for two reasons. First, since there is no single-GPU data for the K80 or P100 runs, the 2-GPU cases start at 100% efficiency. Second, for a parallel device like the GPU strong scalability should not stay close to 100% as it should for a serial code. As the domain gets smaller, the many streaming multi-processors on the GPU handle the domain more effectively in parallel. As a result with smaller partitions and more GPUs there would be no gain in parallel efficiency. The result that the V100 scales worse than the P100 or K80 is an indication of the higher CUDA core count of the V100.

### 3.2.3 Cost

Use of the Google Compute Platform gives the flexibility to test with different types of GPUs on the same virtual node but it also gives a cost breakdown for resources. Prices for resources, which are also a rough measure of power consumption, are listed in the Google Cloud Platform documentation. As of March 2019, on-demand costs for K80, P100, V100 GPUs and virtual CPU cores are:

$$\mathbf{C}_{\text{K80}} = \$0.45/\text{hr} \qquad \mathbf{C}_{\text{P100}} = \$1.46/\text{hr}$$

$$\mathbf{C}_{\text{V100}} = \$2.48/\text{hr} \qquad \mathbf{C}_{\text{core}} = \$0.06322/\text{hr}$$

Such that the total cost $\mathbf{C}_{\text{total}}$ for $N_{\text{GPU}}$ P100 GPUs across $N_{\text{task}}$ tasks over time $t$ hours is

$$\mathbf{C}_{\text{total}} = t \cdot (N_{\text{GPU}} \cdot \mathbf{C}_{\text{P100}} + N_{\text{task}} \cdot \mathbf{C}_{\text{core}}) \tag{3.2}$$

The costs for "pre-defined" CPU and memory configurations are $0.031611 per virtual CPU core per hour, defined by Google to include hyper-threading cores. Since OVERFLOW is run on a 24-core machine without hyper-threading to ensure best parallel performance, the effective cost for this application is $C_{core} = \$0.06322$ per physical core. Table 3.3 shows the same 16 GARFIELD cases from Table 3.2 along with estimates for the OVERFLOW case for a direct cost comparison. The cost savings results for the Kepler series (cases 1-5), Pascal series (cases 7-10) and Volta series (cases 11-13,16) are shown in Fig. 3.7. The cost savings percentages are defined relative to the total estimated cost of OVERFLOW on the CPU:

$$(\mathbf{C}_{\text{total,CPU}} - \mathbf{C}_{\text{total,GPU}})/\mathbf{C}_{\text{total,CPU}} \tag{3.3}$$

For the cases with K80 GPUs, the minimum cost is with 32 GPUs at 5.56¢for the 100 iterations. All K80 cases are run with 32 tasks, meaning the cost of the CPU cores is more than the cost of 1 K80 GPU. For P100 and V100 GPUs the number of GPUs has a larger impact on the total cost. For the Pascal GPUs, the optimal cost point is 4 GPUs. For Volta the optimal number is 2. In all cases the use of GPUs saves on cost compared to using OVERFLOW on only CPU cores. Maximum savings are up to 65% with V100 GPUs. Comparing generations of GPUs, using Pascal reduces cost compared to the Kepler generation. Volta saves significant cost over both the Pascal and Kepler generations. Cost estimates for the 32GB V100 are based on prices for the 16GB V100s available on the Google platform. As a dollar value, the cost of the larger memory model V100 is higher than the 16GB version,

though as an indication of power consumption there is ideally little difference.

| **Case** | **GPUs** | **Time** (s/100it) | **Cost** (¢) | **Savings** (%) |
|---|---|---|---|---|
| 1 | 2× K80 | 153.33 | 5.988 | 34.1 |
| 2 | 4× K80 | 81.88 | 8.696 | 4.3 |
| 3 | 8× K80 | 41.96 | 6.555 | 27.8 |
| 4 | 16× K80 | 22.56 | 5.779 | 36.4 |
| 5 | 32× K80 | 12.19 | 5.560 | 38.8 |
| 6 | 2× P100 | 43.03 | 4.699 | 48.3 |
| 7 | 2× P100 | 44.28 | 4.836 | 46.8 |
| 8 | 4× P100 | 23.03 | 4.383 | 51.7 |
| 9 | 8× P100 | 13.71 | 4.834 | 46.8 |
| 10 | 16× P100 | 7.38 | 4.994 | 45.0 |
| 11 | 1× V100 | 40.29 | 3.341* | 63.2* |
| 12 | 2× V100 | 20.81 | 3.160* | 65.2* |
| 13 | 4× V100 | 13.04 | 3.777* | 58.4* |
| 14 | 2× V100 | 21.10 | 3.203 | 64.7 |
| 15 | 4× V100 | 13.16 | 3.811 | 58.0 |
| 16 | 8× V100 | 7.55 | 4.268 | 53.0 |
| Overflow 24-Cores | | 215.50 | 9.083 | 0.00 |

Table 3.3: Cost estimates for 100 iterations of Garfield and Overflow based on Google Cloud Platform prices. *Cost estimates for the the 32GB V100 cards are not available on Google and are therefore estimated at the 16GB rate.

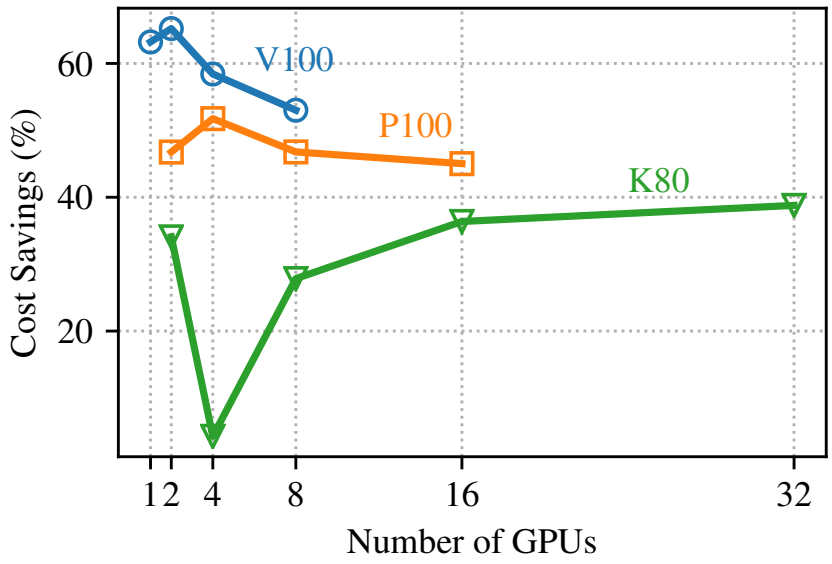



Figure 3.7: Cost savings of Kepler (cases 1-5), Pascal (cases 7-10) and Volta (cases 11-13,16) GPUs.

## 3.3 Elastic UH-60 Rotor

The UH-60 Blackhawk rotor and fuselage underwent wind tunnel and flight tests which have become a popular and challenging data set for CFD validation [69]. One of the main challenges is the aeroelastic coupling that occurs between the blade structure and aerodynamics. Without the correct deflections, the aerodynamics loads from CFD are incorrect; without the correct airloads, the structural model cannot predict the correct deflections. The research of Potsdam et. al. [70] as well as Datta et. al. [71] presented some of the earliest, successful results of coupling CFD to Comprehensive Structural Dynamics (CSD).The result from CFD/CSD coupling is a converged set of structural deflections and aerodynamics loads. One such CFD/CSD converged set of deflections is used for validation of elastic deflections in the presented framework. Given the elastic deflections and resulting aerodynamic forces from the NASA Overflow code, the same deflections are provided to Garfield and airloads

are compared between solvers.

Wind tunnel test 5240 is a high advance-ratio test described in Table 3.4. The high advance ratio means a significant portion of the blade on the retreating side of the rotor is in reverse flow. In combination with a high tip Mach number, the advancing side has a shock near the swept blade tip.

| **Description** | **Symbol** | **Value** |
|---|---|---|
| Advance Ratio | $\mu = V_{tip}/V_\infty$ | 0.37 |
| Blade Loading | $C_T/\sigma$ | 0.09 |
| Blade Tip Mach | $M_{tip}$ | 0.64 |

Table 3.4: Wind tunnel test 5240 parameters.

Garfield uses a different overset grid system from Overflow. Both codes use similar O-O-type structured blade meshes but Overflow uses automatically generated off-body grids while Garfield uses manually generated off-body stretched and Cartesian grids. Three superimposed deflections of the UH-60 blade in Garfield are shown in Fig. 3.8 for $\psi = 0, 90, 180$ degrees. The blade bends and twists elastically based on sectional deflections.

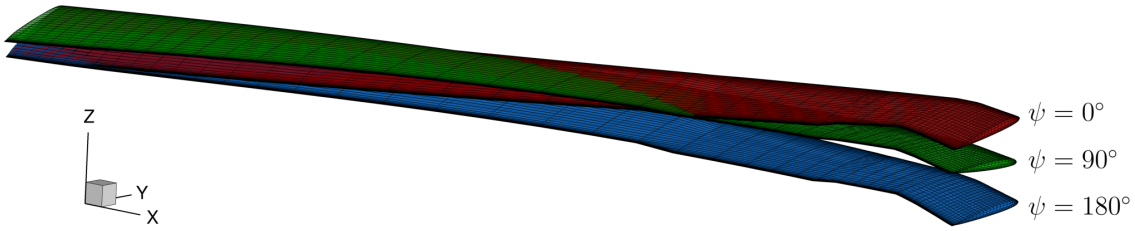
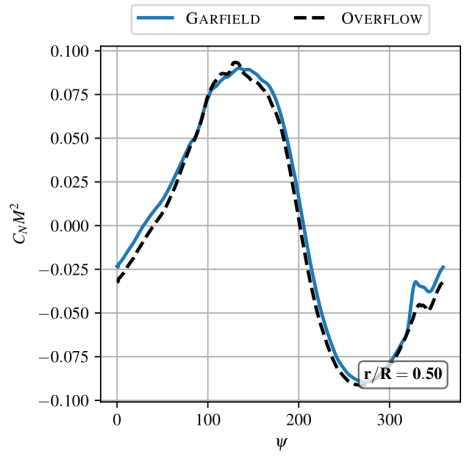



Figure 3.8: Superimposed deflections at three different azimuths: $\psi = 0, 90, 180$.

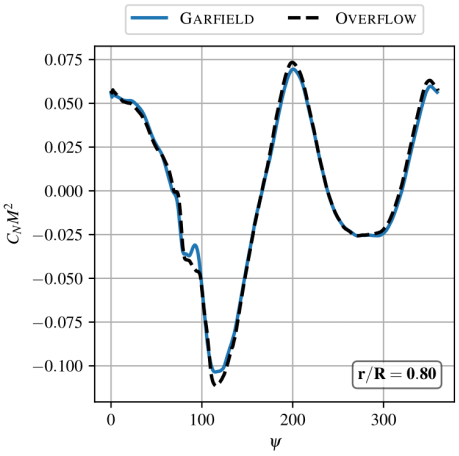
Airload comparisons between results from Overflow and Garfield are shown in Fig. 3.9 for 3 quantities at two blade span-wise locations: 50% and 80%. The normal, chord-wise, and pitching moment coefficients are further multiplied by a Mach number squared, resulting in $C_N M^2$ , $C_C M^2$ , and $C_M M^2$. The mean has been removed from all airloads in Fig. 3.9 to obfuscate results for publication while still allowing for comparison between codes.

At both sections, all airloads agree very well between OVERFLOW and GARFIELD. The largest discrepancies appear in areas of challenging physics. On the advancing side, near $\psi = 90°$, a shock appears at the 80% span, which causes especially high frequency content in the pitching moment. GARFIELD predicts a slightly higher variation in pitching moment from the shock compared to OVERFLOW. At the 50% span, the largest variation appears between $\psi = 300°$ and $\psi = 0°$. This azimuth region is where the reverse-flow separation structures convect aft of the aircraft and affect the passing blade.

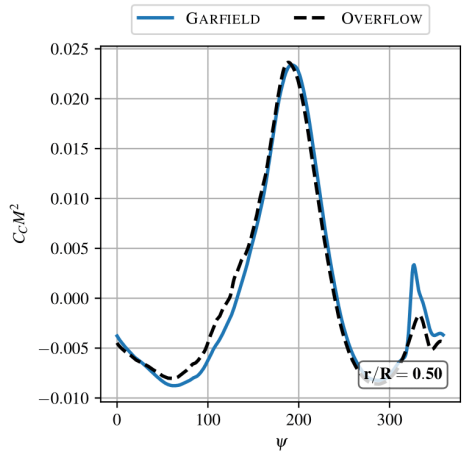
While both GARFIELD and OVERFLOW use the same deflection files, the use of different grids with different resolutions will inevitably result in slight variations in the predicted airloads. Overall there is very good agreement between the two codes and the agreement would not have been obtained without correct treatment of grid motion in the developed overset framework.



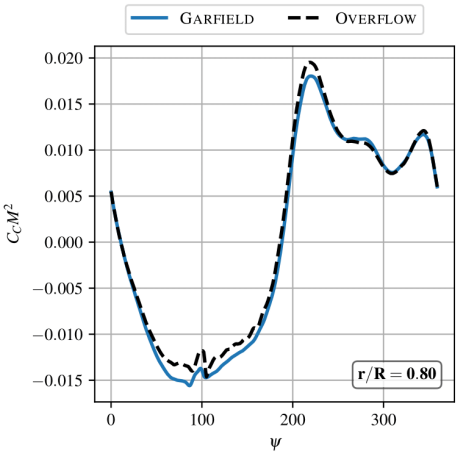

(a) 50% Span: Normal Force



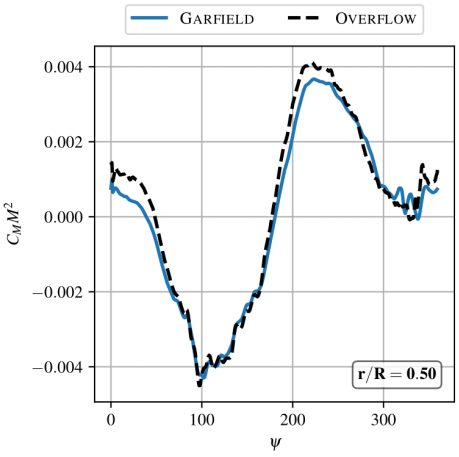

(b) 80% Span: Normal Force



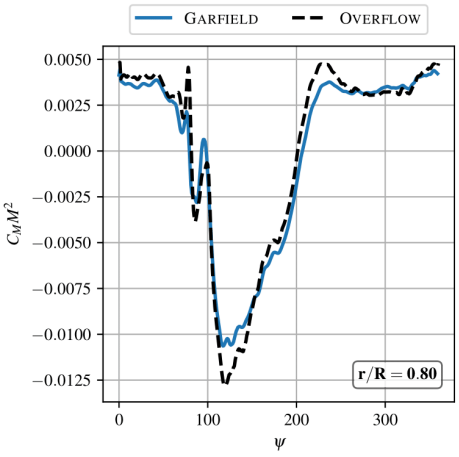

(c) 50% Span: Chord Force




(d) 80% Span: Chord Force




(e) 50% Span: Pitching Moment




(f) 80% Span: Pitching Moment

Figure 3.9: 50% and 80% normal force, chord force, and moment coefficients (mean-removed).

## 3.4 Implicit Algorithms Validation

The methodology chapter, Ch. 2, presents three classes of algorithms for solving the linear system resulting from the implicit discretization of the Navier–Stokes equations. DADI and the Gauss–Seidel algorithms are approximate preconditioners to the linear system. GMRES is a linear-solver approach, which is more involved computationally but also more stable. This section briefly reviews the convergence of the three approaches applied to simple problems to obtain convergence trends.

### 3.4.1 DADI and Red-Black Gauss–Seidel Convergence

An inviscid NACA0012 wing with aspect-ratio 1.0 is used to analyze convergence between DADI and Red-Black Gauss–Seidel methods. Both the Point and Line variations of the Red-Black Gauss–Seidel method are compared to DADI. The low-aspect ratio wing is challenging to converge especially in regions of highly skewed cells near each end of the wing. The selected case is inviscid for simplicity. A solution of the pressure coefficient on the wing and on the center-plane normal to the wing is shown in Fig. 3.10. Streamlines near one end of the blade show the 3-dimensional nature of the flow field.

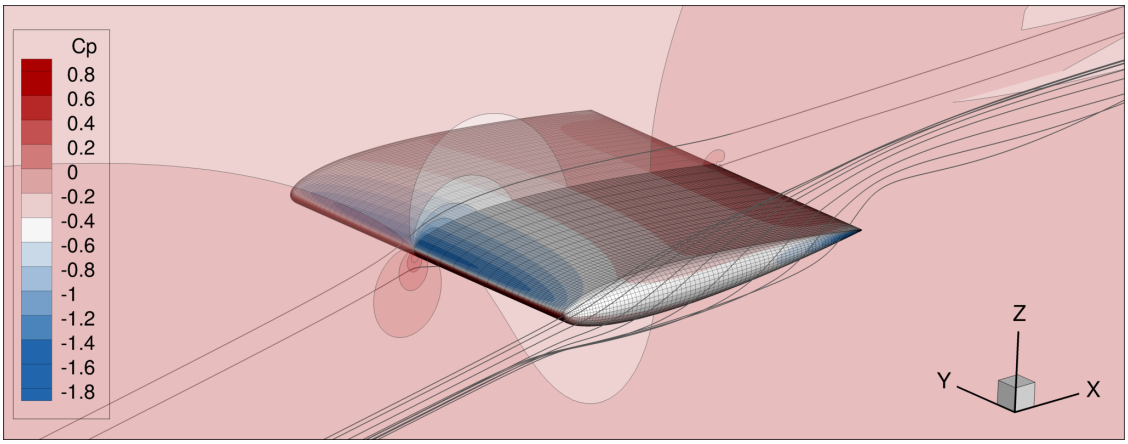



Figure 3.10: Short, inviscid NACA0012 wing used as a representative mesh for convergence analysis.

The goal in converging a steady simulation is to ensure that the numerical residual drops to zero. Fig. 3.11 shows the convergence of the $L_2$-norm of the residual plotted against iteration number. Each method, DADI, Line Red-Black Gauss–Seidel, and Point Red-Black Gauss–Seidel, is run at the maximum stable local timestep, indicated by the CFL number (a higher CFL indicates a higher timestep). For this case the Line Gauss–Seidel method is run with 9 sweeps, 3× for each of the three directions. The Point Gauss–Seidel method is run with 12 sweeps. DADI requires the most number of iterations to converge with a CFL of 8, and almost converges 4 orders in 5000 iterations. The point Gauss–Seidel ("Point GS") method performs significantly better, converging 12 orders in 5000 iterations with a CFL of 20. The line Gauss–Seidel ("Line GS") performs the best, converging 13 orders by 3400 iterations with a CFL of 30. The stalled convergence at $10^{-13}$ is as a result of numerical precision limits for floating point operations. The ability to increase the

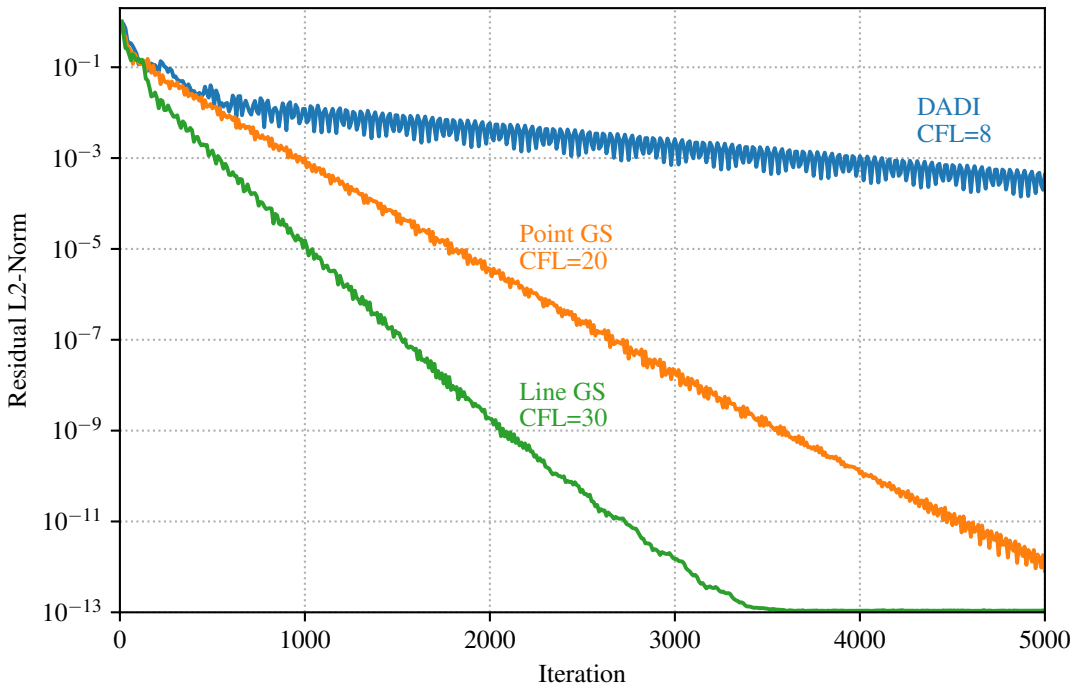



Figure 3.11: Per-Iteration comparison of convergence with DADI and two Gauss–Seidel methods.

timestep (or CFL) for Gauss–Seidel methods translates to improved stability when the same methods are applied to unsteady simulations.

A per-iteration plot does not indicate the practical time costs of a simulation since the iteration time between two methods is not the same. The Gauss–Seidel methods have very different costs, since the Line method requires a block tri-diagonal solve and the Point method does not. A convergence plot over wall-clock time is shown in Fig. 3.12. The fastest converging method is the Point method. The line Gauss–Seidel method still converges slightly faster than DADI.

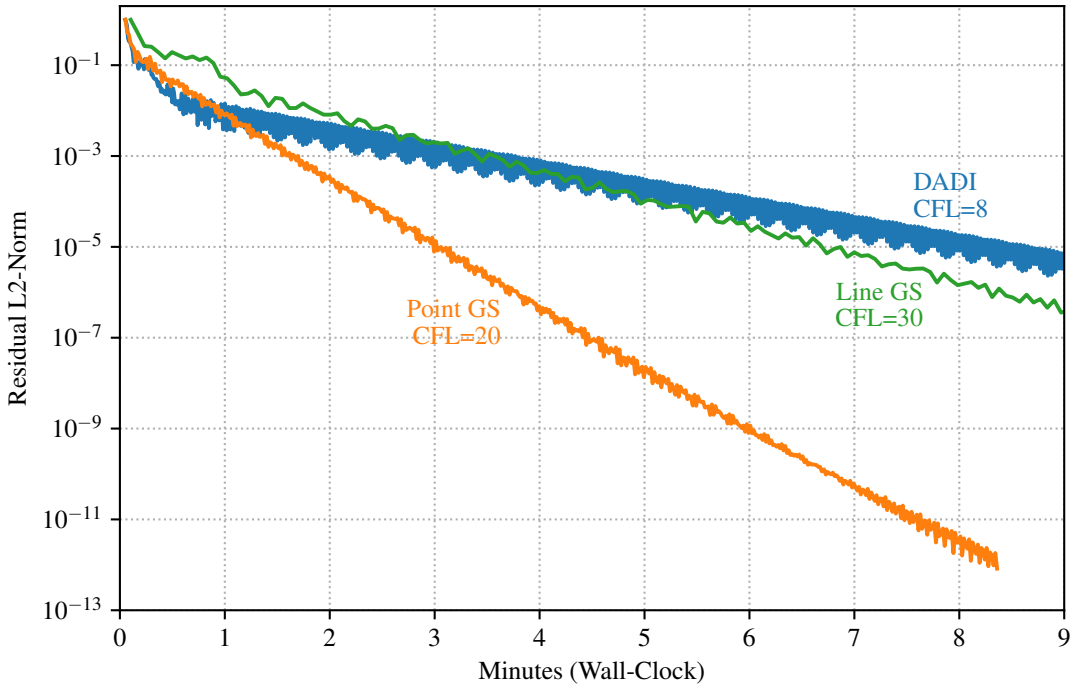



Figure 3.12: Wall-clock time comparison of convergence with DADI and two Gauss–Seidel methods.

Although the Point Gauss–Seidel method is the fastest, the ability to increase the CFL to 30 with the line Gauss–Seidel method is important for time-accurate cases. For a rotor case, for example, the physical timestep is fixed and the effective CFL for the outer portion of the blade is significantly higher than for the inner portion of the blade. Methods with a timestep limitation may be required to take

significantly more sub-iterations with smaller pseudo timesteps to avoid diverging.

### 3.4.2 GMRES Convergence

GMRES is a common linear solver, present in both Garfield and at the framework-level. GMRES can be used with different preconditioners, however in this section only DADI will be used as a baseline for analyzing the benefits of using GMRES. A laminar cylinder is chosen for convergence analysis because the simple case has a steady solution for low Reynolds numbers. The considered case has $M_\infty = 0.3$ freestream flow over a circular cylinder with Reynolds number $Re = 100$. The laminar solution is steady, resulting in vortices on the back of the cylinder. A pressure solution with streamlines is shown in Fig. 3.13. The low Reynolds number makes this case challenging to converge with only the DADI preconditioner. As mentioned in Sec. 2.2.4, DADI uses viscous eigenvalues to approximate the influence of viscous fluxes in the implicit terms of the linear system. For a low Reynolds number case where viscosity has a large influence on the flow physics, the DADI algorithm stability is found to be limited to a timestep corresponding to a CFL of 5.

A per-iteration comparison of convergence between DADI and GMRES is shown in Fig. 3.14. GMRES is used with 5 Krylov vectors and iterations, meaning within the GMRES procedure there is a loop performing 5 residual calculations and

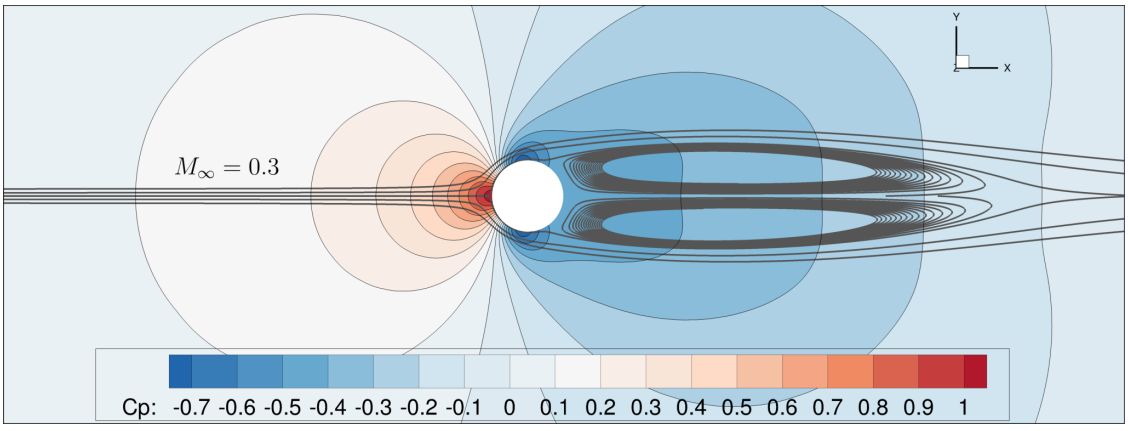



Figure 3.13: Pressure solution over a laminar cylinder with $M_\infty = 0.3$ and $Re = 100$.

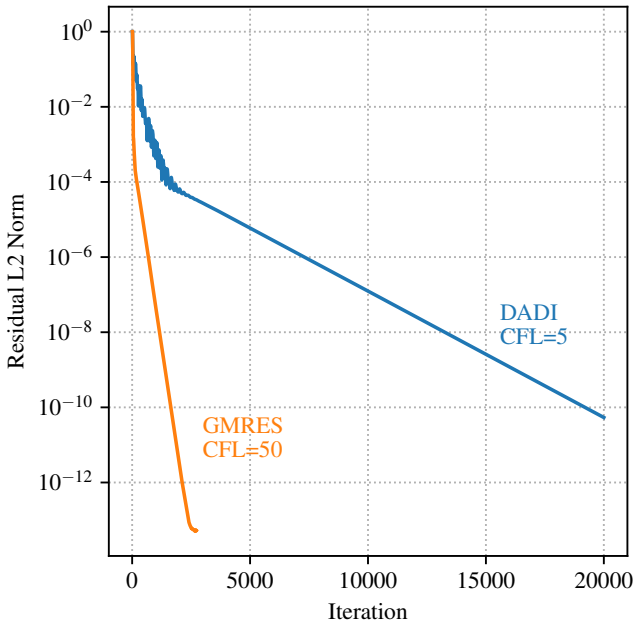



Figure 3.14: Per-Iteration comparison of convergence with DADI and GMRES methods.

5 preconditions. Each of these steps is very computationally expensive and as such, it is expected that GMRES performs so much better than the preconditioner DADI alone per iteration.

A wall-clock time comparison is a much more fair way to compare the routines. The convergence over time is shown in Fig. 3.15, with the two convergence lines much closer together compared to the per-iteration plot. Even with GMRES performing significantly more work than DADI per iteration, over time GMRES converges much faster than DADI. A large factor in the improved convergence rate with GMRES is added stability for the implicit scheme and the ability to increase the timestep size. Whereas DADI alone is limited to a CFL of 5 for this case, GMRES is able to run ten times the timestep size with a CFL of 50.

Similar to the results from the DADI and Gauss–Seidel analysis, a major benefit of added stability is for time-accurate applications where all parts of the flow domain must be converged well to ensure a physical solution. GMRES here is used

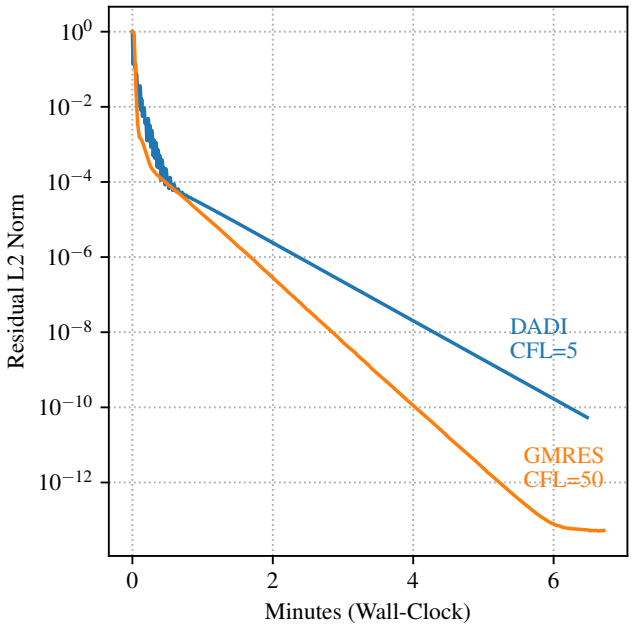



Figure 3.15: Wall-clock time comparison of convergence with DADI and GMRES methods.

with DADI as preconditioner however it can also use the Red-Black Gauss–Seidel methods. GMRES is unconditionally stable without a preconditioner, however it will only converge well for most engineering applications with the help of a good preconditioner.

## 3.5 Overset-GMRES

The developed Overset-GMRES (O-GMRES) algorithm is analyzed for two simple cases. The first is for the 2D Poisson equations and the second is for a single-solver, overset, inviscid simulation of a triangular wedge. The goal of these two simple cases is to establish expected behavior and convergence improvements with the O-GMRES method before moving to more complex simulations.

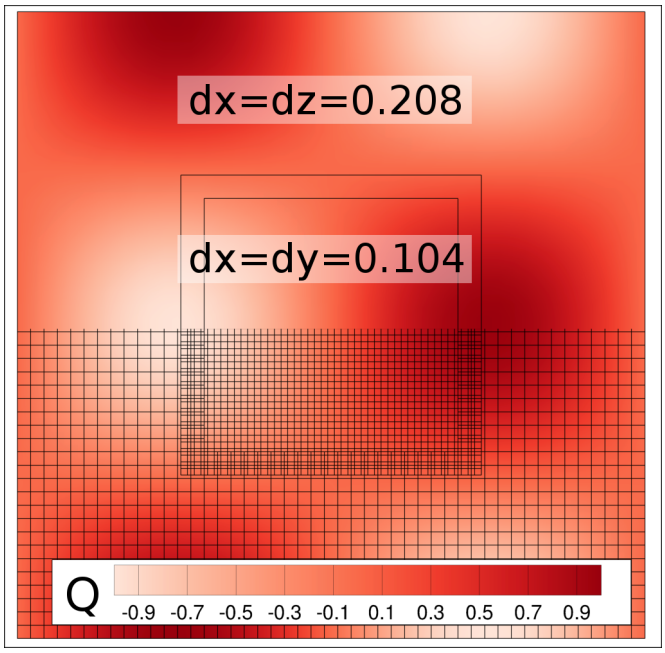
### 3.5.1 Poisson Equation

The first case considers the 2D Poisson equation solved on two overset grids, each with $48 \times 48$ points. The governing equation is shown in Eq. (3.4) and the exact solution is shown in Eq. (3.5). The mesh system and solution contours are shown in Fig. 3.16(a). Both the coarse and fine meshes are square and centered at $(x, y) = (0.5, 0.5)$. The coarse and fine meshes have widths 1.0 and 0.5 respectively. The boundary of the outer mesh is fixed at the exact and the boundary of the interior mesh is interpolated from the outer mesh. The *iBlank* map is shown for half of each mesh in Fig. 3.16(b), where the "blanked" nodes from each mesh are light gray and the field points are blue. The *iBlank* map indicates the outer points of the interior mesh are interpolated from the coarse mesh and a center square from the coarse mesh is interpolated from the fine mesh.

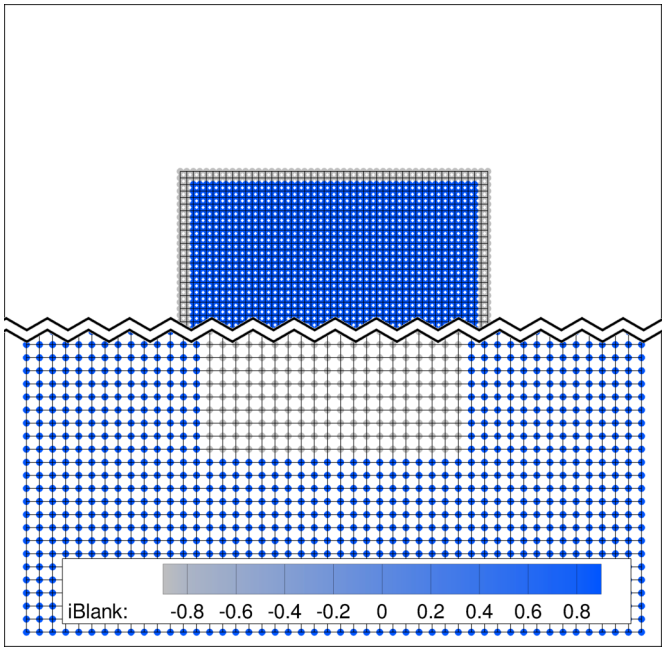
$$\frac{\partial^2 q}{\partial x^2} + \frac{\partial^2 q}{\partial y^2} = S(x, y) \qquad 0 \leq x \leq 1, \quad 0 \leq y \leq 1 \tag{3.4}$$

$$q = sin(2\pi x)cos(2\pi y), \qquad S(x, y) = -8\pi^2 sin(2\pi x)cos(2\pi y) \tag{3.5}$$

The 2D Poisson case is small enough to solve with a single processor and a version of the Poisson solver has been implemented for comparison which builds a contiguous array for all grids. The preconditioner and matrix-vector product for the single-array code is very similar to the 1D example from Alg. 2 and Alg. 3, from Ch. 2. The interpolation weights are extracted from TIOGA to ensure correct preconditioning and matrix-vector product near fringe points. O-GMRES, which is intended for distributed memory, is mathematically equivalent to the single-array implementation. Using $m = 50$ Krylov vectors both codes converge identically within 15 iterations (each iteration in this case is a restart of the GMRES procedure), as shown in Fig. 3.17(a). Differences in the residual below $1 \times 10^{-12}$ are from machine




(a) Solution and Grid

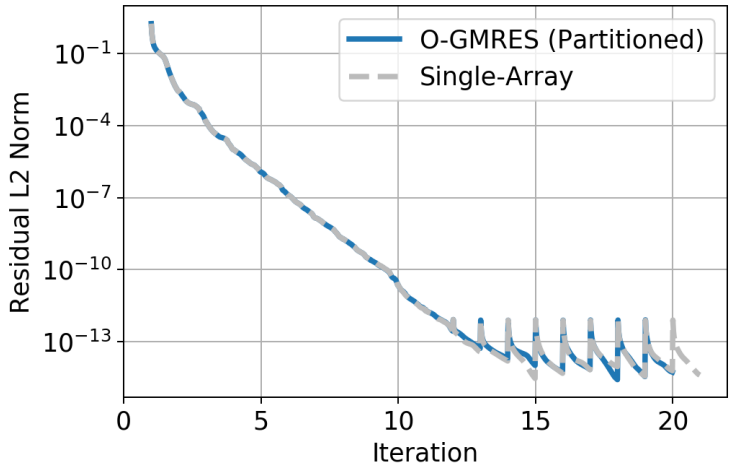



(b) Grid *iBlanks*

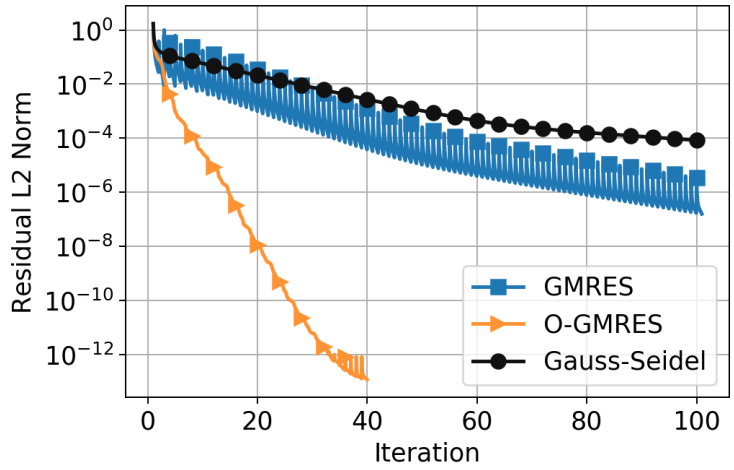
Figure 3.16: Solution, grid system, and *iBlanks* for the overset Poisson equation.

precision limitations.

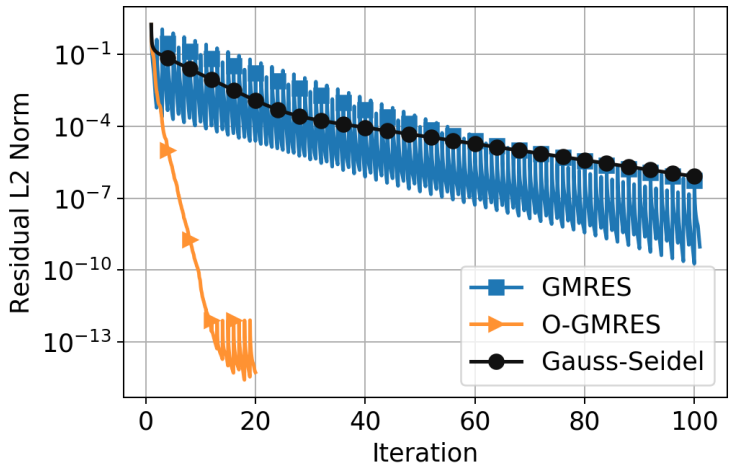
Fig. 3.17(b),(c) and (d) compare the convergence between the traditional GMRES, O-GMRES, and Gauss–Seidel methods. Linear solver iterations refers to the number of Krylov iterations in the case of GMRES and O-GMRES and simply the number of sweeps for the Gauss–Seidel method. Fig. 3.17 plots the residual history for all iterations and linear-solver iterations. The GMRES and Gauss–Seidel methods, representative of the current approach taken in many CFD frameworks, handle fringe points explicitly since exchange of data occurs outside of the linear solver iterations. The traditional GMRES converges with a "saw-tooth" pattern. During the linear solver iterations GMRES converges well but since each mesh is solved independently, the linear system is updated every iteration causing an increase in the residual. O-GMRES considers the full overset system including the interpolated points and the convergence per iteration follows the convergence of the Krylov iterations.



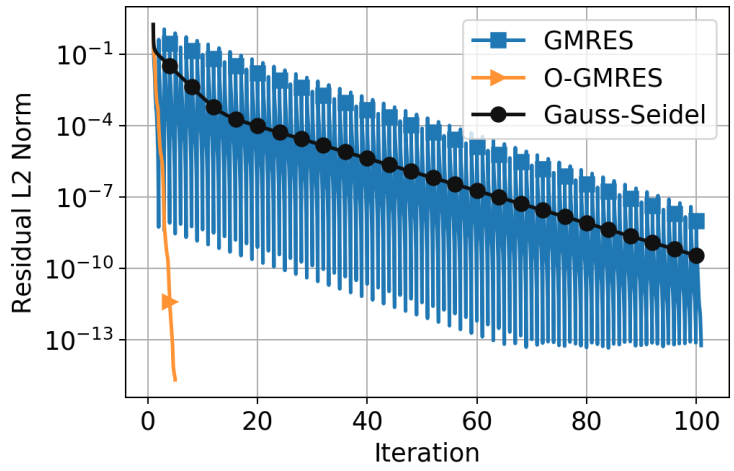

(a) O-GMRES and Single-Array GMRES.




(b) 20 Linear Solver Iterations




(c) 50 Linear Solver Iterations




(d) 100 Linear Solver Iterations

Figure 3.17: Convergence results using different linear solver methods: GMRES, O-GMRES, and Gauss–Seidel

### 3.5.2 Single-solver, Euler Equations

The first case considered for O-GMRES analysis with GARFIELD is inviscid flow around a 2D triangular wedge. The solution, shown in Fig. 3.18(a), shows a wedge in Mach 0.2 free-stream with vortices shedding behind the body. The unsteady problem has been studied by other research groups [72, 73] for analysis of temporal and spatial accuracy. For this case, first-order implicit-Euler time marching is used with a $5^{\text{th}}$ order WENO [23] spatial reconstruction scheme. The grids coarsen in the wake of the wedge and as a result the shed vortices dissipate quickly. The grid system is therefore not ideal for vortex preservation but is sufficient for convergence analysis.

The triangular wedge has sides of length $1m$ with inviscid grid spacing near the wall of $0.005m$. The resolutions of the background grids are $0.08m$ and $0.16m$ for the fine and coarse grids respectively, shown in Fig. 3.18. The physical timestep corresponds to a CFL of 3 in the finest Cartesian nested mesh with respect to freestream conditions.

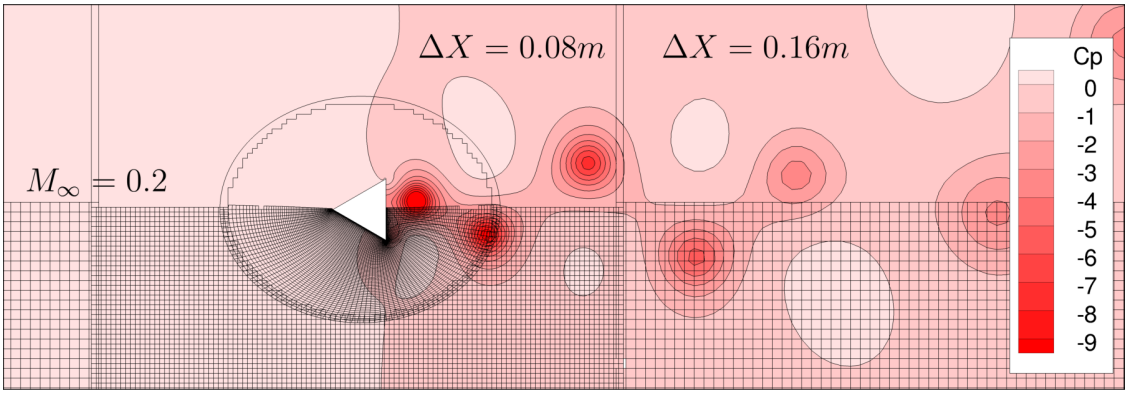
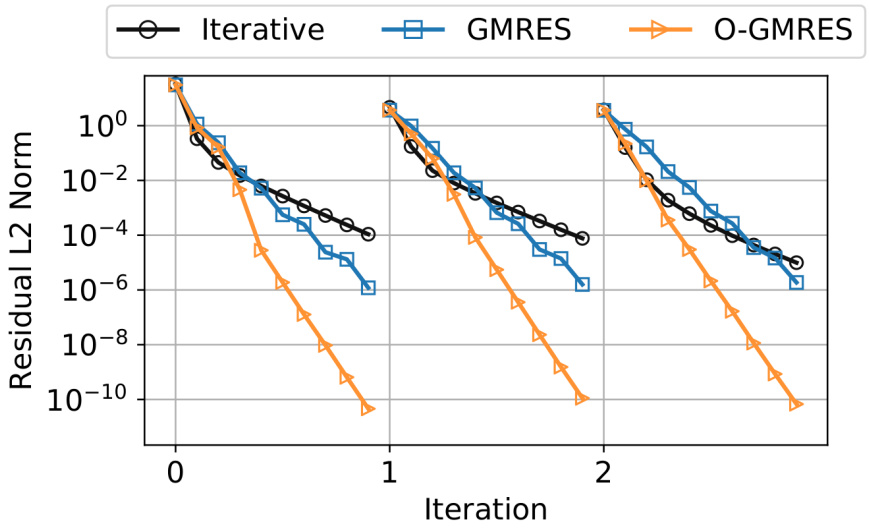


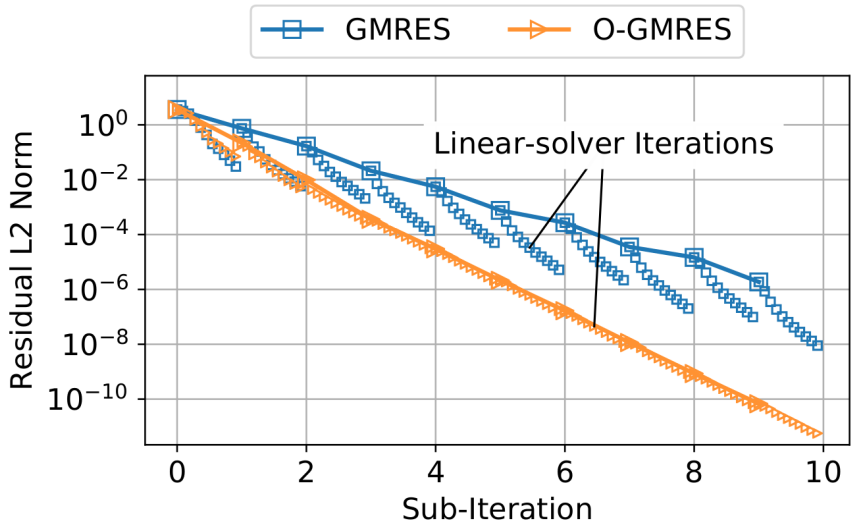

Figure 3.18: Solution and grid system for time-accurate wedge

The solution was allowed to advance to a time $t^*$ when vortices shed into the background meshes. The simulation is re-started with three methods: "Iterative", "GMRES", and "O-GMRES". The "Iterative" method refers to the preconditioner alone applied to approximate the inversion of the linear system. A mixed Point-Line

Red-Black Gauss–Seidel method with 3 sweeps was used for the preconditioner of the wedge mesh and DADI was used for the two background Cartesian meshes. The iterative method used 10 times the number of sub-iterations for a fair-cost comparison with 10 Krylov iterations used in both GMRES methods.




(a) Time-accurate Convergence




(b) Iteration 2 Non-linear and Linear Convergence

Figure 3.19: Sub-iteration and linear solver convergence for 2D wedge case.

Convergence comparison for three linear solvers is shown in Fig. 3.19(a) for three timesteps of the restarted simulation. The iterative methods converges the worst, achieving roughly 5 orders drop in $10 \times 10$ sub-iterations. The de-coupled GMRES method does slightly better, converging roughly 6 orders in 10 sub-iterations. O-GMRES converges over 10 orders, performing much better than any other method.

A zoomed in view of iteration 2 is shown in Fig. 3.19(b) for the two GMRES methods. Within each sub-iteration the linear solver is converging well for both GMRES and O-GMRES. With GMRES, however, the convergence is not consistent with the sub-iteration convergence. Though the linear solver is converging over two orders, the resulting non-linear convergence is less than one order per sub-iteration. With O-GMRES, on the other hand, the overall non-linear convergence closely follows the linear solver convergence.

## 3.6 Conclusions

The verification and validation of the present overset framework, including the GPU accelerated flow solver, is organized into five sections. The first section used the Onera-M6 wing case to demonstrate simple, overset mesh capabilities with Tioga. Results are validated against experimental data and simulation results from NASA's Overflow code run with the same mesh. The pressure solution from Garfield matches very well with both experimental values and those from Overflow.

The Onera-M6 case is subsequently also used for performance, scalability, timing, and cost analysis of the GPU accelerated solver. Performance is analyzed, highlighting the need to minimize CPU-GPU data transfers. A strong scalability study on different GPU generations verifies Garfield scales well on three different compute clusters. The timing study concludes Garfield on the newest GPUs runs 5.35× faster than Overflow running on 24 CPU cores. Notably, a per-node comparison adds a multiplier on this speedup, since it is common to have 4-8 GPUs

available per node. Finally, a cost comparison is made using estimates from the Google Cloud Platform. Not only does using GPUs result in faster simulation times, it results in lower costs. Using cost as a metric of power consumption, using Garfield also saves energy compared to Overflow.

The motion module of the framework, as well as grid-motion terms within Garfield, are validated against Overflow for a UH-60 Blackhawk rotor in high-speed forward flight. Using the same set of elastic deflections, Garfield results match very well with results from Overflow. Because the rotor blades undergo large bending and torsional deformation, the airloads from CFD would not be correct if the deflections were not properly handled.

The newly implemented implicit numerical schemes within Garfield are validated by comparing convergence to the baseline DADI method. Both line and point variations of the Red-Black Gauss–Seidel method perform better than DADI for a representative inviscid 3D wing. Convergence results with GMRES are also compared to DADI for a laminar cylinder case. GMRES converges significantly faster, both per iteration and per wall-clock time. A major benefit in using GMRES and Gauss–Seidel methods is the ability to converge challenging time-accurate cases with a larger timestep than would be possible with DADI.

The framework-level Overset-GMRES (O-GMRES) algorithm is tested on a 2D Poisson equation to verify the partitioned algorithm is equivalent to a single-matrix system. O-GMRES is further validated against the de-coupled GMRES algorithm and simple iterative approaches by showing drastically accelerated convergence. Convergence improvements are seen for the overset Poisson solver, as well as for overset, inviscid simulation of flow over a wedge.

# Chapter 4: Notional Helicopter Configuration

The goal of this chapter is to demonstrate the ability of the developed framework to simulate large, complex simulations for the purpose of interactional aerodynamic analysis. Three sections are presented, which build upon a notional helicopter configuration based on the ROtor Body INteraction (ROBIN) experiments by Mineck and Gorton [3]. The first section describes the experiment, presents the CFD grid system, and compares recorded fuselage pressure for a case involving a main rotor and fuselage. Predictions from the CFD framework are compared to experimental results as well as CFD results from another research group.

The second section extends the baseline configuration to analyze the interactional aerodynamics with the inclusion of a tail rotor and hub. The full configuration is run and main rotor thrust is compared to the CFD result without the tail or hub. As the most complex simulation run for this dissertation, the full configuration is also used to draw conclusions about convergence challenges and computational load-balancing solutions for the heterogeneous framework. The complete vehicle configuration is also simulated with a cross-wind to demonstrate the framework capabilities.

The third section analyzes rotor interactions using different combinations of isolated main and tail rotors. Harmonics of the tail rotor thrust are compared between cases for insight into vibration resulting from the interactional aerodynamics. The main and tail rotor are run for two flight conditions: forward-flight without a cross-wind and forward-flight with a cross-wind.

The selected configurations for simulation are notional in that then involve general design choices commonly applied to real configurations. The notional configurations, however, preserve key computational challenges. Some of the simulation challenges considered include:

1. Ensuring adequate mesh refinement to accurately capture and convect vorticity without excessive dissipation.

2. Converging the residual of the governing equations sufficiently for a time-accurate solution.

3. Load-balancing the many meshes across computational resources (CPUs and GPUs).

When analyzing the flow-field of the large configuration, iso-surfaces of vorticity or Q-criterion help visualize the locations of interesting flow features. A mathematical and qualitative description of Q-criterion and vorticity is included in Appx. A.

## 4.1 Rotor + Fuselage Simulation

### 4.1.1 Case Description

Experiments with the ROBIN fuselage and rotor were performed at the NASA Langley $14 \times 22$ foot subsonic wind tunnel. The ROBIN fuselage geometry is defined in the work of Mineck and Gorton [3] and shown in Fig. 4.1. The fuselage includes a pylon to represent the fairing around the engines and transmission. The length of the body is $l = 78.7$ inches and the rotor is placed over the fuselage at a longitudinal location of $x/l = 0.696$ from the nose and a lateral location of $y/l = 0.051$ from the centerline. The lateral offset indicates the rotor is not centered over the fuselage. The fuselage also has a sideslip-angle of approximately $1.2°$. Figs. 4.1(a) and (b) show two views of the experimental setup in the wind tunnel. The four-bladed rotor

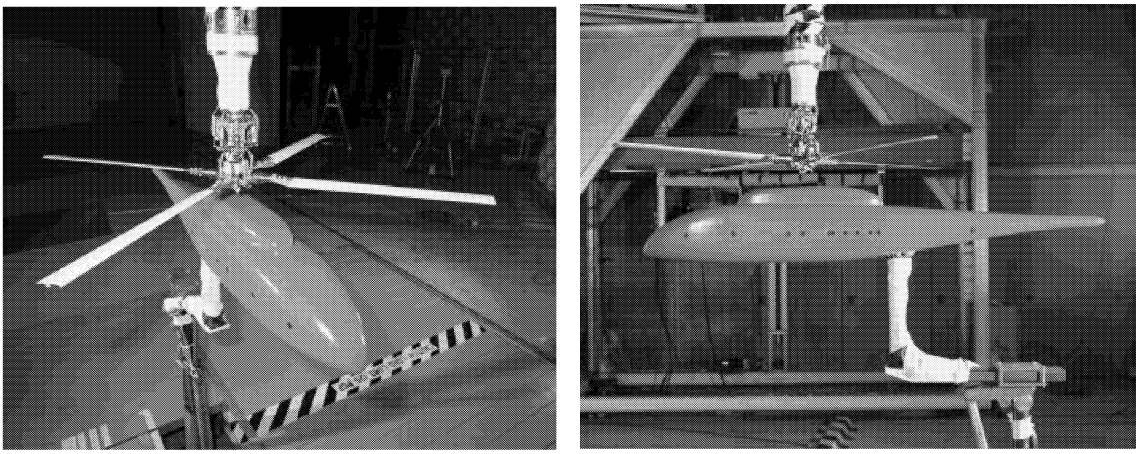
and hub is supported from above whereas the fuselage is supported from below. There is a gap between the rotor and fuselage.

| **Description** | | **Value** |
|---|---|---|
| Radius | $R$ | 33.88 in |
| Chord | $c$ | 2.61 in |
| Number of Blades | $N_b$ | 4 |
| Linear Twist | $\theta_{tw}$ | $-8°$ |
| Collective | $\theta_0$ | $6.7°$ |
| Longitudinal Cyclic | $\theta_{1c}$ | $2.2°$ |
| Lateral Cyclic | $\theta_{1s}$ | $-2.0°$ |
| Tip Mach | $M_{tip}$ | 0.53 |
| Reynolds Number | $Re$ | $8.2 \times 10^5$ |
| Blade loading | $C_T/\sigma$ | 0.064 |
| Advance Ratio | $\mu$ | 0.15 |
| Shaft-tilt | $\alpha_s$ | $3°$ |
| Free-stream Mach Number | $M_\infty$ | 0.08 |

Table 4.1: Main rotor properties and flight condition.

The rotor blades have a rectangular planform with properties summarized in Table 4.1. The airfoil cross-section for the entire blade is a NACA0012. The Reynolds number is based on the tip speed and chord length.

Previous numerical work [74–76] found significant discrepancies in collective pitch between numerical and experimental values, whereas cyclic pitch angles were



(a) Isometric View (b) Side View

Figure 4.1: Experimental Setup [3].

similar. Instead of matching the collective pitch from experiment, the collective pitch is adjusted to match the blade loading between simulation and experiment. The selected collective pitch of $\theta_0 = 6.7°$ results in a time-averaged blade loading of $C_T/\sigma = 0.063$. Cyclic pitch angles are selected based on trim results from Park et. al. [76].

Three different CFD solvers are used for the ROBIN simulation: HamStr, TURNS, and Garfield. Fig. 4.2 shows the surface meshes for the main rotor, run with TURNS, and the fuselage, run with HamStr. The blade mesh is a body conforming O-O type mesh consisting of $207 \times 180 \times 80$ points in the chord-wise, span, and normal directions, respectively. The non-dimensional wall-spacing ($y^+$) is 0.2, and the total number of points for four blade meshes is 11.92 million (2,980,800×4). The fuselage mesh is generated from an all-quadrilateral surface mesh consisting of 122,772 elements. The fuselage grid contains 50 strands layers to create the volume mesh with an initial wall spacing of $y+ = 0.4$. The wall normal direction stretching ratio is 1.25. The total number of hexahedra in the volume is 6,015,828.

All background grids for the rotor-fuselage case are run with Garfield. The

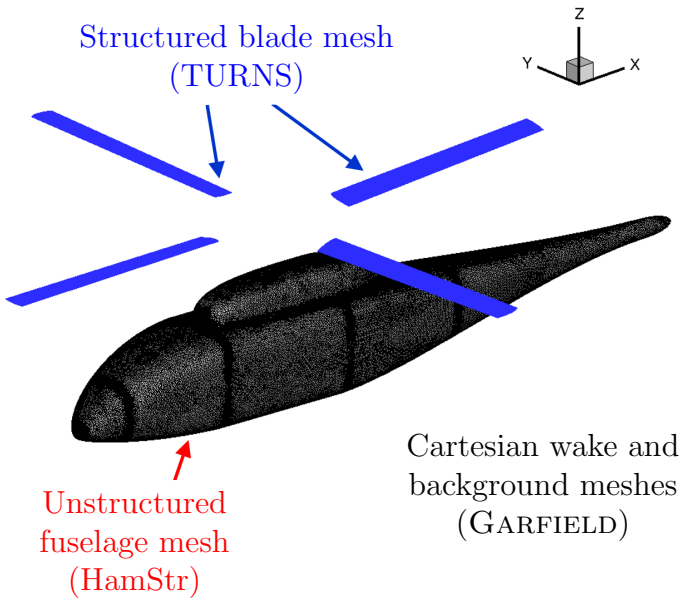
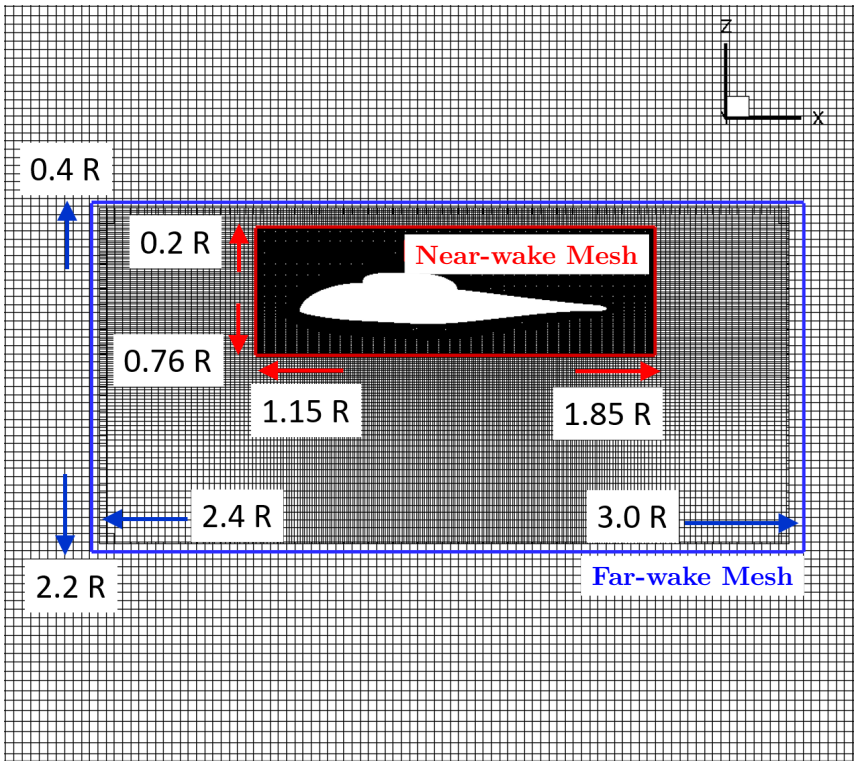
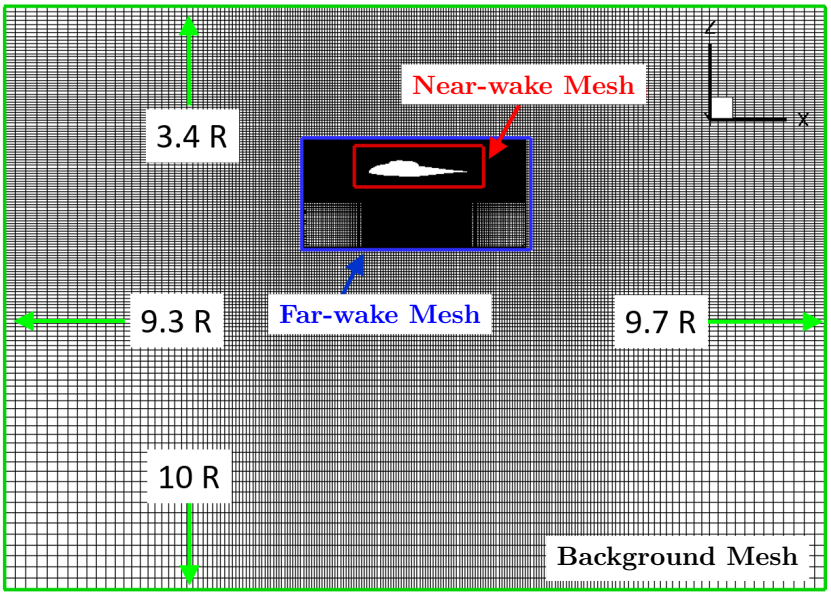



Figure 4.2: Fuselage and blade surfaces for CFD domain.




(a) Near and Far-wake meshes.




(b) Total mesh system including background mesh.

Figure 4.3: Mesh system of the ROBIN simulation in forward flight.

full overset mesh system is shown in Fig. 4.3(a) for the region near the fuselage and (b) for the full background mesh. The near-wake, far-wake, and background regions are all meshed using stretched Cartesian grids. The near-wake mesh has cell spacing of $0.09c$ to accurately capture the blade tip vortex and interactions with the fuselage. The far-wake mesh has spacing of $0.2c$ near the fuselage and $0.8c$ far away. Finally the background mesh stretches from $0.8c$ near the fuselage to $3.5c$ in the far-field. The near-wake, far-wake, and background meshes have 17.97 million, 5.92 million, and 5.46 million nodes respectively.

### 4.1.2 Aerodynamic Results

Fig. 4.4 shows iso-surfaces of vorticity with contours of $z$-velocity to visualize the flow field. Large vortices trail behind the advancing and retreating blades as a result of blade-vortex and vortex-vortex interactions. The vorticity dissipates behind the aircraft in the region where the CFD grid changes from fine to coarse. The vortex dissipation, while not physical, is acceptable for simulation because the influence of a vortex on any surface of the grid diminishes hyperbolically with distance. The vortices far away from the aircraft therefore have negligible influence on the rotor inflow or fuselage surface and the influence gets only smaller as the vorticity convects backwards.

Three points are selected along the longitudinal centerline of the aircraft for comparison with experiment and simulation results from O'Brien [74]. The pressure coefficient $C_P \times 100$ is plotted against the azimuthal position of one blade. Results from the present work are shown in red. Experimental results from Mineck and simulation results from O'Brien are shown in black and blue respectively. O'Brien used the NASA FUN3D solver for all simulations.

Point D9 is near the nose of the aircraft as shown in Fig. 4.5(b). The average pressure coefficient is approximately the same between all three results. The

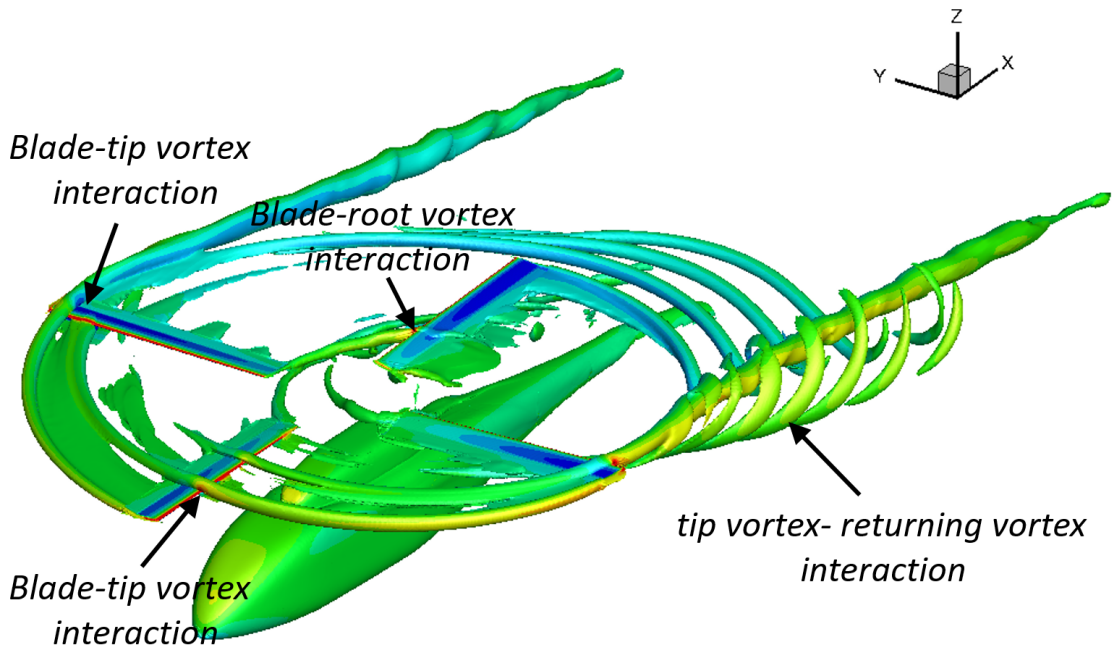
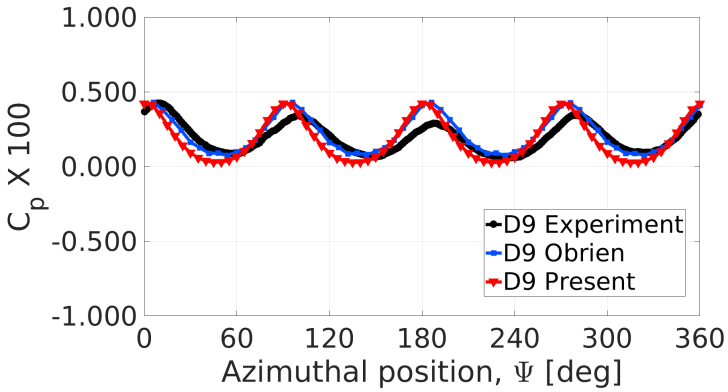



Figure 4.4: Iso-surface of vorticity magnitude (0.1).

experimental results do not show consistent amplitudes between the four pressure peaks from the four blades. CFD results do predict consistent peaks for the 4/rev signal. The peak of the first pressure fluctuation matches well with the present results however the other three peaks appear over-predicted by CFD. Overall the present results match very closely with the results from O'Brien.

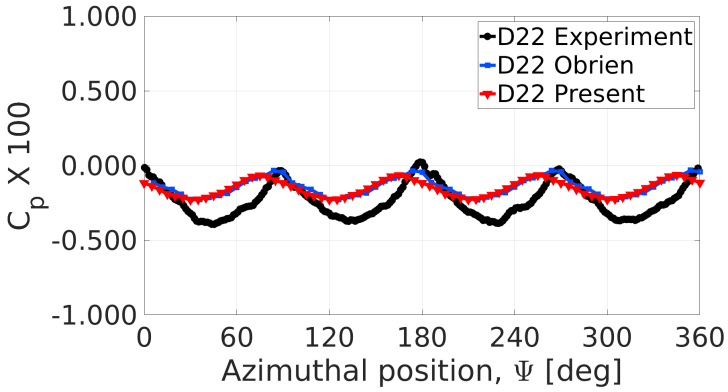
Point D22 is on the pylon, just aft of the hub as shown in Fig. 4.5(d). The presented pressure results again match very well with simulation results from O'Brien. Both CFD results, however, predict a lower amplitude of pressure fluctuation. There appears to also be a slight phase difference between the pressure fluctuations from CFD and experiment. Because point D22 is on the pylon below the root section of the rotor blades, the pressure in this location may be mis-represented in CFD from the lack of a rotor hub and support structure.

Point D14 is on the start of the tail boom of the aircraft as shown in Fig. 4.5(f). The amplitude of the pressure fluctuations matches well between experiment and CFD. Experimental results appear to have some asymmetry, since not all four of the

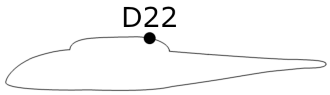



(a) Point D9 Pressure

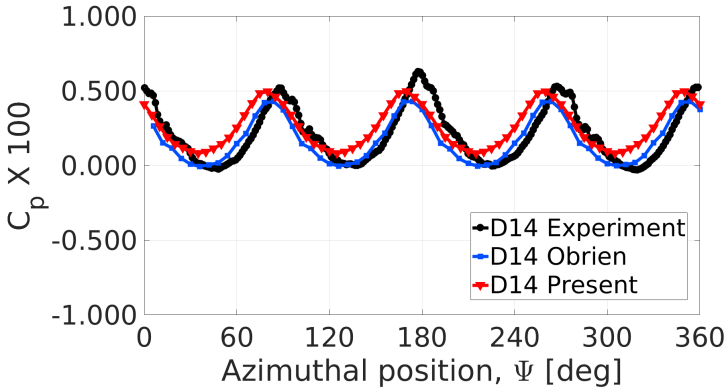

(b) Point D9 Location



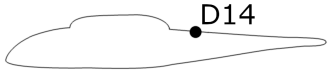

(c) Point D22 Pressure




(d) Point D22 Location

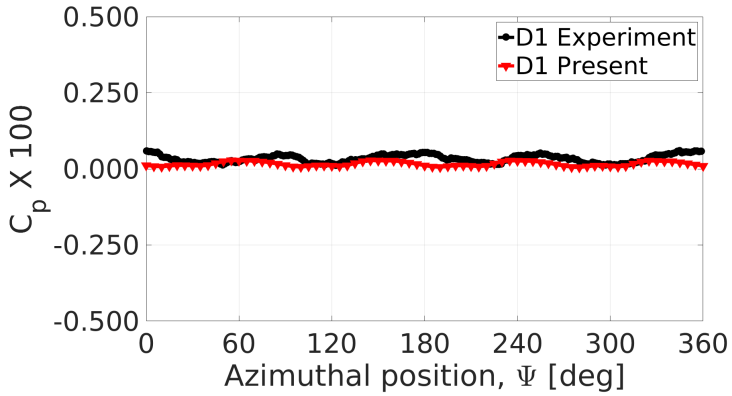



(e) Point D14 Pressure

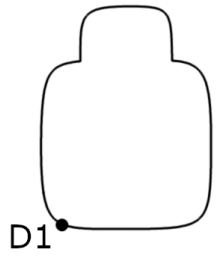



(f) Point D14 Location

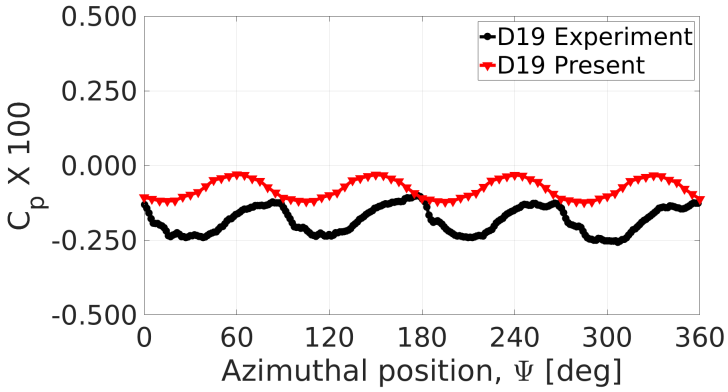
Figure 4.5: Three Center Locations.



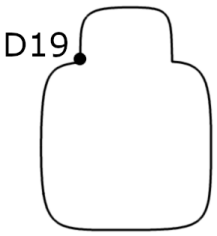

(a) Point D1 Pressure



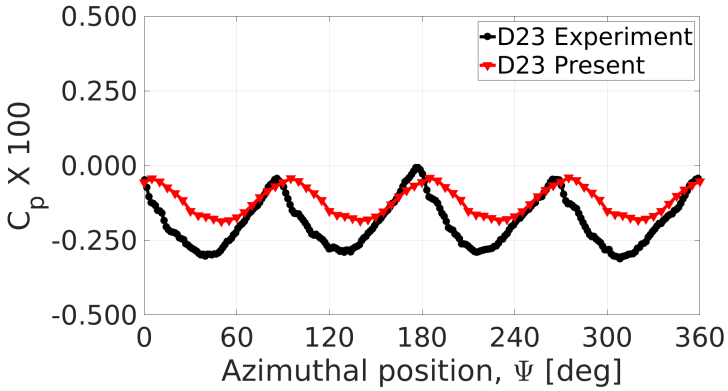

(b) Point D1 Location



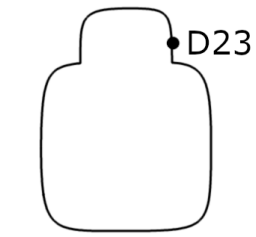

(c) Point D19 Pressure




(d) Point D19 Location




(e) Point D23 Pressure




(f) Point D23 Location

Figure 4.6: Three Lateral Locations.

peaks are the same. The peaks from the present result match better with experiments than the peaks from O'Brien however the valleys from O'Brien match better than those from the present result.

For all points on the lateral cross-section of the fuselage the pressure results from the presented overset framework are only compared to experimental results. Point D1 is on the under-side of the fuselage as shown on the lateral cross-section in Fig. 4.6(b). Neither experiment nor CFD results show meaningful pressure distribution at this point, even with the $\times 100$ multiplier on the pressure coefficient. At an advance ratio of 0.15, the influence from the main rotor is convected backwards fast enough that the under-side of the fuselage experiences very little fluctuations and the pressure is essentially steady.

Point D19 is on the corner of the pylon and main fuselage, as shown on the lateral cross-section in Fig. 4.6(d). The corner point is a very challenging area to mesh properly even with an unstructured mesh. The CFD results show both a phase and magnitude offset from the experiment. The CFD results, however, are smooth compared to the experimental results which show some high-frequency content. The CFD simulation for this data point may benefit from an improved mesh as well as a surface roughness model to capture laminar-turbulent transition and possibly separation.

Point D23 is on the port side of the pylon as shown on the lateral cross-section in Fig. 4.6(f). The present CFD results capture the top peaks of the pressure coefficient however the valleys and overall amplitude of the signal differ from experiment. There is less phase offset that is seen from point D19. The experimental results show a much sharper upper peak to the pressure signal whereas the simulation results are more sinusoidal in shape.

Overall the presented CFD results are able to capture the pressure signature on the fuselage especially in the longitudinal direction. The discrepancies especially for

points near the pylon could be a result of the missing hub and support structure in the CFD simulation. The presence of a hub and support structure may add additional blockage, preventing the blade root vortices from simply convecting downstream.

## 4.2 Rotor + Fuselage + Hub + Tail Simulation

### 4.2.1 Case Description

To demonstrate the full capability of the presented overset framework, a tail rotor and hub are added to the previous ROBIN fuselage and rotor test case. The four main-rotor blades are identical to the previous case. The fuselage mesh is also identical but shifted to be centered under the rotor. A two-bladed tail rotor is simulated with properties described in Table 4.2. Subscripts are provided to distinguish between main and tail values. The Reynolds number is based on the chord and tip Mach number and the $x$-, $y$-, $z$-locations of the tail center of rotation is measured from the center of rotation of the main rotor. The coordinate system for the configuration is shown in Fig. 4.7. The tail rotor spins counter-clockwise about the $y$-axis-direction for a "tractor" design.

| **Description** | | **Value** |
|---|---|---|
| Radius Ratio | $R_{tail}/R_{main}$ | 0.2 |
| Blade Aspect Ratio | | 8.0 |
| Linear Twist | $\theta_{tw}$ | $-6.67°$ |
| Collective | $\theta_0$ | $11.67°$ |
| RPM Ratio | $\Omega_{tail}/\Omega_{main}$ | 4.5 |
| Reynolds Number | $Re$ | $2.52 \times 10^5$ |
| Number of Blades | $N_b$ | 2 |
| $x$-location | $x/R_{main}$ | 1.502 |
| $y$-location | $y/R_{main}$ | 0.1 |
| $z$-location | $z/R_{main}$ | -0.2119 |
| Azimuthal Step | $\Delta\psi_{tail}$ | $0.9°$ |

Table 4.2: Tail rotor properties and flight condition.

The flight condition and main rotor controls are the same as the previous

case from Sec. 4.1. A variation of the forward-flight case is also run to simulate a cross-wind from the port-side of the aircraft. The 33° cross-wind case was selected as a challenging flight condition from a pilot's perspective.

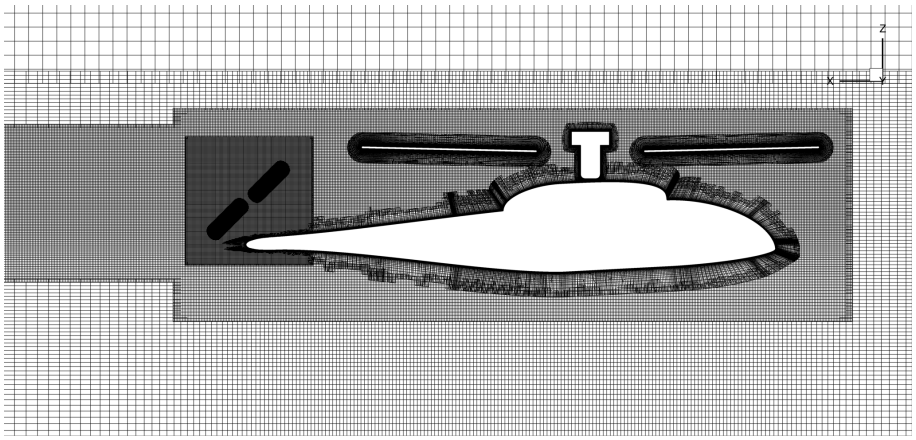



Figure 4.7: Mesh System for full configuration.

A cylindrical, non-rotating hub is added to the mesh system for a first pass at hub-wake analysis. The lower and upper parts of the hub have radii $0.0427R$ and $0.0825R$ respectively. The height of the lower and upper sections are $0.146R$ and $0.06R$ respectively. The hub and fuselage bodies do not overlap to avoid complex connectivity. The hub mesh is unstructured and is run with the HamStr flow solver.

Two additional nested meshes are added to the mesh system used from the rotor-fuselage-only case from Sec. 4.1. The first is a fine, nested mesh with spacing $0.05c$ centered about the tail rotor used to improve wake capturing of the tail rotor. Behind the fuselage, a nested mesh with spacing $0.09c$ extends the existing nested mesh, also for improved wake capture. Because both the existing nested mesh and added wake mesh have the same spacing of $0.09c$ and are coincident at overlapping points, there is no interpolation error between meshes. Both the wake and tail-rotor nested mesh additions are run with the Garfield flow solver.

### 4.2.2 Full Configuration in Forward Flight

Fig. 4.8 shows a top-view of the full configuration solution with iso-surfaces of vorticity colored by $z$-velocity. Shed structures from the wake of the hub are shown to interact with the blade near the $\psi = 0°$ azimuth. Figure 4.9 shows a comparison of the airloads for a single blade between the full configuration (solid) and the previous case results without the tail or hub (dotted). While only a small spike in torque is shown, a large and significant difference in thrust is shown. Furthermore the change in thrust is not always the same, as shown by the over-prediction at $\psi = 0$ and under-prediction one revolution later at $\psi = 360$. The slight phase-shift observed near azimuth $\psi = 180$ is a result of the $y$-direction shift of the rotor. In the rotor-fuselage case the rotor was not centered over the fuselage to match the experiment; in the full configuration, the rotor and hub are centered over the fuselage.

A solution of the full configuration after 4 revolutions is shown in Fig. 4.10 with iso-surfaces of Q-criterion (Q = 0.0008) colored by vorticity. Q-criterion, a relation involving vorticity and strain, is a useful metric for highlighting structures in a flowfield. Flow structures generated from the hub are convected slightly starboard of the helicopter in the direction of the rotating blades. The main rotor wake is well captured by the nested mesh surrounding the rotor and fuselage. Vortices from the main rotor are well defined until they break up passing through the tail rotor.

Fig. 4.11 shows the wake profile behind the helicopter. Contours of $x$-Mach number show the accelerated flow behind the rotor passing directly through the tail rotor. Although the configuration is not trimmed, the average value of stream-wise Mach number behind the fuselage is greater than the free-stream value of 0.08, indicating the rotor is generating forward thrust. The slice $x = 1.502R$, through the tail rotor hub, shows the main rotor wake passing almost directly through the center of the tail rotor plane.

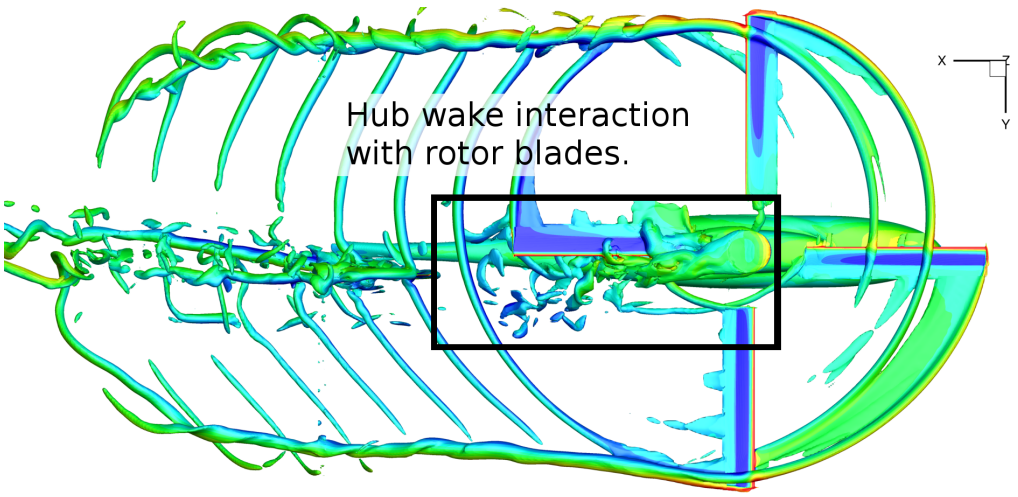



Figure 4.8: Iso-surfaces of vorticity: blade 1 near $\psi = 0$ passes through wake of hub.

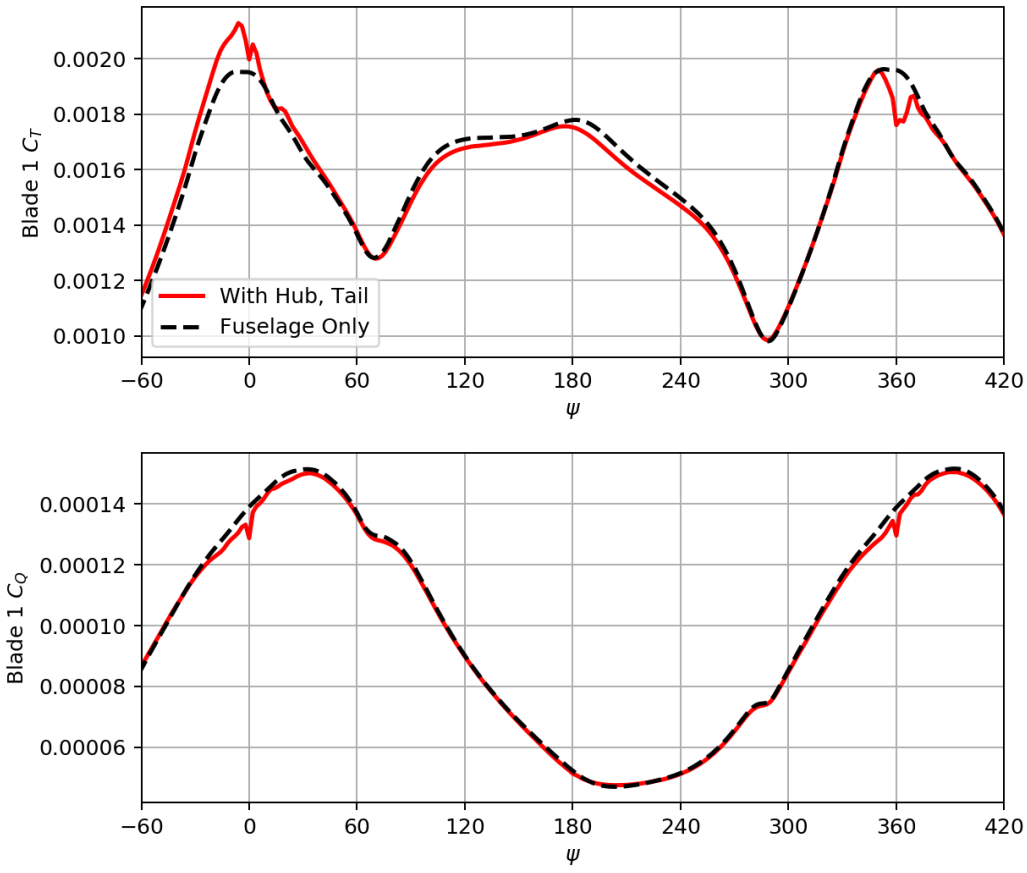



Figure 4.9: Comparison of Blade 1 Loads with and without hub and tail rotor.

Tip vortices from each blade roll up into large, trailed vortices on the advancing and retreating sides behind the aircraft. The tail rotor also shows strong trailed

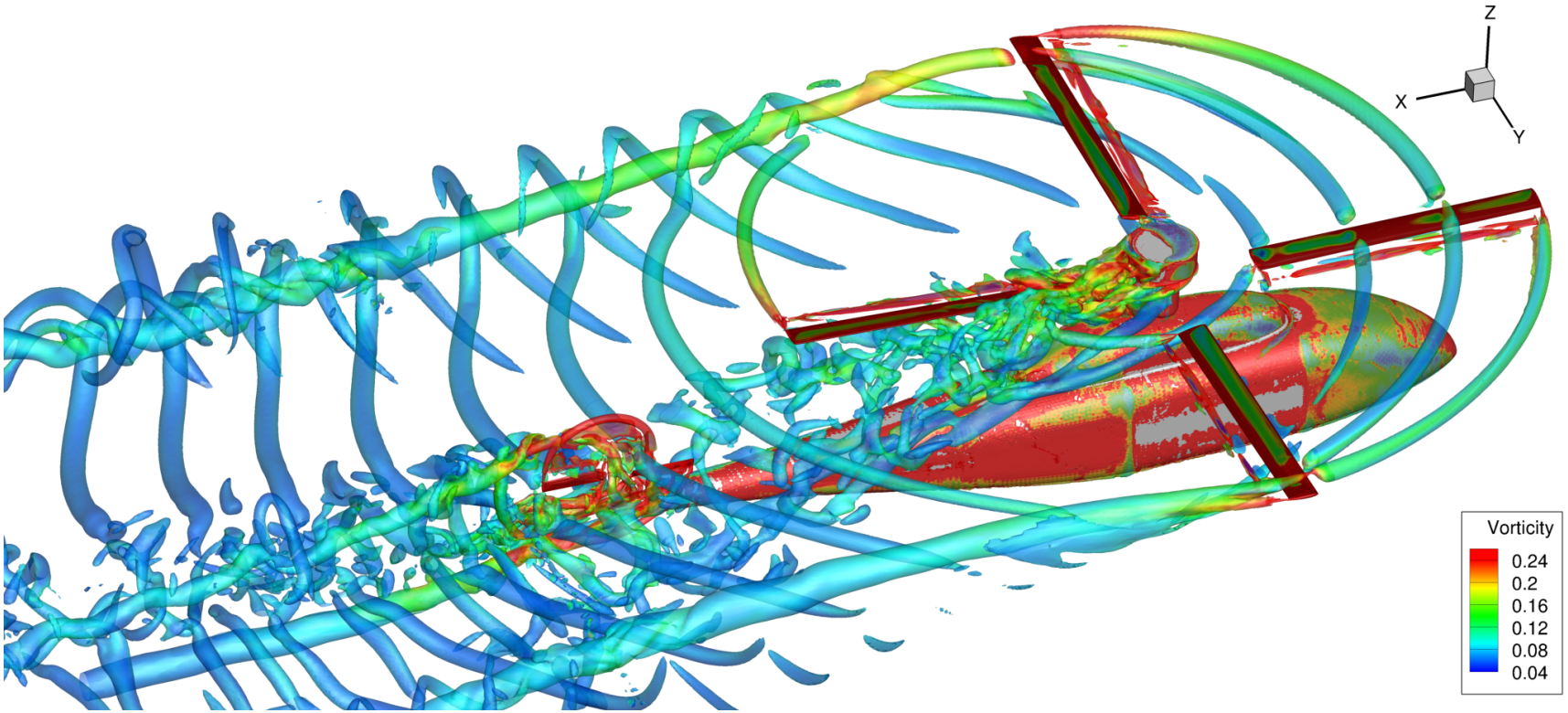



Figure 4.10: Full Solution showing Q-Criterion (Q = 0.0008) colored by Vorticity.

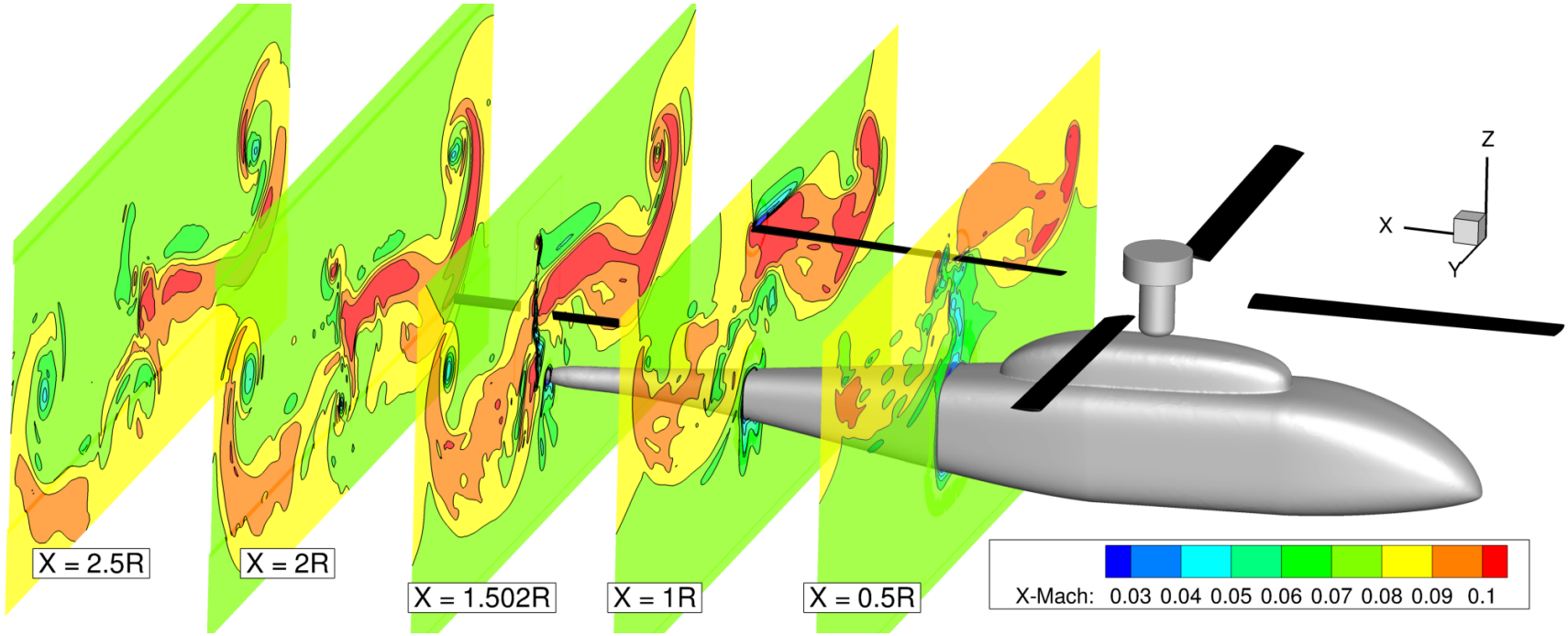



Figure 4.11: Slices behind rotor showing $x$-Mach number.

vortices, although they are harder to identify because they are in the wake of the main rotor and hub.

Just like with the rotor-fuselage-only case from the previous section, Newton-subiterations are used to converge the non-linear residual derivative in time. Between each exchange of overset data between meshes TURNS and HamStr run one subiteration and Garfield does three. Garfield requires the extra subiterations to improve the convergence of the tail rotor, which rotates 0.9° (compared to 0.2° for the main rotor) per timestep. Convergence is most challenging in the blade meshes where skewed cells and large gradients require many subiterations to converge whereas the background meshes need fewer. Even with a total of 27 subiterations ($9 \times 3$), Fig. 4.12 shows less than 1 order of convergence is achieved for the tail rotor. The norm of the volume-scaled residual (from Eq. (2.40)) is used to represent convergence in all cells; plotting the norm of the un-scaled residual bias convergence to that of large cells. After the first 10 subiterations, the convergence of the tail grids appear to stall. In contrast, all nested and background Cartesian meshes achieve more than 2.5 orders of convergence. Garfield uses a Diagonalized ADI scheme to approximate the inversion of the implicit terms from Eq. (2.53); a stronger inversion scheme such as a line Gauss–Seidel method or the addition of a linear solver like GMRES would also help alleviate the convergence issues in the tail rotor meshes. Reducing

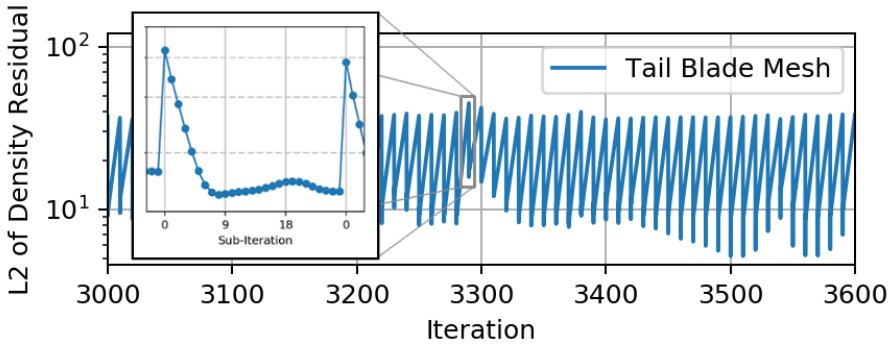



Figure 4.12: Tail-rotor sub-iterative convergence (volume-scaled L2 norm of Density residual).

the timestep of the simulation to 0.25° for the tail rotor would help, however this translates to a 0.056° timestep of the main rotor, 6480 steps per revolution, and a drastic increase in time and resources required for the simulation.

### 4.2.3 Full Configuration in Cross-Wind

A cross-wind from the port-side of the aircraft can result in the large, trailed vorticity from the main rotor blades to enter the tail rotor. Fig. 4.13 shows iso-surfaces of Q-criterion to visualize the location of vortices. The vortex-vortex interactions that trailed behind the aircraft in forward flight now enter the tail rotor plane. The rotation direction of the vortex is opposite to the rotation direction of the tail rotor; this is typically by design on a helicopter to avoid canceling the rotational effect of the tail rotor.

Between Fig. 4.10 and Fig. 4.13 both visualizations use the same iso-surface of Q-criterion yet the cross-wind case shows significantly more chaotic fluid structures. The fuselage is a streamlined body in forward flight; with the cross-wind acting as a large sideslip angle, however, the fuselage operates more as a blunt body and sheds large structures. The tail rotor operates entirely in a "descent" condition; the additional free-stream velocity component introduced by operating the aircraft in a sideslip condition manifests as upwash through the tail rotor disc. The tail rotor blades also shed large, complex fluid structures indicating that the blades are operating at very large angles of attack and have effectively stalled.

The main rotor uses the same blade deflections in both the forward flight case and the cross-wind case. While the forward flight controls are based on trim results from literature, in a cross-wind the main rotor is not trimmed, meaning the flight condition is not necessarily representative of steady, level, sideslip flight; the same goes for tail rotor. Through proper coupling to a rotorcraft comprehensive analysis procedure the flight condition could be studied in detail and simulated using more

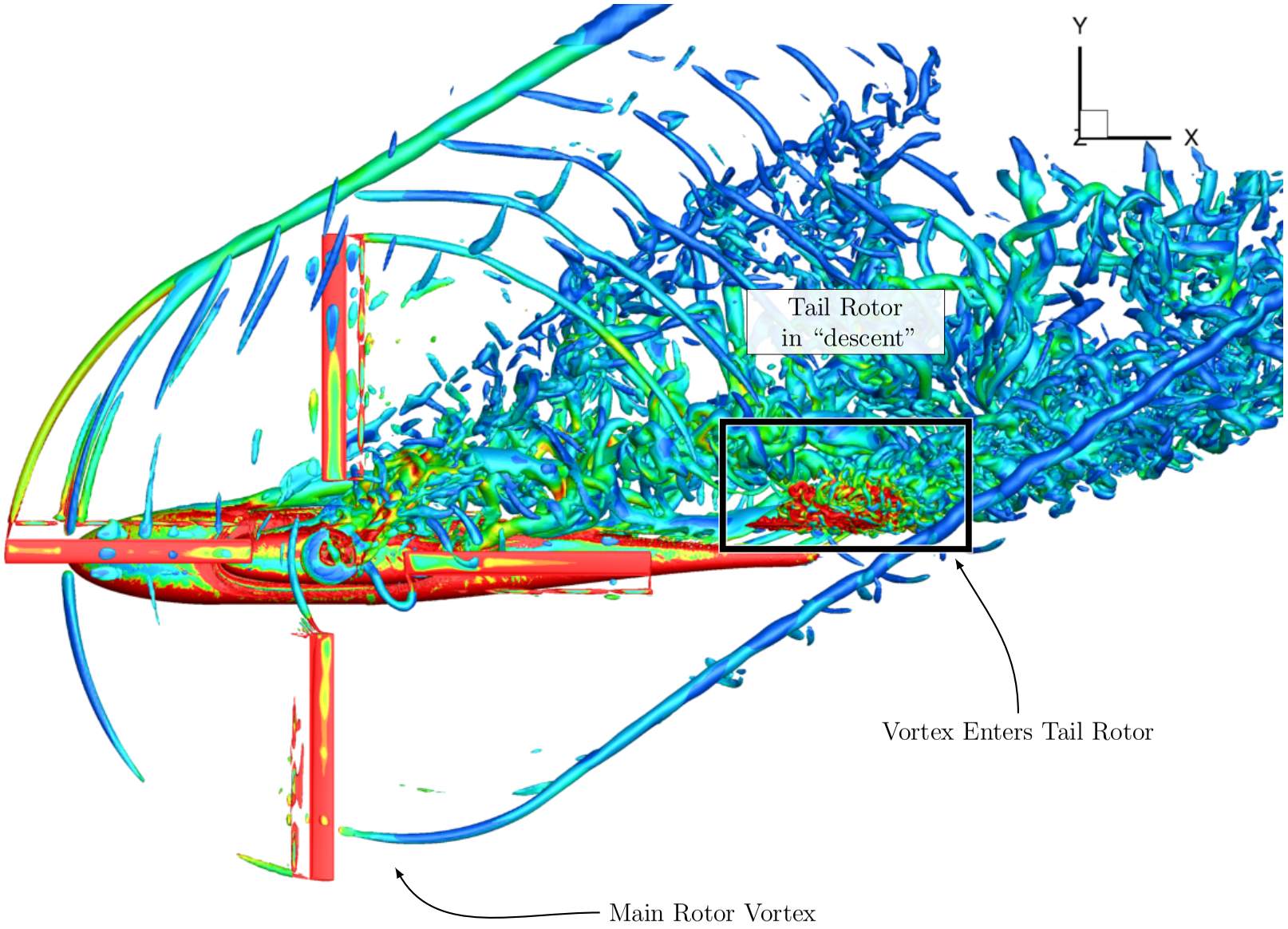



Figure 4.13: Top-view of the cross-wind Q-criterion solution (Q = 0.0008) colored by vorticity.

realistic pilot controls; this is one of the key recommendations for future work.

### 4.2.4 Performance Results

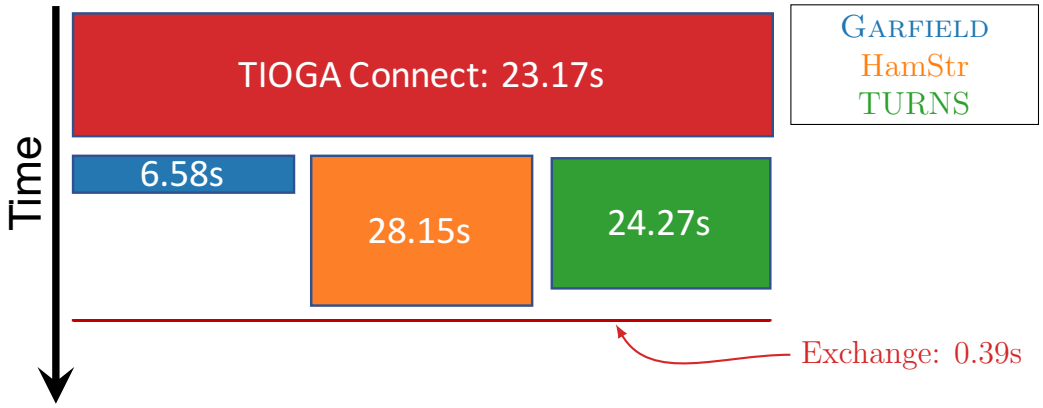
The wall-clock timeline for the simulation is a function of the time taken by connectivity, flow solvers, and data exchange routines. Fig. 4.14(a) shows the timeline for one timestep with each solver taking the same number of sub-iterations. Table 4.3 shows additional metrics for performance analysis. Although GARFIELD handles by far the most number of points, using 6 GPUs the time per iteration is less for GARFIELD than for TURNS or HamStr. The overset framework has to wait for all flow solvers to complete before exchanging data between meshes and therefore the GPUs are sitting idle during this time.

Table 4.3 also shows performance per iteration, per million points. For the fixed set of resources, this metric is one measure of speedup for one code over another for the same number of grid points. For example in this case, GARFIELD on 6 GPUs is running at a $2.04/0.156 \approx 13\times$ speedup over TURNS on 40 CPU cores. The relative speedup multipliers between codes can be used to assign points and resources (CPUs or GPUs) between solvers.

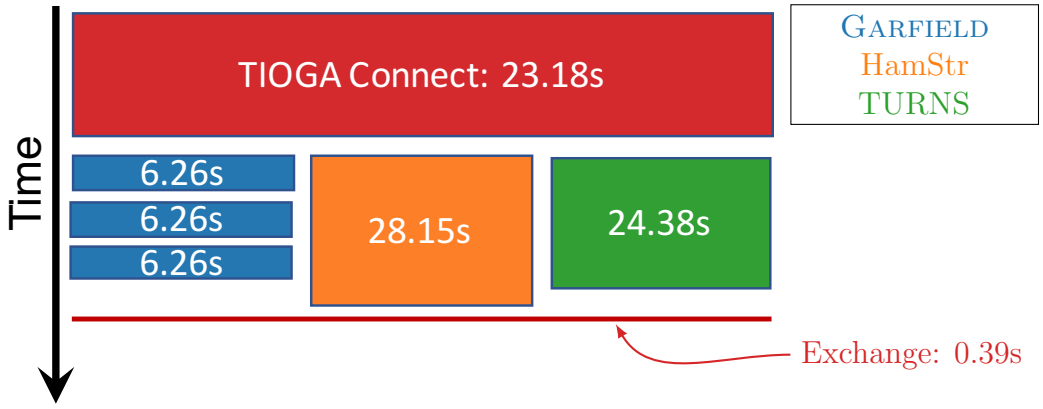
A mis-match between times taken for each solver can also be adjusted by adding a multiplier on the sub-iterations. GARFIELD takes 6.58 seconds using the same number of sub-iterations as HamStr or TURNS; with $3\times$ the number of sub-iterations GARFIELD should still finish before the other flow solvers. Fig. 4.14(b)

| | GARFIELD | HamStr | TURNS |
|---|---|---|---|
| Number of grid points | 42.3 million | 6.9 million | 11.9 million |
| Resources | 6 GPUs | 54 CPU Cores | 40 CPU Cores |
| Time per iteration (s) | 6.58 | 28.15 | 24.27 |
| Time per iteration per million points (s) | 0.156 | 4.07 | 2.04 |

Table 4.3: Solver Timing and Performance.




(a) Equal Sub-iterations for all flow solvers.




(b) 3× Sub-iterations for GARFIELD.

Figure 4.14: Iteration timeline varying sub-iterations in different solvers.

shows the timing result with GARFIELD taking 3× the number of sub-iterations. The effective time per set of sub-iterations decreased from 6.58 to 6.28 seconds because the 3× multiplier is applied within GARFIELD without the overhead from a Python framework call. The extra sub-iterations with GARFIELD can be used to help with the convergence in challenging areas.

## 4.3 Main and Tail Rotor Interactions

The full notional configuration in both forward and cross-wind flight from Sec. 4.2 results in a complex flow-field with interactions between the main rotor,

tail rotor, hub, and fuselage. Interactions between individual components can be better understood by isolating the components and considering sub-sets of the full configuration. This sections considers interactions only between the main and tail rotor. The fuselage and hub are negated and all grids are run using GARFIELD for faster simulation times.

| **Description** | | **Value** |
|---|---|---|
| Radius Ratio | $R_{tail}/R_{main}$ | 0.2 |
| Blade Aspect Ratio | | 8.0 |
| Linear Twist | $\theta_{tw}$ | $-6.67°$ |
| Collective | $\theta_0$ | $11.67°$ |
| RPM Ratio | $\Omega_{tail}/\Omega_{main}$ | 4.0 |
| Reynolds Number | $Re$ | $2.52 \times 10^5$ |
| Number of Blades | $N_b$ | 2 |
| $x$-location | $x/R_{main}$ | 1.502 |
| $y$-location | $y/R_{main}$ | 0.1 |
| $z$-location | $z/R_{main}$ | -0.2119 |
| Azimuthal Step | $\Delta\psi_{tail}$ | $1.0°$ |

Table 4.4: Properties of the tail rotor for main and tail rotor interactions analysis.

The geometries of the main and tail rotors are the same as from Sec. 4.2 however the tail rotor rotation has been slightly slowed to $\Omega_{tail} = 4 \times \Omega_{main}$ instead of 4.5×. The corresponding azimuthal steps of the main and tail rotor simplify to $\Delta\psi_{main} = 0.25°$ and $\Delta\psi_{tail} = 1.0°$. The updated tail rotor properties are summarized in Table 4.4. Three cases are considered for comparison in this section:

- Isolated tail rotor in forward flight.
- Main and tail rotor in forward flight.
- Main and tail rotor in cross-wind flight.

The thrust coefficient of the tail rotor is used as the primary metric for comparison between cases. The magnitude of tail rotor thrust coefficients are compared between cases for analysis of anti-torque efficiency. Furthermore, the frequency

domain of the thrust is analyzed for insight into possible vibration challenges that arise from the interactional aerodynamics.

### 4.3.1 Isolated Tail Rotor: Forward Flight

A tail rotor run without the main rotor in effect becomes a fixed-pitch, low-aspect-ratio rotor problem. The thrust of the tail rotor is plotted against the revolution of the main rotor in Fig. 4.15. The two-bladed tail results in a 2/rev signal with respect to the revolution of the tail rotor. When plotted against a revolution of the main rotor, the 4× multiplier results in a clear 8/rev signal. The average thrust coefficient of the tail rotor is $C_T = 0.00996$.

The 8/rev dominant, expected frequency is further confirmed by analyzing the frequency spectrum of the thrust coefficient. Fig. 4.16 shows the result of a Fourier transform applied to tail rotor thrust in units of 1/rev of the main rotor. The 8/rev is by far the most dominant frequency. Higher harmonics appear primarily at 16/rev and 24/rev. The tail rotor is a fixed-pitch rotor with linear twist and a collective of $\theta_{75} = 11.67°$. The high collective results in a high angle of attack on the interior portion of the blade. The tail rotor also operates at a high advance ratio. The tip Mach number of the tail rotor is $M_{tip} = 0.43$, meaning with a free-stream Macn number of $M_\infty = 0.08$, the advance ratio is 0.186. Separation on the blade as a result of low local Reynolds numbers and high angles of attack are a likely cause of the higher harmonics. In reality, blade motions of the tail rotor also play an important role in the force response however the simulation approximates the tail rotor as rigid.

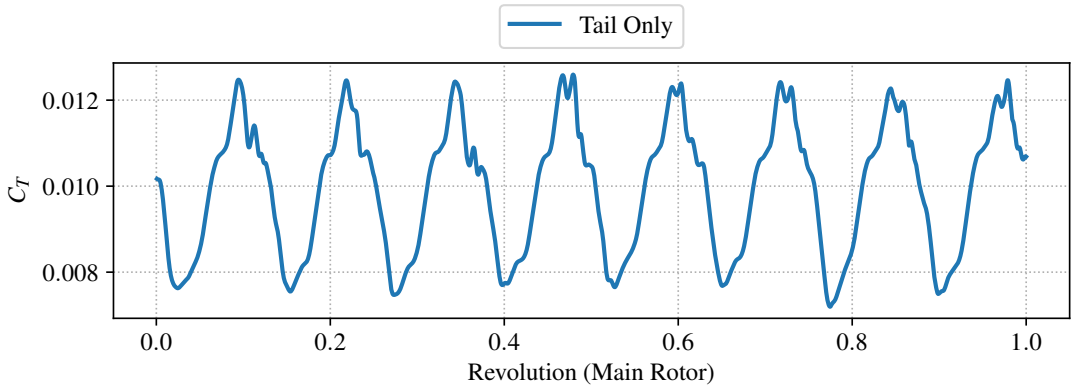



Figure 4.15: Tail rotor thrust coefficient.

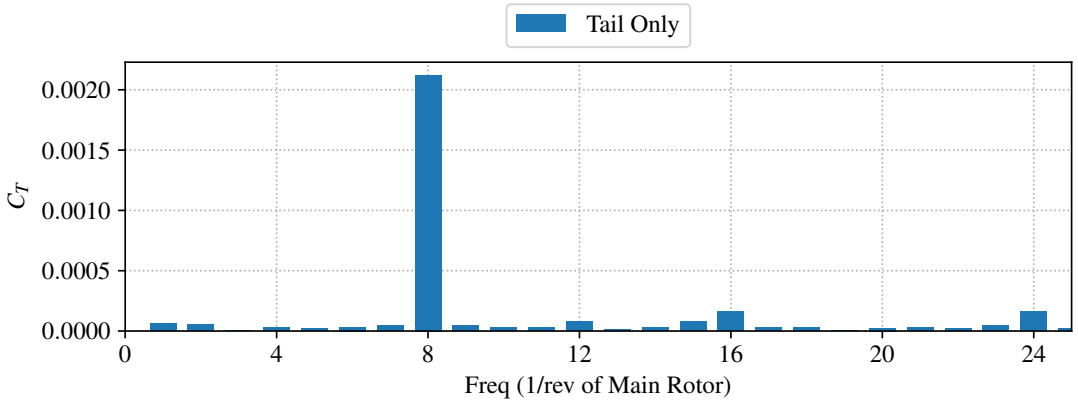



Figure 4.16: Tail rotor mean-removed thrust coefficient frequencies.

### 4.3.2 Main and Tail Rotor: Forward Flight

In the presence of the main rotor, the thrust of the tail rotor can be compared to the isolated tail result. Fig. 4.17 shows the Q-criterion solution colored by vorticity to visualize the flow-field. Vortices from the main rotor can be seen entering the tail rotor at a frequency of 4/rev. Without the presence of a fuselage or hub, the root vortices of the blade combine in to complex structures however they convect starboard of the aircraft and do not appear to interact with the tail rotor.

Fig. 4.18 shows both the thrust coefficients of the isolated tail and main+tail simulations. Again there appears to be a dominant 8/rev. There is a slight phase

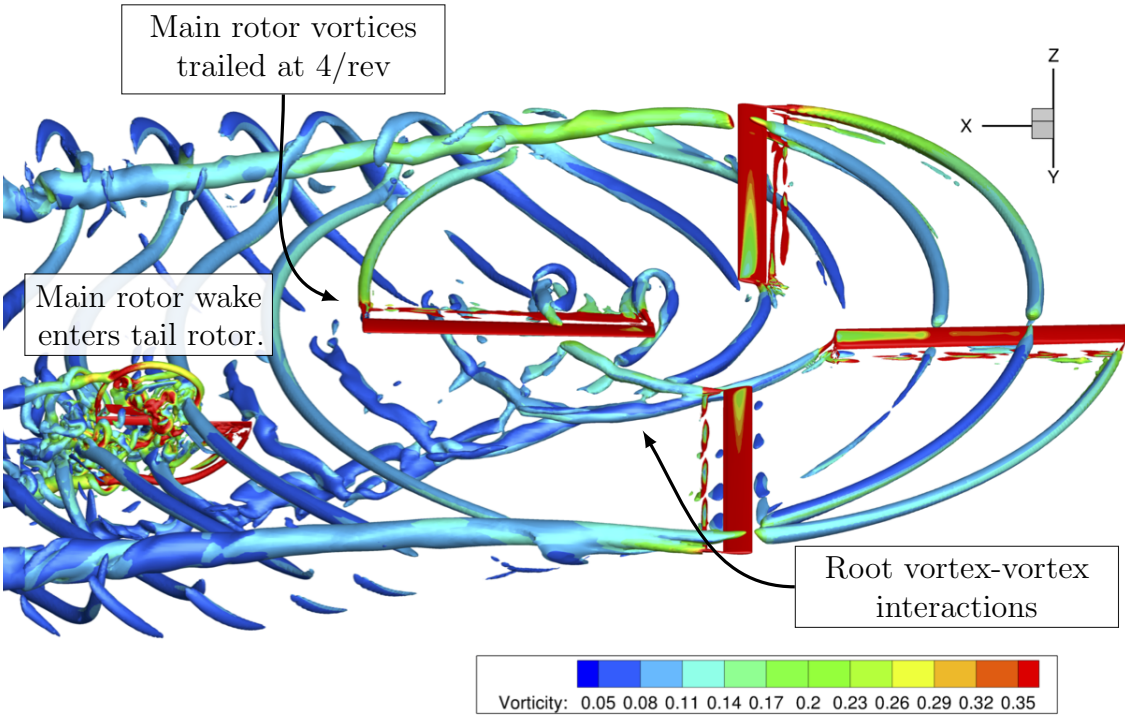



Figure 4.17: Main and tail rotor Q-criterion ($Q = 0.0008$) solution in forward flight

shift in the location of the thrust maxima and minima and the main+tail result displays a dual-peak. The mean thrust of $C_T = 0.01063$ does not change significantly from the value for the isolated tail ($C_T = 0.00996$). The consistent mean-thrust indicates that analyzing the tail rotor in isolation is sufficient for the purposes of determining the anti-torque of the main rotor.

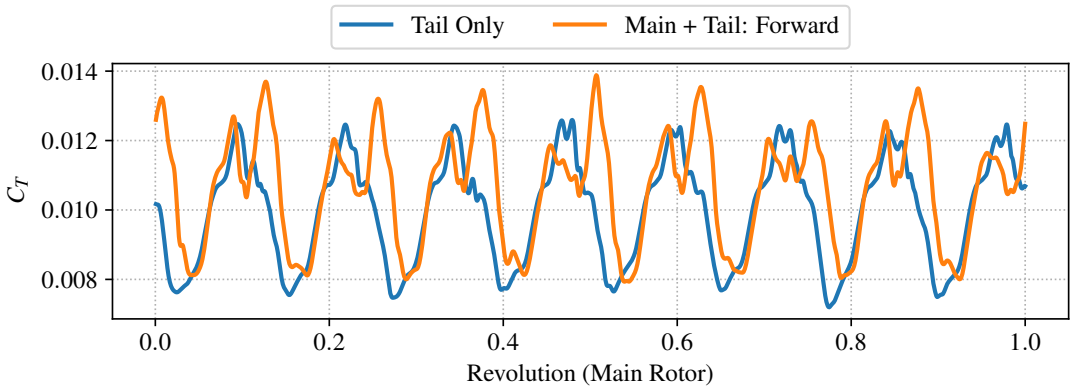



Figure 4.18: Tail rotor thrust coefficient

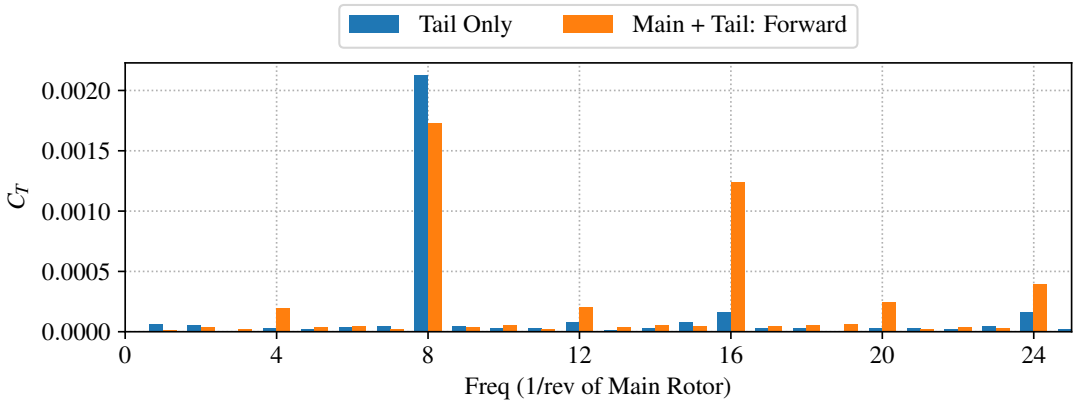



Figure 4.19: Tail rotor thrust coefficient frequencies

The frequency spectrum of the tail rotor thrust is shown in Fig. 4.19 for both the isolated tail and main+tail cases. The main+tail case shows a slightly lower 8/rev frequency however this is still the most prevalent frequency. Interestingly there is a very strong 16/rev component that appears only in the main+tail result. The 16/rev is therefore a result of aerodynamic interactions between the main and tail rotor. The 16/rev can be justified by considering the thrust of a single blade:

$$\begin{aligned} T_{blade} &= \frac{\rho}{2}\int_0^R v_{eff}^2 \cdot C_l dr \\ &\propto v_{eff}^2 \end{aligned} \tag{4.1}$$

where $v_{eff}$ is the effective velocity experienced by the blade. $C_l$ is the lift coefficient at each blade section, presumed to be well behaved for thin airfoils. For a simple isolated rotor in free-stream forward flight, the $v_{eff}$ has a $1/\text{rev}_{tail} = 4/\text{rev}_{main}$ component. Hypothesizing that the main rotor produces an inflow with a $4/\text{rev}_{main}$ component, the effective velocity seen by a tail rotor blade results in 8/rev and 16/rev components:

$$
\begin{aligned}
v_{eff} &\sim sin(4\psi) \cdot sin(\psi_{tail}) = sin^2(4\psi) \\
v_{eff}^2 &\sim sin^4(4\psi) \sim \alpha_1 cos(8\psi) + \alpha_2 cos(16\psi)
\end{aligned}
\tag{4.2}
$$

The 8/rev for each blade results in an overall 16/rev for the tail rotor. The decomposition from the $sin^4()$ term is performed using trigonometric identities.

The simulations results also predict a 4/rev signal for the tail rotor as well as higher harmonics at 12/rev, 20/rev, and 24/rev. Depending on the type of engine and frequency of operation, these higher harmonics may excite engine support and airframe modes. The higher harmonics may also result in intrusive vibrations for the pilot or passengers.

### 4.3.3 Main and Tail Rotor: Cross-wind Flight

The final flight condition for comparison is with the free-stream at a 33° angle from the port side of the aircraft, creating an effective cross-wind. The cross-wind case is used to represent a challenging flight condition from the pilot's perspective because the rolled up tip vortices from the retreating main-rotor blades are convected directly into the tail rotor. The vorticity entering the tail rotor can cause unpredictable loads and vibrations resulting in challenging pilot scenarios.

Fig. 4.20 shows the Q-criterion solution colored by vorticity to visualize the flow-field. In hover or straight forward flight, the typical operation of the tail rotor results in inflow from the starboard side of the aircraft to the port side. As was seen from the full configuration solution in cross-wind from Sec. 4.2.3, the inflow through the tail rotor passes in the opposite direction, from port to starboard. The tail rotor is therefore in a "descent" condition. The tail rotor controls, however, remain at the same collective angle $\theta_{75} = 11.67°$ meaning the entire blade is operating at very high angles of attack. The complex vorticity downwind of the tail rotor indicates the flow

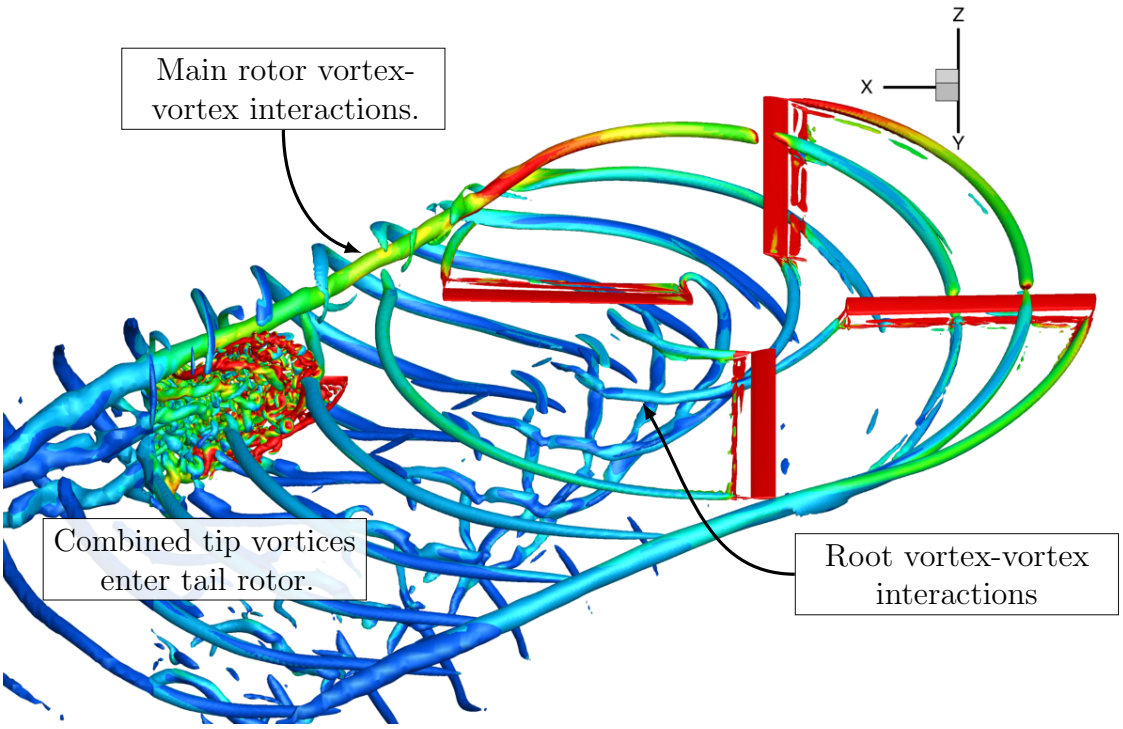



Figure 4.20: Main and tail rotor Q-criterion ($Q = 0.0008$) solution in cross-wind flight.

is largely separated on the blades.

The thrust coefficient of the tail rotor for the cross-wind case is compared to the two forward flight cases (isolated tail, main+tail) in Fig. 4.21. The average thrust stays approximately the same for both forward flight cases. With a cross-wind, however, the average thrust increases significantly. The average thrust for all three compared cases is shown in Table 4.5. The high thrust of the tail rotor in cross-wind is likely a result of the drag component of force and not lift. For the high angles of attack experienced by the tail rotor blade, the blades operate almost as flat plates perpendicular to the flow, with the drag direction aligned with the direction of the desired thrust. So even though the tail rotor blades may be stalled, the rotor still produces much larger counter-torque to the main rotor and may result in corrective yawing motion of the helicopter. As the helicopter yaws left, the tail rotor may enter its own wake resulting in additional complex aerodynamic interactions.

| Case | Average $C_T$ |
|---|---|
| Isolated Tail, Forward Flight | 0.00996 |
| Main+Tail, Forward Flight | 0.01063 |
| Main+Tail, Cross-wind | 0.01545 |

Table 4.5: Average tail rotor thrust coefficient for three cases.

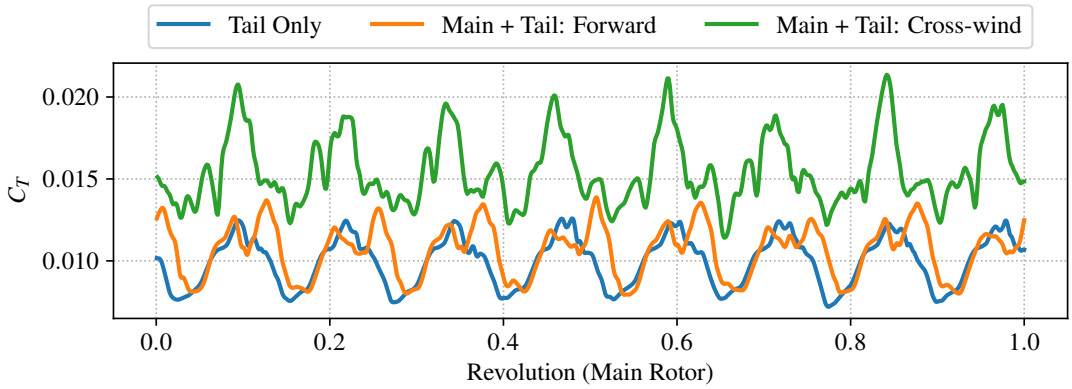



Figure 4.21: Tail rotor thrust coefficient

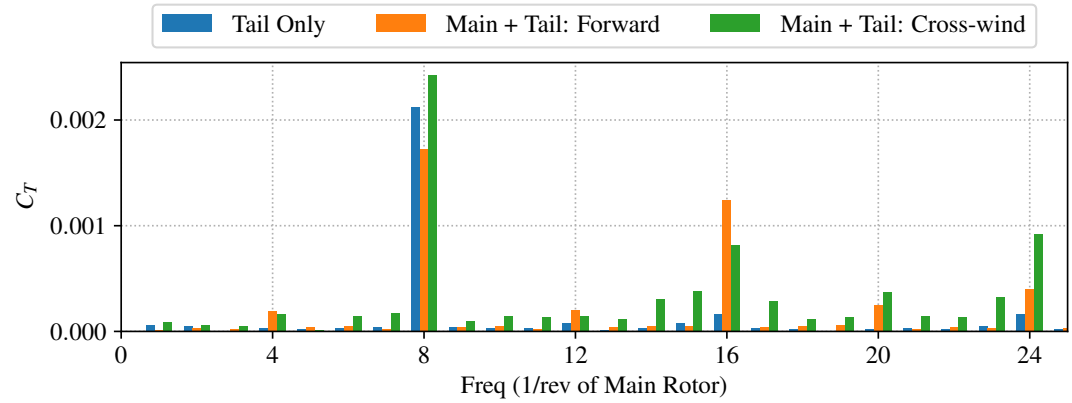



Figure 4.22: Tail rotor thrust coefficient frequencies

The thrust coefficient from Fig. 4.21 also shows more frequency content, as further confirmed by the frequency spectrum plot in Fig. 4.22. The 8/rev is still the dominant frequency, consistent with both forward flight cases. The 16/rev is also prevalent for the cross-wind case, though it is not as strong as the forward flight main+tail case. Notably there is frequency content spread over a much larger range

of high frequencies. With the exception of the 16/rev, all frequencies above 12/rev are higher with a cross-wind. The 24/rev vibratory load amplitude for the cross-wind case is stronger than that of the 16/rev. Depending on the type of engine used in the aircraft, the high frequencies could be in the range of engine resonance for turbine engines.

## 4.4 Conclusions

In this chapter the developed, heterogeneous CFD framework has been applied to a notional helicopter configuration representative of a complex, large-scale rotorcraft application. The cases from this chapter could not be run on CPUs and GPUs together prior to the development of the presented CFD framework. Single-solver or CPU-only frameworks from state-of-the-art CFD software like Helios [38] could simulate the same problems, however at a higher cost of supercomputer cluster usage and without novel compute architectures.

The first case simulated the ROBIN wind-tunnel experiments performed at NASA Langley. The experiments recorded pressure variation at points on the fuselage with a four-bladed main rotor in forward flight. The CFD framework was able to successfully integrate three separate CFD solvers: HamStr, TURNS, and Garfield for the simulation. Overall the pressure results match well between experiment and simulation. Discrepancies between the results especially on the lateral cross-section are likely a result two different sources of error. First, the CFD does not model the support structure or hub for the rotor, nor the support structure of the fuselage. The presence of the rotor support and hub likely cause additional blockage, which would affect the pressure fluctuation amplitude on the fuselage. Second, the CFD does not match the rotor control angles used in experiment. Other research groups had similar issues matching the rotor thrust for the control angles used in the experiment. Instead, the collective pitch of the rotor is adjusted to match the desired thrust

condition. The cyclic pitch values are matched from values in literature and also differ from the experiment. The mis-match could be an offset in the measured blade pitch in the experiment due to elastic deflection or merely experimental error. Another possibility for the discrepancy is correct modeling of any interference or blockage due to the wind tunnel geometry. The wind tunnel is not modeled in CFD.

The second case builds upon the rotor and fuselage case with the addition of a hub and tail rotor. The hub is cylindrical and did not rotate for preliminary analysis. The tail rotor was designed with a symetric airfoil and linear twist; the tail rotor collective was picked based on approximations for main-rotor counter-torque. The multi-rotor system demonstrates the ability of the framework to handle multiple rotors with different axes of rotation. The full configuration, run with HamStr, TURNS, and Garfield, was analyzed for performance by comparing the timeline of events occurring in parallel. Garfield, operating on GPUs, runs faster than either of the two CPU codes despite having significantly more grid points. By assigning a multiplier on the number of sub-iterations for Garfield, the framework benefits from additional convergence without any added wall-clock time.

The final section analyzed three variations of the main and tail rotor without the hub or fuselage with the goal of demonstrating rotor-rotor interactional aerodynamic analysis. The tail rotor with a main rotor in forward flight shows a distinct 16/rev component of thrust that is solely from the presence of the main rotor. The interaction can be justified in an analytical manner with a lower-order blade-element model however without the full CFD the interaction is not numerically precise. The addition of a cross-wind from the port side of the aircraft is intended as a challenging case from the pilot's perspective. With a 33° crosswind the trailed vortices from the main rotor approach the tail rotor and the tail rotor enters a "descent" flight condition. The thrust on the tail rotor increases significantly as a result of the blades operating at very high angles of attack. The tail rotor blades are fully stalled, resulting in

potential control reversal; decreasing the tail rotor pitch may result in increased thrust. The cross-wind case produced additional high-frequency content to the tail rotor thrust. The high-frequency content presents potential issues for vibrations and engine resonance.

Interactions between the main and tail rotors are analyzed by looking at the forces on the tail rotor but the selected flight controls are based on approximations. A more formal analysis of a real configuration would require coupling to a comprehensive rotorcraft analysis tool to trim the aircraft.

The numerical convergence is a primary challenge in rotorcraft simulations because the physical timestep may not be optimal in all parts of the domain. For reasonable simulation times the main rotor may run with a $0.25^\circ$ timestep, however this results in a much larger azimuthal timestep for the tail rotor. The large timestep can cause the tail blades grids to not converge very well, thus bringing into question the validity of the skin friction and forces reported from that region. The challenges faced in overset convergence, however, are characteristics of most overset frameworks which use multiple solvers for complex problems.

Wake complexity in the region behind the hub and tail rotor indicates finer mesh resolution could be used to better resolve interactions. Reducing the cell size to $0.05c$ would improve the interpolation at the boundaries of the blade mesh and also reduce dissipation. Use of the $5^{\text{th}}$ order WENO scheme with Jiang and Shu weights could also be changed to a simple central discretization since there are no discontinuities (like shocks) in the wake region.

Although the presented variations of the ROBIN fuselage and rotor are notional configurations, the simulations challenges and conclusions apply generally to future analysis performed for a real case. The ability to simulate complex interactional problems and extract meaningful engineering conclusions lays the ground-work for CFD as a more efficient and approachable tool for rotorcraft design and analysis.

# Chapter 5: Multi-Solver Overset-GMRES

This chapter presents two applications of the O-GMRES method with the goal of accelerating convergence for unsteady, multi-solver simulations. Analysis of results is performed from the perspective of the numerical convergence, which is governed by the implicit algorithm used to invert the left-hand-side of the governing discretized Navier–Stokes equations Eq. (2.40). Three different formulations for the implicit operator are compared:

1. Preconditioner alone applied to individual meshes. This approach is referred to as "iterative" or by the name of the preconditioner (ie. DADI, Gauss–Seidel).

2. GMRES applied to individual meshes with "frozen" interpolation boundaries. The method is referred to as "GMRES."

3. GMRES applied to the full overset mesh system with implicit treatment of interpolation boundaries. This method is referred to as "O-GMRES."

Two different solvers are used for convergence analysis: Garfield and mStrand. Garfield is used for the background region and mStrand is used for the near-body region. All cases are unsteady, meaning sub-iteration convergence is the primary metric for convergence analysis. In each physical timestep, sub-iterations are used to reduce the linearization error. Within each sub-iteration, the chosen implicit method may use multiple linear iterations in solving the linear system. Pseudo-code for the different time-marching iterations is shown in Alg. 5

Algorithm 5: Time Marching Loops

```
1: for t = 1, 2, ..., T do                                  ▷ Physical Time Iterations
2:     for τ = 1, 2, ..., P do                              ▷ Pseudo Time (Non-linear) Iterations
3:         for k = 1, 2, ..., m do                          ▷ Linear Solver Iterations
4:             ...                                      } Linear Solver
5:         end for                                      } (ie. GMRES, Gauss–Seidel)
6:     end for
7: end for
```

For every timestep $t$, the residual should drop with each sub-iteration $\tau$. Similarly within each sub-iteration $\tau$, the residual should drop within each linear iteration $k$. In both the iterative and traditional GMRES formulations (1 and 2 above), the linear solver loop on line 3 of Alg. 5 is typically hidden inside of the flow solver. In the O-GMRES formulation, the linear solver iterations are performed at the framework level and therefore calls to TIOGA for interpolation between domains can be built into the linear solver.

All cases considered in this chapter are assumed laminar and run without a turbulence model. O-GMRES applied to a multi-solver system at the framework level is a novel approach and therefore considering only the laminar mean-flow equations is the first step towards demonstrating usefulness of the method.

The first section in this chapter presents a shedding, laminar sphere at low Reynolds number. The second presents a laminar rotor designed for use on a Mars helicopter. Both cases involve unsteady fluid structures passing between overset boundaries which can be challenging to sufficiently converge for large timesteps. The final section summarizes findings.

## 5.1 Laminar Flow Over a Sphere

Time-accurate, laminar flow over a sphere is considered with one body mesh and three nested background meshes. The sphere mesh is a "strand-type" [77] mesh

run with mStrand and all background meshes are run with GARFIELD. The Reynolds number is set to $Re = 800$ in order to ensure shedding occurs, convecting unsteady flow structures between meshes. The mesh system is shown in Fig. 5.1. The entire domain is initialized with free-stream values and run to time $t^*$ when the vortices have started shedding behind the sphere. The flow solvers are re-started from time $t^*$ with three time-stepping methods: Iterative, GMRES, and O-GMRES. GARFIELD uses DADI as a preconditioner whereas mStrand uses 5 sweeps of the multi-colored line Gauss–Seidel algorithm.

For a sphere of unit diameter $D = 1m$, background grid spacings of $0.07m$, $0.14m$ and $0.28m$ are used for the three levels of nested Cartesian meshes. Cell volumes in the finest mesh match the approximate volume of the outer-most cells of the 404,600 point near-body mesh. The sphere mesh uses wall spacing of $1 \times 10^{-3}m$ ($y+ \approx 0.1$). The freestream Mach number is 0.2, based on $340m/s$ speed of sound. Small and moderate timestep sizes are used for convergence analysis. The small and moderate timesteps correspond to CFLs of 1.71 and 3.42 in the finest nested Cartesian mesh. In mStrand, Newton timestepping is used with an infinite dual timestep $\Delta\tau = \infty$. In GARFIELD, Quasi-Newton timestepping is used with a local dual-timestep CFL of 200. GARFIELD uses a $5^{\text{th}}$ order WENO [23] spatial reconstruction scheme for all background meshes.

The solution at time $t^*$ is shown in Fig. 5.1. Contours of pressure coefficient are shown on the $y = 0$ slice of the sphere and iso-surfaces of Q-criterion show the vorticity pattern shed behind the sphere. Vortices can be seen convecting through the boundaries of each grid.

For a fair-cost comparison between the Iterative and GMRES methods, the Iterative method uses 20 times the number of sub-iterations to match the 20 Krylov iterations performed with GMRES and O-GMRES. In Figs. 5.2 and 5.3 every $20^{\text{th}}$ sub-iteration is marked for comparison with GMRES and O-GMRES. In the low

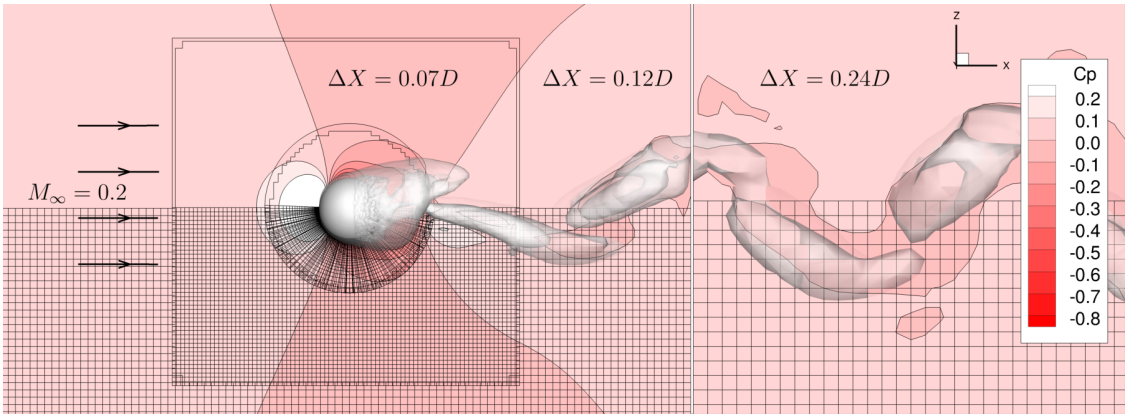
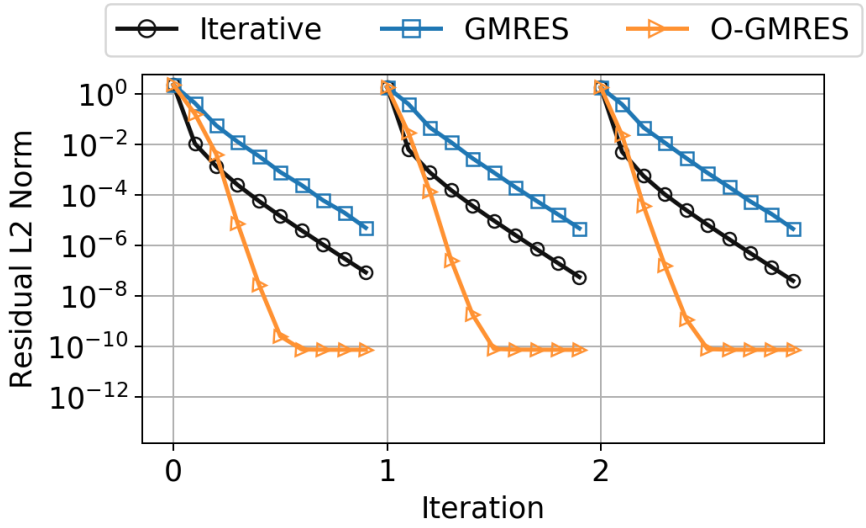


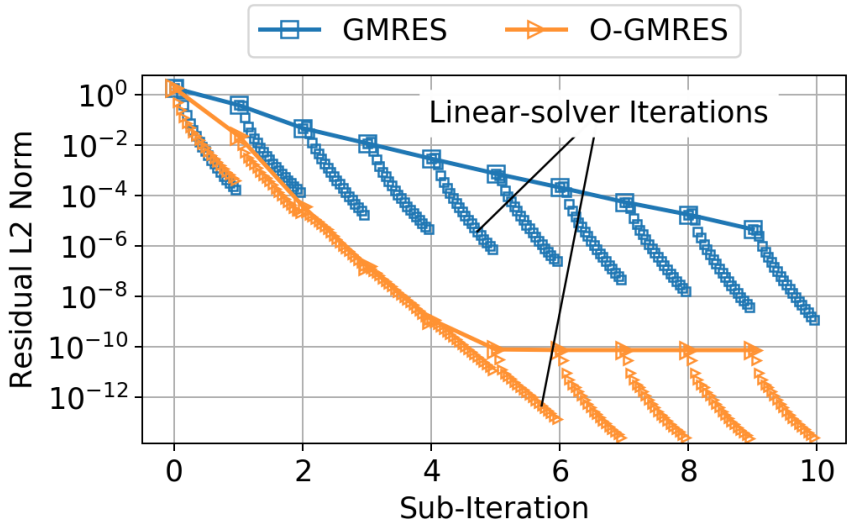

Figure 5.1: Iso-surface of Q-criterion (0.0001) with contours of Pressure Coefficient at time $t^*$.

CFL case, shown in Fig. 5.2, the convergence slopes for the iterative and traditional GMRES methods are approximately the same. The slope for the O-GMRES method is much higher, converging to $10^{-10}$ within 5 sub-iterations before stalling because of the errors from finite difference approximations of the matrix-vector product. The linear solver convergence from Fig. 5.2(b) shows that the GMRES linear solver is converging very well but this does not translate to good convergence of the full overset system. As a result the GMRES case shows a "saw-tooth" pattern with fast convergence in the linear solver but slow convergence overall. The residual reported from the linear solver is based on the Hessenberg least-squares procedure described previously. As a result of the "frozen" fringe points the residual does not reflect changes in interpolated quantities.

In the moderate CFL case, shown in Fig. 5.3(a), the convergence of the Iterative method reduces drastically. The GMRES and O-GMRES methods converge at similar rates however the inclusion of the implicit overset terms results in faster initial convergence for O-GMRES. Compared to the low CFL case there is less "saw-toothed" behavior for the GMRES method, indicating the convergence of the linear residual is not straying far from the convergence of the actual full overset system. The similar rates of convergence between GMRES and O-GMRES indicates




(a) Time-accurate Convergence

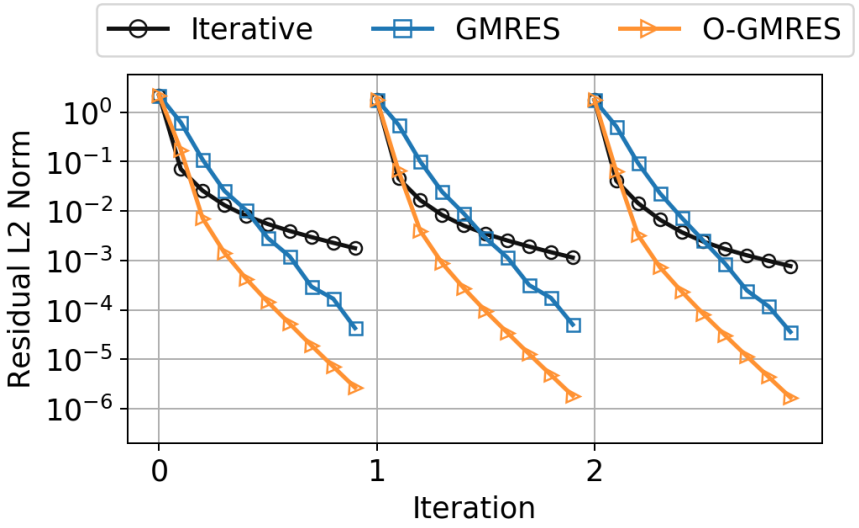



(b) Iteration 2 Non-linear and Linear Convergence

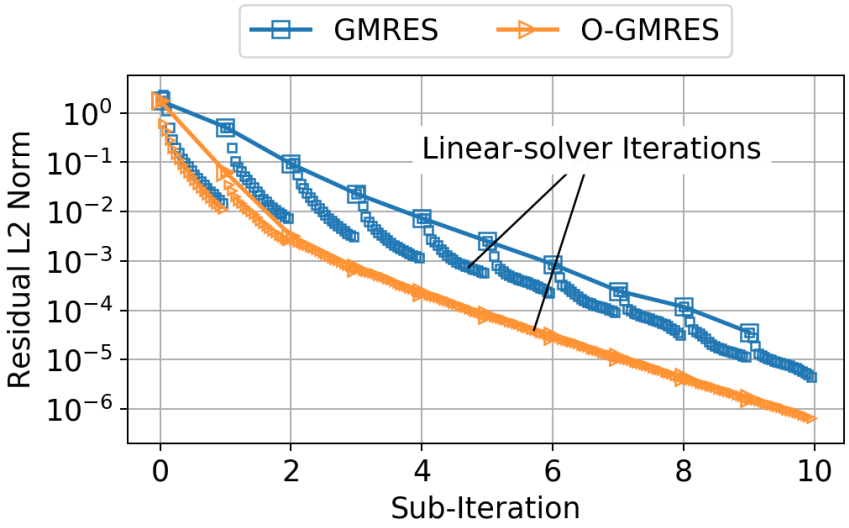
Figure 5.2: Small timestep (CFL=1.71) non-linear and linear convergence.

the convergence rate for this case is limited more by internal mesh flow features and less from overset boundaries.

A few patterns emerge from the low and moderate CFL cases. First, the Iterative method appears to do very well initially. This is slightly misleading since the Iterative method is taking 20 times the number of sub-iterations as the GMRES methods. As a result, the Iterative method has a better approximation of the backwards differencing source term from Eq. (2.40). Second, the linear-solver convergence for O-GMRES follows the overall sub-iterative convergence for the global system except for the first sub-iteration. In the first sub-iteration, the linearized system is a poor approximation to the non-linear equations.




(a) Time-accurate Convergence




(b) Iteration 2 Non-linear and Linear Convergence

Figure 5.3: Moderate timestep (CFL=3.42) non-linear and linear convergence.

## 5.2 Laminar Coaxial Rotors

The second case for O-GMRES analysis considers a set of laminar, coaxial rotors for a more realistic and complex problem. In most full-size rotorcraft applications where Reynolds numbers are in the millions, purely laminar flow is a poor assumption of the rotor physics. At the micro unmanned-aerial-vehicle (UAV) scale or in different climates, the Reynolds number may be significantly reduced and laminar flow is a useful assumption. Recent research at the University of Maryland on low-Reynolds number rotor blades and vehicle design for a Mars helicopter present a unique and interesting case for CFD analysis. Winslow et. al. researched low-Reynolds number blade performance resulting in a thin, cambered flat-plate design [78]. Escobar et. al. [79] combined the work of Winslow with design choices from the University of Maryland Helicopter Design Team from 2000 [80] to design a rotor for the Martian atmosphere, where Reynolds number is low and Mach number is high.

The Mars rotor from Escobar et. al. is run with mStrand for the four blades of the coaxial system and Garfield for all background meshes. For the selected case, the untwisted blades have an aspect-ratio of 4.61, root cutout of 13%, tip Mach

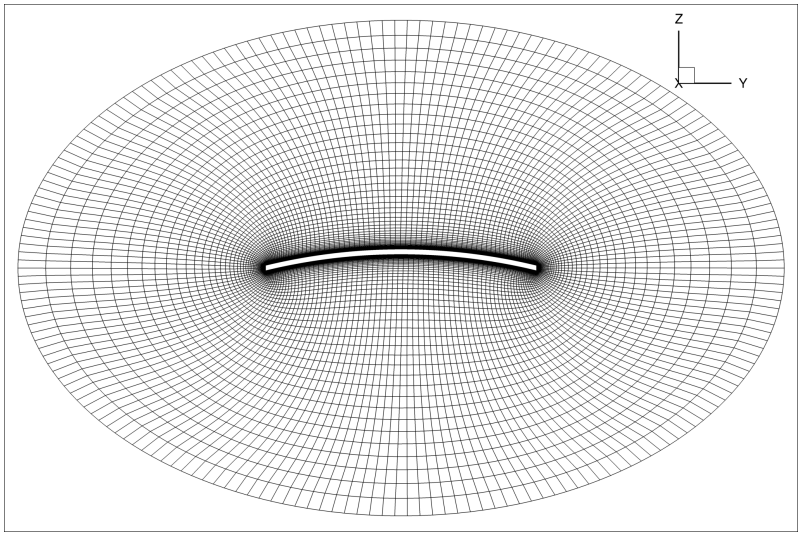



Figure 5.4: Blade cross-section showing cambered, 2% thick flat plate.

number of $M_{tip} = 0.21$, Reynolds number of $Re = 10,000$, and collective of $\theta = 10°$. The cambered flat plate airfoil, shown in Fig. 5.4, is 2% thick with slight camber and sharp leading and trailing edge corners. The spacing between rotors is set to 10%R. Each blade has $2.85 \times 10^6$ points with wall spacing of $5.5 \times 10^{-6}$ chords. The wall cell size corresponds to a very small non-dimensional wall spacing of $y+ \approx 0.003$.

The ratio of specific heats, $\gamma$, is kept at 1.4 for the value on Earth. The solution was advanced to time $t^*$ with the Iterative method for 3 revolutions with azimuthal timestep $\Delta\psi = 0.5°$. The solution after 3 revolutions at time $t^*$ is shown in Fig. 5.6 and is the point from which convergence studies are restarted.

The grid system is shown in Fig. 5.5. Four blades meshes are used, two per rotor, each with 2.8 million points. Both clock-wise and counter-clockwise blades are identical mirror images of each other. The three nested background meshes from fine to coarse contain 1.7, 8.9, and 4.6 million points respectively. With the exception of

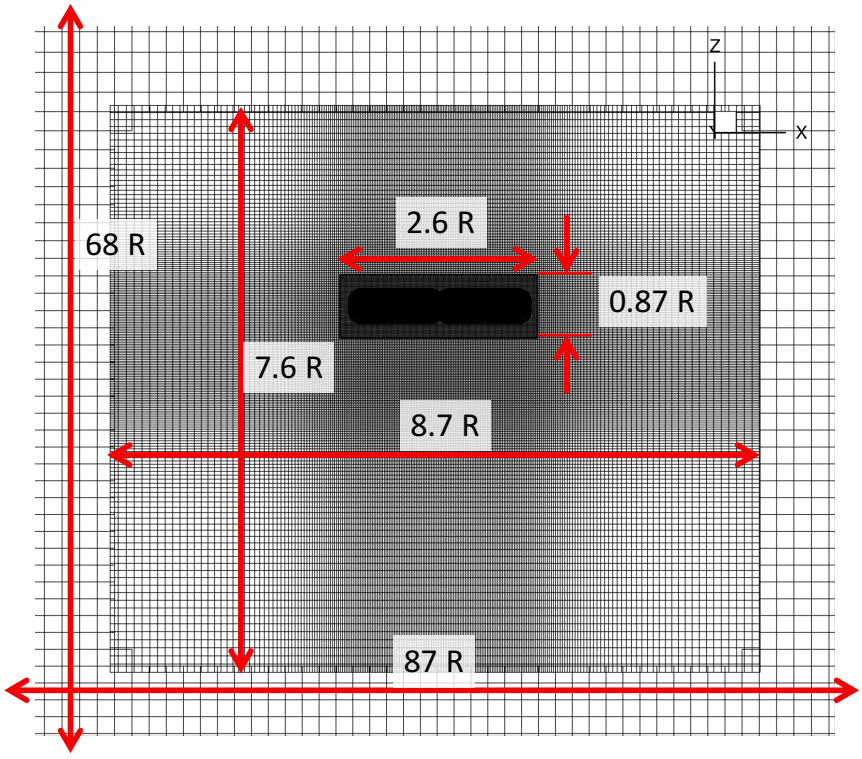



Figure 5.5: Overset grid system for the laminar coaxial rotors.

the finest level, which has uniform spacing of 7% chord, the other background meshes use a hyperbolic tangent function to smoothly stretch the grid in each direction. In mStrand, the multi-colored Gauss–Seidel method is used for the blade meshes. In the background Garfield used DADI for all meshes instead of the red-black method. Results titled as "Iterative" for this case corresponds to DADI applied in the off-body with multi-colored Gauss–Seidel in the near-body and without the use of any linear solver.

At restart point $t^*$ the rotors are aligned along the X-axix. Convergence is compared over 10 timesteps of $\Delta\psi = 1°$, after which the upper and lower rotors have rotated $+10°$ and $-10°$ respectively about the +Z axis. The rotor positions at five azimuths of blade passage are shown in Fig. 5.7, with the upper rotor colored orange

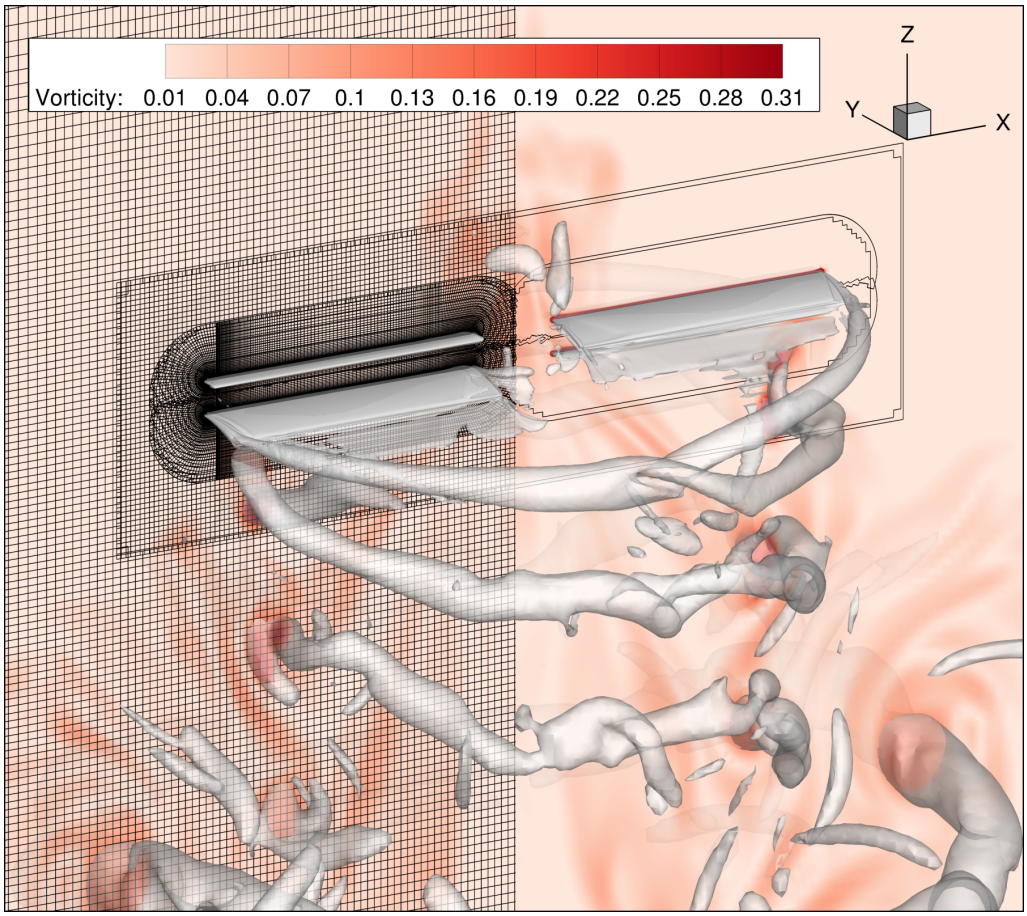
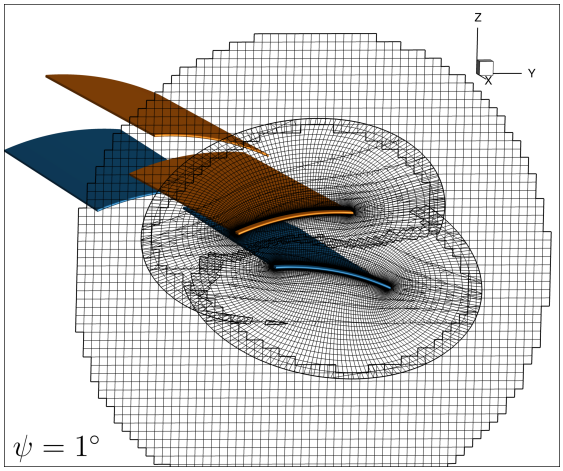
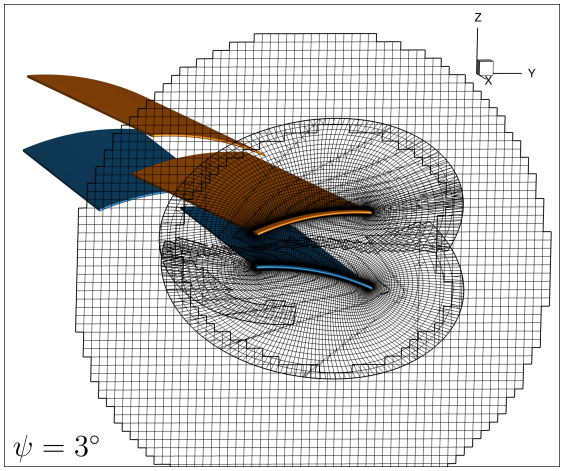


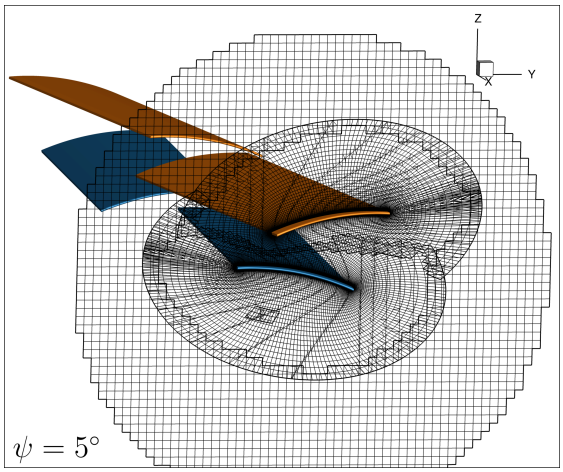

Figure 5.6: Iso-surface of Q-criterion(0.003) with contours of vorticity.



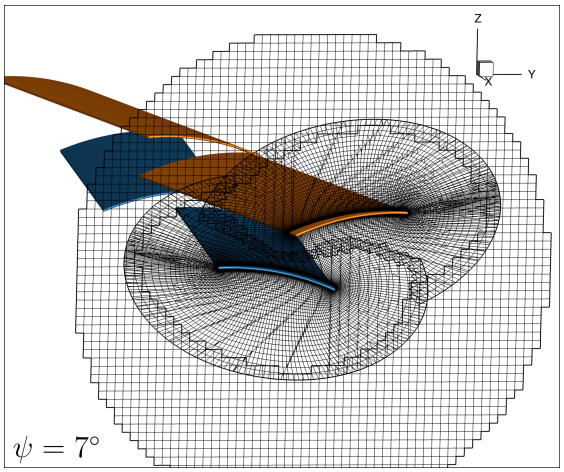

(a) 1° Azimuth



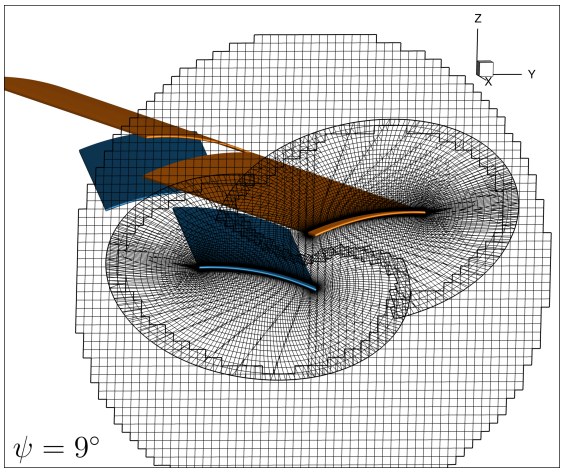

(b) 3° Azimuth




(c) 5° Azimuth




(d) 7° Azimuth




(e) 9° Azimuth

Figure 5.7: Grid and slice location at five azimuths.

and lower rotor colored blue. The overset grid is shown on an X-slice of the domain near the blade tip ($x/R = 0.97$). As the blades pass, the fringe- and field-maps of the overset blade grids changes significantly. Furthermore interpolation at these azimuths involve three grids: the upper rotor blade, lower rotor blade, and nested background mesh. Initially at azimuths $\psi = 1°$ and $3°$ (Fig. 5.7(a) and (b), the lower rotor especially is expected to receive significant influence from the upper blade mesh. By $\psi = 7°$ and definitely $\psi = 9°$, the convection of fluid into the lower rotor mesh is back to merely the nested mesh. Considering the direction of fluid convection through meshes (considering grid motion), is an important factor for determining the relevancy of iterpolation regions to the overall solution and convergence.

The sub-iteration convergence for the ten timesteps during blade passage is shown in Fig. 5.8. The solution is restarted from $\psi = 0°$ therefore the first iteration corresponds to solving for $\psi = 1°$ for the upper (counter-clockwise) rotor and $\psi = -1°$ for the lower (clockwise) rotor. All methods use 15 sub-iterations with 20 linear solver iterations. For the GMRES methods this corresponds to 20 Krylov iterations

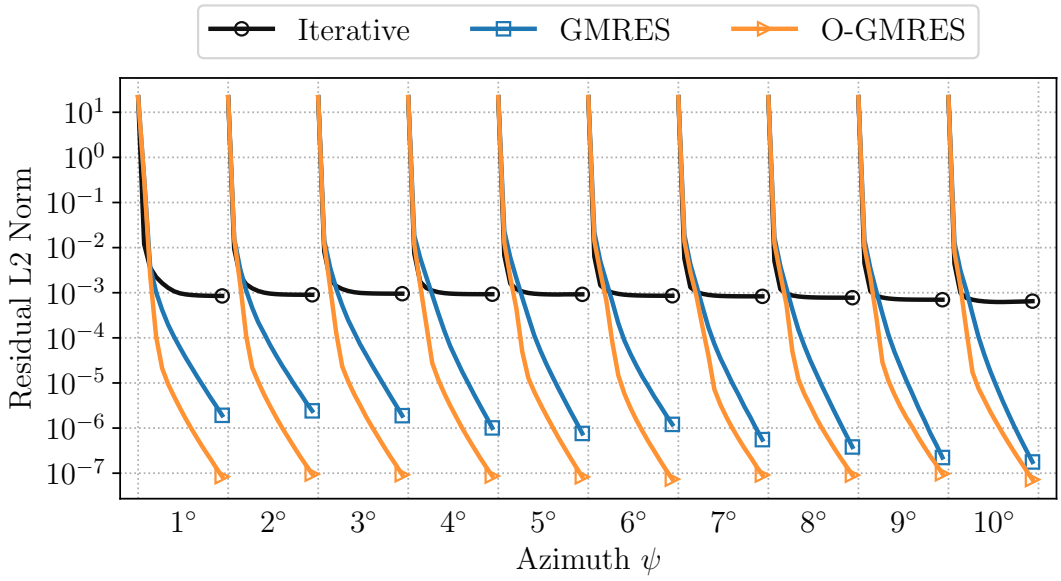



Figure 5.8: Iteration convergence after $t^*$: $\Delta\psi = 1°$. Dual-time CFL is $\infty$ for GMRES and O-GMRES methods and 100 for the Iterative method.

and for the Iterative algorithm this is approximated simply as a 20× multiplier on the number of sub-iterations. Both GMRES and O-GMRES were able to run with an infinite dual-time $\text{CFL}_\tau = \infty$, meaning the local timestep in each cell is the physical timestep. The Iterative method did not converge well with the large timestep and instead was run with dual-time $\text{CFL}_\tau = 100$. Even with the reduced CFL, the Iterative method stalls, indicating further relaxation of the dual-timestep may be required. O-GMRES in general converges better than GMRES especially in the initial azimuths.

The solution and convergence at the five azimuths from Fig. 5.7 can be analyzed in more detail by looking at the vorticity solution near the blade tips and linear solver convergence for GMRES and O-GMRES methods. The vorticity solution is used to visualize important fluid structures convecting across overset boundaries. The vorticity solution at $\psi = 1°$ is shown in Fig. 5.9(a). The lower rotor, seen traveling to the left, is receiving significant flow information directly from the mesh of the upper rotor. The convergence, shown in Fig. 5.9(b), shows the sub-iteration convergence as solid symbols and linear-solver convergence as smaller, hollow symbols. In later sub-iterations O-GMRES converges at approximately the same rate as GMRES however in early sub-iterations, O-GMRES maintains a significantly steeper convergence rate. As a result, O-GMRES requires approximately half the number of sub-iterations to achieve the convergence drop seen in GMRES over 15 sub-iterations. After 3 sub-iterations the linear solver trend from O-GMRES matches very well with the sub-iteration convergence. This indicates after 3 sub-iterations that the linear system is a good approximation to the non-linear problem. With the de-coupled GMRES algorithm, however, a saw-tooth pattern emerges between the sub-iteration and linear iteration trends. Linear convergence for GMRES may be very good but this does not translate to faster non-linear convergence.

At $\psi = 3°$, shown in Fig. 5.10, the same overall observations are amplified

from $\psi = 1°$. Vorticity from the upper rotor is convecting into the interpolation region between the upper rotor, lower rotor, and background mesh. As a result the convergence rate with GMRES reduces earlier by sub-iteration 2, whereas the convergence with O-GMRES remains relatively unaffected.

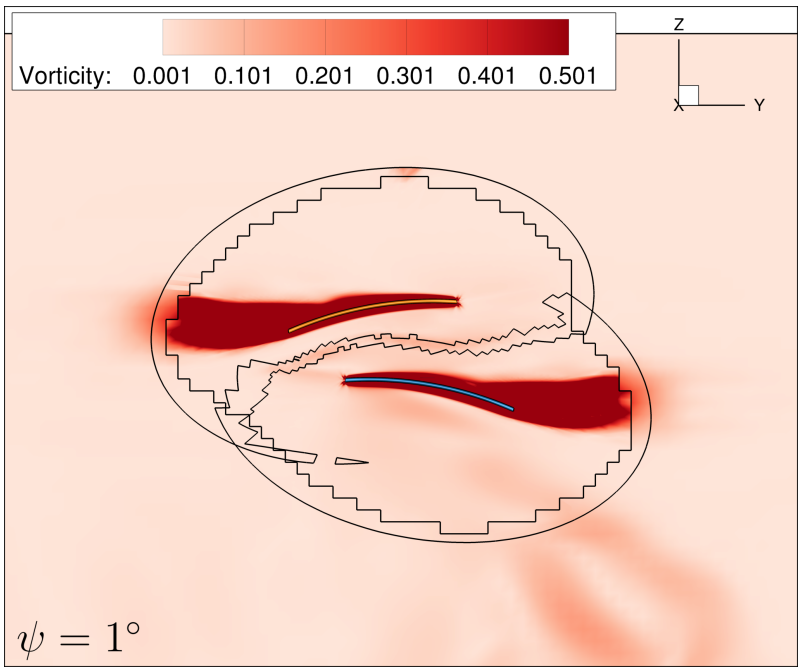
By $\psi = 5°$, shown in Fig. 5.11, and $\psi = 7°$, shown in Fig. 5.12 the vortex from the upper rotor has entered the grid of the lower rotor. The convergence rate with GMRES still reduces drastically after the first sub-iteration. The convergence rate with O-GMRES starts to show two clear kinks in the convergence trend. The first kink occurs after the first sub-iteration and the second occurs after the fourth or fifth sub-iteration. Before sub-iteration 6, the linear solver convergence for O-GMRES does not match well with the sub-iteration convergence. The mis-match indicates convergence is largely affected by the temporal linearization terms. After sub-iteration 5, for $\psi = 5°$, or 6, for $\psi = 7°$, the O-GMRES convergence rate decreases after the second kink but the linear solver convergence matches the non-linear convergence. The kinks in convergence are therefore primarily due to linearization error. The fact that O-GMRES converges faster for the same number of sub-iterations indicates a significant portion of the linearization error is a function of the interpolated region. This conclusion matches the qualitative observation that between $\psi = 5° - 7°$, the vorticity solution crosses the overset boundaries of both the upper and lower rotors.

Finally by $\psi = 9°$ each rotor blade is back to receiving a relatively clear flow solution from the nested mesh. The vortex from the upper blade still passes through the mesh of the lower rotor blade however the convection direction in each mesh indicates interpolation should play less of a role. The convergence result still shows a kink in the O-GMRES solution where the non-linear error is being converged however the kink occurs earlier than for $\psi = 5° - 7°$. This indicates non-linear solution error involves less influence from interpolation regions. O-GMRES and GMRES appear to converge toward the a similar convergence level and rate by sub-iteration 14. As

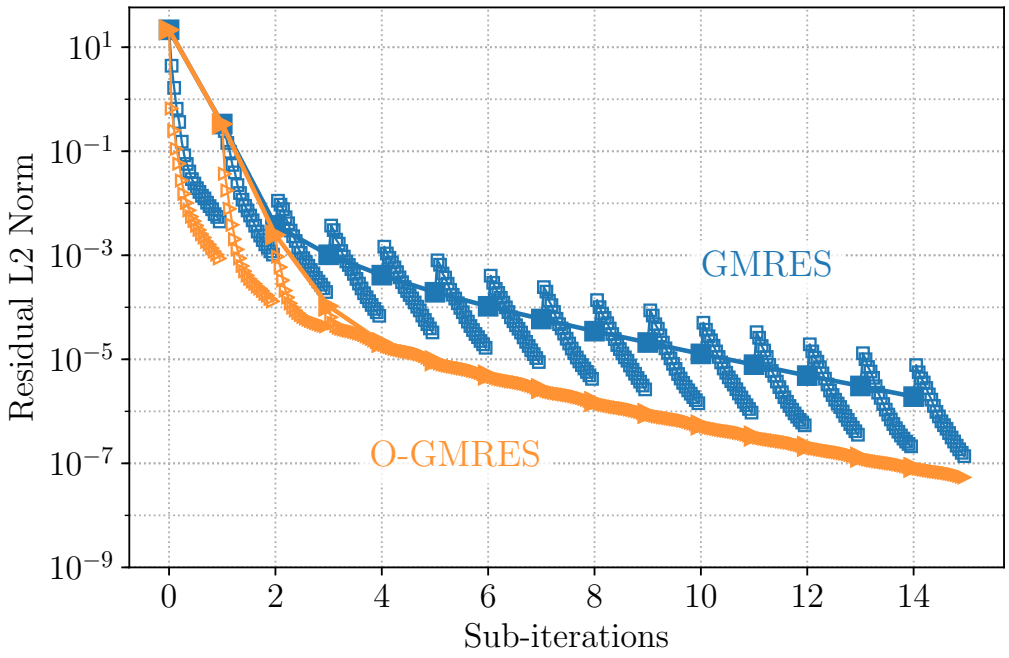
interpolation plays less of a role in the numerical error, it makes sense that both O-GMRES and GMRES would perform similarly. The linear solver convergence for GMRES, however, still demonstrates the saw-tooth pattern whereas GMRES converges much more consistently between linear and non-linear sub-iterations.

For all timesteps, the benefit of O-GMRES occurs primarily in the first few sub-iterations especially at azimuths where interpolation of fluid structures occurs on the convection-inflow-side of the grid. Travel of information for convection-dominated flows is governed by the wave speeds. In a numerical stencil where a cell is downwind of an interpolated region with high gradients, O-GMRES greatly out-performs GMRES. Once the blades pass, large gradients may still be interpolated at the boundary however the boundary is now downwind of most interior cells. In a numerical stencil where field cells are upwind of an interpolation region, there is less of a benefit for O-GMRES compared to the traditional GMRES.

After the first few sub-iterations, the non-linear error of the system is reduced enough that the linear convergence for O-GMRES is representative of the non-linear convergence. This is not the case for the traditional GMRES convergence, which instead displays a saw-toothed pattern. The O-GMRES result presents an interesting prospect for further convergence accelleration. With the linear system adequately representative of the non-linear system, increasing the number of Krylov iterations with each sub-iteration would take advantage of the quadratic convergence rate of the GMRES algorithm while still properly handling interpolation boundaries.

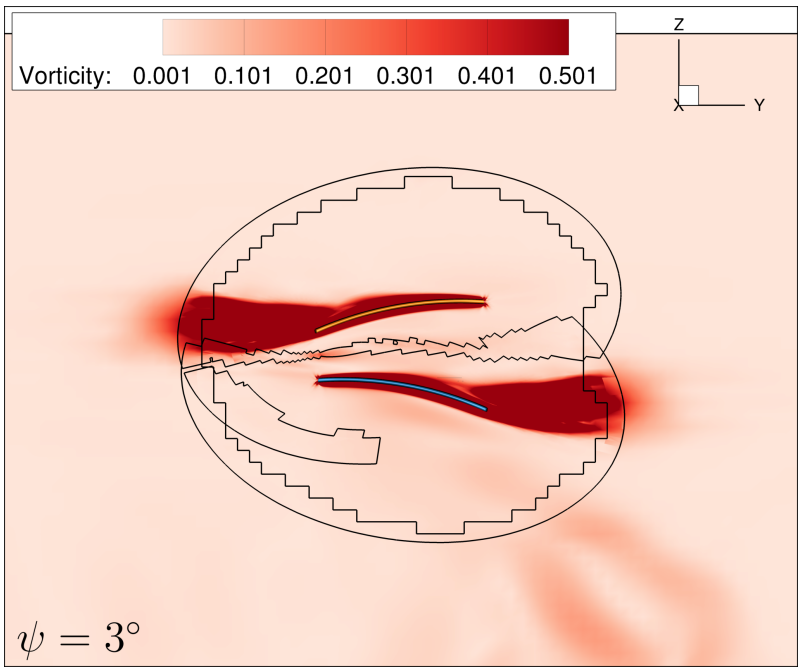



(a) Vorticity at 1°

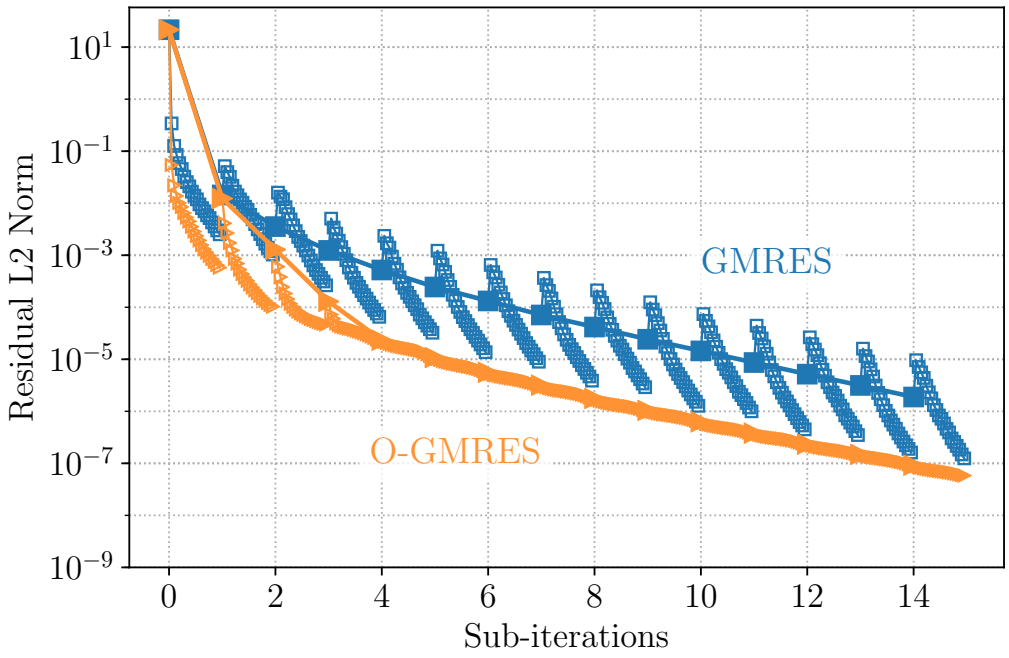



(b) Convergence at 1°

Figure 5.9: Vorticity solution near the tip of the blades and convergence for $\psi = 1°$.

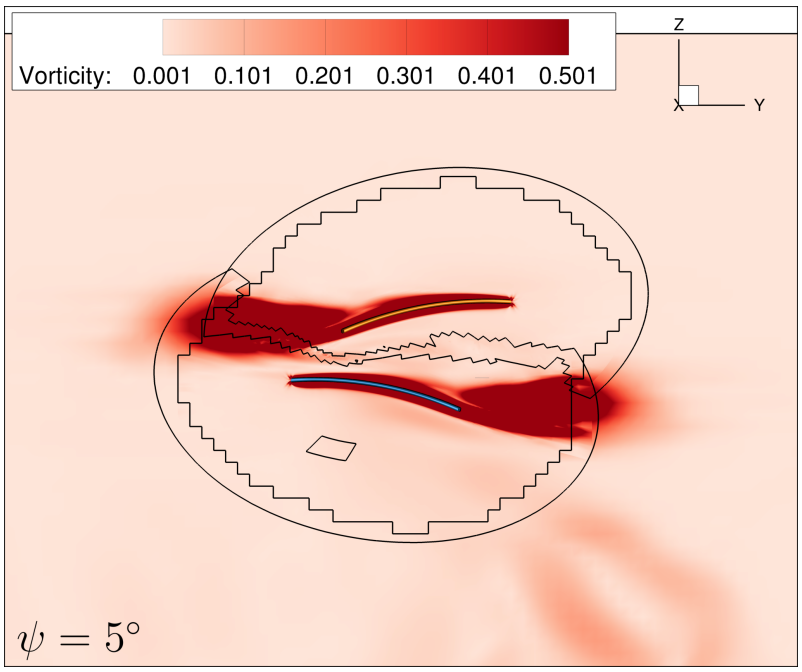



(a) Vorticity at 3°

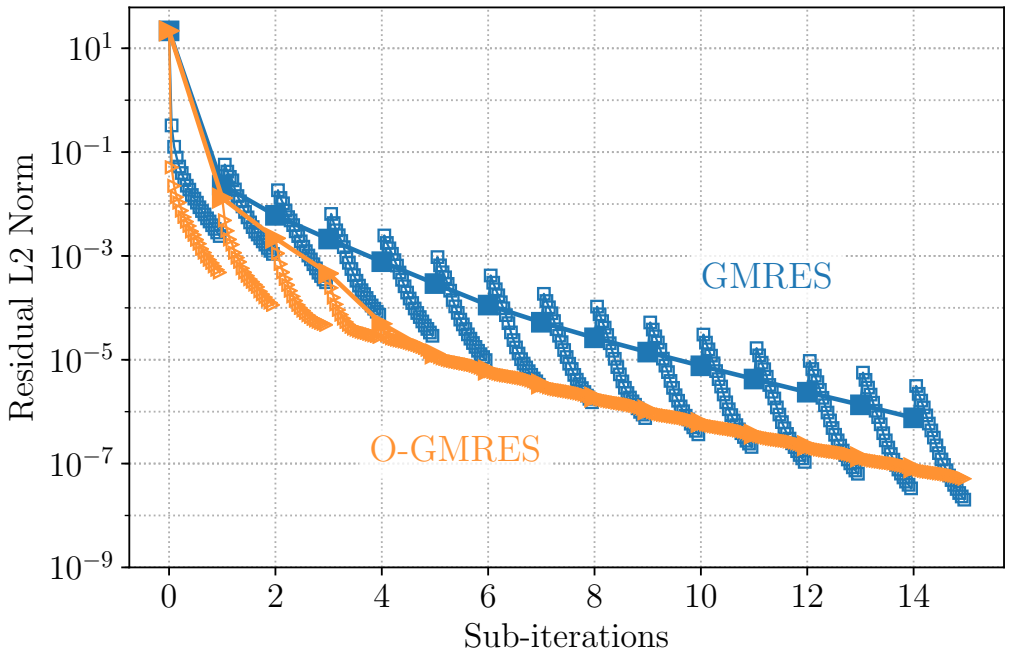



(b) Convergence at 3°

Figure 5.10: Vorticity solution near the tip of the blades and convergence for $\psi = 3°$.

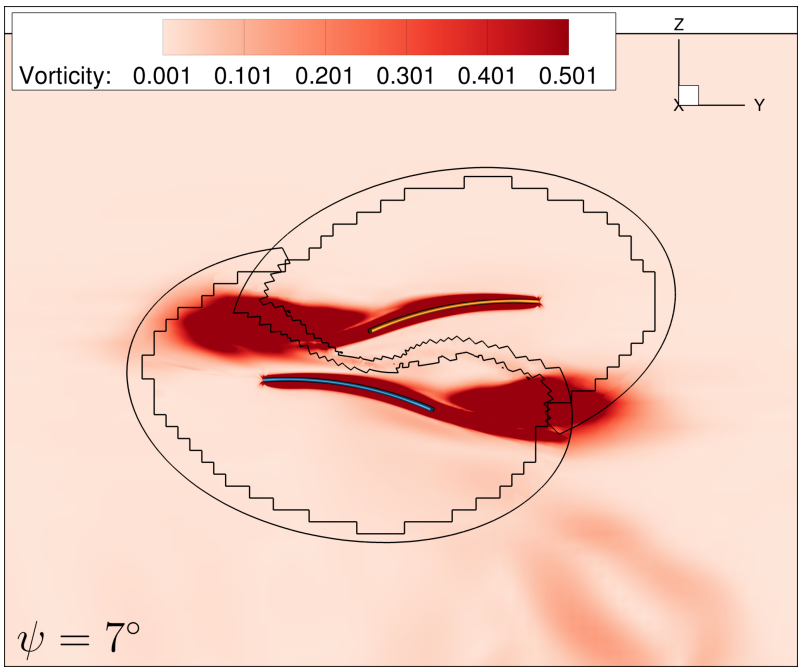



(a) Vorticity at 5°

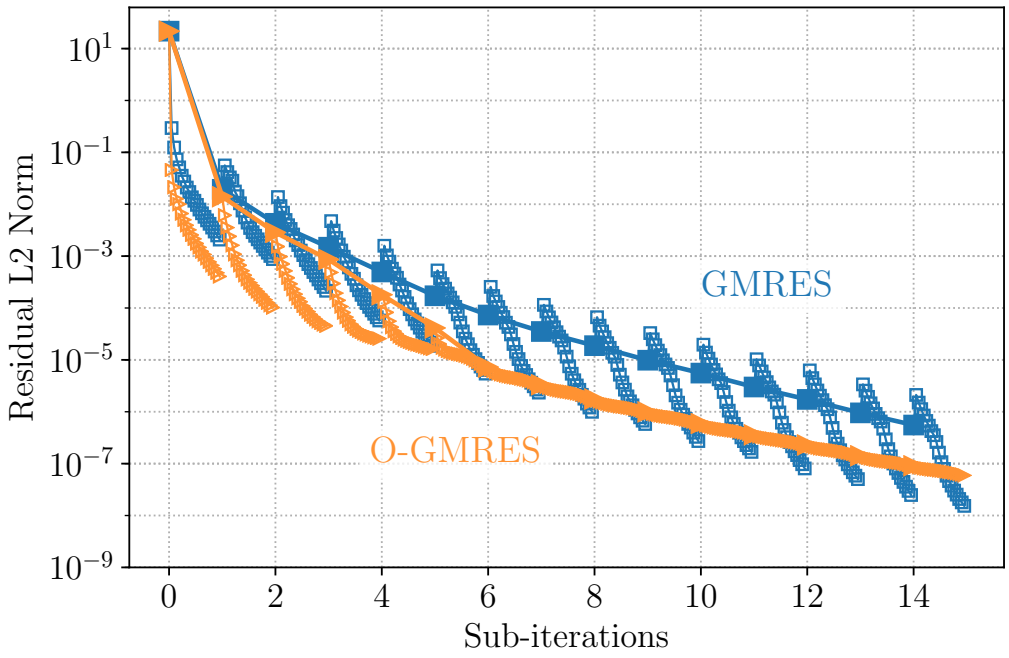



(b) Convergence at 5°

Figure 5.11: Vorticity solution near the tip of the blades and convergence for $\psi = 5°$.

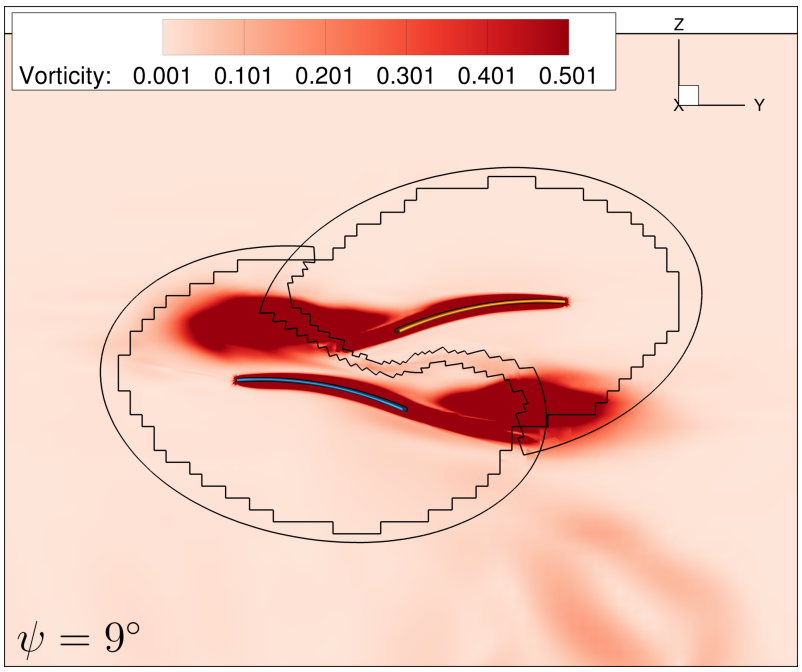



(a) Vorticity at 7°

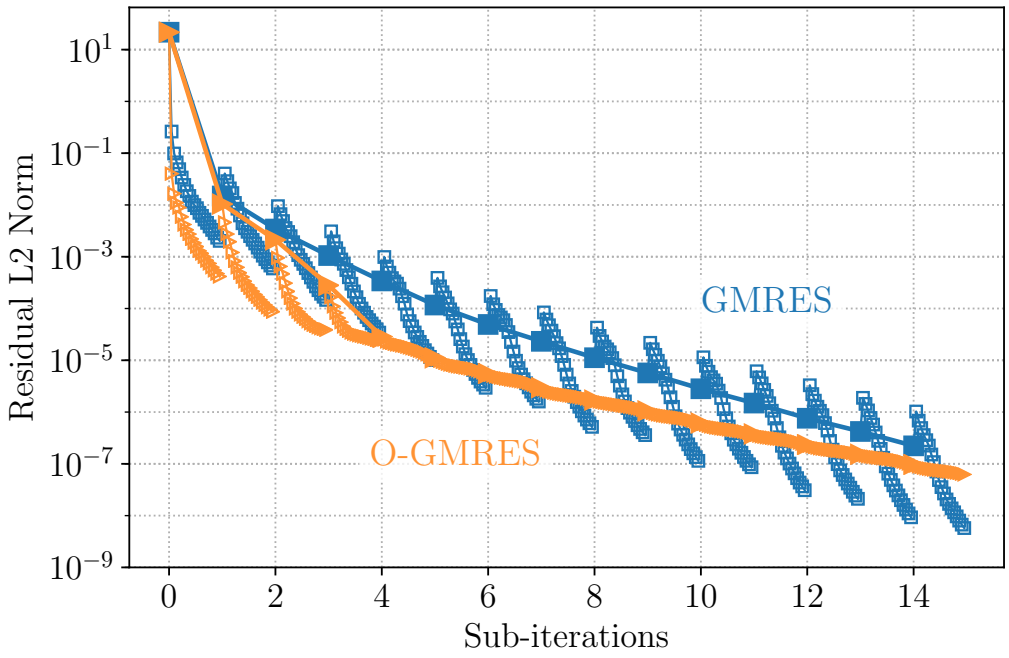



(b) Convergence at 7°

Figure 5.12: Vorticity solution near the tip of the blades and convergence for $\psi = 7°$.




(a) Vorticity at 9°




(b) Convergence at 9°

Figure 5.13: Vorticity solution near the tip of the blades and convergence for $\psi = 9°$.

## 5.3 O-GMRES Conclusions

Two cases are presented for analysis of the developed O-GMRES algorithm. The framework-level algorithm incorporates interpolation between meshes as part of the implicit system for overset problems with the goal of accelerating convergence. Two different solvers are used, mStrand and Garfield, and three different implicit methods are compared. The first method uses the preconditioner available from each solver to approximate the inversion of the linear system. The second method uses a de-coupled GMRES algorithm to solve the linear system in each mesh independently. The final method uses the developed O-GMRES algorithm. In all compared cases, both sub-iterations and linear solver iterations are used for convergence analysis.

The first considered case is a sphere in freestream at $Re = 800$, which results in unsteady vortices shed behind the sphere. mStrand is used for the sphere mesh and Garfield is used for all background meshes. Convergence with all three implicit methods are compared for two different timestep sizes. For a fair comparison, GMRES and O-GMRES are run with 10 sub-iterations of 20 Krylov (linear) iterations and the iterative scheme is run with $20 \times 10$ sub-iterations. For the small timestep, the simple iterative procedure with just the preconditioner out-performs the traditional GMRES method. The multiplier on sub-iterations for the iterative scheme results in better approximation to the time-accurate terms in initial sub-iterations. This explains the initial rapid convergence of the Iterative method. O-GMRES out-performs both GMRES and iterative methods. The O-GMRES method stalls after dropping 10 orders as a result of finite-difference limitations for floating point arithmetic. In the case with the moderate timestep, the iterative method struggles to converge and the traditional GMRES clearly showed an advantage. O-GMRES and GMRES converged with the same slope, however O-GMRES initially converged significantly faster.

A detailed view of the sub-iteration and linear iteration convergence is compared for both small and moderate timesteps. In both cases, GMRES shows discrepancy between convergence of the linear solver and convergence of the overall non-linear equations. This trend is a result of the fixed values at interpolation boundaries. Effectively the solution that the linear system is converging toward is inconsistent with the actual overall overset system. When the overset points are included as part of the implicit system, O-GMRES shows consistent drop in the residual between both sub-iterations and linear iterations.

The second case considers a set of coaxial rotors designed for use on Mars. The cambered, flat-plate airfoils are meant for low-Reynolds number flight conditions. A 10° window during blade passage is studied in detail. Most notably from the general convergence trend, the Iterative method convergence stalls even with a relaxation applied to the timestep. GMRES and O-GMRES convergence do not stall. Similar to the results from the sphere run with moderate timestep, O-GMRES and GMRES eventually converge at similar rates. The primary benefit from O-GMRES is from faster initial convergence.

Analysis of sub-iterations and linear iterations at azimuths 1°, 3°, 5°, 7°, and 9° reveals patterns from the physics that affect the convergence trend. When regions of large-gradient interpolation appear upstream of blade meshes, O-GMRES greatly out-performs GMRES. Once the blades pass and interactional aerodynamics play less of a roll, O-GMRES and GMRES converge more similarly.

In both the sphere case and coaxial rotor case, O-GMRES converges more consistently than GMRES. As the interpolation regions changed for the case of passing blades, O-GMRES maintains consistent convergence over the 10° passage window. Convergence with the traditional GMRES method, however, is adversely affected by the presence of overset interpolation. In an interactional aerodynamics case where the overset connectivity may be constantly changing from the presence of

multiple rotors, O-GMRES offers a more reliable approach to obtain good numerical convergence.

Comparing patterns of sub-iteration and linear-solver convergence leads to additional convergence acceleration possibilities. All cases considered in this chapter use constant numbers of sub-iterations and linear-solver iterations. A realistic application might instead vary the number based on convergence or sub-iteration step. For example, initial sub-iterations may benefit from the use of fewer linear solver iterations in order to quickly reduce linearization error. Later sub-iterations may benefit from more linear iterations to take advantage of the quadratic convergence rate of the GMRES algorithm.

Timing of O-GMRES and GMRES has not been considered as a metric for comparison for two reasons. First, the difference in algorithms simplifies to additional data-exchange routines called during the linear iteration process. The performance of the data-exchange routines is both case and hardware dependent. Second, a real engineering application of the framework would apply practical limits to linear solver steps to avoid unnecessary computation in initial sub-iterations.

In addition to aforementioned use of variable linear solver steps, future development and investigations of O-GMRES can include using the overset algorithm only on sub-sets of meshes. Similar to how Sec. 2.6 describes separating stationary and reconnecting meshes to avoid unnecessary computation, O-GMRES could be applied only to moving, reconnecting meshes where the change in interpolation regions could have a significant and adverse affect on convergence. For the coaxial rotor case, for example, all background meshes could be solved using the traditional approach while the blade meshes use O-GMRES to ensure consistent convergence.

# Chapter 6: Conclusions

The demands of commercial urban air mobility and next-generation military aircraft have resulted in the evolution of the helicopter to include complex configurations with many interacting rotors and lifting surfaces. Improved understanding of interactional aerodynamics between components is crucial to the safety and viability of future vertical take-off and landing vehicles. Complex configurations may require higher-fidelity computational aerodynamic tools to be used for analysis to ensure accurate representation of the problem.

Just as the definition of rotorcraft has evolved, so has the definition of the modern supercomputer. Improvements in computational fluid dynamics tools since the late $20^{\text{th}}$ century has largely followed advancements in computer science and engineering to develop faster, cheaper computers. From single-core to multi-core to distributed memory computers, each new hardware generation resulted in new software written to achieve faster simulation times. The modern trend of massively parallel compute architectures like the GPU present a promising new era of supercomputing.

From the perspective of aerodynamic analysis, full-scale rotorcraft CFD is not new and can be accomplished using a plethora of industry, government, or academic codes on traditional supercomputers with large simulation times. Similarly from the computer science perspective, using GPUs to accelerate numerical methods is also well established. The intersection of these two perspectives, however, presents promising conjecture and is the root motivation for this dissertation research.

Research from this dissertation has been presented with the goal of achieving six primary objectives:

1. Develop a GPU-accelerated, time-accurate Navier–Stokes solver for overset applications.

2. Design and implement a framework for the overset multi-solver paradigm amenable to heterogeneous architectures like the GPU

3. Identify and derive a robust, flexible algorithm for the overset multi-solver paradigm that considers fringe points implicitly.

4. Verify and validate the individual components of an overset framework.

5. Apply the overset framework to a full, notional rotorcraft configuration.

6. Apply the implicit overset algorithm to a coaxial rotor configuration.

The first three objectives are development-based and require advancing algorithms or technology beyond the state-of-the-art whereas the last three are more application focused. This chapter reviews a summary of the presented work to achieve each objective. Contributions to the state-of-the-art are highlighted. Finally, future research directions are recommended based on findings from the current work.

## 6.1 Algorithm and Framework Development

Addressing the first objective, the GPU-accelerated CFD solver Garfield has been developed to approximate the Reynolds-Averaged Navier–Stokes equations on structured, curvi-linear meshes. The finite-volume code has been heavily adapted to operate on multiple, moving and deforming grids run in parallel across multiple GPUs. Each algorithm within the flow solver has been designed to run efficiently in

parallel on the GPU and all data remains on the GPU to avoid costly transfers back to CPU memory.

The preconditioner of the implicit linear system in particular poses algorithmic challenges because many common approaches use serial algorithms designed for a CPU core, a serial architecture. In Garfield, three variations of the Red-Black Gauss–Seidel method, have been designed specifically for efficiency on GPU architectures.

To combine Garfield with other, CPU-based codes, a light-weight Python framework has been designed which interfaces many different numerical codes. In addition CPU-based CFD solvers, the framework utilizes a grid-motion module for rigid or elastic deflection of any grid topology (structured or unstructured). For overset communication, the open-source overset grid assembler Tioga is used. Furthermore Tioga has been adapted to efficiently handle either CPU or GPU data to drastically reduce data-transfer and simulation times. The designed framework satisfies the requirements of the second objective.

The third objective of improving the convergence for challenging overset cases is accomplished with the development of the Overset GMRES method. As a distributed formulation of the GMRES algorithm, O-GMRES is implemented at the framework level to consider interpolated boundaries as part of the implicit system of equations. In contrast, traditional Schwartz-alternating methods consider each grid separately.

## 6.2 Framework Applications

### 6.2.1 Verification and Validation

Subsets of the full framework have been validated using simpler, well-understood aerodynamic cases. For accuracy validation of Garfield, the GPU code is compared to NASA's Overflow code for the same overset mesh system of an Onera M6 wing.

Both codes result in almost identical pressure distribution over the wing, indicating the flow solver and overset interpolation routines behave as expected.

The Onera M6 case is also used for a performance comparison of CPU and GPU codes. The NASA Overflow code solves almost identical equations as Garfield however timing results show Garfield runs 5.35× faster on a single GPU compared to a single CPU. Extrapolating speedup to modern clusters containing 4-8 GPUs per node, Garfield easily demonstrates an order of magnitude in performance gain over the highly-optimized CPU code. Furthermore, the scalability of Garfield tested on different GPU architectures on up to 32 GPUs demonstrates the code scales well across multiple processors.

NASA's Overflow code has also been used to verify proper treatment of elastic deflection within the developed mesh motion module. A UH-60 rotor in high-speed forward flight is used for comparison. Without the proper elastic deflection treatment in CFD, the predicted airloads will not be correct. Assuming proper treatment within NASA's Overflow solver, if the motion module is provided the same deflections and results in the same airloads, the deflections and grid-motion in Garfield must also be correctly handled. Airload results between Overflow and Garfield are very similar and minor differences appear only in regions of challenging flow physics, such as the shock on the advancing side of the rotor. The minor discrepancies can be attributed to different grids used in each solver.

The developed Red-Black Gauss–Seidel preconditioners as well as the implicit GMRES linear solver have been verified within Garfield using simple aerodynamic problems. For rotor-blade-like meshes, the Gauss–Seidel method out-performs the baseline DADI implementation by converging in less wall-clock time and with a larger allowable timestep. For a laminar cylinder case, the use of GMRES also results in faster convergence and a larger maximum allowable timestep.

Finally, benefits of the Overset GMRES algorithm has been analyzed for two

simple problems: the Poisson heat-equation and unsteady, inviscid flow over a wedge. As a linear partial differential equation, the Poisson equation solved with O-GMRES can be accomplished by increasing the number of linear (Krylov) iterations. As the number of linear iterations is increased, O-GMRES converges in fewer total iterations. With the traditional GMRES, however, linear iterations do not include updated interpolated quantities resulting in a saw-tooth convergence pattern and significantly more iterations. The saw-tooth pattern also appears in the time-accurate convergence of the inviscid wedge case. At each timestep, O-GMRES shows consistent convergence between linear iterations and the non-linear Newton iterations.

### 6.2.2 Notional Helicopter Configuration

The ROBIN fuselage and rotor have been used as the starting point for applying the full, overset, heterogeneous framework to a realistic configuration. Experimental wind-tunnel results from rotor-fuselage interaction cases have been simulated in the present framework using three different in-house CFD solvers: HamStr, TURNS, and Garfield. Previous comparisons of the experiment and CFD have found discrepancies between the required rotor controls for trimmed flight. Because the trim analysis is beyond the scope of this work, the cyclic control angles from literature have been used and the collective estimated to match the main rotor lift. Overall, the pressure results in the longitudinal direction match well between experiment and CFD. On the lateral cross section the pressure history matches less well, likely as a result of ignoring the hub and support structure in CFD meshes. Most importantly the ability to model the interactional aerodynamics and extract meaningful data for the rotor-fuselage case is demonstrated.

The most complex case run for this work extends the rotor-fuselage case to include a hub and tail rotor. The large simulation is representative of a full configuration and highlights many challenges, including convergence and load balancing.

The tail rotor rotates at a higher RPM than the main rotor, resulting in a larger azimuthal timestep. The larger step causes the residual of the tail rotor to stall in Garfield, indicating the need for improved implicit numerical methods. Although Garfield is assigned significantly more grid points compared to the CPU-based solvers, Garfield runs over 3× faster than the other solvers. The mis-match in performance can be offset by adding a 3× multiplier to the number of sub-iterations performed in Garfield. The additional sub-iterations are free from a wall-clock perspective and contribute to improved convergence in Garfield grids.

Between the initial rotor-fuselage simulation and the rotor-fuselage-hub-tail simulation, the recorded thrust on the main rotor experiences a significant blip at azimuth $\psi = 0°$. The change in thrust at this azimuth is a result of the hub, which sheds structures and causes separation on the rotor blade passing in its wake. While the presence of the hub affects the rotor blade thrust, the torque is hardly affected.

Aerodynamic interactions between the main and tail rotors have been analyzed by running the two rotors without the fuselage or hub. Three different cases are used to draw conclusions about the tail rotor thrust (and corresponding main-rotor anti-torque). First, the isolated tail rotor shows the expected, dominant 8/rev signal in forward flight. In the presence of the main rotor, however, a strong 16/rev component of thrust appears. This higher frequency can be justified assuming a 4/rev variation in inflow to the tail rotor but without the full CFD result the analysis is not trivial. Finally a cross-wind added to the configuration shows the tail in a "descent" flight condition results in massively separated flow on the tail rotor blades. Despite increased counter-torque to the main rotor and resulting aircraft yaw, the separation on the blades indicates the pilot may not have the control authority to correct the tail thrust without significant and possibly dangerous decrease of the tail rotor blade pitch.

### 6.2.3 Overset GMRES

The first case used to analyze performance of the O-GMRES algorithm applied to a multi-solver problem is unsteady, laminar flow over a sphere. Three background meshes are used with Garfield and the near-body sphere mesh is solved using mStrand. The convergence using two different timesteps both show the O-GMRES method converges better than the traditional GMRES method or simple pre-conditioning approach. Just as with the wedge case used for single-solver validation, the traditional GMRES results in a saw-tooth pattern for sub-iteration convergence. O-GMRES, on the other hand, converges consistently between linear and non-linear (Newton) iterations.

A laminar rotor designed for use on Mars is used as the final case for O-GMRES analysis. Again Garfield is used for all background grids and mStrand is used for the four coaxial, counter-rotating blades. Focusing on the azimuths of blade passage, convergence in a 10° window is analyzed between the iterative preconditioner, traditional GMRES, and O-GMRES. A large azimuthal timestep of $\psi = 1°$ is used to further challenge convergence. Interestingly, the convergence with only the iterative preconditioner stalls. The traditional GMRES method converges, however the total convergence varies with each iteration as the interpolation between grids changes. Furthermore the linear solver iterations from GMRES indicate the same saw-tooth pattern observed from previous results also applies to this case. The O-GMRES method, on the other hand, converges consistently throughout the entire 10° passage window. Linear solver convergence indicates after the first four sub-iterations that the non-linear residual is sufficiently converged and the convergence is primarily governed by the linear convergence.

Convergence analysis for O-GMRES has been conducted between all grids for a fixed number of linear and non-linear iterations. The convergence results can be

used to draw conclusions about more efficient application of the framework. For example, a realistic application may only require O-GMRES to be applied to sub-sets of grids in areas of large gradients. The number of linear iterations in initial Newton steps can also be reduced, since convergence appears dominated by the non-linear, time-differencing terms.

## 6.3 Contributions and Future Work

Results from the heterogeneous framework design presented in this dissertation are being considered for implementation in the Helios (CREATE AV$^{TM}$) software suite. Modifications to the open-source connectivity code for Python wrapping and GPU memory interpolation are available to the public through GitHub. The modifications are minor, however result in significant improvements in speedup when CPU and GPU codes are used together. Performance results from this dissertation provide a guideline for the advancement of industry and government research codes to take advantage of new, available supercomputer hardware.

The Garfield source code is another key contribution, since it demonstrates an order of magnitude speedup over comparable CPU codes like NASA's Overflow. Most importantly the speedup and scalability of Garfield is obtained for realistic use cases on currently available clusters. Algorithm implementation details, for example of the Gauss–Seidel-type methods, suggests techniques for improved convergence accelerated on parallel architectures like the GPU.

The developed Overset GMRES framework is also being considered for inclusion in the Helios (CREATE AV$^{TM}$) framework. An extension of the algorithm is under consideration for intra-mesh interpolation, which occurs in strand meshes near locations of concave geometry. Improving the convergence of challenging, overset applications is key to attaining physical, trustworthy results, especially as aircraft configurations become more complex.

Future recommended work for the Garfield solver should consider improving the treatment of viscous terms for the Red-Black Gauss–Seidel method. Currently, viscous eigenvalue approximations are used which perform well for high Reynolds number simulations but poorly for cases like the laminar Mars rotor. The GMRES routines within Garfield should also be extended to consider the turbulence model equations. Currently the turbulence model equation is de-coupled from the mean-flow equations, which severely limits the simulation timestep size. Since the turbulence model equation is a conservation equation, it can be added to the other five equations to create a single residual vector minimized during the GMRES procedure.

Future work for the O-GMRES framework should include convergence analysis using variable timestep sizes (dual-time CFL) and variable numbers of linear solver iterations. In initial Newton iterations, for example, only 5 Krylov iterations may be required whereas in final Newton iterations 40-50 Krylov iterations may be used. The additional Krylov iterations should improve convergence as a result of the quadratic convergence rate of the GMRES algorithm.

Capability enhancements to the developed framework should consider a more generic treatment of grid motion to allow nested, elastic frames of motion. Currently, the blades of a rotor system are allowed to deform given an deflection file but the rotor location is fixed. For application involving a propeller on a flexing wing, the rotor frame should be allowed to deform with the wing.

The developed O-GMRES method demonstrates improved convergence compared to traditional methods, however additional research is required to analyze the benefits of improved convergence on engineering quantities. One possible application area is the investigation of hub drag by looking at a breakdown of different components. Only recently have state-of-the-art CFD frameworks started looking at complex hub geometries with the goal of understanding specific sources of hub drag. Since the hub consists of highly complex connections to the fuselage and blades, the

hub region in CFD can result in a significant amount of interpolated fringe areas. Without proper convergence, the pressure and skin-friction drag may not be properly captured by the numerics. Rotor hub drag is especially important at high advance ratios and the CFD for such a case would be complemented well with additional wind or water tunnel experiments.

# Appendix A: Flow Visualization with Vorticity and Q-Criterion

When analyzing the solution of large simulations, iso-surfaces of Vorticity or Q-criterion help visualize the locations of interesting flow features. Vorticity is the magnitude of the curl of the velocity, defined as

$$\omega = \|\nabla \times \overrightarrow{V}\| \tag{A.1}$$

and Q-criterion is a function of the rotation and strain tensors

$$Q_c = \frac{1}{2}\left(\|\mathbf{\Omega}\|^2 - \|S\|^2\right) \tag{A.2}$$

where the rotation and strain tensors respectively are

$$\mathbf{\Omega}_{i,j} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \qquad S_{i,j} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{A.3}$$

# Bibliography


[1] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer, "Top500 list - november 2018."

[2] K. Rupp, M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten, "42 years of microprocessor trend data," 2017.

[3] R. E. Mineck and S. A. Gorton, "Steady and periodic pressure measurements on a generic helicopter fuselage model in the presence of a rotor," Tech. Rep. NASA TM-2000-210286, NASA, June, 2010.

[4] R. W. Prouty, *Military helicopter design technology*. Malabar, Florida: Krieger Publishing Company, 1998.

[5] B. Roget, J. Sitaraman, and A. M. Wissink, "Maneuvering rotorcraft simulations using CREATE-AV(TM) Helios," in *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, jan 2016.

[6] R. Magnus and H. Yoshihara, "Inviscid transonic flow over airfoils," *AIAA Journal*, vol. 8, pp. 2157–2162, dec 1970.

[7] J. D. Cole and E. M. Murman, "Calculation of plane steady transonic flows," *AIAA Journal*, vol. 9, pp. 114–121, jan 1971.

[8] A. Jameson, W. Schmidt, and E. Turkel, "Numerical solution of the euler equations by finite volume methods using runge kutta time stepping schemes," in *14th Fluid and Plasma Dynamics Conference*, American Institute of Aeronautics and Astronautics, jun 1981.

[9] T. H. Pulliam and J. L. Steger, "Implicit finite-difference simulations of three-dimensional compressible flow," *AIAA Journal*, vol. 18, pp. 159–167, Feb 1980.

[10] T. Pulliam and D. Chaussee, "A diagonal form of an implicit approximate-factorization algorithm," *Journal of Computational Physics*, vol. 39, no. 2, pp. 347 – 363, 1981.

[11] S. Obayashi and K. Kuwahara, "LU factorization of an implicit scheme for the compressible navier-stokes equations," in *17th Fluid Dynamics, Plasma Dynamics, and Lasers Conference*, American Institute of Aeronautics and Astronautics, jun 1984.

[12] T. Cebeci and A. M. O. Smith, "A finite-difference method for calculating compressible laminar and turbulent boundary layers," *Journal of Basic Engineering*, vol. 92, no. 3, p. 523, 1970.

[13] B. Baldwin and H. Lomax, "Thin-layer approximation and algebraic model for separated turbulentflows," in *16th Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, jan 1978.

[14] D. A. Johnson and L. S. King, "A mathematically simple turbulence closure model for attached and separated turbulent boundary layers," *AIAA Journal*, vol. 23, pp. 1684–1692, nov 1985.

[15] Spalart, P. R., and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodyanmics Flows," in *AIAA-Paper-1992-0439, 30th AIAA Sciences Meeting and Exhibit*, January 6–9, 1992.

[16] B. Launder and D. Spalding, "The numerical computation of turbulent flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 3, no. 2, pp. 269 – 289, 1974.

[17] F. R. Menter, "Two-equation eddy-viscosity turbulence models for engineering applications," *AIAA Journal*, vol. 32, pp. 1598–1605, aug 1994.

[18] A. Bagai and J. G. Leishman, "Rotor free-wake modeling using a pseudoimplicit relaxation algorithm," *Journal of Aircraft*, vol. 32, pp. 1276–1285, nov 1995.

[19] J. P. Yin and S. R. Ahmed, "Helicopter main-rotor/tail-rotor interaction," *Journal of the American Helicopter Society*, vol. 45, pp. 293–302, oct 2000.

[20] F. X. Caradonna, C. Tung, and A. Desopper, "Finite difference modeling of rotor flows including wake effects," *Journal of the American Helicopter Society*, vol. 29, pp. 26–33, apr 1984.

[21] D. Jude and J. D. Baeder, "Extending a three-dimensional GPU RANS solver for unsteady grid motion and free-wake coupling," in *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, jan 2016.

[22] B. Koren, *A robust upwind discretization method for advection, diffusion and source terms*, pp. 117–138. Notes on Numerical Fluid Mechanics, Vieweg, 1993.

[23] G.-S. Jiang and C.-W. Shu, "Efficient implementation of weighted eno schemes," *Journal of Computational Physics*, vol. 126, no. 1, pp. 202 – 228, 1996.

[24] A. N. Brooks and T. J. Hughes, "Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 32, pp. 199–259, sep 1982.

[25] B. Cockburn and C.-W. Shu, "Runge–kutta discontinuous galerkin methods for convection-dominated problems," *Journal of Scientific Computing*, vol. 16, pp. 173–261, Sep 2001.

[26] N. Nguyen, J. Peraire, and B. Cockburn, "An implicit high-order hybridizable discontinuous galerkin method for linear convection–diffusion equations," *Journal of Computational Physics*, vol. 228, pp. 3232–3254, may 2009.

[27] F. X. Caradonna and M. P. Isom, "Subsonic and transonic potential flow over helicopter rotor blades," *AIAA Journal*, vol. 10, pp. 1606–1612, dec 1972.

[28] T. Roberts and E. Murman, "Solution method for a hovering helicopter rotor using the euler equations," in *23rd Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, jan 1985.

[29] C. Chen and W. Mccroskey, "Numerical simulation of helicopter multi-bladed rotor flow," in *26th Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, jan 1988.

[30] G. R. Srinivasan, J. D. Baeder, S. Obayashi, and W. J. Mccroskey, "Flowfield of a lifting rotor in hover - a navier-stokes simulation," *AIAA Journal*, vol. 30, pp. 2371–2378, oct 1992.

[31] J. Benek, J. Steger, and F. C. Dougherty, "A flexible grid embedding technique with application to the euler equations," Fluid Dynamics and Co-located Conferences, American Institute of Aeronautics and Astronautics, Jul 1983.

[32] E. Duque and G. R. Srinivasan, "Numerical simulation of a hovering rotor using embedded grids," AHS Forum, American Helicopter Society, June 1992.

[33] J. U. Ahmad and R. C. Strawn, "Hovering rotor and wake calculations with an overset-grid navier-stokes solver," vol. 2 of *AHS Forum*, pp. 1949–1959, American Helicopter Society, 01 1999.

[34] H. Pomin and S. Wagner, "Navier-stokes analysis of helicopter rotor aerodynamics in hover and forward flight," *Journal of Aircraft*, vol. 39, pp. 813–821, sep 2002.

[35] D. Jespersen, T. Pulliam, and P. Buning, "Recent enhancements to overflow," Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan 1997.

[36] R. Biedron and J. Thomas, "Recent enhancements to the fun3d flow solver for moving-mesh applications," Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan 2009.

[37] V. Sankaran, A. Wissink, A. Datta, J. Sitaraman, M. Potsdam, B. Jayaraman, A. Katz, S. Kamkar, B. Roget, D. Mavriplis, H. Saberi, W.-B. Chen, W. Johnson, and R. Strawn, "Overview of the Helios version 2.0 computational platform for rotorcraft simulations," Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan. 2011.

[38] A. Wissink, B. Jayaraman, A. Datta, J. Sitaraman, M. Potsdam, S. Kamkar, D. Mavriplis, Z. Yang, R. Jain, J. Lim, and R. Strawn, "Capability enhancements in version 3 of the Helios high-fidelity rotorcraft simulation code," Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan. 2012.

[39] M. Ghoreyshi, M. E. Young, A. J. Lofthouse, A. Jirásek, and R. M. Cummings, "Numerical Simulation and Reduced-Order Aerodynamic Modeling of a Lambda Wing Configuration," *Journal of Aircraft*, vol. 55, pp. 549–570, June 2016.

[40] H. Gopalan, C. Gundling, K. Brown, B. Roget, J. Sitaraman, J. D. Mirocha, and W. O. Miller, "A coupled mesoscale–microscale framework for wind resource estimation and farm aerodynamics," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 132, pp. 13 – 26, 2014.

[41] H. A. Schwarz, "II. Ueber einen Grenzübergang durch alternirendes Verfahren.." Wolf J. XV. 272-286., 1870.

[42] A. Mota, I. Tezaur, and C. Alleman, "The schwarz alternating method in solid mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 319, pp. 19–51, June 2017.

[43] B. Smith, P. Bjorstad, and W. Gropp, *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge: Cambridge University Press, 2004.

[44] M. Galbraith, R. Knapke, P. Orkwis, and J. Benek, "A discontinuous galerkin chimera scheme with implicit artificial boundaries," Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan 2013.

[45] M. J. Brazell, J. Sitaraman, and D. J. Mavriplis, "An overset mesh approach for 3d mixed element high-order discretizations," *Journal of Computational Physics*, vol. 322, pp. 33–51, Oct. 2016.

[46] W. D. Henshaw, "On multigrid for overlapping grids," *SIAM Journal on Scientific Computing*, vol. 26, pp. 1547–1572, Jan. 2005.

[47] W. Anderson, R. D. Rausch, and D. L. Bonhaus, "Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids," *Journal of Computational Physics*, vol. 128, no. 2, pp. 391 – 408, 1996.

[48] V. K. Lakshminarayan, J. Sitaraman, B. Roget, and A. M. Wissink, "Development and validation of a multi-strand solver for complex aerodynamic flows," *Computers & Fluids*, vol. 147, pp. 41–62, Apr. 2017.

[49] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789 – 828, 1996.

[50] N. H. Decker, V. K. Naik, and M. Nicoules, "Parallelization of a class of implicit finite difference schemes in computational fluid dynamics," *International Journal of High Speed Computing*, vol. 05, pp. 1–50, mar 1993.

[51] R. S. Williams, "What's next? [the end of moore's law]," *Computing in Science & Engineering*, vol. 19, pp. 7–13, mar 2017.

[52] E. S. Larsen and D. McAllister, "Fast matrix multiplies using graphics hardware," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, SC '01, (New York, NY, USA), pp. 55–55, ACM, 2001.

[53] B. P. Pickering, C. W. Jackson, T. R. Scogland, W.-C. Feng, and C. J. Roy, "Directive-based gpu programming for computational fluid dynamics," *Computers & Fluids*, vol. 114, pp. 242 – 253, 2015.

[54] K. Soni, D. D. Chandar, and J. Sitaraman, "Development of an overset grid computational fluid dynamics solver on graphical processing units," *Computers & Fluids*, vol. 58, pp. 1 – 14, 2012.

[55] D. C. Jespersen, "Acceleration of a cfd code with a gpu," *Scientific Programming*, vol. 18, pp. 193–201, 01 2010.

[56] E. Nielsen and A. Walden, "Preparing the FUN3D CFD solver for the exascale era," in *Supercomputing Conference*, The International Conference for High Performance Computing, Networking, Storage, and Analysis, Nov 2018.

[57] P. Roe, "Approximate riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, vol. 43, no. 2, pp. 357 – 372, 1981.

[58] R. Borges, M. Carmona, B. Costa, and W. S. Don, "An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws," *Journal of Computational Physics*, vol. 227, no. 6, pp. 3191 – 3211, 2008.

[59] J. Blazek, *Computational Fluid Dynamics : Principles and Applications.* Burlington: Elsevier, 2006.

[60] A. Harten and J. M. Hyman, "Self adjusting grid methods for one-dimensional hyperbolic conservation laws," *Journal of Computational Physics*, vol. 50, no. 2, pp. 235 – 269, 1983.

[61] J. D. Baeder, S. Medida, and T. S. Kalra, *OVERTURNS Simulation of S-76 Rotor in Hover*. AIAA SciTech, American Institute of Aeronautics and Astronautics, Jan 2014.

[62] Lakshminarayan, V. K., and Baeder, J. D., "Computational Investigation of Micro Hovering Rotor Aerodynamics," *Journal of the American Helicopter Society*, vol. 55, no. 1, pp. 14–29, 2010.

[63] Y. S. Jung, B. Govindarajan, and J. Baeder, "Turbulent and unsteady flows on unstructured line-based hamiltonian paths and strands grids," *AIAA Journal*, vol. 55, pp. 1986–2001, jun 2017.

[64] G. Klopfer, C. Hung, R. V. der Wijngaart, and J. Onufer, "A diagonalized diagonal dominant alternating direction implicit (D3ADI) scheme and subiteration correction," in *29th AIAA, Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, jun 1998.

[65] A. C. Kirby, M. J. Brazell, Z. Yang, R. Roy, B. R. Ahrabi, M. K. Stoellinger, J. Sitaraman, and D. J. Mavriplis, "Wind farm simulations using an overset hp-adaptive approach with blade-resolved turbine models," *The International Journal of High Performance Computing Applications*, p. 109434201983296, mar 2019.

[66] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2003.

[67] M. Blanco and D. W. Zingg, "Fast Newton-Krylov Method for Unstructured Grids," *AIAA Journal*, vol. 36, pp. 607–612, April 1998.

[68] V. Schmitt and F. Charpin, "Pressure distributions on the onera-m6-wing at transonic mach numbers," *Experimental Data Base for Computer Program Assessment*, vol. 138, 05 1979.

[69] Norman, T. R., Shinoda, P., Peterson, R. L., and Datta, A., "Full-Scale Wind Tunnel Test of the UH-60A Airloads Rotor," in *67th Annual Forum Proceedings of the American Helicopter Society*, May 3–5, 2011.

[70] M. Potsdam, H. Yeo, and W. Johnson, "Rotor airloads prediction using loose aerodynamic/structural coupling," *Journal of Aircraft*, vol. 43, pp. 732–742, may 2006.

[71] A. Datta, J. Sitaraman, I. Chopra, and J. D. Baeder, "CFD/CSD prediction of rotor vibratory loads in high-speed flight," *Journal of Aircraft*, vol. 43, pp. 1698–1709, nov 2006.

[72] L. Wang and D. Mavriplis, "Implicit solution of the unsteady euler equations for high-order accurate discontinuous galerkin discretizations," Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Jan 2006.

[73] Jung, Y. S., Govindarajan, B., and Baeder, J., "Unstructured/Structured Overset Methods for Flow Solver Using Hamiltonian Paths and Strand Grids," AIAA-Paper-2016-1056, 54th AIAA Aerospace Sciences Meeting, January 4–8, 2016.

[74] D. M. O'Brien, "Analysis of computational modeling techniques for complete rotorcraft configurations," phd. dissertation, School of Aerospace Engineering, Georgia Institute of Technology, 2006.

[75] B.-S. Lee, M.-S. Jung, O.-J. Kwon, and H.-J. Kang, "Numerical simulation of rotor-fuselage aerodynamic interaction using an unstructured overset mesh technique," *International Journal of Aeronautical and Space Sciences*, vol. 11, pp. 1–9, Mar. 2010.

[76] "Simulation of unsteady rotor-fuselage aerodynamic interaction using unstructured adaptive meshes," *Journal of the Korean Society for Aeronautical & Space Sciences*, vol. 33, pp. 11–21, Feb. 2005.

[77] R. Meakin, A. Wissink, W. Chan, and S. Pandya, "On strand grids for complex flows," in *18th AIAA Computational Fluid Dynamics Conference*, p. 3834, 2007.

[78] J. Winslow, H. Otsuka, B. Govindarajan, and I. Chopra, "Basic understanding of airfoil characteristics at low reynolds numbers (104–105)," *Journal of Aircraft*, vol. 55, pp. 1–12, 12 2017.

[79] D. Escobar, I. Chopra, and A. Datta, "Aeromechanical loads on a mars coaxial rotor," AHS Forum, American Helicopter Society, May 2018.

[80] A. Datta, J. Bao, O. Gamard, D. Griffiths, L. Liu, G. Pugliese, B. Roget, and J. Sitaraman, "Design of a martian autonomous rotary-wing vehicle," *Journal of Aircraft*, vol. 40, pp. 461–472, 05 2003.