High-Order Incompressible Computational Fluid Dynamics on Modern Hardware Architectures

by

Niki Andreas Loppi

Submitted to the Department of Aeronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at

IMPERIAL COLLEGE LONDON

April 2019

Abstract

In this thesis, a high-order incompressible Navier-Stokes solver is developed in the Python-based PyFR framework. The solver is based on the artificial compressibility formulation with a Flux Reconstruction (FR) discretisation in space and explicit dual time stepping in time. In order to reduce time to solution, explicit convergence acceleration techniques are developed and implemented. These techniques include polynomial multigrid, a novel locally adaptive pseudo-time stepping approach and novel stability-optimised Runge-Kutta schemes.

Choices regarding the numerical methods and implementation are motivated as follows. Firstly, high-order FR is selected as the spatial discretisation due to its low dissipation and ability to work with unstructured meshes of complex geometries. Being discontinuous, it also allows the majority of computation to be performed locally. Secondly, convergence acceleration techniques are restricted to explicit methods in order to retain the spatial locality provided by FR, which allows efficient harnessing of the massively parallel compute capability of modern hardware. Thirdly, the solver is implemented in the PyFR framework with cross-platform support such that it can run on modern heterogeneous systems via an MPI + X model, with X being CUDA, OpenCL or OpenMP. As such, it is well-placed to remain relevant in an era of rapidly evolving hardware architectures.

The new software constitutes the first high-order accurate cross-platform implementation of an incompressible Navier-Stokes solver via artificial compressibility. The solver and the convergence acceleration techniques are validated for a range of turbulent test cases. Furthermore, performance of the convergence acceleration techniques is assessed with a 2D cylinder test case, showing speed-up factors of over 20 relative to global RK4 pseudo-time stepping when all of the technologies are combined. Finally, a simulation of the DARPA SUBOFF submarine model is undertaken using the solver and all convergence acceleration techniques. Excellent agreement with previous studies is obtained, demonstrating that the technology can be used to conduct high-fidelity implicit Large Eddy Simulation of industrially relevant problems at scale using hundreds of GPUs.

Acknowledgments

I would like to express my most sincere gratitude to Dr. Peter Vincent for his teaching, support and guidance. I could not have wished for a better supervisor.

I am also deeply grateful to Dr. Freddie Witherden for his invaluable help with the PyFR codebase. I wish him best of luck in his new assistant professor appointment. I also would like to thank Professor Antony Jameson for the privilege to collaborate with him and visit his research group at Stanford University. I also wish to thank Assistant Professor Brian Vermeire for collaboration and hosting my research visit at Concordia University.

This research would have not been possible without support from BAE Systems Maritime - Submarines, who funded the project under an EPSRC iCASE studentship. I thank the BAE staff involved in this project.

I wish to thank all new friends that I have met during the PhD. They have made this journey an unforgettable experience. I also thank my friends back at home in Helsinki, who have always been there for me despite the distance.

Finally, I would like to thank my parents, Anne and Vesa, for their encouragement and continuous support. I am dedicating this thesis to them.

Declaration of Originality

All contents of this thesis are author's original work undertaken in the department of Aeronautics at Imperial College London unless otherwise acknowledged. None of the material has been submitted to any other institution.

Copyright Declaration

List of Publications

Journal Publications

1. **A High-Order Cross-Platform Incompressible Navier-Stokes Solver via Artificial Compressibility with Application to a Turbulent Jet**. N. A. Loppi, F. D. Witherden, A. Jameson, P. E. Vincent. Computer Physics Communications, Volume 233, Pages 193-205, 2018

2. **Optimal Runge-Kutta Schemes for Pseudo Time-Stepping with High-Order Unstructured Methods**. B. C. Vermeire, N. A. Loppi, P. E. Vincent. Journal of Computational Physics, Volume 383, Pages 55-71, 2019

3. **Locally Adaptive Pseudo-Time Stepping for High-Order Flux Reconstruction**. N. A. Loppi, F. D. Witherden, A. Jameson, P. E. Vincent. Submitted for publication in Journal of Computational Physics

Oral Presentations

1. **Locally Adaptive Dual Time Stepping for High-Order Flux Reconstruction via Embedded Pair RK-Schemes and a PI-Controller**. N. A. Loppi, F. D. Witherden, A. Jameson, P. E. Vincent. International Conference on Spectral and High-Order Methods 2018, 9-13 July 2018. London, UK

2. **High-Order Incompressible Computational Fluid Dynamics on GPUs and Co-Processors** . N. A. Loppi, F. D. Witherden. A. Jameson, P. E. Vincent. United States National Congress on Computational Mechanics, 7-20 July 2017. Montreal, Canada

3. **Optimal Runge-Kutta Schemes for Pseudo Time Stepping with High-Order Methods**. B. C. Vermeire (presenting author), N. A. Loppi, P. E. Vincent. United States National Congress on Computational Mechanics, 7-20 July 2017. Montreal, Canada

4. **A High-Order GPU-Accelerated Incompressible Navier-Stokes Solver via Artificial Compressibility** N. A. Loppi, F. D. Witherden, A. Jameson, P. E. Vincent. International Conference on Spectral and High-Order Methods 2016, 27 June - 1 July 2016. Rio de Janeiro, Brazil

Poster Presentations

1. **A High-order Cross-platform Incompressible Navier-Stokes Solver via Artificial Compressibility with Application to Submarine Hydrodynamics**. N. A. Loppi, F. D. Witherden, P.E. Vincent. Society for Industrial and Applied Mathematics - Computational Science and Engineering Conference, 25 February - 1 March 2019. Spokane, Washington, USA

2. **High-Order Incompressible CFD on Modern Hardware** . N. A. Loppi, F. D. Witherden. A. Jameson, P. E. Vincent. Finnish Science 100 Symposium, 4 November 2017. London, UK

Contents

List of Figures

List of Tables

Nomenclature

The thesis adopts a convention where repeated dummy indices on the right-hand-side are summed if the indices are not present on the left-hand-side as

$$C_{ijk} = A_{ijl}B_{ilk} = \sum_{l} A_{ijl}B_{ilk} \ .$$

The limits are implied from the context. Furthermore, only Eucledian vectors are written in bold unless otherwise stated.

Coordinate Systems

$\mathcal{M}$	Mapping from transformed to physical coordinates
$\tilde{x}$, $\tilde{y}$, $\tilde{z}$	Transformed coordinates
x, y, z	Physical coordinates

Domains

Ω	Physical solution domain
Ω_e	All elements of type e
$\tilde{\Omega}_e$	Standard element of type e
$\partial\tilde{\Omega}_e$	Boundary of $\tilde{\Omega}_e$
N_D	Number of dimensions

Functions

δ_{ij}	Dirac delta
det	Determinant of matrix
dim	Dimensions of matrix

Indices

α	Field variable

e	Element type
i, j, k	Dummy indices
l	Polynomial multigrid level
n	Element number

Number Sets

$\mathbb{C}$	Complex Numbers
$\mathbb{R}$	Real Numbers

Operators

$\mathcal{B}$	Ghost solution at boundary interface
$\mathcal{C}$	Common solution at element interface
$\mathcal{F}^{\mathrm{e}}_{\alpha}$	Inviscid part of $\mathcal{F}_{\alpha}$
$\mathcal{F}^{\mathrm{v}}_{\alpha}$	Viscous part of $\mathcal{F}_{\alpha}$
$\mathcal{F}_{\alpha}$	Common flux at element interface

Polynomial Expansions

$\mathcal{P}$	Vector space of polynomials
L_{ei}	Orthonormal (modal) basis polynomial i for element type e
l_{ei}	Nodal basis polynomial i for element type e
P	Polynomial order

Diacritic notations

$\hat{\square}$	Vector of unit magnitude
$\square^{f}$	Quantity at flux points
$\square^{u}$	Quantity at solution points
$\square^{(s,q)}$	Component of Runge-Kutta scheme with stage count s and temporal order q
$\square^{\perp}$	Normal to the element interface
$\square^{\mathrm{e}}$	Inviscid part of flux
$\square^{\mathrm{v}}$	Viscous part of flux
$\square^{fq}$	Quantity at interface quadrature points
$\square^{q}$	Quantity at quadrature points
$\tilde{\square}$	Transformed quantity

Chapter 1

Introduction

1.1 High-Order Methods

The Computational Fluid Dynamics (CFD) community benefits from a wide range of methods for solving flow problems. Currently, nominally second-order accurate Finite Volume (FV) methods underpin the majority of industrial CFD, due to their robustness, and ability to work on unstructured meshes of complex geometries. However, over the past three decades, significant research has been undertaken into developing methods with increased spatial orders-of-accuracy. These high-order methods have been shown to offer superior accuracy to low-order methods, with the same or lower computational cost [1, 2], and have demonstrated potential for performing scale-resolving turbulent simulations more efficiently than their low order counterparts. However, issues remain, especially in terms of industrial adoption. These include a high memory footprint for implicit time stepping, and the challenge of generating high-order curved element meshes [2, 3].

Early attempts at developing high-order accurate spatial discretisations were based on Finite Difference (FD) and purely spectral approaches [4, 5]. These methods are computationally very efficient but they require using curvilinear block structured or fully structured grids which limits their use in applications that involve highly complex geometries. The first spatially compact high-order approach suitable for unstructured grids was the Discontinuous Galerkin (DG) method by Reed and Hill [6] for simulating neutron transport. Later, it was popularised and applied to CFD by Cockburn et al. [7, 8]. DG is formulated using the integral representation of the governing system by combining a Finite Element (FE) type polynomial approximation inside the element, and a FV type interface flux. Other popular high-order schemes with a resemblance to nodal DG are the Spectral Difference (SD) methods by Liu et al. [9, 10], which are a generalised version of the staggered grid Chebyshev multidomain method by Kopriva and Kolias [11]. A more recent approach is the Flux Reconstruction (FR) method proposed by Huynh [12] which unifies nodal DG and SD schemes within a single framework.

1.2 Modern Hardware

When combined with explicit time stepping, discontinuous high-order schemes, including all schemes in the FR framework, retain a compact stencil making them well-suited for solving hyperbolic conservation laws. This spatial locality results in minimal communication between elements, which is of particular importance for modern hardware platforms that are characterised by an abundance of compute capability relative to memory bandwidth, extensive parallelism, and low memory per core. GPU-accelerated high-order implementations of hyberbolic systems have been studied extensively over the past decade. One of the earliest attempts was by Klockner et al. [13], who used an explicit GPU-accelerated DG method to solve Maxwell's equations. Subsequently, Castonguay et al. [14, 15] and Lopez et al. [16] developed GPU-accelerated compressible Navier-Stokes solvers with FR discretisation. More recently, Witherden et al. [17, 18] and Jacobs et al. [19] introduced open-source Python-based frameworks for solving the compressible Navier-Stokes equations that support all modern backends via run-time code generation. Both frameworks, PyFR by Witherden et al. and OpenSBLI by Jacobs et al., are massively parallel via the MPI + X approach, where X can be OpenMP for conventional CPUs, CUDA for Nvidia GPUs or OpenCL for AMD GPUs. PyFR uses the FR discretisation in space and also supports heterogeneous computing [20]. OpenSBLI uses the high-order FD method in space and allows easy system extensions with nearly-mathematical syntax. The utility of both frameworks have been demonstrated for a range of compressible flow problems [1, 21, 22, 23].

Significant research has also been undertaken into high-order methods with implicit time-stepping [24, 25, 26, 27, 28, 29]. However, published research on their implementation for modern hardware appears to be very limited because of challenges related to very high storage requirements and increased global communication. Some of the early attempts demonstrating the potential and also highlighting the challenges of implicit time stepping on modern hardware, include work by Watkins et al. [30] and Lou et al. [31] on steady Euler equations and work by Aissa et al. [32] on Reynolds-averaged Navier-Stokes equations (RANS).

1.3 Incompressible Flows

Flows are known to behave as incompressible when the Mach number $M = V/c$, describing the ratio of the advection velocity V and the speed of sound c, approaches zero. Real-life applications involving effectively incompressible flows can be found in a range of sectors and fields of research, including the maritime and automotive industries, meteorology, hydrology and astrophysics.

Solving incompressible flow problems using the compressible Navier-Stokes equations at low M is known to be sub-optimal. Rieper [33] specified three deficiencies related to the approach when M approaches zero:

- **Stiffness**: Large disparity between the acoustic wave speed and advection velocity imposes a very strict limit on the explicit time-step size.
- **Cancellation**: The background pressure is proportional to $1/M^2$. At low M, the pressure fluctuations relative to the background pressure can become so small that they cannot be captured within machine precision. This causes numerical round-off error.
- **Accuracy**: In [34], it was shown via asympotic analysis that standard Riemann

solvers introduce artificial diffusion that is proportial to $1/M$. When local M approaches zero, the excess artificial diffusion can deteriorate solution accuracy.

More effective solution strategies for incompressible flows can be divided into three categories. The first solution strategy is to simulate incompressible flows using a compressible solver but at a compromise M [35, 36, 37]. Instead of prescribing M based on the physics of the application, which may be very low, it can be set in the range of $M \approx 0.2 - 0.3$, yielding a reasonable balance between physical accuracy and the effects of low-Mach deficiencies. The second solution strategy is low-Mach preconditioning in which the governing equations are modified to specifically tackle the aforementioned deficiencies. The system preconditioning techniques [38] decrease the acoustic wave speed close to the advection speed by altering the characteristics of the system. This procedure allows a much larger explicit time-step to be taken, but the temporal accuracy is lost unless the approach is combined with the dual time stepping technique [39, 38]. In addition to preconditioning the system, numerical flux preconditioners i.e. low-Mach Riemann solvers can be used adjust the amount of artificial viscosity in the low-Mach regime. These low-Mach Riemann solvers have been demonstrated to improve accuracy and stability especially at very low M [33]. The third solution strategy is solving the incompressible Navier-Stokes equations, where the density is kept constant and pressure relates directly to the divergence of the velocity field. This reduces the number of governing equations to be solved since energy does not have to be considered.

The most common techniques for solving the incompressible Navier-Stokes equations are the pressure-based operator splitting methods, in which the computation of the velocity field and pressure field is decoupled. A typical algorithm of such type consists of first performing an explicit advection step for computing an intermediate velocity field, followed by solving the pressure with a second order elliptic Poisson equation. This pressure is then used to correct the velocity field. The high-order continuous Galerkin (CG) method, as in the Nektar++ solver [40, 41, 42], has been favoured as the spatial discretisation scheme in pressure-based formulations since it requires less globally coupled degrees of freedom for solving the Poisson equation compared to DG. Nevertheless, several variants of pressure-based methods with DG have also been proposed [43, 44]. Early attempts of this type were found to suffer from instabilities with coarse grids and small time steps, and only recent advances in stabilisation techniques have allowed the approach to be applied to implicit LES simulations [45]. Another discretisation technique that has showed potential for pressure-based incompressible formulations is the Hybridized Discontinuous Galerkin (HDG) method [46, 47]. HDG retains the advantages of a discontinuous discretisation while reducing the number of globally coupled degree by forming the global linear system over a set of variables which exist only at the element boundaries. This so-called trace is assembled by applying static condensation to the degrees of freedom located inside the elements. Furthermore, for applications only involving simple geometries, FD methods with immersed boundary treatment on cartesian grids, as in the Incompact3D solver [48, 49, 50, 51], are an attractive alternative to FE type methods. Their main advantage is that they allow an efficient Poisson solve via Fast Fourier Transforms.

In pressure-based incompressible methods, solving the Poisson equation discretely leads to a linear system, solution of which is known to be a challenge on modern hardware architectures due to global couplings. The literature on pressure-based incompressible algorithms targeting modern hardware is limited. Roca el al. [52] have proposed a GPU-accelerated sparse matrix-vector product for incompressible HDG. Comprehensive work on the topic has been undertaken by Karakus et al. [53] who introduced a GPU-accelerated DG solver that uses a semi-lagragian subcycling approach and computes the pressure with a preconditioned conjugate gradient method. The technology appears promising but its source-code is not public and its applications have been so far limited to 2D laminar test cases. A project that is also closely related to incompressible CFD is the development of the GPU-accelerated algebraic multigrid library AmgX [54] by Nvidia. The library can be used to outsource the Poisson solve to Nvidia GPUs.

An alternative to pressure-based methods is the Artificial Compressibility Method (ACM) of Chorin [55]. Rather than projecting the pressure with a Poisson equation, a coupling pressure term is added to the continuity equation, which leads to a hyperbolic system that satisfies the divergence free velocity constraint in the limit of steady state. Similar to low-Mach precondition techniques, ACM alters the characteristic wave speeds to reduce system stiffness which destroys time accuracy and dual time stepping [56] is required for unsteady simulations. In the dual time stepping approach, physical time is discretised with a backward difference scheme, whose solution is found by marching the governing equations in pseudo time. Therefore, in the context of the ACM, the divergence free velocity constraint is satisfied at each physical time step up to a given tolerance. The ACM formulation is well-suited for modern hardware architectures if explicit dual time stepping is used together with a high-order discontinous discretisation, such as FR, due to minimal communication between elements.

There have been various attempts to apply the ACM in a high-order context. Bassi [57] first succeeded in applying the approach with DG to solve steady incompressible flow problems. Later, Liang et al. [58] extended the method to SD schemes in 2D, providing support for unsteady flows via dual time stepping. Recently, Cox et al. [59] successfully applied the method with FR in 2D, and introduced fully implicit pseudo-time stepping for accelerating the convergence.

1.4 Motivation

In this project, a high-order incompressible Navier-Stokes solver is developed in the Python-based PyFR (www.pyfr.org) [17] framework. The solver is based on the ACM formulation with the FR discretisation in space and explicit dual time stepping in time. In order to reduce time to solution, existing and new explicit convergence acceleration techniques are developed and implemented.

Choices regarding the numerical methods and implementation are motivated as follows. Firstly, high-order FR is selected as the spatial discretisation due to its low dissipation and ability to work with unstructured meshes of complex geometries. Being discontinuous, it also allows the majority of the computation to be performed locally. Secondly, convergence acceleration techniques are restricted to explicit methods in order to retain the spatial locality provided by FR, which allows efficient harnessing of the massively parallel compute capability of modern hardware. Thirdly, the solver is implemented in the PyFR framework with cross-platform support such that it can run on modern heterogeneous systems via an MPI + X model, with X being CUDA, OpenCL or OpenMP. As such, it is well-placed to remain relevant in an era of rapidly evolving hardware architectures.

The new software constitutes the first high-order accurate cross-platform implementation of an incompressible Navier-Stokes solver via artificial compressibility. The technology has applications in a range of sectors, including the maritime and automotive industries, such as studying the wakes of submarines to improve their tail design or minimising the drag of racing cars. All solver technologies presented in this thesis are released open-source as part of the PyFR code base.

1.5 Outline

The thesis is structured as follows. Chapter 2 summarises the FR discretisation for mixed unstructured grids, and Chapter 3 introduces relevant time-integration techniques. Chapter 4 describes the unsteady artificial compressibility method for computing incompressible flows with FR. Furthermore, the chapter details the cross-platform implementation of the unsteady artificial compressibility method and the required changes to the PyFR framework. The remainder of the thesis focuses on explicit convergence acceleration techniques, each given in a separate chapter. Chapter 5 focuses on P-multigrid, Chapter 6 focuses on locally adaptive pseudo-time stepping and Chapter 7 on FR-optimal Runge-Kutta schemes. All convergence acceleration chapters include the relevant theory, implementation, performance study and validation. The performance study is undertaken with a 2D cylinder to allow comparison across different technologies, whereas validation is undertaken with a range of turbulent test cases. In Chapter 8, the utility of the solver at scale is demonstrated with a simulation of the DARPA SUBOFF submarine model. Finally, conclusions are drawn in Chapter 9.

Chapter 2

Flux Reconstruction

2.1 Formulation

The presentation of the FR discretisation for unstructured grids closely follows those of [17, 60]. Consider a finite solution domain $\mathbf{\Omega}$ in Euclidean space $\mathbb{R}^{N_D}$ with a coordinate system $\mathbf{x} = x_i \in \mathbb{R}^{N_D}$, where N_D is the number of dimensions. A conservation law in the domain takes the form

$$\frac{\partial u_\alpha}{\partial t} = -\nabla \cdot \mathbf{f}_\alpha \ , \tag{2.1}$$

where $u_\alpha = u_\alpha(\mathbf{x}, t)$ is the solution state for a field variable α and $\mathbf{f}_\alpha = \mathbf{f}_\alpha(u_\alpha, \nabla \mathbf{u}_\alpha)$ is the associated flux. The domain is discretised using a set of suitable element types ε, such as line elements in $N_D = 1$, quadrilaterals and triangles in $N_D = 2$, and hexahedra and tetrahedra in $N_D = 3$. The elements in the discretised domain must conform according to

$$\mathbf{\Omega} = \bigcup_{e \in \varepsilon} \mathbf{\Omega}_e \ , \qquad \mathbf{\Omega}_e = \bigcup_{n=1}^{|\mathbf{\Omega}_e|} \mathbf{\Omega}_n \ , \qquad \mathbf{\Omega} = \bigcap_{e \in \varepsilon} \bigcap_{n=1}^{|\mathbf{\Omega}_e|} \mathbf{\Omega}_n = \emptyset \ , \tag{2.2}$$

where the subscript e refers to an element type and $|\mathbf{\Omega}_e|$ is the number of elements of that type.

For efficient numerical implementation, all operations are performed in a transformed space with a coordinate system $\tilde{\mathbf{x}} = \tilde{x}_i \in \mathbb{R}^{N_D}$. Each physical element $\boldsymbol{\Omega}_e$ is mapped into its respective transformed standard element $\tilde{\boldsymbol{\Omega}}_e \in \mathbb{R}^{N_D}$ via a mapping function

$$\tilde{\mathbf{x}} = \mathcal{M}_{en}^{-1}(\mathbf{x}) \ , \tag{2.3}$$

$$\mathbf{x} = \mathcal{M}_{en}(\tilde{\mathbf{x}}) \ . \tag{2.4}$$

The Jacobian matrices and determinants related to the mapping are defined as

$$\mathbf{J}_{en}^{-1} = J_{enij}^{-1} = \frac{\partial \mathcal{M}_{eni}^{-1}}{\partial x_j} \ , \qquad \mathbf{J}_{en} = J_{enij} = \frac{\partial \mathcal{M}_{eni}}{\partial \tilde{x}_j} \ , \tag{2.5}$$

$$\mathcal{J}_{en}^{-1} = \det \mathbf{J}_{en}^{-1} = \frac{1}{\mathcal{J}_{en}} \ , \qquad \mathcal{J}_{en} = \det \mathbf{J}_{en} \ . \tag{2.6}$$

Using the above relations, the transformed flux and transformed gradient of the solution can be written as a function of the physical solution as

$$\tilde{\mathbf{f}}_{en\alpha}(\tilde{\mathbf{x}}, t) = \mathcal{J}_{en}(\tilde{\mathbf{x}})\mathbf{J}_{en}^{-1}(\mathcal{M}_{en}(\tilde{\mathbf{x}}))\mathbf{f}_{en\alpha}(\mathcal{M}_{en}(\tilde{\mathbf{x}}), t) \ , \tag{2.7}$$

$$\widetilde{\nabla}\mathbf{u}_{en\alpha}(\tilde{\mathbf{x}}, t) = \mathbf{J}_{en}^{T}(\tilde{\mathbf{x}})\nabla\mathbf{u}_{en\alpha}(\mathcal{M}_{en}(\tilde{\mathbf{x}}), t) \ , \tag{2.8}$$

where $\widetilde{\nabla} = \partial/\partial\tilde{x}_i$. Moreover, Equation 2.1 can be expressed in terms of the transformed divergence of the transformed flux as

$$\frac{\partial u_{en\alpha}}{\partial t} = -\mathcal{J}_{en}^{-1}\widetilde{\nabla} \cdot \tilde{\mathbf{f}}_{en\alpha} \ . \tag{2.9}$$

Figure 2-1 illustrates the mapping between physical and transformed spaces for a quadrilateral element.

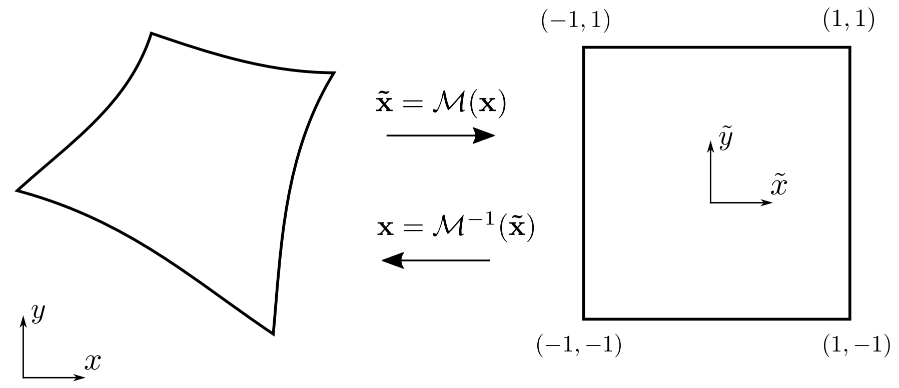

$(-1,1)$
$(1,1)$
$\tilde{\mathbf{x}} = \mathcal{M}(\mathbf{x})$
$\tilde{y}$
$\tilde{x}$
$\mathbf{x} = \mathcal{M}^{-1}(\tilde{\mathbf{x}})$
y
x
$(-1,-1)$
$(1,-1)$

Figure 2-1: Mapping between physical and transformed spaces for a quadrilateral element.

In the FR method, piece-wise discontinuous polynomials of order P are used to represent the solution within each element. Take $\{\tilde{\mathbf{x}}^u_{ei}\}$ to be a set of solution points associated with a given standard element type, where $0 \le i < N_e$, with N_e being the number of solution points. The solution points are often referred to as degrees of freedom (per field variable) and their number depends on the selected polynomial order and point distribution. Examples of such point distributions are Gauss-Legendre or Gauss-Lobatto points, when $N_D = 1$, Witherden-Vincent points [61] for triangles, when $N_D = 2$, and Shunn-Ham points [62] for tetrahedra, when $N_D = 3$. For tensor-product elements $N_e = (P+1)^{N_D}$. In addition to the solution points, a set of element-type-specific flux points $\{\tilde{\mathbf{x}}^f_i\}$, where $0 \le i < N^f_e$, with N^f_e being the number of flux points, are defined at the element interfaces. For tensor product elements $N^f_e = N_{\text{face}}(P+1)^{N_D-1}$, where N_{face} is the number of faces. Each element interface, apart from those at the domain boundaries, contains the flux points of two neighbouring elements. For a given flux point pair eij and $e'i'j'$, the mapping function must return the same physical location according to

$$\mathcal{M}_{en}(\tilde{\mathbf{x}}^f_{ei}) = \mathcal{M}_{e'n'}(\tilde{\mathbf{x}}^f_{e'i'}) \ . \tag{2.10}$$

Figure 2-2 illustrates a second order quadrilateral element with Gauss-Legendre solution points and flux points neighbouring a second order triangular element with

Williams-Shunn solution points and Gauss-Legendre flux points.

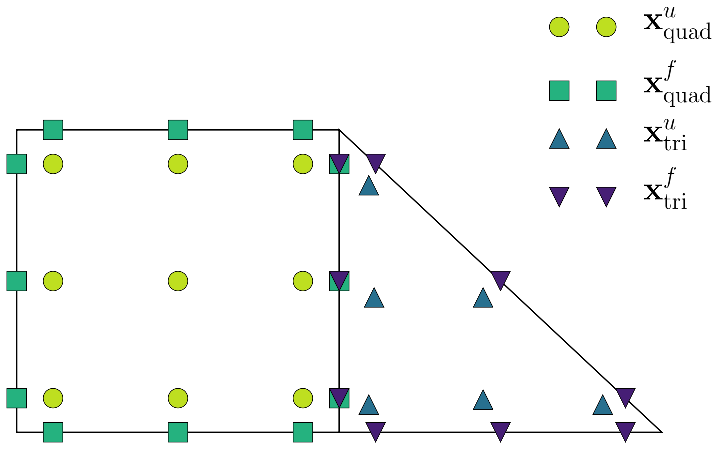

$\mathbf{x}^u_{\text{quad}}$
$\mathbf{x}^f_{\text{quad}}$
$\mathbf{x}^u_{\text{tri}}$
$\mathbf{x}^f_{\text{tri}}$

Figure 2-2: A second order quadrilateral element with Gauss-Legendre solution points and flux points neighbouring a second order triangular element with Williams-Shunn solution points and Gauss-Legendre flux points.

The set of solution points $\{\tilde{\mathbf{x}}^u_{ei}\}$ can be used to define a nodal basis set $\{l_{ei}(\tilde{\mathbf{x}})\}$ of order $P(N_e)$ that spans a polynomial space $\mathcal{P}$ and satisfies the property $l_{ei}(\tilde{\mathbf{x}}^u_{ej}) = \delta_{ij}$. Following the methodology in [63], the nodal basis can be formed by first defining any orthonormal basis set $\{L_{ei}(\tilde{\mathbf{x}})\}$ that spans $\mathcal{P}$, and calculating the entries in the associated Vandermonde matrix $\mathcal{V}_{eij} = L_{ei}(\tilde{\mathbf{x}}^u_{ej})$. Subsequently, a nodal basis can be constructed as $l_{ei}(\tilde{\mathbf{x}}) = \mathcal{V}^{-1}_{eij} L_{ej}(\tilde{\mathbf{x}})$. Figure 2-3 plots the nine nodal basis functions of a 2nd order quadrilateral element with Gauss-Legendre solution points, illustrated as blue nodes. For each basis function, the value at the associated solution point takes a value of one whereas the values at the rest of the solution points are zero.

For computing the right-hand-side, the first step is obtaining the discontinuous solution at the flux points from the interpolating solution polynomial as

$$u^f_{ejn\alpha} = u^u_{ein\alpha} l_{ei}(\tilde{\mathbf{x}}^f_{ej}) \ . \tag{2.11}$$

These values are used to find a common interface solution at the flux points via

$$\mathcal{C}u^f_{ein\alpha} = \mathcal{C}u^f_{e'i'n'\alpha} = \mathcal{C}(u^f_{ein\alpha}, u^f_{e'i'n'\alpha}) \ , \tag{2.12}$$

where $\mathcal{C}(u_R, u_L)$ is a scalar function that returns the common solution, with R and L denoting the interpolated solution states on the right and left side of the boundary, respectively. The second step is to compute the gradient of the solution at the solution points. For this purpose, an FR correction function $\mathbf{g}^f_{ei}(\tilde{\mathbf{x}})$ is formed, which satisfies

$$\hat{\tilde{\mathbf{n}}}_{ej} \cdot \mathbf{g}^f_{ei}(\tilde{\mathbf{x}}^f_{ej}) = \delta_{ij} \ , \tag{2.13}$$

where $\hat{\tilde{\mathbf{n}}}_{ej}$ is the outward facing normal vector. This allows the transformed gradient of the solution at the solution points to be computed as

$$\tilde{\nabla}\mathbf{u}^{u}_{ein\alpha} = \left(\hat{\tilde{\mathbf{n}}}\tilde{\nabla}\right)_{ej} \cdot \mathbf{g}^f_{ej}(\tilde{\mathbf{x}}^u_{ei})\{\mathcal{C}_\alpha u^f_{ejn\alpha} - u^f_{ejn\alpha}\} + u^u_{ekn\alpha}\tilde{\nabla} l_{ek}(\tilde{\mathbf{x}}^u_{ei}) \ , \tag{2.14}$$

which transforms into physical space as

$$\nabla\mathbf{u}^u_{ein\alpha} = \mathbf{J}^{-T}_{ein}\tilde{\nabla}\mathbf{u}^{u}_{ein\alpha} \ , \tag{2.15}$$

where $\mathbf{J}^{-T}_{ein} = \mathbf{J}^{-T}_{en}(\tilde{\mathbf{x}}_{ei})$. As a third step, the transformed discontinuous flux at the solution points is evaluated as

$$\tilde{\mathbf{f}}^u_{ein\alpha} = \mathcal{J}_{ein}\mathbf{J}^{-1}_{ein}\mathbf{f}_\alpha(u^u_{ein\alpha}, \nabla\mathbf{u}^u_{ein\alpha}) \ , \tag{2.16}$$

and its normal component at the flux points is evaluated as

$$\tilde{f}^{f\perp}_{ein\alpha} = l_{ej}(\tilde{\mathbf{x}}^f_{ei})\hat{\tilde{\mathbf{n}}}_{ei} \cdot \tilde{\mathbf{f}}^u_{ejn\alpha} \ . \tag{2.17}$$

The fourth step is to compute the common normal fluxes at the flux point pairs with a function $\mathcal{F}_\alpha(u_R, \nabla\mathbf{u}_R, u_L, \nabla\mathbf{u}_L, \hat{\mathbf{n}})$, that comprises of performing a Riemann solve for the inviscid part and using the LDG approach for the viscous part. The common values are assigned as

$$\mathcal{F}_\alpha f^{f\perp}_{ein\alpha} = -\mathcal{F}_\alpha f^{f\perp}_{e'i'n'\alpha} = \mathcal{F}_\alpha(u^f_{ein}, \nabla \mathbf{u}^f_{ein}, u^f_{e'i'n'}, \nabla \mathbf{u}^f_{e'i'n'}, \hat{\mathbf{n}}_{ein}) \ , \tag{2.18}$$

where

$$\nabla \mathbf{u}^f_{ein\alpha} = l_{ejn}(\tilde{\mathbf{x}}^f_{ei}) \nabla \mathbf{u}^u_{ejn\alpha} \ , \tag{2.19}$$

and they are transformed into the standard element space via

$$\mathcal{F}_\alpha \tilde{f}^{f\perp}_{ein\alpha} = \mathcal{J}^f_{ein} n_{ein} \mathcal{F}_\alpha f^{f\perp}_{ein\alpha} \ , \tag{2.20}$$

with n_{ein} being the magnitude of the physical normal at a flux point. Finally, the divergence of the continuous flux can be computed as

$$\left(\widetilde{\nabla} \cdot \tilde{\mathbf{f}}\right)^u_{ein\alpha} = \widetilde{\nabla} \cdot \mathbf{g}^f_{ej}(\tilde{\mathbf{x}}^u_{ei}) \{\mathcal{F}_\alpha \tilde{f}^{f\perp}_{ejn\alpha} - \tilde{f}^{f\perp}_{ejn\alpha}\} + \tilde{\mathbf{f}}^u_{ekn\alpha} \cdot \widetilde{\nabla} l_{ek}(\tilde{\mathbf{x}}^u_{ei}) \ . \tag{2.21}$$

This serves as the discrete representation of the flux divergence, and the solution can be advanced in time by integrating

$$\frac{\partial u_{ein\alpha}}{\partial t} = -\left(\mathcal{J}^{-1}\right)^u_{ein} \left(\widetilde{\nabla} \cdot \tilde{\mathbf{f}}\right)^u_{ein\alpha} \ . \tag{2.22}$$

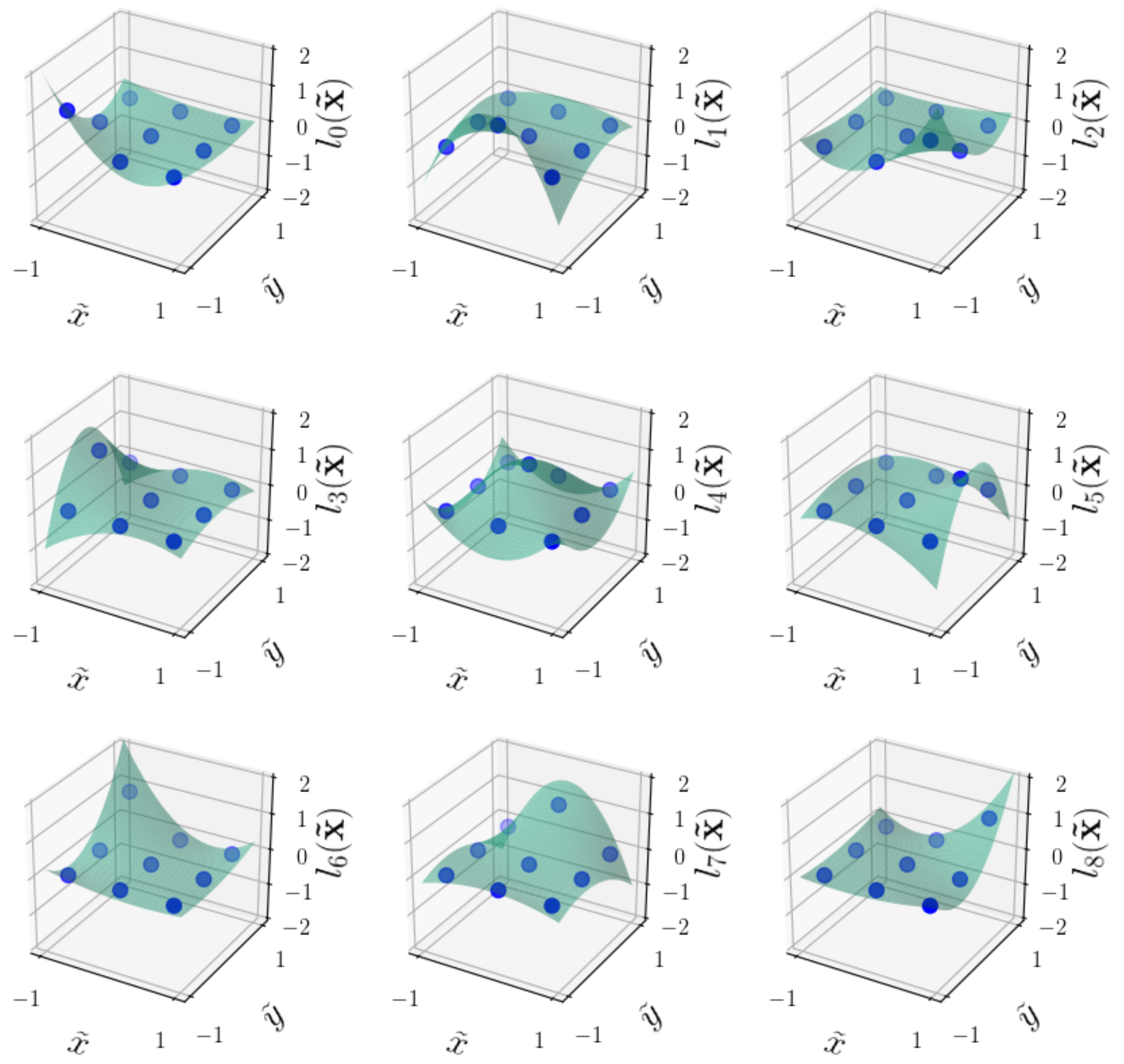


Figure 2-3: Basis functions of a second order quadrilateral element with Gauss-Legendre solution points, illustrated as blue nodes.

2.2 Correction Functions

The FR correction functions $\mathbf{g}^f_{ei}(\tilde{\mathbf{x}})$ distribute the effects of the common interface solution and flux states to the element-interior degrees of freedom, which allows information to propagate between neighbouring elements. The selection of the FR correction function affects the stability and dispersion relations of the FR scheme. In the original paper on FR, Huynh [12] identified correction functions for recovering existing nodal DG and SD schemes for 1D linear advection, and a family of alternative correction functions with varying stability and spatial accuracy properties. These schemes are generally referred to as the g-schemes. Building on Huynh's work on 1D linear advection, Vincent al. [64] identified a one parameter family of correction functions for recovering a range of schemes, referred to as the Vincent-Castonguay-Jameson-Huynh (VCJH) schemes, which they proved to be energy stable. The range of VCJH schemes was extended to advection-diffusion problems in 1D by Castonguay [65], and more recently Vincent et al. [66] extended the VCHJ family to multi-parameter schemes. Furthermore, Allaneau and Jameson [67], Williams and Jameson [68], and Zwanenburg and Nadarajah [69] have showed that all energy stable VCHJ schemes can be cast as filtered DG schemes. The 1D correction functions can be extended to multiple dimensions using tensor products for hexahedral and quadrilateral elements as shown in [12, 70]. Correction functions for simplex elements, including triangles and tetrahedra, have been formulated by Huynh [71], Castonguay et al. [72] and Williams and Jameson [73, 68].

As seen from Equations 2.21 and 2.14, only the divergence of the FR correction function is required for the FR algorithm. The divergence of the FR correction function must sit in the same space as the solution

$$\widetilde{\nabla} \cdot \mathbf{g}^f_{ei}(\tilde{\mathbf{x}}) \in \mathcal{P}_P, \tag{2.23}$$

which makes it of order $P+1$ for tensor product elements. For simplex elements the vector correction function sits in a Raviart-Thomas space [74] of order P. As shown in [18], the divergence of the DG correction function for a flux point j at a standard element interface $\partial\tilde{\Omega}_{ei}$ can be computed as

$$\begin{aligned}\widetilde{\nabla} \cdot \mathbf{g}^f_{e(ij)}(\tilde{\mathbf{x}}) &= L_{ek}(\tilde{\mathbf{x}}) \int_{\partial\tilde{\Omega}_{ei}} \hat{\tilde{\mathbf{n}}} \cdot \mathbf{g}^f_{e(ij)}(\tilde{\mathbf{s}}) L_{ek}(\tilde{\mathbf{s}}) \mathrm{d}\tilde{\mathbf{s}} &(2.24)\\ &= L_{ek}(\tilde{\mathbf{x}}) \int_{\partial\tilde{\Omega}_{ei}} l_{e(ij)}(\tilde{\mathbf{s}}) L_{ek}(\tilde{\mathbf{s}}) \mathrm{d}\tilde{\mathbf{s}}\ . &(2.25)\end{aligned}$$

For convenience, the flux point index convention used previously in this instance is divided into two components i and j, where i runs over the standard element interfaces and j runs over the flux points on that interface. Figure 2-4 plots $\widetilde{\nabla} \cdot \mathbf{g}^f_{e(ij)}(\tilde{\mathbf{x}})$ for a second order quadrilateral element with Gauss-Legendre flux points, illustrated as blue nodes. The index $i = 0$ corresponds to the interface at constant $\tilde{y} = -1$, $i = 1$ to the interface at constant $\tilde{x} = 1$, $i = 2$ to the interface at constant $\tilde{y} = 1$, and $i = 3$ to the interface at constant $\tilde{x} = -1$.

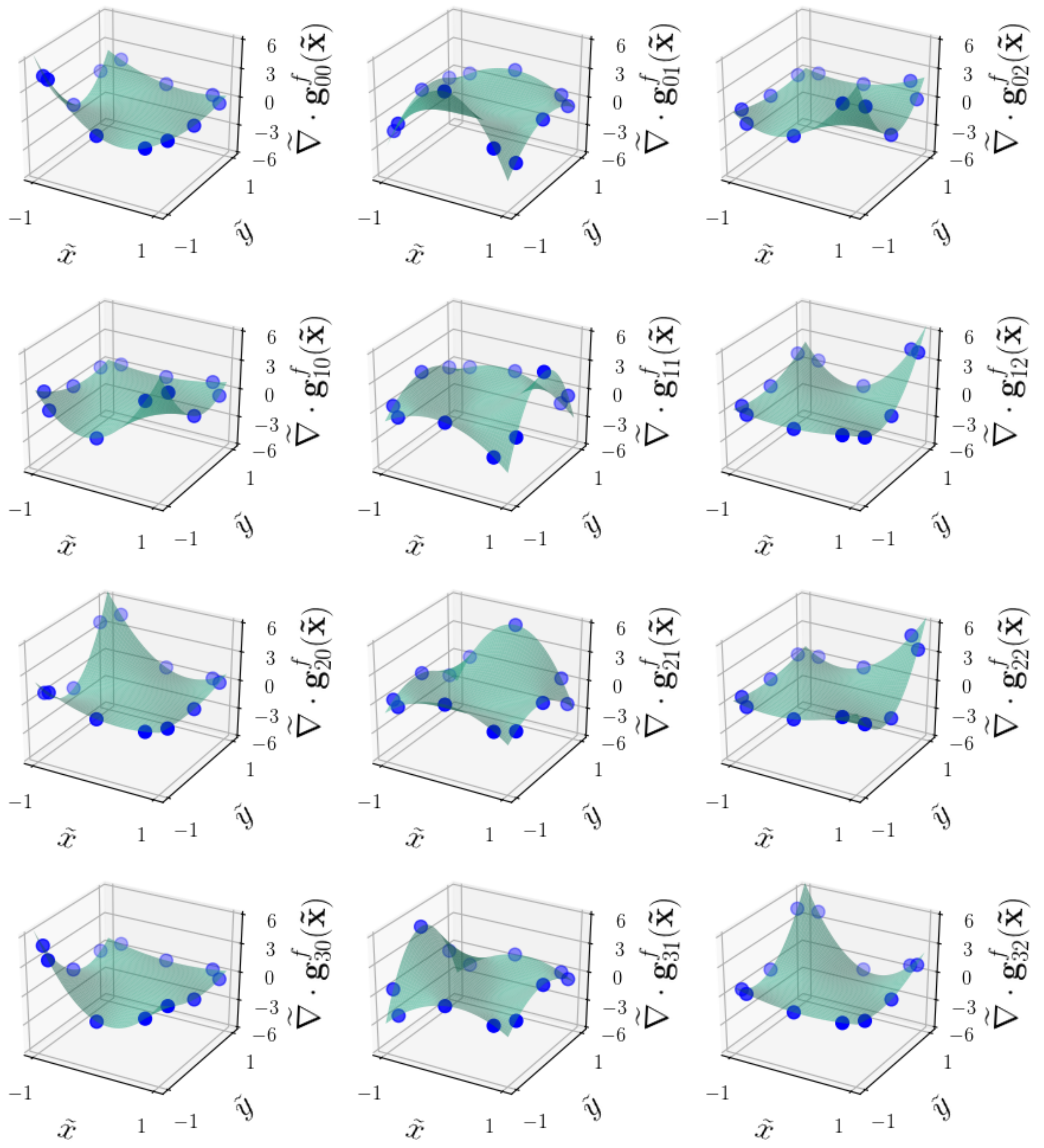


Figure 2-4: Divergence of the correction functions $\widetilde{\nabla} \cdot \mathbf{g}^f_{e(ij)}(\tilde{\mathbf{x}})$ for a second order quadrilateral element with Gauss-Legendre flux points, illustrated as blue nodes.

2.3 Polynomial Aliasing

Aliasing driven instabilities affect a range of high-order FE methods. In DG formulations that use the integral weak form, the origin of aliasing driven instabilities manifests from under-integration of non-linear terms [75]. However, in FR which is based on a differential formulation, their origin is more subtle [76].

Consider a 1D interpolating solution polynomial

$$\tilde{u}(\tilde{x}) = \tilde{u}_i^u l_i(\tilde{x}) \ \in \mathcal{P}_P \ . \tag{2.26}$$

For a non-linear flux function $f = u^2$, the true flux polynomial would sit in a higher order space as

$$\tilde{f}^{\text{true}}(\tilde{x}) = [\tilde{u}(\tilde{x})]^2 \ \in \mathcal{P}_{2P} \ . \tag{2.27}$$

However, since a discontinous flux polynomial $\tilde{f}^{\text{aprx}}(\tilde{x})$ in the FR algorithm is constructed as

$$\tilde{f}^{\text{aprx}}(\tilde{x}) = \tilde{f}_i^u l_i(\tilde{x}) \in \mathcal{P}_P \ , \tag{2.28}$$

where $\tilde{f}_i^u = [\tilde{u}(\tilde{x}_i^u)]^2$ are only evaluated at $P+1$ solution points, an implicit collocation projection of the $\mathcal{P}_{2P}$ polynomial into the $\mathcal{P}_P$ space occurs. This results in unresolved modes ($P+1$ to $2P$) having their energies erroneously aliased to the resolved modes (1 to P). In the context of FR, Jameson and Vincent et al. [77, 76, 64, 66] demonstrated that these aliasing errors lead to instability. Specifically, they considered a broken Sobolev norm of the solution defined as

$$||u||_{P,2}^2 = \sum_{n=0}^{|\Omega|} \int_{x_n}^{x_{n+1}} (u_n)^2 + \frac{c}{2}\left(\frac{\partial^P u_n}{\partial x^P}\right)^2 \mathrm{d}x \ , \tag{2.29}$$

where c is the free VCJH correction function parameter. For energy stability

$$\frac{\mathrm{d}}{\mathrm{d}t}||u||_{P,2}^2 \leq 0 \ . \tag{2.30}$$

For non-linear problems it can be shown [77] that

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}||u||_{P,2}^{2} = \Theta + \sum_{n=0}^{|\Omega|} \epsilon_n \ , \tag{2.31}$$

where Θ is a term that can be proven to be $\Theta \leq 0$ for VCJH schemes, and

$$\epsilon_n = \mathcal{J}_n \tilde{\epsilon}_i \int_{-1}^{1} \frac{\partial \tilde{u}}{\partial \tilde{x}} L_i(\tilde{x}) \ \mathrm{d}\tilde{x} \ , \tag{2.32}$$

where

$$\tilde{\epsilon}_i = \int_{-1}^{1} \tilde{f}^{\mathrm{aprx}} L_i \ \mathrm{d}\tilde{x} - \int_{-1}^{1} \tilde{f}^{\mathrm{true}} L_i \ \mathrm{d}\tilde{x} \tag{2.33}$$

are the aliasing errors and L_i are the normalised Legendre polynomials. Clearly, if these aliasing errors are non-zero, ϵ_n is non-zero. Since the sign of ϵ_n cannot be guaranteed, the stability condition in Equation 2.30 may be violated and aliasing driven instabilities can occur.

To circumvent aliasing errors, three approaches can be considered [18]. Firstly, a favourable set of solution points, such as Gauss-Legendre points, can be used to minimise the error in Equation 2.33 as shown in [76]. Secondly, an exponential filter can be applied to the high order modes as they tend to be most affected by aliasing as shown in [63, 78]. Thirdly, projection to the solution space can be improved with anti-aliasing. In this approach, instead of evaluating $\tilde{f}_i^u$ directly from the flux function, they are computed via an L^2-projection to reduce or completely eliminate the error defined by Equation 2.33. The modal expansion that is obtained via L^2-projection can be expressed as

$$f^{\star}(\tilde{x}) = \gamma_i^{\star} L_i(\tilde{x}) \ , \tag{2.34}$$

where

$$\gamma_i^{\star} = \int_{-1}^{1} L_i(\tilde{x}) f^{\mathrm{true}}(\tilde{x}) \ \mathrm{d}\tilde{x} \tag{2.35}$$

are the modal expansion coefficients that minimise the L^2-error between $f^\star(\tilde{x})$ and $f^{\text{true}}(\tilde{x})$. The expansion coefficients can be computed exactly up to machine precision with a quadrature rule of sufficient degree as

$$\int_{-1}^{1} L_i(\tilde{x}) f^{\text{true}}(\tilde{x})\ \mathrm{d}\tilde{x} = w^q_j L_i(\tilde{x}^q_j) f^{\text{true}}(\tilde{x}^q_j)\ , \tag{2.36}$$

where the superscript q refers to a variable at quadrature points and w^q_j are the quadrature weights. The number of quadrature points required for a given degree depends on the rule. For example, the Gauss-Legendre quadrature rule can integrate polynomials of degree $2N^q-1$ and lower exactly with N^q number of quadrature points. As a guide, a flux polynomial of order P should be anti-aliased using a quadrature with $N^q = (P+2)$ in 1D.

In PyFR, volume flux anti-aliasing is performed by first over-sampling the solution and its gradient at a set of higher order quadrature points as

$$u^q_{ejn\alpha} = u^u_{ein\alpha} l^u_{ei}(\tilde{\mathbf{x}}^q_{ej})\ , \tag{2.37}$$

$$\nabla u^q_{ejn\alpha} = \nabla u^u_{ein\alpha} l^u_{ei}(\tilde{\mathbf{x}}^q_{ej})\ , \tag{2.38}$$

and subsequently computing the transformed flux as

$$\tilde{\mathbf{f}}^q_{ejn\alpha} = \mathcal{J}_{ejn} \mathbf{J}^{-1}_{ejn} \mathbf{f}_\alpha(u^q_{ejn\alpha}, \nabla \mathbf{u}^q_{ejn\alpha})\ . \tag{2.39}$$

This allows the anti-aliased flux at the solution points to be computed as

$$\tilde{\mathbf{f}}^u_{ein\alpha} = L_{ek}(\tilde{\mathbf{x}}^u_{ei}) w^q_{ej} L_{ek}(\tilde{\mathbf{x}}^q_{ej}) \tilde{\mathbf{f}}^q_{ejn\alpha}\ . \tag{2.40}$$

If the elements are curved, the transformation into physical space in Equation 2.22 can also introduce aliasing. This can be addressed with flux divergence anti-aliasing, in which the transformed fluxes and gradients at the quadrature points are computed following the procedures in Equations 2.37 - 2.39, and the divergence of the transformed flux at the quadrature points is computed as

$$\left(\widetilde{\nabla} \cdot \tilde{\mathbf{f}}\right)^{q}_{ein\alpha} = \widetilde{\nabla} \cdot \mathbf{g}^{f}_{ej}(\tilde{\mathbf{x}}^{q}_{ei})\{\mathcal{F}_{\alpha}\tilde{f}^{f\perp}_{ejn\alpha} - \tilde{f}^{f\perp}_{ejn\alpha}\} + \tilde{\mathbf{f}}^{q}_{ekn\alpha} \cdot \widetilde{\nabla} l_{ek}(\tilde{\mathbf{x}}^{q}_{ei}) \ . \tag{2.41}$$

The anti-aliased divergence of flux at the solution points can be computed as

$$\left(\widetilde{\nabla} \cdot \tilde{\mathbf{f}}\right)^{u}_{ein\alpha} = L_{ek}(\tilde{\mathbf{x}}^{u}_{ei}) w^{q}_{ej} L_{ek}(\tilde{\mathbf{x}}^{q}_{ej}) \left(\widetilde{\nabla} \cdot \tilde{\mathbf{f}}\right)^{q}_{ein\alpha} \ . \tag{2.42}$$

Anti-aliasing can also be applied at the interfaces in case of non-linear interface flux functions. In this approach, the solution and its divergence are first interpolated to quadrature points at the boundaries as

$$u^{fq}_{ejn\alpha} = u^{u}_{ein\alpha} l_{ei}(\tilde{\mathbf{x}}^{f}_{ej}) \ , \tag{2.43}$$

$$\nabla u^{fq}_{ejn\alpha} = \nabla u^{u}_{ein\alpha} l_{ei}(\tilde{\mathbf{x}}^{f}_{ej}) \ , \tag{2.44}$$

where the superscript fq denotes a quantity at the interface quadrature points. Using these values, the transformed common interface flux can be computed via

$$\mathcal{F}_{\alpha} f^{fq\perp}_{ejn\alpha} = -\mathcal{F}_{\alpha} f^{fq\perp}_{e'j'n'\alpha} = \mathcal{F}_{\alpha}(u^{fq}_{ejn}, \nabla \mathbf{u}^{fq}_{ejn}, u^{fq}_{e'j'n'}, \nabla \mathbf{u}^{fq}_{e'j'n'}, \hat{\mathbf{n}}_{ejn}) \ , \tag{2.45}$$

$$\mathcal{F}_{\alpha} \tilde{f}^{fq\perp}_{ein\alpha} = \mathcal{J}^{fq}_{ein} n_{ein} \mathcal{F}_{\alpha} f^{fq\perp}_{ein\alpha} \ . \tag{2.46}$$

Finally, the anti-aliased surface flux at the flux points can be computed as

$$\mathcal{F}_{\alpha} f^{f\perp}_{ejn\alpha} = L^{f}_{ek}(\tilde{\mathbf{x}}^{f}_{ei}) w^{fq}_{ej} L^{f}_{ek}(\tilde{\mathbf{x}}^{fq}_{ej}) \mathcal{F}_{\alpha} f^{fq\perp}_{ejn\alpha} \ , \tag{2.47}$$

where L^{f}_{ek} are the interface basis functions and w^{fq}_{ej} are their associated weights.

Chapter 3

Dual Time Stepping

3.1 Overview

The dual time stepping integration technique treats an unsteady problem as consecutive steady-state problems. Most commonly, the physical time is treated implicitly with a backward difference formula (BDF) schemes allowing large physical steps to be taken, and the solution to the steady-state problem is found iteratively by marching the equations in pseudo-time. In this study, the pseudo-time stepping techniques are restricted to explicit Runge-Kutta (RK) schemes to allow efficient use of modern hardware architectures. Since temporal accuracy is unnecessary in pseudo-time, the explicit RK schemes can be combined with any convergence acceleration techniques, such as P-multigrid and local pseudo-time stepping. Section 3.2 details explicit RK schemes, Section 3.3 details implicit BDF schemes and Section 3.4 details the dual time formulation of the two.

3.2 Explicit Runge-Kutta Schemes

Consider an ODE of the form

$$\frac{du}{dt} = g(t, u) \ . \tag{3.1}$$

Explicit RK methods for solving Equation 3.1 can be written in the general form

$$u^{n+1} = u^n + \Delta t \sum_{i=1}^{s} b_i k_i \ , \tag{3.2}$$

$$k_i = g(t + c_i \Delta t, u^n + \Delta t \sum_{j=1}^{i-1} a_{ij} k_j) \ , \tag{3.3}$$

where n denotes a time level, s is the stage count, $\mathbf{A} = a_{ij}$ is an $s \times s$ strictly lower triangular matrix, and $\mathbf{b} = b_i$ and $\mathbf{c} = c_i$ are vectors of length s. The coefficients of a generalised explicit RK scheme are typically presented in a Butcher tableau [79]

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array} \ . \tag{3.4}$$

The constraints for an explicit RK scheme are

$$c_i = \sum_{j=1}^{s} a_{ij} \ , \tag{3.5}$$

$$\sum_{i=1}^{s} b_i = 1 \ , \tag{3.6}$$

$$a_{ij} = 0, \ \ j \geq i \ . \tag{3.7}$$

By applying an order q accurate RK scheme to a linear test problem $du/dt = \omega u$, where $\omega \in \mathbb{C}$, a single time step can be expressed as

$$u^{n+1} = P^{(s,q)}(z) u^n \ , \tag{3.8}$$

where $z = \omega \Delta t$ and

$$P^{(s,q)}(z) = \sum_{j=0}^{s} \gamma_j z^j \tag{3.9}$$

is the associated stability polynomial [80]. The coefficients of the stability polynomial must satisfy

$$\gamma_j = \frac{1}{j!}, \ \ 0 \leq j \leq q \ , \tag{3.10}$$

to obtain order q temporal accuracy. The stability polynomial of an explicit RK scheme can be calculated from the Butcher tableau as

$$P^{(s,q)}(z) = 1 + z\mathbf{b}^T \left(\mathbf{I} - z\mathbf{A}\right)^{-1} \mathbf{e} = \frac{|\mathbf{I} - z\mathbf{A} + z\mathbf{e}\mathbf{b}^T|}{|\mathbf{I} - z\mathbf{A}|} \ , \tag{3.11}$$

where $\mathbf{e}$ is a vector of ones. The stability region S of the RK scheme is defined as

$$S = \left\{ z \in \mathbb{C} : |P^{(s,q)}(z)| \leq 1 \right\} \ , \tag{3.12}$$

and the linear stability condition is

$$\omega \Delta t \subseteq S \ . \tag{3.13}$$

Simply put, the method remains stable for a given Δt if $\omega \Delta t$ lies within a unit circle. Furthermore, the same analysis applies to equations involving spatial derivatives such as linear advection $\partial u / \partial t = \partial u / \partial x$. In this case, it is required that

$$|P^{(s,q)}(\omega^\delta \Delta t)| \leq 1 \ \forall \omega^\delta \tag{3.14}$$

where ω^δ are the eigenvalues associated with the spatial discretisation of $\partial u / \partial x$ which can be obtained via von Neumann analysis. This topic will be revisited in Chapter 7.

Embedded Runge-Kutta pairs are a family of RK schemes of consecutive temporal orders q and $q-1$ for which $\mathbf{A}$ and $\mathbf{c}$ are equal. This property allows the temporal truncation error of the embedded pair to be estimated as

$$\xi(t + \Delta t) = \Delta t \sum_{i=1}^{s} (b_i^{(s,q)} - b_i^{(s,q-1)}) k_i \ . \tag{3.15}$$

For a given truncation error tolerance, the time-step size can be controlled such that the scheme remains stable. This procedure will be detailed in Chapter 6. The Butcher tableaus of the classical RK4 scheme and the RK3(2)4[2R+] embedded pair [81], which are used in the numerical experiments in later Chapters, are shown in

Tables 3.1 and 3.2.

Table 3.1: Butcher tableau of the classical RK4 scheme.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Table 3.2: Butcher tableau of the RK3(2)4[2R+] embedded pair, where the first row of **b** are the $b_i^{(4,3)}$ coefficients resulting in third order temporal accuracy and the second row are the $b_i^{(4,2)}$ coefficients resulting in second order accuracy.

0				
c_2	$\frac{11847461282814}{36547543011857}$			
c_3	$\frac{1017324711453}{9774461848756}$	$\frac{3943225443063}{7078155732230}$		
c_4	$\frac{1017324711453}{9774461848756}$	$\frac{8237718856693}{13685301971492}$	$-\frac{346793006927}{4029903576067}$	
$b_i^{(4,3)}$	$\frac{1017324711453}{9774461848756}$	$\frac{8237718856693}{13685301971492}$	$\frac{57731312506979}{19404895981398}$	$-\frac{101169746363290}{37734290219643}$
$b_i^{(4,2)}$	$\frac{15763415370699}{46270243929542}$	$\frac{514528521746}{5659431552419}$	$\frac{27030193851939}{9429696342944}$	$-\frac{69544964788955}{30262026368149}$

$c_i = A_{i,i-1} + \sum_{j=1}^{i-2} + b_{j-2}^{(4,3)},\ 2 \leq i \leq 4$

3.3 Implicit Backward Difference Formula Schemes

BDF schemes are a range of implicit multistep methods that can be formulated as

$$u^{n+1} = \sum_{i=0}^{s-1} B_{i+1} u^{n-i} + \Delta t B_0 g(t^{n+1}, u^{n+1}) \ , \tag{3.16}$$

where B_i are the BDF-coefficients. Table 3.3 shows the BDF-coefficients for the most commonly used schemes. The stability polynomial of BDFs can be defined as

$$P(z) = \sum_{i=0}^{s-1} B_{i+1}\xi^i + (1 - B_0 z)\xi^s \ . \tag{3.17}$$

The stability condition is that all roots of the stability polynomial must satisfy $\{z \in \mathbb{C} : |\xi| \leq 1\}$. The BDF schemes are A-stable up to 2nd order. With higher order BDFs, two unstable regions form near the real axis. However, these regions are very small for the third order accurate BDF3, meaning it is still stable for a wide range of z. Moreover, it has been successfully applied to unsteady flow simulations [82].

Table 3.3: Coefficients for the BDFs

	B_0	B_1	B_2	B_3
Backward-Euler	1	1	–	–
BDF2	$\frac{2}{3}$	$\frac{4}{3}$	$-\frac{1}{3}$	–
BDF3	$\frac{6}{11}$	$\frac{18}{11}$	$-\frac{9}{11}$	$\frac{2}{11}$

3.4 Dual Time Formulation

A dual time formulation of an ODE can be written as

$$\frac{\partial u}{\partial \tau} + \frac{\partial u}{\partial t} = g(t, u) \ , \tag{3.18}$$

where $\partial u/\partial \tau$ is the pseudo-time derivative which is marched towards zero and $\partial u/\partial t$ is the physical time derivative. Consider treating the ODE implicitly in physical time. Using a BDF2 scheme to discretise the physical time derivative gives

$$\frac{\partial u}{\partial \tau} = -\frac{3u^{n+1} - 4u^n + u^{n-1}}{2\Delta t} + g(t^{n+1}, u^{n+1}) \ , \tag{3.19}$$

where n is the physical time-level counter. Furthermore, consider treating the ODE explicitly in pseudo-time. Using a forward Euler scheme to discretise the pseudo-time derivative gives

$$\frac{u^{n+1,m+1} - u^{n+1,m}}{\Delta\tau} = -\frac{3u^{n+1,m+1} - 4u^{n} + u^{n+1}}{2\Delta t} + g(t^{n+1}, u^{n+1,m}) \ , \tag{3.20}$$

where $\Delta\tau$ is the pseudo-time step and m is the pseudo-time level counter. Note that the leading term in the physical time derivative is treated implicitly also in pseudo-time which is referred to as point-implicit source term treatment [84]. Subsequently, expanding the term as

$$u^{n+1,m+1} = u^{n+1,m} + u^{n+1,m+1} - u^{n+1,m} \ , \tag{3.21}$$

substituting it to Equation 3.20 and grouping the pseudo-time terms gives

$$\begin{aligned}\left(1 + \frac{3\Delta\tau}{2\Delta t}\right) u^{n+1,m+1} - \left(1 + \frac{3\Delta\tau}{2\Delta t}\right) u^{n+1,m} = -\Delta\tau\Big[&\frac{3u^{n+1,m} - 4u^{n} + u^{n+1}}{2\Delta t} \\ &+ g(t^{n+1}, u^{n+1,m})\Big] \ . \end{aligned} \tag{3.22}$$

Solving for $u^{n+1,m+1}$ gives the final discretised form of

$$u^{n+1,m+1} = u^{n+1,m} - \frac{\Delta\tau}{1 + \frac{3\Delta\tau}{2\Delta t}} \left[\frac{3u^{n+1,m} - 4u^{n} + u^{n+1}}{2\Delta t} + g(t^{n+1}, u^{n+1,m})\right] \ . \tag{3.23}$$

When the solution is integrated with respect to pseudo-time, the physical time derivative can be considered as a source term for the right-hand side that is updated at every pseudo-time step. After each pseudo-time step, m is incremented by one. Pseudo-time steps are repeated until the solution in physical time is deemed to have converged. Convergence criteria include that the point-wise L^2 or L^∞-norm of the pseudo-time residuals are reduced below a given tolerance, or a maximum number of pseudo-time steps have been exceeded. After each successful pseudo-time cycle, m is restarted from zero, and n is incremented by one.

Chapter 4

Artificial Compressibility Method

4.1 Formulation

In the ACM formulation of the incompressible Navier-Stokes equations, the conservative variables in three dimensions are

$$u_\alpha = \begin{Bmatrix} p \\ v_x \\ v_y \\ v_z \end{Bmatrix} , \tag{4.1}$$

where p is the pressure and v_x, v_y, and v_z are the velocity components in the x, y, and z directions, respectively. The fluxes are defined as

$$\mathbf{f}_\alpha = \mathbf{f}^{\mathrm{e}}_\alpha - \mathbf{f}^{\mathrm{v}}_\alpha = \begin{Bmatrix} f^{\mathrm{e}}_{\alpha x} \\ f^{\mathrm{e}}_{\alpha y} \\ f^{\mathrm{e}}_{\alpha z} \end{Bmatrix} - \begin{Bmatrix} f^{\mathrm{v}}_{\alpha x} \\ f^{\mathrm{v}}_{\alpha y} \\ f^{\mathrm{v}}_{\alpha z} \end{Bmatrix} , \tag{4.2}$$

where

$$f^{\mathrm{e}}_{\alpha x} = \begin{Bmatrix} \zeta v_x \\ v_x^2 + p \\ v_x v_y \\ v_x v_z \end{Bmatrix} , f^{\mathrm{e}}_{\alpha y} = \begin{Bmatrix} \zeta v_y \\ v_y v_x \\ v_y^2 + p \\ v_y v_z \end{Bmatrix} , f^{\mathrm{e}}_{\alpha z} = \begin{Bmatrix} \zeta v_z \\ v_z v_x \\ v_z v_y \\ v_z^2 + p \end{Bmatrix} , \tag{4.3}$$

$$f^{\mathrm{v}}_{\alpha x} = \nu \begin{Bmatrix} 0 \\ \frac{\partial v_x}{\partial x} \\ \frac{\partial v_y}{\partial x} \\ \frac{\partial v_z}{\partial x} \end{Bmatrix} , f^{\mathrm{v}}_{\alpha y} = \nu \begin{Bmatrix} 0 \\ \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial y} \\ \frac{\partial v_z}{\partial y} \end{Bmatrix} , f^{\mathrm{v}}_{\alpha z} = \nu \begin{Bmatrix} 0 \\ \frac{\partial v_x}{\partial z} \\ \frac{\partial v_y}{\partial z} \\ \frac{\partial v_z}{\partial z} \end{Bmatrix} , \tag{4.4}$$

with ζ being the artificial compressibility relaxation factor and ν the kinematic viscosity.

The artificial compressibility equations are equivalent to the incompressible Navier-Stokes equations apart from the modified continuity equation. The basis for the continuity equation is

$$\frac{\partial \rho^*}{\partial \tau} + \frac{\partial \rho v_x}{\partial x} + \frac{\partial \rho v_y}{\partial y} + \frac{\partial \rho v_z}{\partial z} = 0 \ . \tag{4.5}$$

where $\partial\rho^*/\partial\tau$ is the artificial density pseudo-time derivative that disappears in the limit of steady-state and ρ is the constant physical density [55]. The artificial density pseudo-time derivative can be expanded with the chain rule as

$$\frac{\partial \rho^*}{\partial \tau} = \frac{\partial \rho^*}{\partial p}\frac{\partial p}{\partial \tau} \ . \tag{4.6}$$

From the definition of isentropic speed-of-sound, the artificial wave speed can be defined as

$$c = \sqrt{\frac{\partial p}{\partial \rho^*}} \ . \tag{4.7}$$

Solving Equation 4.7 for $\partial\rho^*/\partial p$ and substituting it into Equation 4.6 gives

$$\frac{\partial \rho^*}{\partial t} = \frac{1}{\zeta}\frac{\partial p}{\partial \tau} \ , \tag{4.8}$$

where $\zeta = c^2$. Substituting Equation 4.8 into Equation 4.5 yields the ACM continuity equation

$$\frac{\partial p}{\partial \tau} = -\left(\frac{\partial \zeta v_x}{\partial x} + \frac{\partial \zeta v_y}{\partial y} + \frac{\partial \zeta v_z}{\partial z}\right) \ . \tag{4.9}$$

Due to the modified continuity equation, the ACM formulation of the incompressible Navier-Stokes equations is hyperbolic in nature, allowing artificial pressure waves with finite speeds. These waves distribute the pressure and disappear in the limit of pseudo steady-state. Eigenvalues of the inviscid flux Jacobian matrices

$$\mathbf{J}_i = \frac{\partial f^{\mathrm{e}}_{\alpha i}}{\partial u} \ , \tag{4.10}$$

are

$$\lambda_i = \{v_i - c_i,\ v_i,\ v_i,\ v_i + c_i\} \ , \tag{4.11}$$

where $c_i = \sqrt{v_i^2 + \zeta}$ is the pseudo speed of sound in directions $i \in \{x,\ y,\ z\}$. The pseudo speed of sound monotonically decreases with decreasing artificial compressibility relaxation factor ζ which can be adjusted to reduce the system stiffness. The common interface flux is defined as

$$\mathcal{F}_\alpha(u_{\mathrm{L}}, \nabla\mathbf{u}_{\mathrm{L}}, u_{\mathrm{R}}, \nabla\mathbf{u}_{\mathrm{R}}, \hat{\mathbf{n}}) = \mathcal{F}^{\mathrm{e}}_\alpha(u_{\mathrm{L}}, u_{\mathrm{R}}, \hat{\mathbf{n}}) - \mathcal{F}^{\mathrm{v}}_\alpha(u_{\mathrm{L}}, \nabla\mathbf{u}_{\mathrm{L}}, u_{\mathrm{R}}, \nabla\mathbf{u}_{\mathrm{R}}, \hat{\mathbf{n}}) \ , \tag{4.12}$$

where $\mathcal{F}^{\mathrm{e}}_\alpha(u_{\mathrm{L}}, u_{\mathrm{R}}, \hat{\mathbf{n}})$ is the inviscid part and $\mathcal{F}^{\mathrm{v}}_\alpha(u_{\mathrm{L}}, \nabla\mathbf{u}_{\mathrm{L}}, u_{\mathrm{R}}, \nabla\mathbf{u}_{\mathrm{R}}, \hat{\mathbf{n}})$ is the viscous part. Throughout the study, the inviscid part is computed with the Rusanov Riemann solver [83] as

$$\mathcal{F}^{\mathrm{e}}_\alpha(u_{\mathrm{L}}, u_{\mathrm{R}}, \hat{\mathbf{n}}) = \hat{\mathbf{n}} \cdot \{\frac{1}{2}(\mathbf{f}^{\mathrm{e}}_{\mathrm{L}} + \mathbf{f}^{\mathrm{e}}_{\mathrm{R}})\} + \frac{1}{2}\max(\lambda_n)(u_{\mathrm{L}} - u_{\mathrm{R}}) \ , \tag{4.13}$$

where $\lambda_n = V_n + c_n$, with $c_n = \sqrt{V_n^2 + \zeta}$, $V_n = (\hat{\mathbf{n}} \cdot \mathbf{v}_{\mathrm{R}} + \hat{\mathbf{n}} \cdot \mathbf{v}_{\mathrm{L}})/2$ and $\mathbf{v} = v_i$. Furthermore, the viscous interface flux is computed using the local discontinuous Galerkin (LDG) approach following [63] as

$$\mathcal{F}^{\mathrm{v}}_\alpha(u_{\mathrm{L}}, \nabla\mathbf{u}_{\mathrm{L}}, u_{\mathrm{R}}, \nabla\mathbf{u}_{\mathrm{R}}, \hat{\mathbf{n}}) = \hat{\mathbf{n}} \cdot \{(\frac{1}{2} + \beta)\mathbf{f}^{\mathrm{v}}_{\mathrm{L}} + (\frac{1}{2} - \beta)\mathbf{f}^{\mathrm{v}}_{\mathrm{R}}\} + \tau(u_{\mathrm{L}} - u_{\mathrm{R}}) \ , \tag{4.14}$$

where $\beta = 0.5$ and $\tau = 0.1$ are prescribed to control the degree of upwinding/downwinding and the solution jump penalisation at the interface. The common interface solution needed for the gradient computation is calculated as

$$\mathcal{C}_\alpha(u_\mathrm{L}, u_\mathrm{R}) = \left(\frac{1}{2} - \beta\right) u_\mathrm{L} + \left(\frac{1}{2} + \beta\right) u_\mathrm{R} \ . \tag{4.15}$$

The dual time stepping formulation applied to the artificial compressibility equations can be written as

$$\frac{\partial u_\alpha}{\partial \tau} = -\left(I_\alpha \frac{\partial u_\alpha}{\partial t} + \nabla \cdot \mathbf{f}_\alpha\right) \ , \tag{4.16}$$

where $I_\alpha = \{0\ 1\ 1\ 1\}^T$ is employed as a coefficient for the physical time derivative to eliminate it from the continuity equation, forcing the solution to be driven towards a divergence free state according to

$$\lim_{\tau \to \infty} \left[\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}\right] = 0 \ . \tag{4.17}$$

For an s-stage RK scheme in pseudo-time and an s_b-stage BDF scheme in physical time, the equation for performing a single pseudo-time step becomes

$$u_\alpha^{n+1,m+1} = u_\alpha^{n+1,m} + \sum_{i=1}^{s} \frac{\Delta\tau b_i}{\alpha_\mathrm{PI}} k_i \ , \tag{4.18}$$

$$\begin{aligned} k_i = &- \nabla \cdot \mathbf{f}_\alpha(u_\alpha^{n+1,m} + \Delta\tau \sum_{j=1}^{i-1} a_{ij} k_i) \\ &- \frac{I_\alpha}{\Delta t B_0}\left(u_\alpha^{n+1,m} + \sum_{j=0}^{s_b-1} B_{j+1} u_\alpha^{n-j}\right) \ , \end{aligned} \tag{4.19}$$

where $\alpha_\mathrm{PI} = 1 + b_i \Delta\tau / B_0 \Delta t$ the scaling coefficient which purpose is to limit $\Delta\tau$ when $\Delta\tau \gg \Delta t$ [84]. With explicit pseudo-time steppers this is rarely the case and $\alpha_\mathrm{PI} = 1$ is enforced in this study.

4.2 Implementation

4.2.1 PyFR Overview

PyFR is a cross-platform framework for solving advection-diffusion problems using the FR approach. One of the main advantages of PyFR is that it produces platform portable code with a single implementation using Python and Mako. Figure 4-1 provides an overview of the framework. PL is the hardware independent Python layer which constitutes the main body of the framework. PL initialises the data layout, precomputes all necessary matrices, and defines the chain of kernel calls that drive the simulation. Distributed memory parallelism and input/output are also handled by PL. Mesh partitioning, when required, is outsourced to METIS [85].

The kernels called by PL can be divided into matrix multiplication kernels MKs and pointwise kernels PKs. The MKs are used for all operations where a time-wise constant operator matrix multiplies a large state matrix, such as interpolation, extrapolation and anti-aliasing. An example is interpolation of field variables from the solution points onto the flux points, which is performed for a given element type as

$$\mathbf{U}_e^f = \mathbf{M}_e^0 \mathbf{U}_e^u \ , \tag{4.20}$$

where

$$(\mathbf{U}_e^u)_{j(n\alpha)} = u_{ejn\alpha}^u \ , \qquad \dim \mathbf{U}_e^u = N_e \times N_v |\mathbf{\Omega}_e| \ , \tag{4.21}$$

$$(\mathbf{U}_e^f)_{i(n\alpha)} = u_{ein\alpha}^f \ , \qquad \dim \mathbf{U}_e^f = N_e^f \times N_v |\mathbf{\Omega}_e| \ , \tag{4.22}$$

$$(\mathbf{M}_e^0)_{ji} = l_{ej}(\tilde{\mathbf{x}}_{ei}^f) \ , \qquad \dim \mathbf{M}_e^0 = N_e^f \times N_e. \tag{4.23}$$

All MKs are off-loaded to GEMM (GEneral Matrix Multiply) subroutines from vendor supplied BLAS (Basic Linear Algebra Subprograms) libraries or bespoke GiMMiK [86] and LIBXSMM [87] kernels. The latter two kernel libraries leverage the *a priori* known structure/sparsity of the operator matrices and can considerably reduce wallclock time in certain circumstances [86].

PKs are used for operations that require pointwise calculations, such as computing non-linear fluxes, Riemann solves and boundary conditions. They are built at runtime by passing platform-unified Mako kernel templates to a Mako derived templating engine that produces low level platform specific source code. The low level code is then compiled as shared libraries and linked to the PL at runtime. The templating engine automatically generates appropriate indexing for vectorisation and efficient memory accesses, which vary across platforms. With CUDA and OpenCL backends, PyCUDA and PyOpenCL [88] Python wrappers handle the kernel compilation, memory allocation, data transfers and execution.

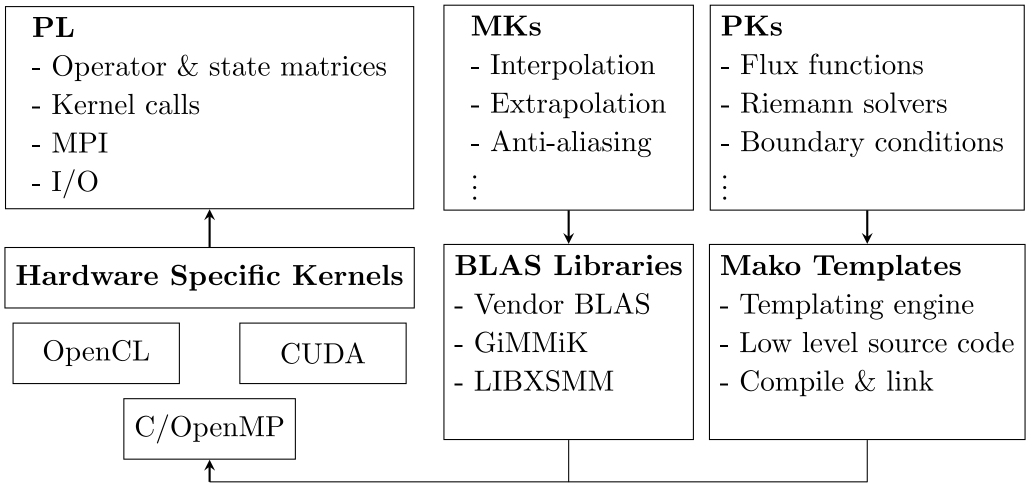


Figure 4-1: The structure of the PyFR framework.

4.2.2 AC Systems

The artificial compressibility solver was implemented as modules under the *pyfr/solvers/* directory. Specifically, two solvers were implemented; *aceuler* containing only the inviscid part and *acnavierstokes* containing the full set of equations. Figure 4-2 illustrates the structure of the ACM solvers in PL, where the class names have been unified to represent both *aceuler* and *acnavierstokes* solvers.

At the top of the ACM solver hierarchy is the `ACSystem` class which inherits a pipeline of kernel calls to perform the FR discretisation of an arbitrary advection-diffusion equation as described in Chapter 2. To complete the discretisation for the ACM equations, `ACSystem` also binds four subclasses: `ACElements`, `ACInternalInt`, `ACMPIInt` and `ACBoundaryInt` that set the state vector u_α and register all PKs for computing the ACM fluxes.

The `ACElements` class registers the PKs for computing fluxes $\tilde{\mathbf{f}}^u_\alpha$ at internal solution points. The common interface flux treatment $\mathcal{F}_\alpha \tilde{\mathbf{f}}^f_\alpha$ is divided into MPI-rank-local PKs and message passing PKs, which are registered in `Internal` and `MPIInt` classes, respectively. This approach allows the data exchange and rank-local computation to happen simultaneously [18]. Additionally, kernels for computing the common interface solution $\mathcal{C}_\alpha u^f_\alpha$ are needed for all interface classes that require the viscous flux calculation. The `BoundaryInt` class registers all boundary condition PKs. All PKs can be found under the *pyfr/solvers/ac*/kernels* directories.

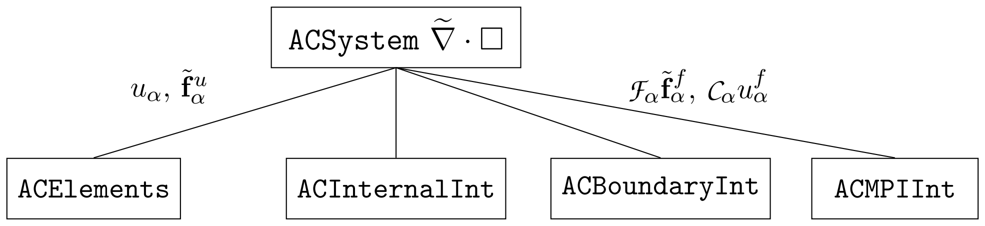

ACSystem $\widetilde{\nabla} \cdot \square$
u_α, $\tilde{\mathbf{f}}^u_\alpha$
$\mathcal{F}_\alpha \tilde{\mathbf{f}}^f_\alpha$, $\mathcal{C}_\alpha u^f_\alpha$
ACElements
ACInternalInt
ACBoundaryInt
ACMPIInt

Figure 4-2: The class hierarchy of the artificial compressibility systems.

Figure 4-3 demonstrates the two-step procedure for registering a PK to the PL. As an example, only a PK for computing the discontinuous flux $\tilde{\mathbf{f}}^u_\alpha$ that is registered to the `ACNavierStokesElements` PL class is considered. First, the kernel must be registered to the backend by providing the location of the Mako template, which is done on line 6. Second, the kernel must be added to the pointwise kernels dictionary as a lambda function, which is done on line 11. The lambda function must specify keyword arguments for all variables that are passed from PL to PK, which include pointers to the input and output matrices `u` (state) and `f`(flux), a pointer to the mapping matrix `smats`, and auxiliary constant variables to facilitate templating `tplargs` and `dims`.

Figure 4-4 shows the templated implementation of the discontinuous flux kernel `tflux` that is registered to PL in Figure 4-3. Input and output arguments on lines 5 to 6 correspond to the keyword arguments in the lambda function. The template consists of `inviscid_flux` and `viscous_flux_add` macro functions on the lines 21 and 36, which work as individual building blocks for the template. These macro functions, which form the expression for the inviscid and viscous fluxes, are expanded on lines 10 and 11. Within the macros, the flux expression is unrolled using a Python-like loop syntax which has direct access to the templating variables passed from Python via `tplargs`. The ${} tags denote a placeholder for variable substitution.

```python
class ACNavierStokesElements(BaseACFluidElements,
                             BaseAdvectionDiffusionElements):
    def set_backend(self, backend, nscalupts, nonce):
        super().set_backend(backend, nscalupts, nonce)

        backend.pointwise.register('pyfr.solvers.acnavstokes.kernels.tflux')

        tplargs = dict(ndims=self.ndims, nvars=self.nvars,
                       c=self.cfg.items_as('constants', float))

        self.kernels['tdisf'] = lambda: backend.kernel(
            'tflux', tplargs=tplargs, dims=[self.nupts, self.neles],
            u=self.scal_upts_inb, smats=self.smat_at('upts'),
            f=self._vect_upts
        )
```

Figure 4-3: Registering the `tflux` PK. which computes the discontinuous flux $\tilde{\mathbf{f}}^{u}_{\alpha}$, to the `ACNavierStokesElements` PL class.

```
<%inherit file='base'/>
<%namespace module='pyfr.backends.base.makoutil' name='pyfr'/>

<%pyfr:kernel name='tflux' ndim='2'
              u='in fpdtype_t[${str(nvars)}]'
              smats='in fpdtype_t[${str(ndims)}][${str(ndims)}]'
              f='inout fpdtype_t[${str(ndims)}][${str(nvars)}]'>
    // Compute the flux (F = Fi + Fv)
    fpdtype_t ftemp[${ndims}][${nvars}];
    ${pyfr.expand('inviscid_flux', 'u', 'ftemp')};
    ${pyfr.expand('viscous_flux_add', 'u', 'f', 'ftemp')};

    // Transform the fluxes
% for i, j in pyfr.ndrange(ndims, nvars):
    f[${i}][${j}] = ${' + '.join('smats[{0}][{1}]*ftemp[{1}][{2}]'
                                  .format(i, k, j)
                                  for k in range(ndims))};
% endfor
</%pyfr:kernel>

<%pyfr:macro name='inviscid_flux' params='s, f'>
    fpdtype_t v[] = ${pyfr.array('s[{i}]', i=(1, ndims + 1))};
    fpdtype_t p = s[0];

    // Mass flux
% for i in range(ndims):
    f[${i}][0] = ${c['ac-zeta']}*v[${i}];
% endfor

    // Momentum fluxes
% for i, j in pyfr.ndrange(ndims, ndims):
    f[${i}][${j + 1}] = v[${i}]*v[${j}]${' + p' if i == j else ''};
% endfor
</%pyfr:macro>

<%pyfr:macro name='viscous_flux_add' params='uin, grad_uin, fout'>
% for i, j in pyfr.ndrange(ndims, ndims):
    fout[${i}][${j+1}] += -${c['nu']}*grad_uin[${i}][${j+1}];
% endfor
</%pyfr:macro>
```

Figure 4-4: The Mako template to generate the `tflux` PK.

4.2.3 Boundary Conditions

In PyFR, boundary conditions are imposed via three ghost states. First, in Equation 4.13 at the boundaries, the right-hand-side solution state u_R is replaced with a prescribed inviscid ghost state $\mathcal{B}^{\text{Rie}}u_{\text{R}}$. Second, in Equation 4.14 at the boundaries, the right-hand-side solution state u_R is replaced with a viscous ghost state $\mathcal{B}^{\text{LDG}}u_{\text{R}}$, which can differ from the inviscid ghost state. Third, also in Equation 4.14, the right-hand-side gradient state $\nabla\mathbf{u}_{\text{R}}$ is replaced with a gradient ghost state $\mathcal{B}^{\text{LDG}}\nabla\mathbf{u}_{\text{R}}$.

Standard Boundary Conditions

The following boundary conditions are available for the ACM solvers. A velocity inlet condition is imposed via ghost states defined as

$$\mathscr{B}^{\text{Rie}} u_\alpha = \{p_{\text{L}} \;\; v_x^b \;\; v_y^b \;\; v_z^b\}^T \ , \tag{4.24}$$

$$\mathscr{B}^{\text{LDG}} u_\alpha = \mathscr{B}^{\text{Rie}} u_\alpha \ , \tag{4.25}$$

$$\mathscr{B}^{\text{LDG}} \nabla \mathbf{u}_\alpha = 0 \ , \tag{4.26}$$

where the superscript b denotes a user-specified free-stream value and the subscript L denotes the domain-side state. A pressure outlet condition is imposed via ghost states defined as

$$\mathscr{B}^{\text{Rie}} u_\alpha = \{p^b \;\; v_{x\text{L}} \;\; v_{y\text{L}} \;\; v_{z\text{L}}\}^T \ , \tag{4.27}$$

$$\mathscr{B}^{\text{LDG}} u_\alpha = \mathscr{B}^{\text{Rie}} u_\alpha \ , \tag{4.28}$$

$$\mathscr{B}^{\text{LDG}} \nabla \mathbf{u}_\alpha = 0 \ . \tag{4.29}$$

A no-slip wall is imposed via ghost states defined as

$$\mathscr{B}^{\text{Rie}} u_\alpha = \{p_{\text{L}} \;\; 2v_x^w - v_{x\text{L}} \;\; 2v_y^w - v_{x\text{L}} \;\; 2v_z^w - v_{x\text{L}}\}^T \ , \tag{4.30}$$

$$\mathscr{B}^{\text{LDG}} u_\alpha = \{p_{\text{L}} \;\; v_x^w \;\; v_y^w \;\; v_z^w\}^T \ , \tag{4.31}$$

$$\mathscr{B}^{\text{LDG}} \nabla \mathbf{u}_\alpha = \nabla \mathbf{u}_{\alpha\text{L}} \ , \tag{4.32}$$

where the superscript w denotes the wall-velocities which are zero for stationary wall. A slip-wall is imposed via ghost states defined as

$$\mathscr{B}^{\text{Rie}} u_\alpha = \{p_{\text{L}} \quad v_{x\text{L}} - 2\hat{n}_x V_{n\text{L}} \;\; v_{y\text{L}} - 2\hat{n}_y V_{n\text{L}} \;\; v_{z\text{L}} - 2\hat{n}_z V_{n\text{L}}\}^T \ , \tag{4.33}$$

$$\mathscr{B}^{\text{LDG}} u_\alpha = \mathscr{B}^{\text{Rie}} u_\alpha \ , \tag{4.34}$$

$$\mathscr{B}^{\text{LDG}} \nabla \mathbf{u}_\alpha = 0 \ . \tag{4.35}$$

Characteristic Riemann-Invariant Boundary Condition for ACM

In addition to the standard boundary conditions, a new characteristic boundary condition was developed. The boundary condition was found to be substantially less reflective than standard velocity inlet and pressure outlet BCs. The characteristic BC is based on Riemann invariants derived in [57]. The boundary condition is parametrised by a target free-stream pressure p_∞ and velocity $\mathbf{v}_\infty$. In addition, a free-stream artificial compressibility factor ζ_∞ is given, which is often required to be significantly larger compared to the global ζ in Equation 4.3. It was found empirically that setting ζ_∞ to be larger than the global artificial compressibility factor ζ made the boundary less reflective.

The 1D Riemann invariants in the interface normal direction are defined as

$$\Gamma_\infty = p_\infty + \frac{1}{2}\left[v_\infty^\perp(v_\infty^\perp - c_\infty) - \zeta_\infty \log(v_\infty^\perp + c_\infty)\right] \ , \tag{4.36}$$

$$\Gamma_L = p_\mathrm{L} + \frac{1}{2}\left[v_\mathrm{L}^\perp(v_\mathrm{L}^\perp + c_\mathrm{L}) + \zeta_\infty \log(v_\mathrm{L}^\perp + c_\mathrm{L})\right] \ , \tag{4.37}$$

where p_L is the pressure interpolated from the interior solution, $v_\mathrm{L}^\perp = \hat{\mathbf{n}} \cdot \mathbf{v}_L$, with $\mathbf{v}_L$ being the velocity interpolated from the interior domain, $v_\infty^\perp = \hat{\mathbf{n}} \cdot \mathbf{v}_\infty$, $c_\mathrm{L} = \sqrt{(v_\mathrm{L}^\perp)^2 + \zeta_\infty}$, and $c_\infty = \sqrt{(v_\infty^\perp)^2 + \zeta_\infty}$. Using these relations, the normal velocity at the boundary can be computed using a Newton-Raphson method as

$$v_b^{\perp,i+1} = v_b^{\perp,i} - \frac{f(v_b^{\perp,i})}{f'(v_b^{\perp,i})} \ , \tag{4.38}$$

where the superscript i is the iteration counter, and

$$f(v_b^\perp) = c_b v_b^\perp + \zeta_\infty \log(v_b^\perp + c_b) + \Gamma_\infty - \Gamma_\mathrm{L} \ , \tag{4.39}$$

$$f'(v_b^\perp) = 2\frac{(v_b^\perp)^2 + \zeta_\infty}{c_b} \ , \tag{4.40}$$

with $c_b = \sqrt{(v_b^\perp)^2 + \zeta_\infty}$. The process converges to machine precision in less than 5 iterations which in PyFR are fully unrolled by the templating engine. Using the converged $v_b^{\perp}$, the individual velocity components at the boundaries can be calculated as

$$\mathbf{v}_b = \begin{Bmatrix} \mathbf{v}_{\mathrm{L}} + \hat{\mathbf{n}}(v_b^{\perp} - \hat{\mathbf{n}} \cdot \mathbf{v}_{\mathrm{L}}) & \text{if } \hat{\mathbf{n}} \cdot \mathbf{v}_{\mathrm{L}} \geq 0 \\ \mathbf{v}_{\infty} + \hat{\mathbf{n}}(v_b^{\perp} - \hat{\mathbf{n}} \cdot \mathbf{v}_{\infty}) & \text{if } \hat{\mathbf{n}} \cdot \mathbf{v}_{\mathrm{L}} < 0 \end{Bmatrix} . \tag{4.41}$$

Finally, the boundary ghost states are prescribed as

$$\mathcal{B}^{\mathrm{Rie}} u_{\alpha} = \mathcal{B}^{\mathrm{LDG}} u_{\alpha} = \begin{Bmatrix} \Gamma_{\mathrm{L}} - \frac{1}{2}\left[v_b(v_b^{\perp} + c_b + \zeta_{\infty} \log(v_b^{\perp} + c_b)\right] \\ \mathbf{v}_b \end{Bmatrix} , \tag{4.42}$$

$$\mathcal{B}^{\mathrm{LDG}} \nabla \mathbf{u}_{\alpha} = 0 \; . \tag{4.43}$$

4.2.4 Dual Integrator

Dual time stepping was implemented as modules under the *pyfr/integrators/* directory. At the top of the class hierarchy is the `DualIntegrator` (`DI`) class which is a composite of `DualController` (`DC`) and `DualStepper` classes (`DS`). The `DC` class handles the physical time management and plugin calls, and `DS` sets the physical time-stepping scheme. Currently available options for physical time-stepping are: Backward-Euler, BDF2 and BDF3. For solving the pseudo-steady-state problem, `DI` initialises an instance of `DualPseudoIntegrator` (`DPI`) class which is a composite of `DualPseudoController` (`DPC`) and `DualPseudoStepper` (`DPS`). The `DPC` class handles the pseudo-time management i.e. convergence monitoring, and implements pseudo-time step control techniques. Currently available options for pseudo-time step control are: None and local-PI, which is detailed in Chapter 6. The `DPS` sets the pseudo-time stepping scheme. Currently available options for pseudo-time stepping are: Euler, tvd-RK3, RK4, RK34, RK45.

Masking axnbpy Kernels

To facilitate the implementation of `DI`, some changes had to be made to the PyFR framework. The first of these was the implementation of masking axnbpy kernels.

The axnbpy kernels are PKs that compute point-wise sums of the form

$$\mathbf{A_0} = c_0\mathbf{A}_0 + c_1\mathbf{A}_1 + c_2\mathbf{A}_2 \; ..., \tag{4.44}$$

where $\mathbf{A}_i$ are solution-sized matrices and c_i are arbitrary real number coefficients. The previous versions of the axnbpy kernels were unaware of the packing methodology making them unable to mask the entries associated with each field variable. This functionality is needed when the physical time-derivate is added to the momentum equations, and when computing the residuals of independent field variables.

PyFR utilises the Array of Structures of Arrays (AoSoA) packing methodology which is illustrated in Figure 4-5. The different shades illustrate the structures i.e. conserved variables in state matrices. AoSoA combines the advantages of Array of structures (AoS) and Structure of Arrays (SoA) methodologies. Firstly, it allows accessing different variables with a constant stride independent of element type. Secondly, it allows coalesced memory access for a single field-variable if k is provided according to hardware specifications.

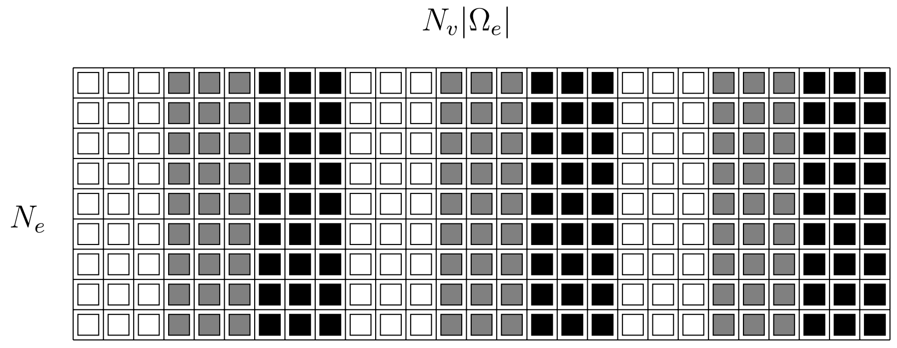


Figure 4-5: AoSoA ($k = 3$) packing methodology with $N_v = 3$.

The masking axnpby kernels were written for all backends separately. Figure 4-6 shows the implementation for the CUDA backend. In the code, the field variables as masked by the `SOA_IX(j, k, ncola)` function on lines 17, 24 and 31 that returns the correct global indices via

$$SOA\_IX(j, v, N_v) = \left(\frac{j}{k} \cdot N_v + v\right) k + j \bmod k \;, \tag{4.45}$$

where j is a column index, v is the field variable being masked and N_v is the total number of field variables. Figure 4-7 shows an example of the usage in a PL function that computes the right-hand side in Equation 4.16. In the code, line 3 computes the negative divergence of the flux, line 6 gets the coefficients for the physical time stepping scheme, lines 9-10 prepare the kernel and the matrices to be added, and line 11 adds the kernel to the execution queue. The masking is parametrised by the `self._subdims` variable which takes values $[1{:}N_v]$ to add the physical time derivative only to the momentum equations.

```
<%inherit file='base'/>
<%namespace module='pyfr.backends.base.makoutil' name='pyfr'/>

__global__ void
axnpby(int nrow, int ncolb, int ldim, fpdtype_t* __restrict__ x0,
       ${', '.join('const fpdtype_t* __restrict__ x' + str(i)
                   for i in range(1, nv))},
       ${', '.join('fpdtype_t a' + str(i) for i in range(nv))})
{
    int i = blockIdx.y*blockDim.y + threadIdx.y;
    int j = blockIdx.x*blockDim.x + threadIdx.x;
    int idx;

    if (j < ncolb && a0 == 0.0)
    {
    % for k in subdims:
        idx = i*ldim + SOA_IX(j, ${k}, ${ncola});
        x0[idx] = ${pyfr.dot('a{l}', 'x{l}[idx]', l=(1, nv))};
    % endfor
    }
    else if (j < ncolb && a0 == 1.0)
    {
    % for k in subdims:
        idx = i*ldim + SOA_IX(j, ${k}, ${ncola});
        x0[idx] += ${pyfr.dot('a{l}', 'x{l}[idx]', l=(1, nv))};
    % endfor
    }
    else if (j < ncolb)
    {
    % for k in subdims:
        idx = i*ldim + SOA_IX(j, ${k}, ${ncola});
        x0[idx] = ${pyfr.dot('a{l}', 'x{l}[idx]', l=nv)};
    % endfor
    }
}
```

Figure 4-6: The axnpby kernel for the CUDA backend that is able to mask individual field variables.

```
    def _rhs_with_dts(self, t, uin, fout):
        # Compute -div f
        self.system.rhs(t, uin, fout)

        # Coefficients for the physical stepper
        svals = [sc/self._dt for sc in self._stepper_coeffs]

        # Physical stepper source addition -div f - dQ/dt
        axnpby = self._get_axnpby_kerns(len(svals) + 1, subdims=self._subdims)
        self._prepare_reg_banks(fout, self._idxcurr, *self._stepper_regidx)
        self._queue % axnpby(1, *svals)
```

Figure 4-7: PL function for computing the right-hand side in 4.16 by leveraging the masking axnbpy kernel.

Convergence Monitoring

With dual time stepping, it is necessary to monitor the convergence of individual field variables via global norms such as the L^2-norm defined as

$$L^2_\alpha = \sqrt{\frac{\sum_{e=1}^{|\varepsilon|}\sum_{n=1}^{|\Omega_e|}\sum_{i=1}^{N_e}(R^u_{ein\alpha})^2}{|\varepsilon||\Omega_e|N_e}}\ , \tag{4.46}$$

or L^∞-norm defined as

$$L^\infty_\alpha = \max|R^u_{ein}|_\alpha\ , \tag{4.47}$$

where

$$R^u_{ein\alpha} = \frac{u^u_{ein\alpha}(\tau+\Delta\tau) - u^u_{ein\alpha}(\tau)}{\Delta\tau}\ . \tag{4.48}$$

To allow this, new reduction kernels were implemented for each backend. Note that the terminology below is adopted from the CUDA programming model. In the OpenCL programming model, shared memory is called local memory and a thread block is called a work group, but their functions are the same.

Computing global residual norms efficiently for individual field variables can be challenging due to the AoSoA packing methodology and hierarchical memory architecture of modern hardware. A reduction operation is not arithmetically intensive, as the number of active threads are recursively halved. Therefore, the reduction algorithm was developed to maximise the memory bandwidth by leveraging the shared memory of each thread block, the latency of which can be up to 100 times lower than that of the global device memory. Furthermore, the reduction kernel implementation was tailored for the AoSoA packing methodology.

Figure 4-8 illustrates the operation of the reduction kernel. In the first stage, when all the threads are active, the algorithm is cascaded making use of every thread to calculate the element-wise residual by reducing sequentially along the number of solution points (y-direction). Simultaneously, each thread masks the element-wise residuals of each variable and loads them into shared memory arrays (N_v shared memory arrays in each block). This approach maximises the amount of work when all the parallel threads are active. After the first step, each block computes the block-wise residual with a strided reduction. Mako templating is used to unroll the reduction loop and to manage the field variable masking. Moreover, a synchronisation barrier is placed after each stage of the reduction. Often the x-dimension of the solution matrix is not divisible by the block dimension. The remainder elements which fall into the last block are reduced with a variable-sized loop using interleaved addressing [89]. By doing this the shared memory arrays do not have to be initialised as zeros. After all stages of the reduction, 0-index threads of each block contain the block-wise residuals each field variable, which are subsequently copied back to global memory and then to the host. Finally, the reduction across blocks and MPI ranks is delegated to NumPy and MPI4Py on the host.

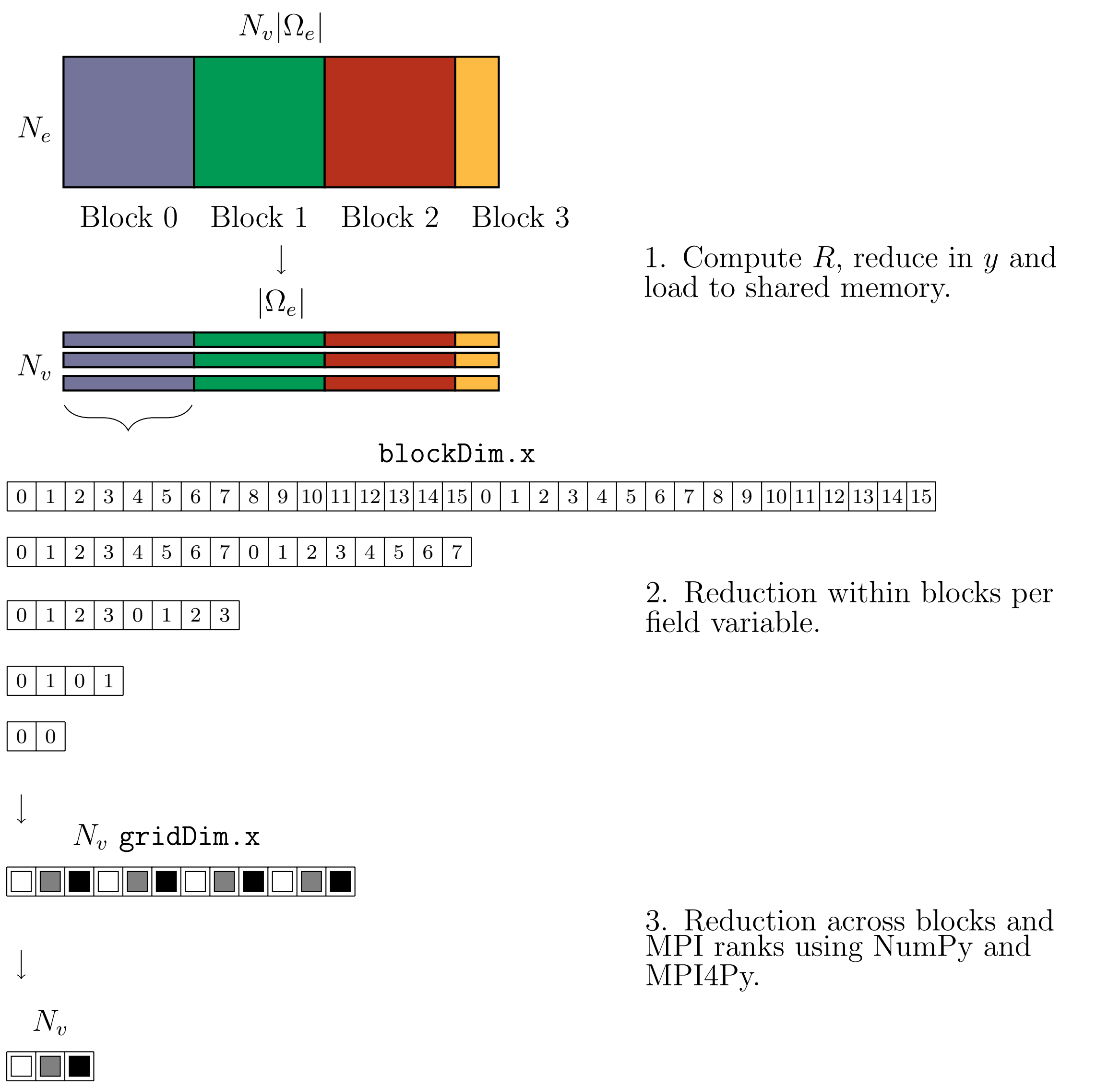


Figure 4-8: Structure of the reduction kernel implementation for computing a global norm of residuals for individual field variables. The numbering corresponds to thread indices `threadIdx.x`.

Chapter 5

P-multigrid

5.1 Overview

The efficiency of dual time stepping depends on the rate of convergence in pseudo-time. The disadvantage of using explicit schemes in pseudo-time is that their stability region is limited by the CFL condition making them inefficient at eliminating low frequency error modes on fine grids [90]. The issue can be addressed by adopting implicit schemes to allow much larger pseudo-time steps [59]. However, the trade-off with implicit schemes is that their storage requirements are significantly higher due to the size of the flux Jacobian matrices, which so far appears to have prohibited their use in large scale 3D applications at high orders. Moreover, many methods for solving the resulting linear system, such as LU-SGS or ILU-GMRES, are not scale invariant and increasing the amount of parallelism reduces the numerical effectiveness of the preconditioner.

The first convergence acceleration technique that was implemented was the P-multigrid approach (perhaps better referred to as multi-P). It attempts to circumvent the CFL condition by correcting the solution at different polynomial levels. Without altering the computational grid, low order representations of the solution, which exhibit a less restrictive CFL limit, can be used to propagate information with a larger $\Delta\tau$. Another quality of P-multigrid is that when the solution is projected to a lower order space, the low frequency error modes appear as higher frequencies respective to the resolution, which allows explicit steppers to eliminate them more effectively.

5.2 Methodology

Dual time stepping with P-multigrid follows the procedures detailed in Chapter 4 with the difference that an additional source term arising from a lower order representation of the solution is added to Equation 4.16. The spatially discrete form of the governing equation is

$$\frac{\partial u^u_{einl\alpha}}{\partial \tau} = R^u_{einl\alpha} - r^u_{einl\alpha} \ , \tag{5.1}$$

where

$$R^u_{einl\alpha} = -\left[\left(\mathcal{J}^{-1}\right)^u_{einl} \left(\tilde{\nabla} \cdot \tilde{\mathbf{f}}\right)^u_{einl\alpha} + I_\alpha \frac{\partial u^u_{einl\alpha}}{\partial t} \right] \ , \tag{5.2}$$

with l denoting the P-multigrid level and $r_{einl\alpha}$ being the additional source term which is zero when $l = P$. Following the methodologies in [58] and [90], a single P-multigrid V-cycle can be formulated according to Algorithm 1.

In this work, the restriction operator I_r is constructed using the generalised Vandermonde matrix as

$$I_r = \mathcal{V}^T_{el'} \tilde{I} \mathcal{V}^{-T}_{el} \ , \tag{5.3}$$

where $\mathcal{V}_{el'} = L_{eil'}(\tilde{\mathbf{x}}^u_{ejl'})$, $\mathcal{V}_{el} = L_{eil}(\tilde{\mathbf{x}}^u_{lej})$, with l' denoting the level onto which the solution is projected, and $\tilde{I}$ is an non-square matrix of zeros, except on its main diagonal, where it has entries of one. The approach corresponds to transforming the nodal solution into a modal representation, removing the highest order mode, and transforming back to a nodal basis. Prolongation is simply performed as Lagrangian interpolation according to

$$I_p = l_{eil}(\tilde{\mathbf{x}}^u_{ejl'}) \ . \tag{5.4}$$

In addition the pseudo-time step size at each level is scaled by a user-specified parameter α_τ as $\Delta\tau_{l'} = \alpha_\tau \Delta\tau_l$, with $l' < l$, due to less restrictive CFL limits at lower levels.

```
for $l \in \{P, ..., l_{min}+1\}$ do
    for $i \in \{0, ..., Niters_l\}$ do
        Smooth according to Equation 5.1
    end
    Calculate residual defect $d_l = r_l - R(u_l)$
    Restrict solution $u^0_{l-1} = \mathrm{I_r}(u_l)$ and store it
    Restrict defect $d_{l-1} = \mathrm{I_r}(d_l)$
    Evaluate source $r_{l-1} = R(u^0_{l-1}) + d_{l-1}$.
end
for $l \in \{l_{min}, ..., P\}$ do
    for $i \in \{0, ..., Niters_l\}$ do
        Smooth according to Equation 5.1
    end
    Calculate correction $\Delta_l = u^0_l - u_l$
    Prolongate correction $\Delta_{l+1} = \mathrm{I_p}(\Delta_l)$
    Add correction $u_{l+1} = u_{l+1} + \Delta_{l+1}$
end
for $i \in \{0, ..., Niters_{l_{max}}\}$ do
    Post-smoothing at the highest level according to Equation 5.1
end
```

Algorithm 1: A single P-multigrid V-cycle between polynomial levels P and $l_{\min}$, where $Niters_l$ denotes the number of smoothing iterations at level l. All indices apart from the level index have been dropped for simplicity.

5.3 Implementation

The P-multigrid implementation is structured around a DualMultiPIntegrator (DMPI) class which replaces the standard DualPseudoIntegrator (DPI) if P-multigrid convergence acceleration is enabled. The DMPI class initialises several MultiPPseudoIntegrator (MPPI) classes, one for each P-multigrid level. All MPPIs are a composite of PseudoController (PC) and PseudoStepper (PS), similar to the standard (DPI), but with certain overridden methods. This allows different pseudo-time stepping schemes and controller methodologies to be used at different levels. In addition, each MPPI initialises an instance of a System class which can compute $\widetilde{\nabla} \cdot \tilde{\mathbf{f}}$ at each level and also allows level-dependent anti-aliasing options. Different MPPIs only communicate via the restriction and prolongation operations that are specified by DMPI. Figure 5-1 illustrates the structure of the P-multigrid implementation.

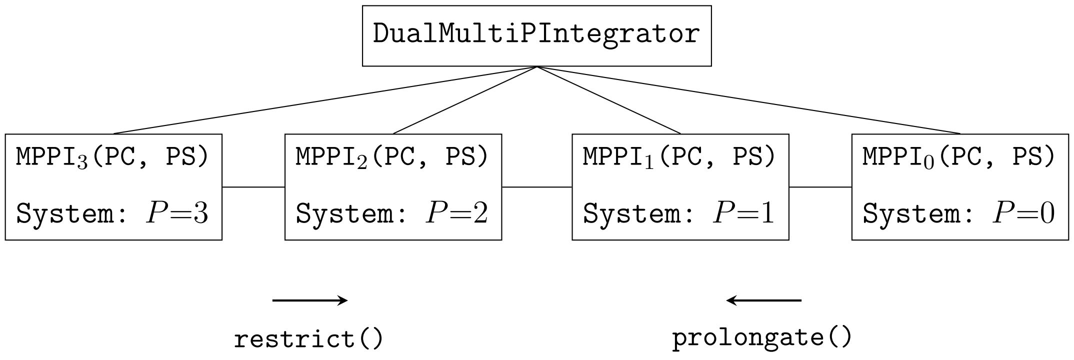


Figure 5-1: The structure of the P-multigrid implementation, consisting of an DualMultiPIntegrator class that launches several MultiPPseudoIntegrators that comprise of a PseudoController and a PseudoStepper.

5.4 2D Cylinder at $Re = 100$

5.4.1 Problem Specification

The performance of P-multigrid was first studied with a laminar 2D cylinder test case at $Re = 100$, based on the cylinder diameter and free-stream velocity. Two simulations were performed: Case Cyl-1 using RK4 pseudo-time stepping and case Cyl-2 using P-multigrid with RK4 smoothing.

Figure 5-2 shows the mixed unstructured computational grid used in the simulations. With the cylinder diameter being 1, the domain consists of a half-circular inflow section with a diameter of 50, and a rectangular wake section with a length of 70. The boundary layer mesh is an O-grid with a height of 0.5 containing 264 quadrilateral elements, whereas the rest of the domain was meshed with 19, 174 triangles. All boundary elements are quadratically curved. The Riemann-invariant-based boundary condition was prescribed at all far-field boundaries with $p_\infty = 1$, $\mathbf{v}_\infty = \{1\ 0\}^T$ and $\zeta_\infty = 100$. The boundary condition was found to be substantially less reflective than conventional BCs used in [58, 59]. A no-slip condition was prescribed on the cylinder surface. The initial condition at $t = 0$ was set as $u_\alpha = \{1\ 1\ 0\}^T$.

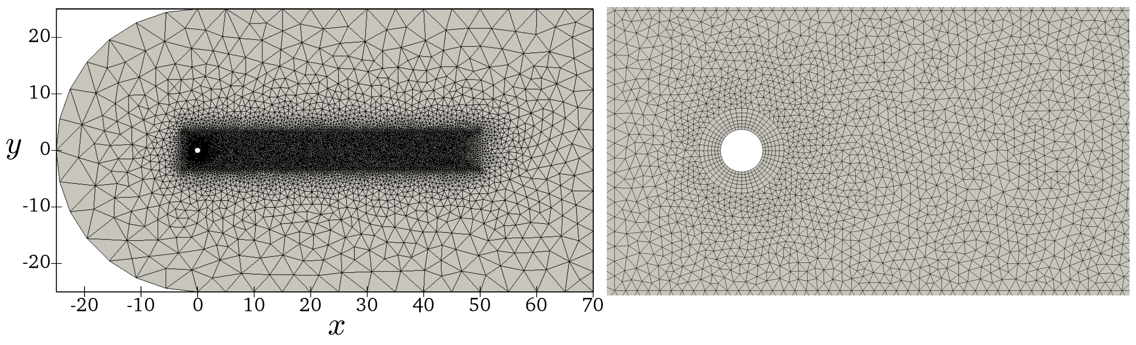


Figure 5-2: The computational grid used in the 2D cylinder simulations (left). A zoom view close to the cylinder (right).

Both cases were run with $P = 4$ and $\zeta = 4$ on a single Nvidia P100 GPU using double precision. The Gauss-Legendre solution point distribution was used for quadrilateral elements and the Williams-Shunn solution point distribution was used for triangular elements. The Gauss-Legendre flux point distribution was used for all interfaces. All flux evaluations were performed without anti-aliasing. Physical time was discretised with a BDF2 scheme, where $\Delta t = 2.5 \times 10^{-2}$. To ensure impartial performance comparisons, the fixed pseudo-time step size $\Delta\tau = 4 \times 10^{-4}$ was found by optimising the convergence via manual bisection for Cyl-1. In Cyl-2, P-multigrid convergence acceleration was performed with a 5-level cycle 1-1-1-1-2-1-1-1-3, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4, and α_τ was set to 2.0. Both cases were run with a convergence tolerance such that the point-wise L^2-norm of the largest velocity residual always driven below 10^{-5}. The L^2-norm of the velocity divergence, which is directly proportional to pressure residual, was approximately 10^{-4} for both cases.

5.4.2 Results

Table 5.1 shows the runtime parameters of the cases together with mean drag coefficients $\overline{C}_D$, Strouhal numbers St and wall-times. The statistics were measured for a period of 100 time units in a fully developed quasi-steady-state from $t = 250.1250$ to $t = 350.1250$. It can be seen that both cases produce identical mean $\overline{C}_D$ and St, which are in line with those of Cox et al. ($\overline{C}_{Dref} = 1.339$ and $St_{ref} = 0.164$), who used implicit pseudo-time stepping with LU-SGS. Moreover, Figure 5-3 shows that temporal evolution of the drag coefficients overlap. The graph of Cyl-2 has been shifted in t such that it is in phase with Cyl-1 because of slight differences in initial transients. From the wall-times, it can be seen that P-multigrid gives a speed-up factor of 6.27 compared to RK4 pseudo-time stepping.

Table 5.1: Summary of the 2D cylinder cases at $Re = 100$, where PS denotes the pseudo-stepper, P-MG denotes P-multigrid, WT denotes the wall-time, and SUF denotes the speed-up factor relative to Cyl-1.

Case	PS	P-MG	α_τ	WT	SUF	$\overline{C}_D$	St
Cyl-1	RK4	Off	-	13:22:23	1.00	1.339	0.166
Cyl-2	RK4	On	2.0	02:07:57	6.27	1.339	0.166

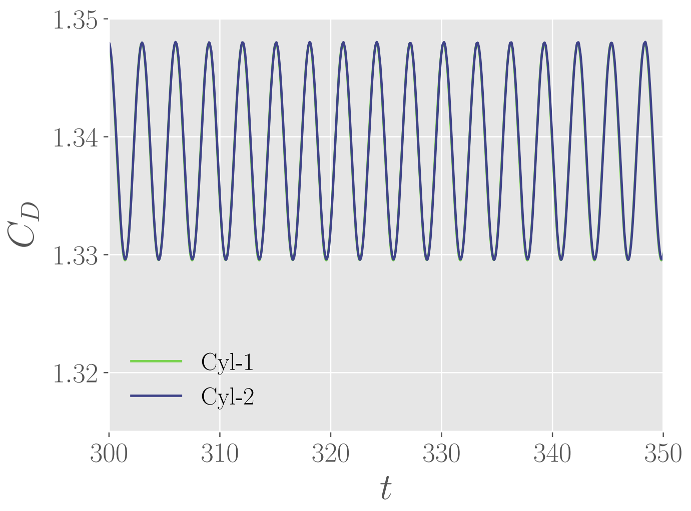


Figure 5-3: Temporal evolution of the drag coefficient for the 2D cylinder case between $t = 300$ and $t = 350$. The graph of Cyl-2 has been shifted in t such that it is in phase with Cyl-1.

5.5 3D Taylor-Green Vortex at $Re = 1,600$

5.5.1 Problem Specification

A 3D Taylor-Green vortex test case at a Reynolds number $Re = 1,600$ based on the diameter and peak velocity of the initial vortices was studied to assess the performance of P-multigrid for a turbulent flow problem and to verify platform independence. In the test case, a set of large vortices interact with each other, transition to turbulence, and finally decay via viscosity. Initial conditions defining vortices with a diameter of 1 and a peak velocity of 1 were prescribed as

$$v_x = \sin(x)\cos(y)\cos(z) \ , \tag{5.5}$$

$$v_y = -\sin(x)\cos(y)\cos(z) \ , \tag{5.6}$$

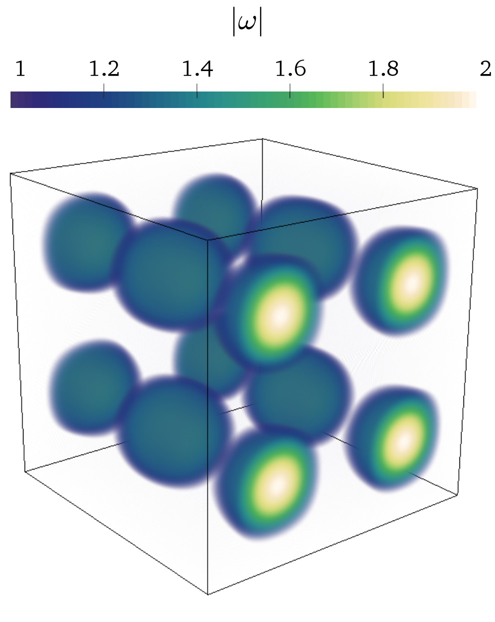
$$v_z = 0 \ , \tag{5.7}$$

$$p = 1 + \frac{1}{16}\left[\cos(2x) + \cos(2y)\right]\left[\cos(2z) + 2\right] \ , \tag{5.8}$$

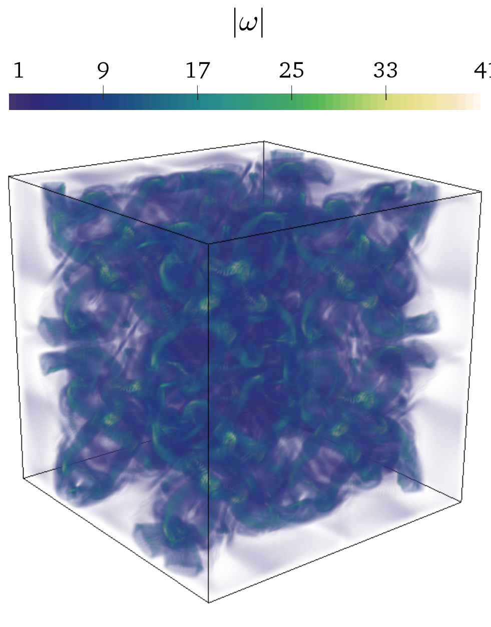
in a periodic domain $-\pi \leq x, y, z \leq \pi$. Figure 5-4 shows volume renderings of the vorticity magnitude for the initial condition and during the enstrophy peak at $t = 8$.

A total of three double precision simulations were performed on two state-of-the-art platforms. Case TGV-1 was run with the CUDA backend on two Nvidia Tesla P100 GPUs using RK4 pseudo-time stepping. Case TVG-2 was run with the CUDA backend on two Nvidia Tesla P100 GPUs using P-multigrid with RK4 smoothing. Case TGV-3 was run with the OpenMP backend on two Intel Xeon Phi 7210 KNL manycore processors using P-multigrid with RK4 smoothing. Due to the extra memory required by the dual time stepping registers and P-multigrid levels, the simulations could not fit into the memory of a single P100 GPU. For an unbiased comparison, all simulations were launched with the optimal GEMM kernel configuration which was found a priori by systematically studying the effect of GiMMiK and LIBXSMM cut-off parameters on the performance. On P100 GPUs, the optimal performance was achieved by offloading all matrix multiplications with number-of-non-zero elements less than 1800 to GiMMiK, which means that only the restriction and prolongation operator between $P = 4$ and $P = 3$ were computed by cuBLAS. On Intel Xeon Phi 7210 KNLs, LIBXSMM outperformed any combination of MKL and GiMMiK and was thereby globally enforced by setting its cut-off based on matrix size to 100,000. Moreover, to achieve optimal performance, MCDRAM of the Intel Xeon Phi 7210 KNL processors was configured to work in the flat mode, and an MPI + OpenMP approach with two MPI ranks per node was used to saturate the interconnects.

All cases were performed on a hexahedral mesh of $52 \times 52 \times 52$ equisized elements using PyFR v1.7.5. The solution polynomial order used in this study was $P = 4$ with a Gauss-Legendre solution point distribution. The Gauss-Legendre flux point distribution was used for the interfaces. All flux evaluations were performed without anti-aliasing. No subgrid-scale turbulence model or spatial filtering was applied, and the simulation can be considered as implicit LES. Physical time was integrated using a BDF2 scheme with a time step size $\Delta t = 0.006$, and the artificial compressibility factor was fixed as $\zeta = 3$. To ensure impartial comparisons, all simulations where performed with a pseudo-time step size $\Delta\tau = 0.0014$ that was optimised for TGV-1 via bisection approach. A 5-level cycle 1-1-1-1-2-1-1-1-3, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4, was used in the P-multigrid accelerated simulations TGV-2 and TGV-3 with $\alpha_\tau = 1.75$. In this instance, to exclude the effects of convergence monitoring on the measured wall-times, all simulations were performed with a fixed number of pseudo-iterations per physical time step, specifically three cycles for the P-multigrid accelerated cases, and 75 RK4 iterations for the single-level case. These values were selected heuristically to ensure both the P-multigrid and single-level cases achieved similar levels of convergence. Furthermore, it was verified *a posteriori* that the velocity field divergence was on average 1.25 times lower for the P-multigrid accelerated cases compared with the single level case, leading to conservative estimates for any P-multigrid speed-up. Table 6.1 shows a summary of the cases.



(a)



(b)

Figure 5-4: Vorticity magnitude for the TGV-2 case at (a) $t = 0$ and (b) $t = 8$.

Table 5.2: Summary of Taylor-Green Vortex simulations.

Case	P-Multigrid	Iterations	Backend	Platform
TGV-1	Off	75 RK4	CUDA	P100
TGV-2	On	3 cycles	CUDA	P100
TGV-3	On	3 cycles	OpenMP	KNL

5.5.2 Results

Figure 5-5 plots the temporal evolution of $\mathcal{D}$, the solution-point-wise L^2-norm of the velocity field divergence evaluated at the end of each physical time step, for TGV-1 and TGV-2. It was found that $\mathcal{D}$ was on average 1.25 times lower for the P-multigrid accelerated cases compared with the single level case. It can also be seen that the effectiveness of P-multigrid is most apparent at the beginning of the simulation, when lengths scales are larger, due to more effective low wave number smoothing. When the large vortices break into smaller structures near the enstrophy peak at $t = 8$, the low wave number smoothing associated with P-multigrid becomes less important, and both TGV-1 and TGV-2 converge to the same level.

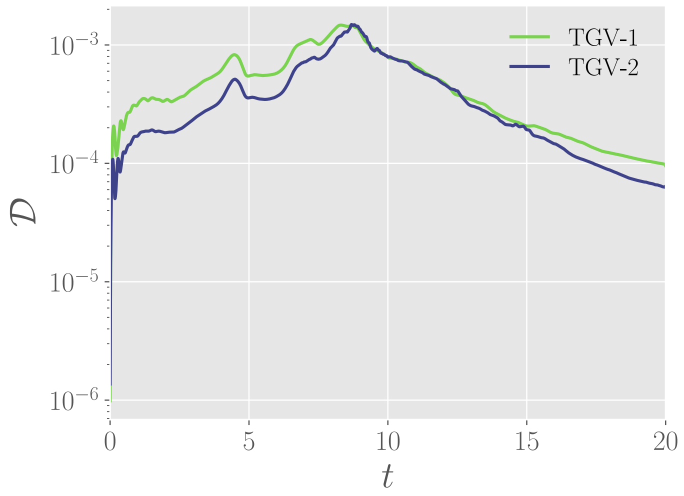


Figure 5-5: The temporal evolution of the solution-point-wise L^2-norm of the velocity field divergence $\mathcal{D}$ evaluated at the end of each physical time step.

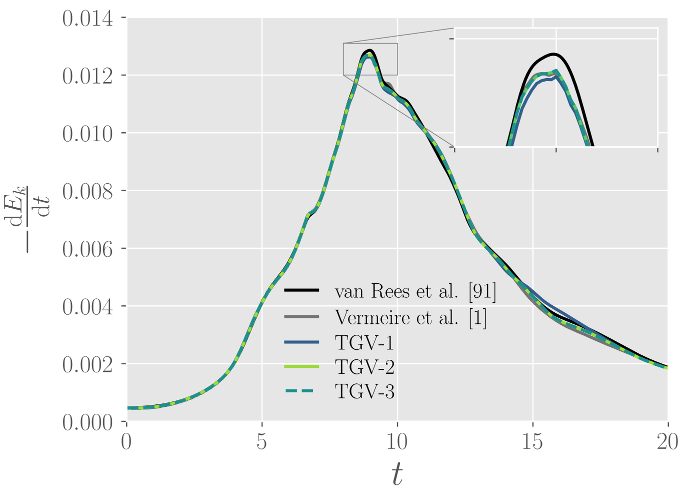
Figure 5-6 shows the dissipation of the total kinetic energy

$$-\frac{\mathrm{d}E_k}{\mathrm{d}t} = -\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{|\Omega|}\int_\Omega \frac{\mathbf{v}\cdot\mathbf{v}}{2}\mathrm{d}\mathbf{x}\right) , \tag{5.9}$$

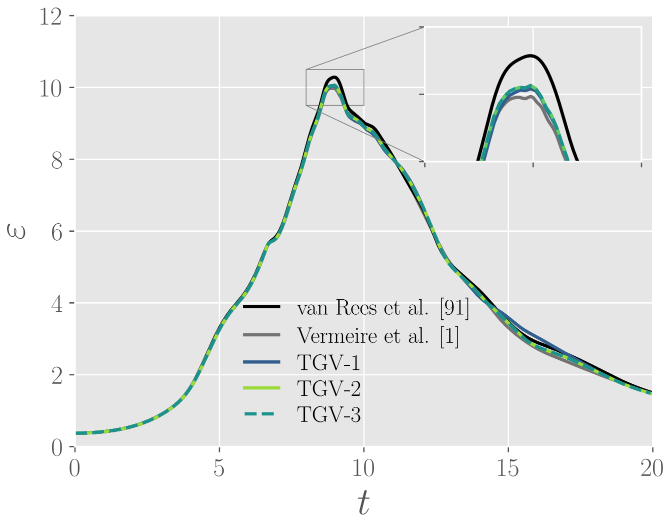
and the temporal evolution of enstrophy

$$\varepsilon = \frac{1}{|\Omega|}\int_\Omega \frac{\boldsymbol{\omega}\cdot\boldsymbol{\omega}}{2}\mathrm{d}\mathbf{x} , \tag{5.10}$$

where $\boldsymbol{\omega}$ is the vorticity vector. The volume integrals were computed using quadrature rules, the quadrature nodes being the 4th order Gauss-Legendre solution points. Figure 5-6 also includes reference results from van Rees et al. [91] who used an incompressible pseudo-spectral code with 512^3 degrees of freedom, and compressible PyFR results at $M = 0.1$ from Vermeire et al. [1] with identical resolution to the current setup. Three observations can be made. Firstly, results of TGV-2 and TGV-3 are identical which verifies the platform and backend independence. Secondly, the P-multigrid accelerated TVG-2 produces slightly more accurate results than TGV-1, which is in line with the better convergence observed in Figure 5-5. Thirdly, the P-multigrid accelerated TGV-2 results are more accurate than the low-Mach compressible solution from Vermeire et al. [1] at the enstrophy peak at $t = 8$, and homogeneous turbulence decay phase $t > 15$, which suggest that the ACM formulation captures the incompressible physics better than a compressible low-Mach approach.



(a)



(b)

Figure 5-6: The temporal evolution of (a) kinetic energy dissipation and (b) enstrophy.

Table 5.3 shows the wall-times for each case. The results confirm that P-multigrid yields an over 3.5 times speed-up compared to single level pseudo-time stepping, while maintaining slightly better solution accuracy and convergence. The results also show that the OpenMP backend on Intel Xeon Phi 7210 KNLs is approximately 2.5 times slower than the CUDA backend on Nvidia Tesla P100s. To put the wall-times in context, the compressible $M = 0.1$ simulation of Vermeire et al. was repeated on the same Nvidia Tesla P100 hardware as Cases 1 and 2, using the configuration file provided in the Electronic Supplementary Material of [1]. The wall-time of the compressible $M = 0.1$ simulation with $P = 4$ and adaptive explicit RK45 time stepping was 05:44:02 on two Nvidia Tesla P100 GPUs. This is 1.6 times longer than the time required for TGV-2 to complete, further demonstrating the utility of an ACM approach with P-multigrid convergence acceleration.

Table 5.3: Wall-times of Taylor-Green vortex simulations.

	TGV-1	TGV-2	TGV-3
Wall-time (hh:mm:ss)	12:45:44	03:38:12	09:07:14

5.5.3 Strong Scaling

A strong scaling study was undertaken to demonstrate the scalability of the solver. A turbulent jet test case at $Re = 10,000$ was used in the study to obtain scaling numbers that are more descriptive of a practical application. The scaling runs were performed on a mesh with 247,250 hexahedral and 596,500 prismatic elements with $P = 4$. The scaling was studied with and without P-multigrid convergence acceleration. With P-multigrid, the cycle was prescribed as 1-1-1-1-2-1-1-1-3, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4, with $\alpha_\tau = 1.7$, and 3 cycles were performed within each physical time-step. Without P-multigrid 75 RK4 pseudo-time steps were performed within each physical time-step. Both pseudo-time stepping techniques lead to $\mathcal{D} \approx 8 \times 10^{-4}$. The full results of the simulation have been published in [60]. Apart from a different pseudo-time stepping technique, the jet test case configuration is identical to the one used for validation in Chapter 7.

Figure 5-7 shows strong scaling from 9 fully loaded Nvidia P100 GPUs to 144 Nvidia P100 GPUs with and without P-multigrid acceleration. Strong scaling in both cases is almost linear up to 36 Nvidia P100 GPUs after which both cases start to tail off. The P-multigrid accelerated case experiences quicker decline due to the presence of low-order iterations with fewer degrees of freedom. For example, on 144

Nvidia P100 GPUs, there are only 1.64 $P = 0$ solution points per CUDA core, whereas the corresponding number at $P = 4$ is over 146.

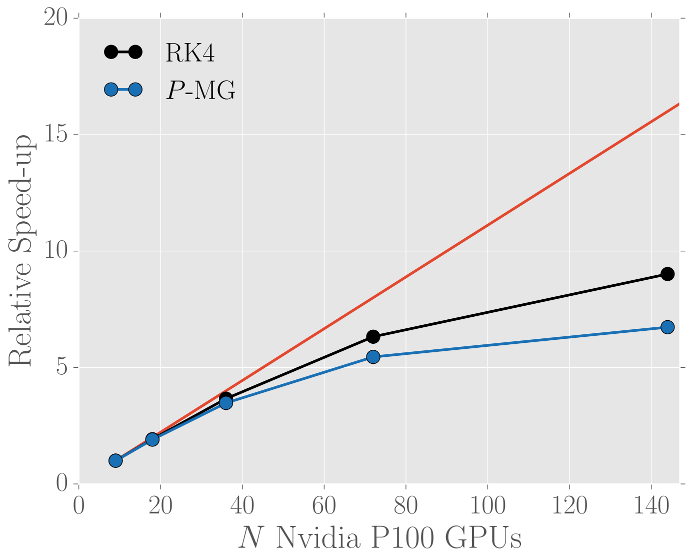


Figure 5-7: Strong scaling of the incompressible jet on Nvidia Tesla P100 GPUs with P-multigrid (P-MG) and without P-multigrid (RK4) .

Chapter 6

Locally Adaptive Pseudo-Time Stepping

6.1 Overview

Local pseudo-time stepping is a widely-used technique for accelerating steady-state convergence explicitly [92, 93, 38]. The majority of current local pseudo-time stepping approaches compute the local pseudo-time step size for element n of type e from the hyperbolic CFL criterion as

$$\Delta\tau_{en} < \frac{h_{en}}{\max(|\lambda_i|_{en})} \ , \tag{6.1}$$

where h_{en} is the characteristic length of the element, and $\max(|\lambda_i|_{en})$ is the local spectral radius of the inviscid flux Jacobian. For viscous dominated flows it is also necessary to consider the parabolic CFL limit defined as

$$\Delta\tau_{en} < \frac{h_{en}^2}{2\mu} \ , \tag{6.2}$$

where $\mu = \rho\nu$, with ρ being the density for compressible Navier-Stokes formulations. Definition of h is often based on heuristics and it can have a considerable effect on the performance and accuracy of local time stepping as shown in [94]. With unstructured grids two popular definitions are the radius of an inscribed sphere or the distance to the element boundary along a streamline [95, 96].

6.2 Methodology

As an alternative to previous approaches, a local pseudo-time stepping technique based on adaptive error control with embedded pair RK schemes is proposed. The main advantage of the technique is that it does not require an expression for the characteristic element size, which is difficult to obtain reliably for curved mixed-element meshes. It also allows a finer level of locality for high-order nodal discretisations, such as FR, since the local time-steps can vary between solution points and field variables. Additionally, it is well-suited to work with P-multigrid convergence acceleration since the pseudo-time steps can be locally projected onto different solution bases.

Embedded pair RK schemes give two approximations of the solution at the next time level, each depending on a different set of b coefficients. In pseudo-time, the difference between these two approximations, often called a truncation error, can be approximated as

$$\xi^u_{ejn\alpha}(\tau + \Delta\tau^u_{ejn\alpha}) = \Delta\tau^u_{ejn\alpha} \sum_{i=1}^{s} (b_i^{(s,q)} - b_i^{(s,q-1)}) k^u_{iejn\alpha} \tag{6.3}$$

for a field variable α at solution point j within element n of type e. In the current study, the local pseudo-time steps $\Delta\tau^u_{ejn\alpha}$ will be controlled such that the truncation error remains small, and the method remains stable. Specifically, a normalised error

$$\sigma^u_{ejn\alpha} = \frac{\xi^u_{ejn\alpha}(\tau + \Delta\tau^u_{ejn\alpha})}{\kappa} \ , \tag{6.4}$$

is kept close to unity, where κ is a user-specified error tolerance parameter. With a PI-controller, the pseudo-time step adjustment factor is computed as

$$(f_{\mathrm{PI}})^u_{ejn\alpha} = \left[(\sigma_{\mathrm{curr}})^u_{ejn\alpha}\right]^{-\phi/q} \left[(\sigma_{\mathrm{prev}})^u_{ejn\alpha}\right]^{-\chi/q} \ , \tag{6.5}$$

where $\phi = 0.4$, $\chi = 0.7$ [79], and the subscripts curr and prev denote the errors computed at the current and previous pseudo-time steps. For improved robustness, the PI-adjustment factor is subsequently scaled by a safety factor f_{safe}, and limited by maximum and minimum factors $f_{\max}$ and $f_{\min}$ as

$$f^u_{ejn\alpha} = \min(f_{\max}, \max(f_{\min}, f_{\text{safe}}(f_{\text{PI}})^u_{ejn\alpha})) \ . \tag{6.6}$$

The new pseudo-time step sizes are computed as

$$(\Delta\tau_{\text{new}})^u_{ejn\alpha} = \max(\Delta\tau_{\min}, \max(\Delta\tau_{\max}, f^u_{ejn\alpha}\Delta\tau^u_{ejn\alpha})) \ , \tag{6.7}$$

with absolute minimum and maximum pseudo-time step limits $\Delta\tau_{\min}$ and $\Delta\tau_{\max}$. In this work, the maximum pseudo-time step limit is defined as $\Delta\tau_{\max} = f_\tau\Delta\tau_{\min}$, where f_τ is a user-specified parameter.

The approach differs from physical time-step adaptation in several ways. Firstly, in physical time-step adaptation, it is often the case that a global physical time-step is adjusted by controlling a global error. In locally adaptive pseudo-time stepping, all operations are kept completely local and independent for each field variable. Secondly, in physical time-step adaptation it is common practice to reject steps that result in $\sigma > 1$, and retake them with a smaller step size. In locally adaptive pseudo-time stepping, retaking steps is unnecessary as long the they remain stable. By setting the adjustment factors $f_{\max}$ conservatively e.g. $f_{\max} = 1.01$ and $f_{\min} = 0.98$, the error can be allowed to exceed the tolerance by a small value momentarily as it will be corrected back within the limits during the next step. Third, in locally adaptive pseudo-time stepping, absolute minimum and maximum limits should be prescribed for the pseudo-time step size to restrict large oscillations. The minimum limit also serves as the initial value for the pseudo-time steps.

6.3 Implementation

The embedded pair explicit Runge-Kutta schemes were implemented as `PS` classes and the local PI-controller was implemented as a `PC` class. In addition, two PKs had to be implemented; one for computing the local error estimates and the other for multiplying the right-hand-side of Equation 4.16 with the local pseudo-time steps.

6.4 Combining with P-Multigrid

To further accelerate pseudo-steady-state convergence, the locally adaptive pseudo-time stepping technique can be coupled with the P-multigrid. Since each `MPPI` in P-multigrid comprises of an arbitrary `PC` and `PS`, locally adaptive pseudo-time stepping can be enabled by simply prescribing `PC` as the PI-controller and `PS` as any of the embedded pair explicit RK schemes. Two different approaches for combining P-multigrid were studied. The first approach let all `MPPI`s perform pseudo-time step control independently of each other. Despite its potential for being fully automated, it lacked robustness in finding the local pseudo-time steps at the intermediate levels and was therefore discarded. The second approach performs the pseudo-time step control only at the highest level `MPPI` and the local pseudo-time steps are projected onto different bases locally. Additionally, the projected values at lower levels are increased with a user-specified scaling factor due to less restrictive CFL limits. The second approach was found to be superior in terms of robustness, and it also has a smaller memory footprint and computational overhead, since only the highest level performs the pseudo-time step adaptation.

A single P-multigrid V-cycle with locally adaptive pseudo-time stepping at the highest level can be performed according to Algorithm 2. The pseudo-time steps are restricted to lower levels with

$$I_{r\tau} = \alpha_\tau \mathcal{V}_{el'}^{T} \tilde{I} \mathcal{V}_{el}^{-T} \ , \tag{6.8}$$

where the leading coefficient α_τ is a scaling factor which takes a user-specified value to increase the local pseudo-time steps sizes due to less restrictive CFL limits at lower levels.

6.5 2D Cylinder at $Re = 100$

6.5.1 Problem Specification

The performance of locally adaptive pseudo-time stepping was first studied with a 2D cylinder test case at $Re = 100$ using the same setup as in Section 5.4. Two simulations were performed: Case Cyl-3 with locally adaptive RK3(2)4[2R+] pseudo-time stepping and case Cyl-4 with P-multigrid acceleration and locally adaptive RK3(2)4[2R+] smoothing.

As in Section 5.4, both cases were run with $P = 4$ and $\zeta = 4$ on a single Nvidia P100 GPU using double precision and physical time was discretised with a BDF2 scheme, where $\Delta t = 2.5 \times 10^{-2}$. The locally adaptive pseudo-time stepping parameters were set as $f_{\max} = 1.01$, $f_{\min} = 0.98$, $\kappa = 10^{-6}$ and $\Delta\tau_{min} = 4 \times 10^{-4}$, and $f_\tau = 8.0$ in both cases. P-multigrid convergence acceleration was performed with a 5-level cycle 1-1-1-1-2-1-1-1-3, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4, and $\alpha_\tau = 1.6$. As in Section 5.4, both cases were run with a convergence tolerance such that the point-wise L^2-norm of the largest velocity residual was always driven below 10^{-5}. The L^2-norm of the velocity divergence, which is directly proportional to pressure residual, was approximately 10^{-4} in both cases.

6.5.2 Results

Table 1 shows the runtime parameters, mean drag coefficients $\overline{C}_D$, Strouhal numbers St, and wall-times of the new cases together with the data of cases Cyl-1 and Cyl-2 from Chapter 5. The statistics of the new cases were measured for the same period of 100 time units between $t = 250.1250$ and $t = 350.1250$ as Cyl-1 and Cyl-2. It can be seen that all cases produce identical $\overline{C}_D$ and St. Furthermore, Figure 6-1

```
for $l \in \{P, ..., l_{min}+1\}$ do
    if $l$=$P$ then
        for $i \in \{0, ..., Niters_l\}$ do
            Pseudo-step Equation 5.1 to update $u_l$
            Update $\Delta\tau_l$ with locally adaptive pseudo-time stepping
        end
    else
        for $i \in \{0, ..., Niters_l\}$ do
            Pseudo-step Equation 5.1 to update $u_l$
        end
    end
    Calculate residual defect $d_l = r_l - R(u_l)$
    Restrict solution $u^0_{l-1} = I_r(u_l)$ and store it
    Restrict defect $d_{l-1} = I_r(d_l)$
    Restrict pseudo-time steps $\Delta\tau_{l-1} = I_{r\tau}(\Delta\tau_l)$ with $\alpha_\tau > 1$
    Evaluate source $r_{l-1} = R(u^0_{l-1}) + d_{l-1}$
end
for $l \in \{l_{min}, ..., P\}$ do
    for $i \in \{0, ..., Niters_l\}$ do
        Pseudo-step Equation 5.1 to update $u_l$
    end
    Calculate correction $\Delta_l = u^0_l - u_l$
    Prolongate correction $\Delta_{l+1} = I_p(\Delta_l)$
    Add correction $u_{l+1} = u_{l+1} + \Delta_{l+1}$
end
for $i \in \{0, ..., Niters_P\}$ do
    Pseudo-step Equation 5.1 to update $u_P$
    Update $\Delta\tau_P$ with locally adaptive pseudo-time stepping
end
```
Algorithm 2: A single P-multigrid V-cycle with locally adaptive pseudo-time stepping polynomial between levels P and $l_{\min}$, where $Niters_l$ denotes the number of smoothing iterations at level l. All indices apart from the level index have been dropped for simplicity.

shows that the temporal evolution of the drag coefficients agrees with the previous runs. From the wall-times, it can be seen that locally adaptive pseudo-time stepping leads to a speed-up factor of 4.16 compared to RK4 pseudo-time stepping. Although not as effective as P-multigrid acceleration with a speed-up factor of 6.27, it is still substantial. Combining the two techniques together yields a total speed-up factor of 15.24.

Table 6.1: Summary of the 2D cylinder cases at $Re = 100$, where PS denotes the pseudo-stepper, LAPTS denotes locally adaptive pseudo-time stepping, P-MG denotes P-multigrid, WT denotes the wall-time, and SUF denotes the speed-up factor relative to Cyl-1.

Case	PS	LAPTS	P-MG	f_τ	α_τ	WT	SUF	$\overline{C}_D$	St
Cyl-1	RK4	Off	Off	-	-	13:22:23	1.00	1.339	0.166
Cyl-2	RK4	Off	On	-	2.0	02:07:57	6.27	1.339	0.166
Cyl-3	RK3(2)4[2R+]	On	Off	8.0	-	03:12:53	4.16	1.339	0.166
Cyl-4	RK3(2)4[2R+]	On	On	8.0	1.6	00:52:40	15.24	1.339	0.166

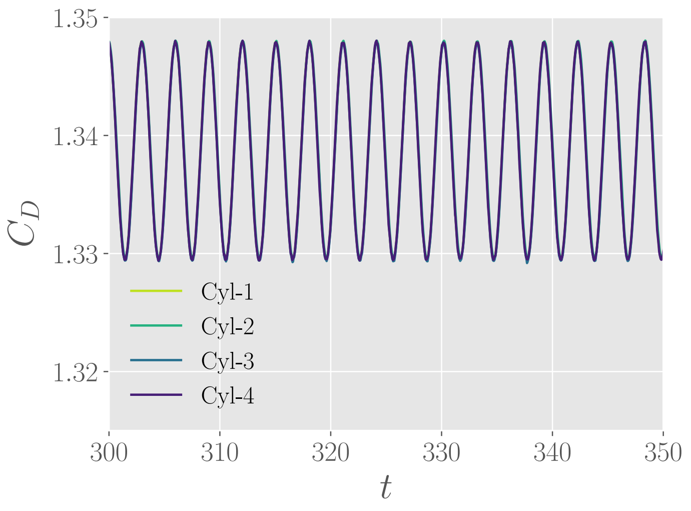
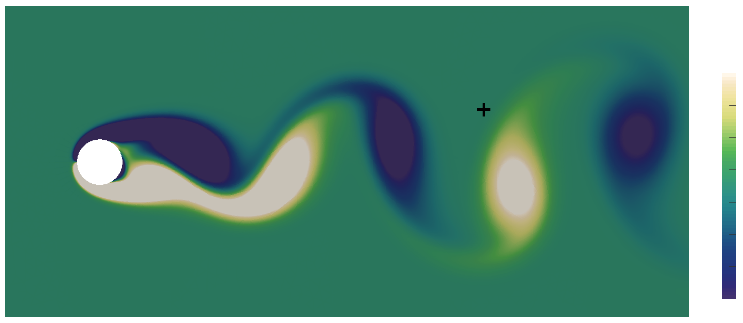


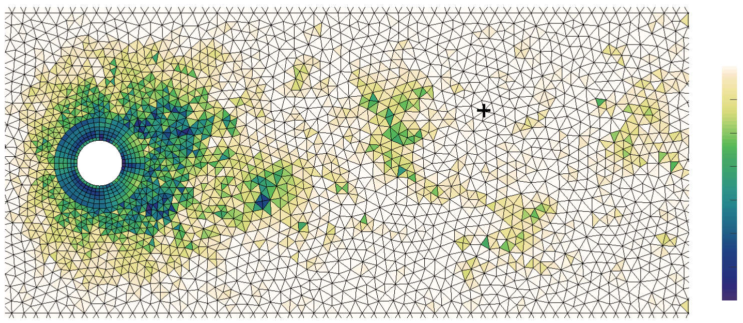
Figure 6-1: Temporal evolution of the drag coefficient for the 2D cylinder case between $t = 300$ and $t = 350$. The graphs of Cyl-2 - Cyl-4 have been shifted in t such that they are in phase with Cyl-1.

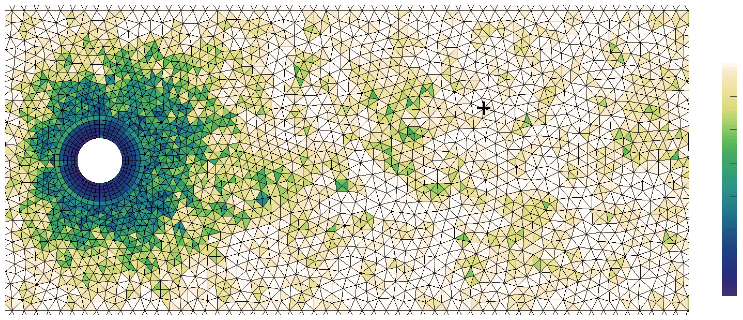
6.5.3 Adaptivity Analysis

To better understand the underlying adaptation mechanisms, temporal and spatial behaviour of local pseudo-time step sizes were studied with Cyl-4. Figures 6-2a-d show the instantaneous z-vorticity field ω_z and fields of local pseudo-time step sizes $\Delta\tau_{en\alpha}$, with $\alpha \in \{p,\ v_x, v_y\}$, at $t = 340.35$. The pseudo-time step sizes were taken from the last pseudo-time step after the convergence criterion had been reached, and averaged over solution points. Two observations can be made. Firstly, the approach is able to adapt to the element size and shape. This is most evident in the structured boundary layer block and its immediate vicinity, where the pseudo-time step sizes grow monotonically with increasing element size. Secondly, the approach is able to vary between different field variables and adapt to their local values. The variation between field variables can be seen in the boundary layer where $\Delta\tau_{enp}$ exhibits considerably larger values than $\Delta\tau_{env_x}$ and $\Delta\tau_{env_y}$. Adaptation to the local values is most visible for $\Delta\tau_{enp}$, where areas of reduced pseudo-time step sizes resemble the von Karman vortices in the wake section. Figure 6-3 shows the temporal evolution of the pseudo-time step sizes $\Delta\tau_\alpha$ for a single solution point of level $l = P$ at $\mathcal{P} = (8.474,\ 1.155)$, which is in a triangular element located near the center of the upper vortex street. The location of the probe is illustrated as a cross in Figures 6-2a-d. It can be seen that all pseudo-time step components experience a periodic reduction with a frequency that is half of the shedding frequency. This demonstrates that all pseudo-time step components at the probe location reduce and rise back up as a vortex passes.

Figures 6-4a-c show a close-up of the pseudo-time step sizes associated with solution points $\Delta\tau^{u}_{ejn\alpha}$ at level $l = P$ near the cylinder trailing edge at $t = 340.35$. It can be seen that the solution points closest to the element corners are the most restrictive, closely followed by the other solution points near the interfaces. The interior solution points are considerably less restrictive than the interface points. This observation supports our assertion that locally pseudo-time stepping approaches based on a reference length would be difficult to implement for nodal high-order schemes.


(a)


(b)


(c)

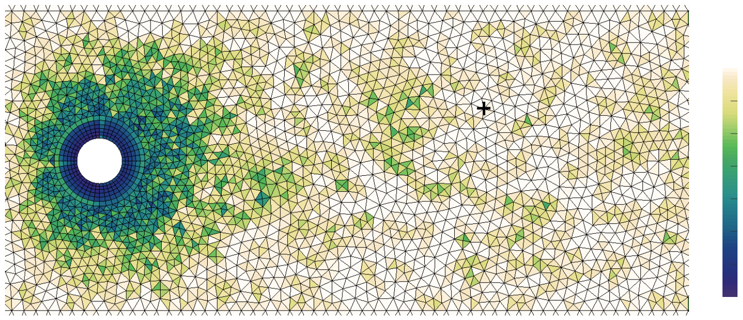

(d)

Figure 6-2: (a) Instantaneous vorticity field ω_z at $t = 340.35$ for Cyl-4. Fields of pseudo-time step sizes (b) $\Delta\tau_{enp}$, (c) $\Delta\tau_{env_x}$ and (d) $\Delta\tau_{env_y}$ at $t = 340.35$ for Cyl-4. The cross is the location of the probe $\mathcal{P}$.

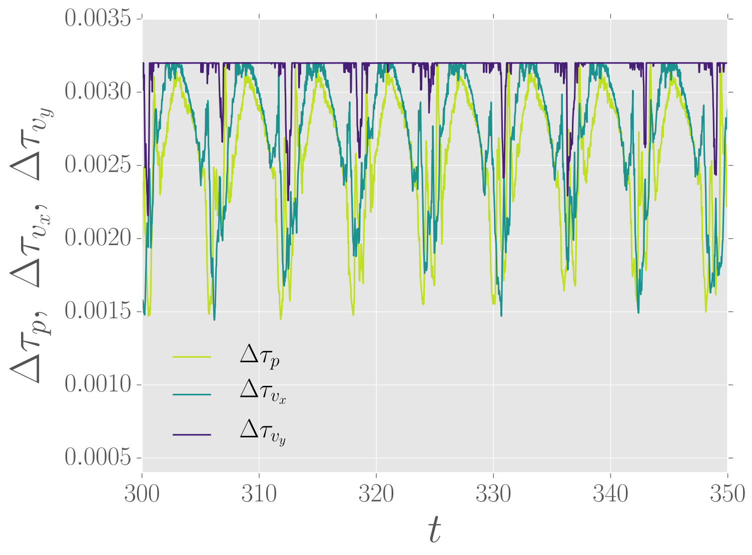
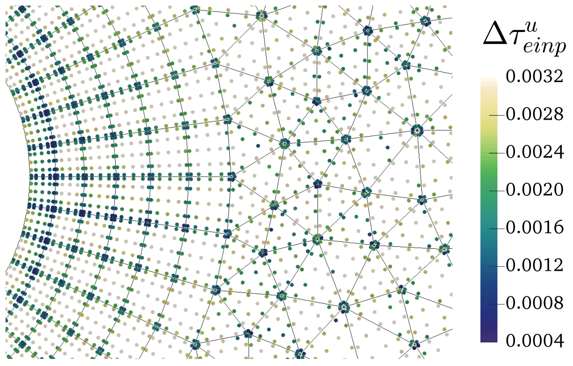
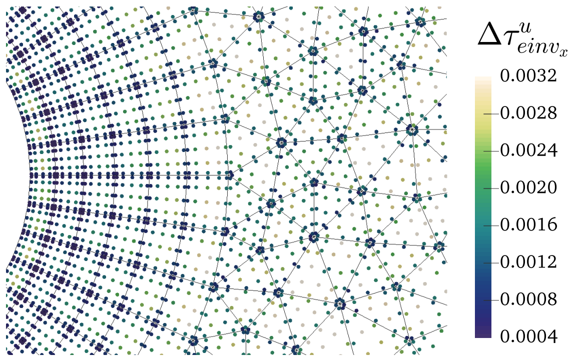


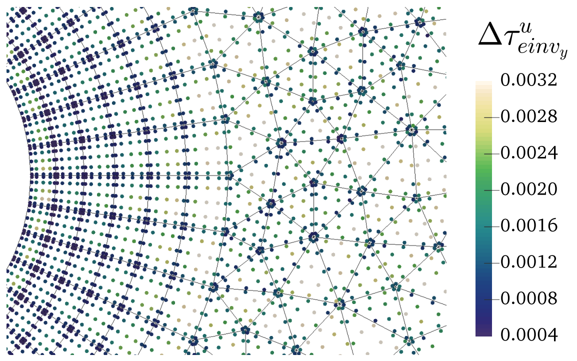
Figure 6-3: Temporal evolution of $\Delta\tau_\alpha$ at the probe location $\mathcal{P}$, which is in a triangular element.



(a)



(b)



(c)

Figure 6-4: Fields of pseudo-time step sizes (a) $\Delta\tau^u_{einp}$, (b) $\Delta\tau^u_{einv_x}$ and (c) $\Delta\tau^u_{einv_x}$ at individual solution points of $l = P$ near the cylinder trailing edge at $t = 340.35$ for Cyl-4.

6.6 SD7003 Airfoil at $Re = 60,000$

6.6.1 Problem Specification

An SD7003 airfoil simulation at $Re = 60,000$ based on the chord length and free-stream velocity was performed to assess locally adaptive pseudo-time stepping in a turbulent case. Figure 6-5 shows the unstructured computational grid used in the study. With the chord length being 1, the domain consists of an inflow section with a diameter of 20, and a rectangular wake section with a length of 20. The domain width in the span-wise direction was selected as 0.2. The total element count was 137,916, consisting of only hexahedra. The Riemann-invariant-based boundary condition was prescribed at all far-field boundaries with $p_\infty = 1$, $\mathbf{v}_\infty = \{1\ 0\ 0\}^T$ and $\zeta_\infty = 100$. This boundary condition was found to be substantially less reflective than conventional BCs in [58, 59]. A no-slip condition was prescribed on the airfoil surface. The initial condition at $t = 0$ was set as $u_\alpha = \{1\ 1\ 0\ 0\}^T$.

One complete simulation with $P = 4$ and $\zeta = 3$ was run on 32 Nvidia P100 GPUs using double precision. The Gauss-Legendre point distribution was used for the solution points and the interfaces. No subgrid-scale turbulence model or spatial filtering was applied, and the simulation can be considered as implicit LES. Physical time was discretised with BDF2, where $\Delta t = 1 \times 10^{-3}$. Locally adaptive pseudo-time stepping with P-multigrid convergence acceleration was used in pseudo-time with $f_{\max} = 1.001$, $f_{\min} = 0.998$, $\kappa = 10^{-6}$, $\Delta\tau_{min} = 3.6 \times 10^{-5}$, and $f_\tau = 7.0$. The P-multigrid cycle was prescribed as 1-1-1-1-4-1-1-1-6, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4, with $\alpha_\tau = 1.7$. Additionally, volume flux anti-aliasing with quadrature degrees 11-9-7-5-3-5-7-9-11 was performed at P-multigrid levels.

The simulation was run in three stages. First, the flow was developed with $P = 1$ up to $t = 25.011$. Second, the simulation was restarted and run up to $t = 45.024$ with $P = 4$ to settle transients. Third, flow statistics were measured between $t = 45.024$ and $t = 70.043$. The case was run with a convergence tolerance such that the point-wise L^2-norm of the largest velocity residual was always driven below 10^{-4}. The L^2-

norm of the velocity divergence, which is directly proportional to pressure residual, was approximately 10^{-3}.

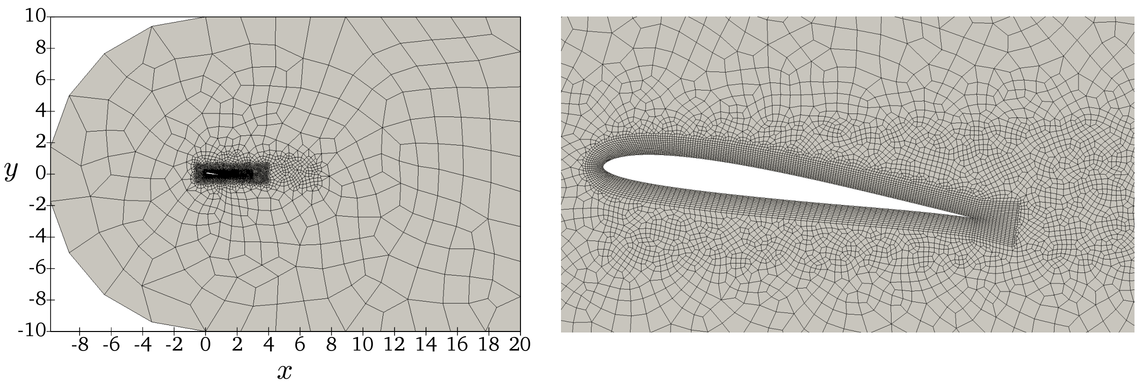


Figure 6-5: The computational grid used in the 3D SD7003 airfoil simulations (left). A zoom close to the airfoil body (right).

6.6.2 Results

Figure 6-6 plots Q-criterion iso-surfaces coloured by instantaneous velocity V at $t = 59.5460$. The simulation captures the characteristic features described in [97]; a laminar separation bubble is formed at the leading edge which breaks up into turbulence via Kelvin-Helmholtz vortices. Qualitatively the turbulent structures are well-resolved throughout the domain.

Table 6.2 shows a comparison of the simulation data with compressible low-Mach data from Beck et al. [97] using DG and Vermeire et al. [1] using FR. In addition, low-speed experimental data from Selig et al. [98] are provided. The separation and reattachement points were measured from the time and span-averaged velocity in the chord-wise direction $\overline{V}_c = v_x \cos(8°) + v_y \sin(8°)$, which is visualised in Figure 6-7. It can be seen that all numerical results over-estimate the drag coefficient $\overline{C}_D$ compared to the experiments. The biggest differences across the numerical results are observed in the lift coefficients $\overline{C}_L$ and reattachement points x_{rea}. The current simulation results are within the bounds of those reported by [97, 1] except for x_{rea} which was found to be higher, but still close to the high-resolution $P = 8$ results of Beck et al.

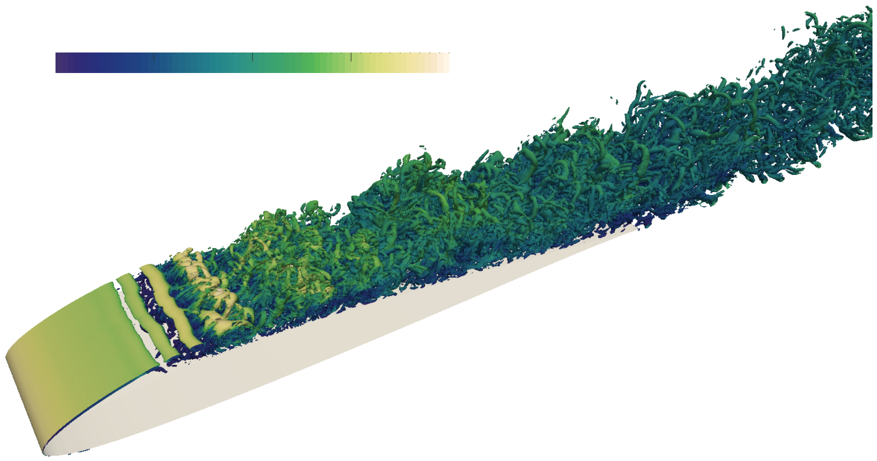


Figure 6-6: Q-criterion iso-surfaces coloured by instantaneous velocity V at t=59.546.

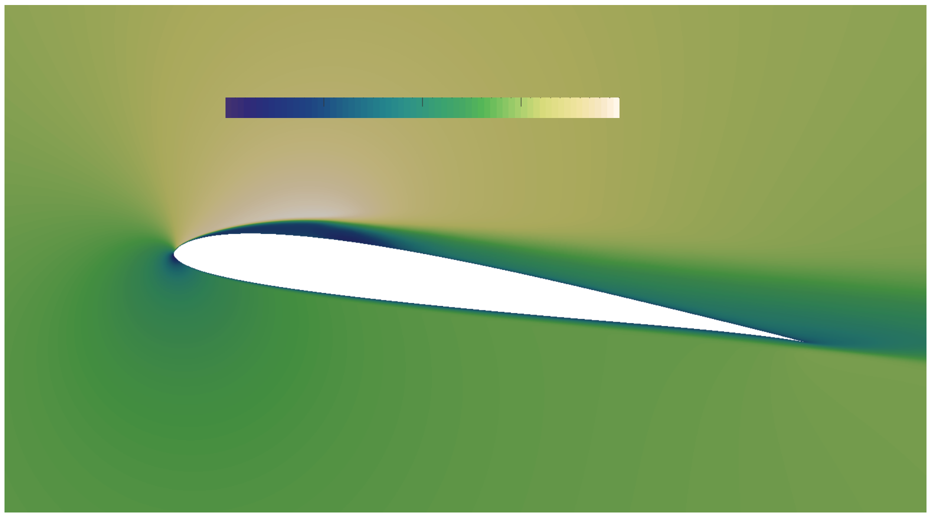


Figure 6-7: Time and span-averaged velocity field in the chord-wise direction $\overline{V_c}$.

Table 6.2: The current simulation data and reference data for the SD7003 case at Re=60,000. INS abbreviates Incompressible Navier-Stokes and NS the compressible Navier-Stokes equations.

	Formulation	Method	$\overline{C}_D$	$\overline{C}_L$	x_{sep}	x_{rea}
Current	AC INS	FR $P = 4$	0.052	0.923	0.033	0.346
Beck et al. [97]	$M = 0.1$ NS	DG $P = 8$	0.050	0.932	0.030	0.336
Beck et al. [97]	$M = 0.1$ NS	DG $P = 3$	0.045	0.923	0.027	0.310
Vermeire et al. [1]	$M = 0.2$ NS	FR $P = 4$	0.059	0.941	0.045	0.315
Selig et al. [98]	-	Exp.	0.029	0.920	-	-

In addition to the complete simulation, four additional runs starting from $t = 45.532$ were undertaken for a single flow pass over the chord. Specifically, case SD-1 was undertaken without any convergence acceleration using global RK4 pseudo-time stepping with a constant $\Delta\tau = 3.7 \times 10^{-5}$ that was optimised with a bisection approach, case SD-2 was performed with P-multigrid acceleration alone, case SD-3 with locally adaptive RK3(2)4[2R+] pseudo-time stepping alone and case SD-4 with a combination of the two. All cases utilised the same run-time parameters as the full run and converged such that the point-wise L^2-norm of the largest velocity residual was driven below 10^{-4}. Table 6.3 shows the runtime parameters, wall-times, and speed-up factors for each case. It can be seen that locally adaptive pseudo-time stepping alone leads to a speed-up factor of 2.46 compared to RK4 pseudo-time stepping. Combining locally adaptive pseudo-time stepping with P-multigrid yields a total speed-up factor of 11.65 compared to RK4 pseudo-time stepping.

Table 6.3: Summary of the SD7003 performance runs, where PS denotes the pseudo-stepper, LAPTS denotes locally adaptive pseudo-time stepping, P-MG denotes P-multigrid, WT denotes the wall-time, and SUF denotes the speed-up factor relative to SD-1.

Case	PS	LAPTS	P-MG	f_τ	α_τ	WT	SUF
SD-1	RK4	Off	Off	-	-	21:43:08	1.00
SD-2	RK4	Off	On	-	1.7	04:28:58	4.84
SD-3	RK3(2)4[2R+]	On	Off	7	-	08:50:47	2.46
SD-4	RK3(2)4[2R+]	On	On	7	1.7	01:51:54	11.65

Chapter 7

Optimal Runge-Kutta Schemes

7.1 Overview

To further accelerate the convergence, optimal explicit RK schemes for high-order FR discretisations were developed [99]. These schemes aim to reduce the total number of pseudo time steps required by increasing the maximum allowable pseudo-time step. The optimisation of the schemes was undertaken by Vermeire, and the implementation and validation by Loppi. Only a summary of the optimisation methodology is given in this Chapter. For details see [99].

7.2 Methodology

A stability polynomial of an explicit RK scheme can be represented as

$$P^{(s,q)}(z) = \sum_{j=0}^{s} \gamma_j z^j \ , \tag{7.1}$$

in which the coefficients γ_j must satisfy

$$\gamma_j = \frac{1}{j!}, \ \ 0 \le j \le q \ , \tag{7.2}$$

to obtain a temporal accuracy of order q. Since the temporal accuracy in pseudo-time is not required, only a first order accurate stability polynomial of the form

$$P^{(s,1)}(z) = 1 + z + \sum_{j=2}^{s} \gamma_j z^j \ , \tag{7.3}$$

is considered. The coefficients $\{\gamma_2, \gamma_3, \ldots, \gamma_s\}$ are optimised to yield the largest possible pseudo-time step $\Delta\tau_{\text{opt}}$ subject to

$$|P^{s,q}(\Delta\tau\omega^\delta)| - 1 \leq 0, \ \ \forall\omega^\delta \ , \tag{7.4}$$

where ω^δ are the eigenvalues associated with a given spatial discretisation which can be obtained via von Neumann analysis.

The von Neumann analysis for FR follows Huynh [12], Vincent et al [100], and Vermeire et al. [101, 102]. Consider a linear advection equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \ , \tag{7.5}$$

with fully upwinded fluxes. The solutions to Equation 7.5 can be represented as plane waves

$$u = e^{I(\theta x - \omega t)} \ , \tag{7.6}$$

where θ is the wave number and $I = \sqrt{-1}$. For the purpose of analysis, a uniform mesh with an element width $h = 1$ is considered. This allows FR to be cast for any element as

$$\frac{\partial \hat{\mathbf{u}}^\delta}{\partial t} = -2\mathbf{D}\hat{\mathbf{u}}^\delta - \left(\hat{f}^{CL} - 2\mathbf{l}^T\hat{\mathbf{u}}^\delta\right)\mathbf{g}_{\xi L}, \tag{7.7}$$

in which the vector notations

$$\hat{\mathbf{u}}^\delta[i] = u_i(-1) \ , \tag{7.8}$$

$$\mathbf{D}[i,j] = \frac{\mathrm{d}l_j}{\mathrm{d}\hat{x}}(\hat{x}_i) \ , \tag{7.9}$$

$$\mathbf{g}_\xi[i] = \frac{\mathrm{d}g_l}{\mathrm{d}\hat{x}}(\hat{x}_i) \ , \tag{7.10}$$

with $\hat{f}^{CL}$ being the upwind interface flux and g_l the left correction function, have been adopted from [99, 100].

For Equation 7.7, Bloch plane wave solutions of the form

$$\hat{\mathbf{u}}^\delta = e^{I\left(n\overline{\theta}^\delta - \omega^\delta t\right)}\hat{\mathbf{v}}^\delta \tag{7.11}$$

are sought, where $\overline{\theta}^\delta$ is a given baseline wavenumber $-\pi \leq \overline{\theta} \leq \pi$ and $\hat{\mathbf{v}}^\delta$ is a vector. Furthermore, the upwind interface flux can be expressed as

$$\hat{f}^{CL} = 2\mathbf{r}^T e^{I\left(n\overline{\theta}^\delta - \overline{\theta}^\delta - \omega^\delta t\right)}\hat{\mathbf{v}}^\delta \ . \tag{7.12}$$

Substituting Equations 7.11 and 7.12 into Equation 7.7 yields an eigenvalue problem

$$\mathbf{Q}\hat{\mathbf{v}}^\delta = \omega^\delta \hat{\mathbf{v}}^\delta \ , \tag{7.13}$$

where

$$\mathbf{Q} = -2I\left[\mathbf{D} + \mathbf{g}_{\xi L}\left(\mathbf{r}^T e^{-I\overline{\theta}^\delta} - \mathbf{l}^T\right)\right] \ , \tag{7.14}$$

ω^δ are the eigenvalues and $\hat{\mathbf{v}}^\delta$ are the right eigenvectors of $\mathbf{Q}$ that can be computed numerically for a given $\overline{\theta}^\delta$. Using these eigenvalues, the maximum allowable pseudo-time step of a given stability polynomial $P^{(s,1)}(z)$ can be computed according to Equation 7.4. Thus, it is possible to optimise $P^{(s,1)}(z)$ by varying its γ_i coefficients such that it yields the largest possible pseudo-time step size. After finding optimal γ_i, a Butcher tableau can be determined. For more details on finding the optimal coefficients and generating the associated Butcher tableau, see [99, 80].

7.3 Optimal schemes for FR-DG

7.3.1 Normal Schemes

By setting the $\mathbf{g}$ to be the FR-DG correction function in the von Neumann analysis, and following the optimisation procedure in [99], the optimal schemes for DG were obtained. The optimisation was undertaken by Vermeire for a range of stage-counts s and solution polynomial orders P. It was observed that for a given polynomial degree, optimal RK schemes with more stages have a larger stability limit, whereas for a given number of stages, optimal RK schemes for higher polynomial degrees have a smaller stability limit.

Figure 7-1 plots normalised maximum pseudo-time steps for FR-DG $P = 4$ optimal schemes with stage-counts between 2 and 7. Figure 7-2 plots a sampling of the eigenspectra for the FR-DG $P = 4$ scheme scaled by the maximum stable pseudo-time step size as $\boldsymbol{\omega}^{\delta}\Delta\tau_{opt}$ alongside contours of $|P^{(s,q)}(z)|$. It was observed that schemes with a low number of stages were unable to use large portions of the region of absolute stability. However as the number of stages, and hence degrees of freedom available for optimisation, was increased, the scaled eigenspectra and boundary of the region of absolute stability began to overlap. This observation suggests that, for an RK scheme with enough stages, the optimiser described in [99] is able to generate a scheme whose stability boundary mimics the shape of the eigenspectra of the spatial discretisation. Based on the stability analysis, the optimised schemes allow maximum stable pseudo-time step sizes (normalised with the stage count) that are nearly a factor of two larger than those for the classical RK4 scheme. The speed-up factors observed in numerical experiments in [99] were found to be in line with these theoretical results. Table 7.1 shows the Butcher tableau for the FR-DG $P = 4$ optimal seven-stage Opt-RK$_{7,1}$ scheme.

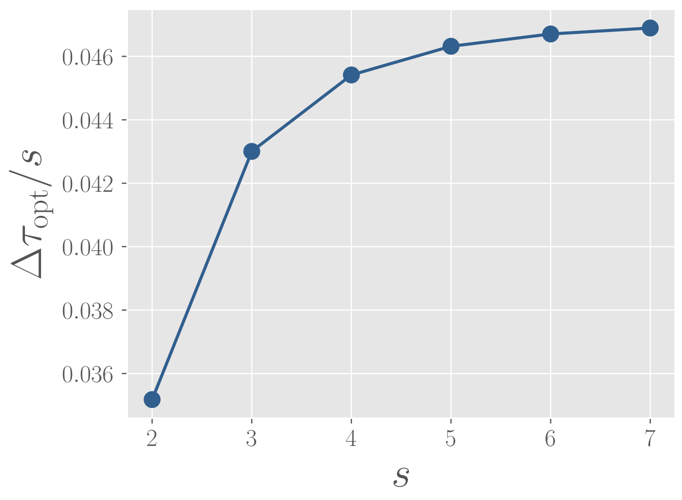


Figure 7-1: Normalised optimal pseudo-time step sizes for FR-DG $P = 4$ optimised schemes with different stage counts.

7.3.2 Embedded Pair Schemes

Building on the work in [99], the same optimisation framework has been used to find optimal embedded pair RK schemes for FR-DG by Vermeire, Loppi and Vincent. The optimal embedded pair RK schemes were found by restricting both schemes of the pair to first order temporal accuracy and optimising such that the stability region of the scheme that takes the step is larger than the scheme that is only used for the truncation error approximation. Table 7.1 shows the Butcher tableau for the FR-DG $P = 4$ optimal ten-stage Opt-RK$_{10,1,1}$ scheme.

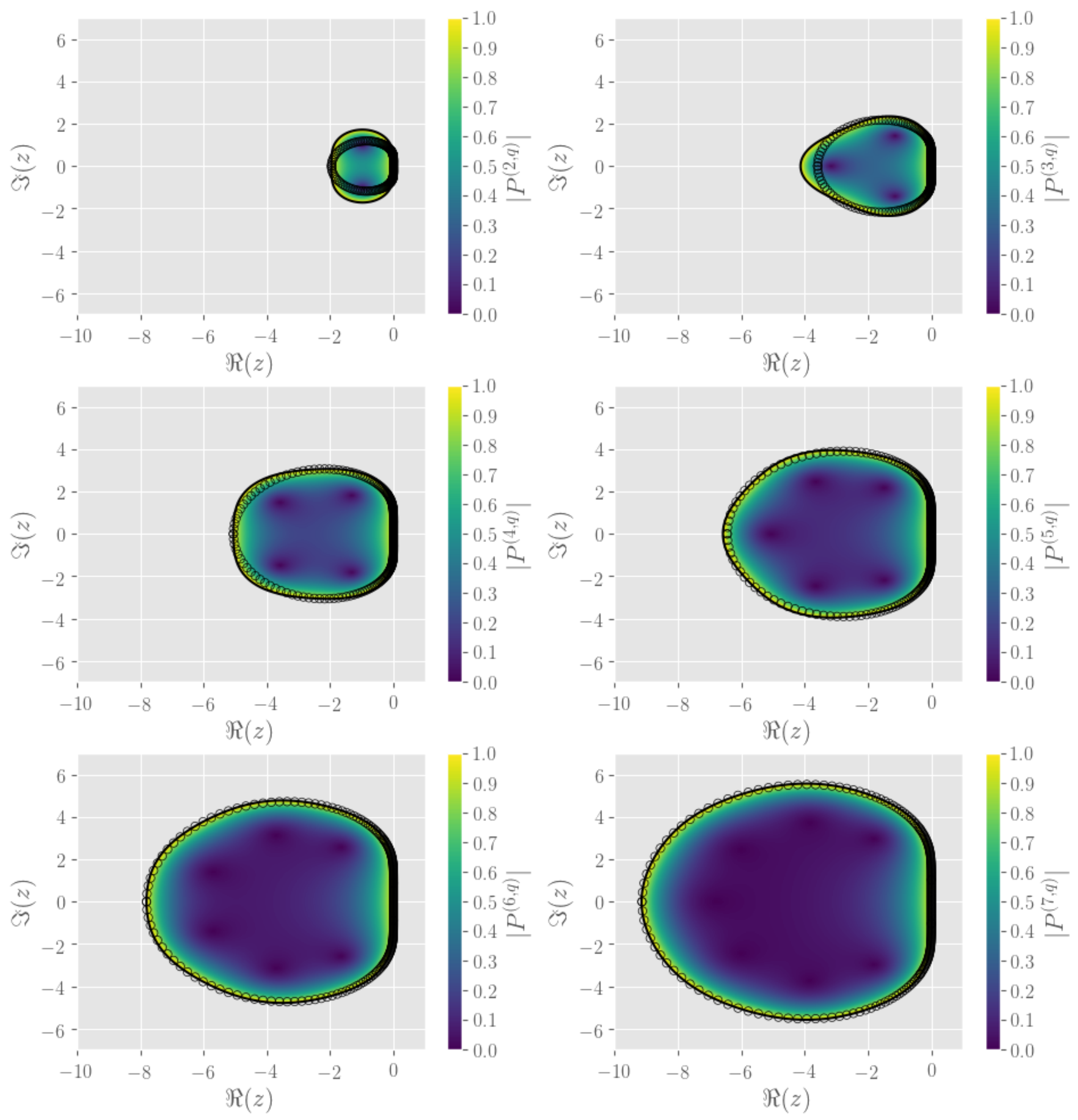


Figure 7-2: Stability polynomial contours for FR-DG $P = 4$ optimised schemes with different stage counts together with $\boldsymbol{\omega}^{\delta}\Delta\tau_{opt}$ obtained via von Neumann analysis (circles).

Table 7.1: Butcher tableau of the FR-DG $P = 4$ optimal seven-stage Opt-RK$_{7,1}$ scheme.

0							
c_2	0.2324511858361562						
c_3	0.2322231685703214	0.2322231685703214					
c_4	0.2322237588440728	0.231622006897831,	0.2318494338899144				
c_5	0.2322351030333376	0.2316575500243345	0.2202846506823094	0.2208563868951234			
c_6	0.2322346239117762	0.2316802573753524	0.2207730890236098	0.2119898751137023	0.2231191886646174		
c_7	0.214206616344267	0.1979168835910834	0.1933813914977442	0.1536900816632531	0.0945522683179592	0.09850693266479	
b_i	0.2083651435574576	0.1912434615654654	0.175992229629985	0.1160474272290253	0.0721803614104373	0.0702944469173733	0.1658769296902561

Table 7.2: Butcher tableau of the FR-DG $P = 4$ optimal ten-stage Opt-RK$_{10,1,1}$ scheme.

0										
c_2	0.1111111111									
c_3	0.1900199097	0.0322023124								
c_4	0.2810938259	0	0.0522395073							
c_5	0.3683599872	0	0	0.0760844571						
c_6	0.4503724121	0	0	0	0.1051831433					
c_7	0.5247721825	0	0	0	0	0.1418944841				
c_8	0.5874505094	0	0	0	0	0	0.1903272683			
c_9	0.6304783975	0	0	0	0	0	0	0.2584104913		
c_{10}	0.6358199324	0	0	0	0	0	0	0	0.3641800675	
b_i	0.4988192238	0	0	0	0	0	0	0	0	0.5011807761
b_i	0.3906281980	0.1179848341	1.7353065354	−7.9567462555	17.3753753701	−23.4057667136	20.5007152462	−11.4042315893	3.6467343745	0

7.4 2D cylinder at Re=100

7.4.1 Problem Specification

The performance of the optimal RK schemes was first studied with the 2D cylinder at $Re = 100$ using the same setup as in Sections 5.4 and 6.5. Four simulations were performed: Case Cyl-5 using optimal seven-stage Opt-RK$_{7,1}$ pseudo-time stepping alone, case Cyl-6 using locally adaptive pseudo-time stepping with the optimal ten-stage embedded pair Opt-RK$_{10,1,1}$ scheme, case Cyl-7 using P-multigrid with seven-stage Opt-RK$_{7,1}$ smoothing, and case Cyl-8 using P-multigrid and locally adaptive pseudo-time stepping with Opt-RK$_{10,1,1}$ smoothing.

A fixed pseudo-time step $\Delta\tau = 1.4 \times 10^{-3}$ was used for Cyl-5 and Cyl-7 which was found via bisection approach that resulted in best performance. The locally adaptive pseudo-time stepping parameters were set as $f_{\max} = 1.01$, $f_{\min} = 0.98$, $\kappa = 10^{-6}$, $\Delta\tau_{min} = 1.9 \times 10^{-3}$, and $f_\tau = 8.0$ for Cyl-6. The same values were used in Cyl-8 apart from f_τ which was prescribed as $f_\tau = 4.0$. The P-multigrid convergence acceleration was performed using a 5-level cycle 1-1-1-1-2-1-1-1-3, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4, with $\alpha_\tau = 2.0$ in Cyl-7 and with $\alpha_\tau = 1.6$ in Cyl-8.

7.4.2 Results

Table 7.3 shows the run-time parameters of all cases together with mean drag coefficients $\overline{C}_D$, Strouhal numbers St, and wall-times. The results of Cyl-1 - Cyl-4 from Sections 5.4 and 6.5 are provided as reference to complete a full comparison of convergence acceleration techniques and their combinations. The statistics of all cases were measured for the same period of 100 time units between $t = 250.1250$ and $t = 350.1250$. It can be seen that all cases produce identical mean $\overline{C}_D$ and St. Furthermore, Figure 7-3 shows that the temporal evolution of the drag coefficients agree with those of Cyl-1 - Cyl-4.

From the wall-times it can be seen that Opt-RK$_{7,1}$ alone yields a speed-up factor of 1.95 compared to RK4 pseudo-time stepping. Using the locally adaptive pseudo-time stepping with Opt-RK$_{10,1,1}$ yields a speed-up factor of 8.80. Using Opt-RK$_{7,1}$ as the P-multigrid smoother yields a combined speed-up of 13.94. P-multigrid and locally adaptive pseudo-time with Opt-RK$_{10,10,1}$ yields a speed-up factor of 21.26. The results demonstrate that all convergence acceleration techniques can be combined for increased performance. However, to allow the combinations to run robustly, the user-specified parameters have to be decreased from their peak values. This behaviour may be partially explained by the fact that the pseudo-time steps at lower P levels considerably exceed the physical time step size which is known to cause instabilities [84]. Nevertheless, the total speed-up factor of over 20 relative to RK4 pseudo-time stepping is substantial, considering that only explicit convergence acceleration techniques are used.

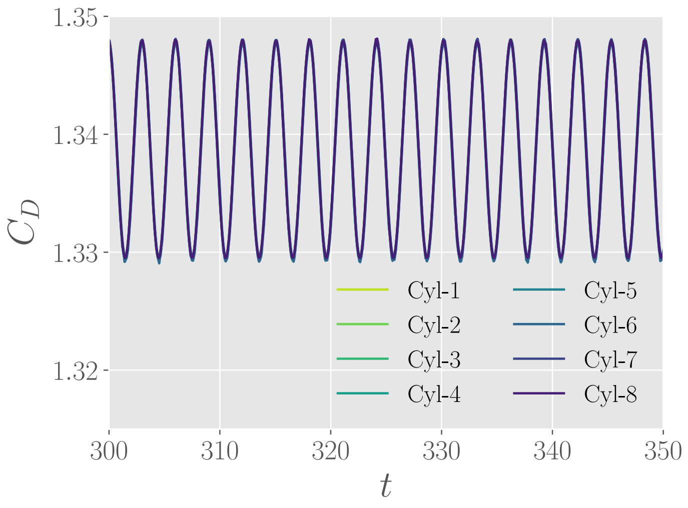


Figure 7-3: Temporal evolution of the drag coefficient for the 2D cylinder case between $t = 300$ and $t = 350$. The graphs of Cyl-2 - Cyl-8 have been shifted in t such that they are in phase with Cyl-1.

Table 7.3: Summary of the 2D cylinder cases at $Re = 100$, where PS denotes the pseudo-stepper, LAPTS denotes locally adaptive pseudo-time stepping, P-MG denotes P-multigrid, WT denotes the wall-time, and SUF denotes the speed-up factor relative to Cyl-1.

	PS	LAPTS	P-MG	f_τ	α_τ	WT	SUF	$\overline{C}_D$	St
Cyl-1	RK4	Off	Off	-	-	13:22:23	1.00	1.339	0.166
Cyl-2	RK4	Off	On	-	2.0	02:07:57	6.27	1.339	0.166
Cyl-3	RK3(2)4[2R+]	On	Off	8.0	-	03:12:53	4.16	1.339	0.166
Cyl-4	RK3(2)4[2R+]	On	On	8.0	1.6	00:52:40	15.24	1.339	0.166
Cyl-5	Opt-RK$_{7,1}$	Off	Off	-	-	06:51:24	1.95	1.339	0.166
Cyl-6	Opt-RK$_{10,1,1}$	On	Off	8.0	-	01:31:12	8.80	1.339	0.166
Cyl-7	Opt-RK$_{7,1}$	Off	On	-	2.0	00:57:32	13.94	1.339	0.166
Cyl-8	Opt-RK$_{10,1,1}$	On	On	4.0	1.6	00:37:44	21.26	1.339	0.166

7.5 Turbulent Jet at $Re = 10,000$

7.5.1 Problem Specification

To validate the optimal RK schemes for a turbulent flow problem, an incompressible round jet at $Re = 10,000$ based on the jet diameter and midline velocity of the inflow was studied. The test case was chosen since experimental data is available for comparison [103], and it has relevance to many industrial application areas and natural flow phenomena, such as hydrojet propulsion, cooling systems, and seafloor plumes. Influential experiments and a general theory of incompressible turbulent round jets are discussed in the review of Lipari and Standsby [104]. Round jets at various Reynolds numbers have also been studied numerically, closest to our setup being DNS by Boersma [105] at $Re = 5,000$ and explicitly filtered LES by Bogey and Bailly [106] at $Re = 11,000$. However, both of these studies have been performed with compressible codes at higher Mach numbers, 0.6 and 0.9, respectively.

Figure 7-4 shows the computational grid in the yz-plane at $x = 0$ together with a schematic of the simulation setup in the xy-plane at $z = 0$. The diameter of the jet was 0.5. The origin was located at the center of the jet as it enters the domain. A two dimensional unstructured circular grid of diameter 24 was extruded in x for a distance of 50 in 250 equally sized steps. The resulting 3D mesh contained 247,250 hexahedral elements in the center of the domain $0 < r < 2.5$, where $r = \sqrt{y^2 + z^2}$, and 596,500 prismatic elements elsewhere. The virtual origin is located at (x_0,0,0), which is the starting point of the self-similar region associated with a linear velocity decay and spreading rate [104].

The jet inflow profile with a peak velocity of 1.0 was imposed as

$$V_{\text{jet}}(r) = 0.5 - 0.5\tanh\left[20\left(r - 0.25\right)\right] \ . \tag{7.15}$$

A no-slip boundary condition was imposed at the vertical front interface outside the jet inflow zone and a pressure outlet boundary condition with $p_\infty = 10$ was used for the outlet. The lateral far-field wall was specified as a non-entraining slip wall boundary which leads to a weak backflow at large r, as fluid is being drawn from the vicinity of the edges. A sponge layer was found to be necessary to dissipate the jet before it impinged on the outlet. Specifically, it was imposed via a spatially dependent source term S defined as

$$S = \left(u_\alpha - u_\alpha^{\text{out}}\right)\left[0.5 + 0.5\tanh\left(0.5\left(x - 45\right)\right)\right] \ , \tag{7.16}$$

where $u_\alpha^{\text{out}} = \{10 \ \ 0 \ \ 0 \ \ 0\}^T$.

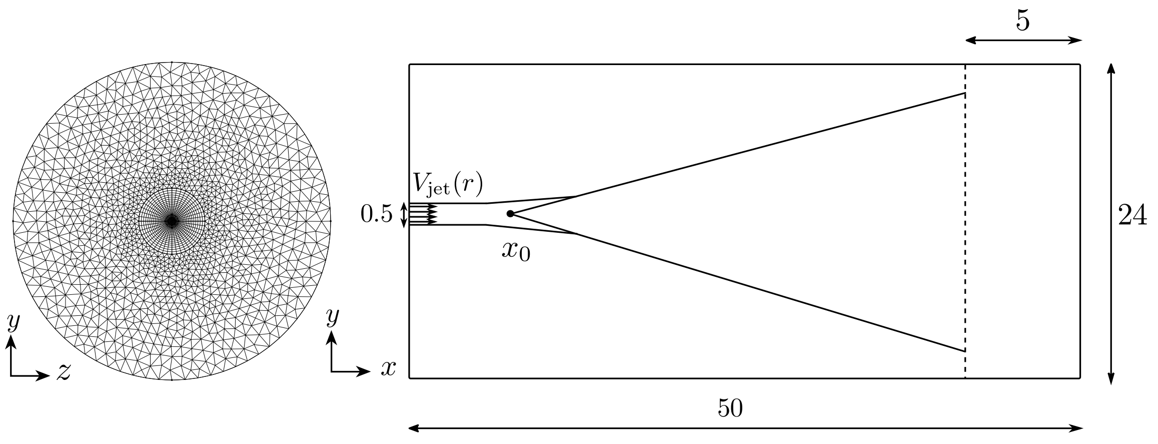


Figure 7-4: Computational grid in the yz-plane at $x = 0$ together with a schematic of the simulation setup in the xy-plane at $z = 0$. The virtual origin is located at (x_0,0,0).

A single simulation was performed with a solution polynomial order $P = 4$, and

$\zeta = 2.5$. The Gauss-Legendre solution point distribution was used for hexahedral elements and the Gauss-Legendre-Williams-Shunn solution point distribution was used for prismatic elements. The Gauss-Legendre flux point distribution was used for quadrilateral interfaces and the Williams-Shunn flux point distribution was used for triangular interfaces. All flux evaluations were performed without anti-aliasing. No subgrid-scale turbulence model or spatial filtering was applied, and the simulation can be considered as implicit LES. The BDF2 scheme was used for physical time and the optimal $RK_{7,1}$ scheme for $P = 4$ was used in pseudo-time. Constant time steps of $\Delta t = 0.004$ and $\Delta\tau = 0.0027$ were used throughout the simulation.

The jet was initially developed up to $t = 520$ to damp initial transients using 15 pseudo-iterations within each physical time step. The simulation was then restarted with a convergence criterion of 5×10^{-5} for the velocity residuals and statistics were collected up to $t = 1220$. The real time steps converged within 18 to 24 pseudo-iterations, resulting in the L^2-norm of divergence $\nabla \cdot \mathbf{v} = \frac{1}{\zeta}\frac{\partial p}{\partial \tau}$ being approximately 6×10^{-4}.

7.5.2 Results

Figure 7-5 shows a volume rendering of the instantaneous velocity field to visualise the shape of the jet. The experimental study by Panchapakesan and Lumley [103] at $M \approx 0.01 - 0.02$ and $Re = 11,000$ is used as a reference for all flow statistics. To find the location of the virtual origin, the time-averaged midline axial velocity decay was shifted in x to fit a linear constant decay rate through the origin. The midline velocity decay shifted by $x_0 = 1.6$ is shown in Figure 7-6a together with the experimental rate observed by Panchapakesan and Lumley [103]. It can be seen that the predicted linear velocity decay region matches with the reference before the sponge starts gradually dissipating the velocity. Figure 7-6b shows the average axial velocity with respect to the self-similarity coordinate η. In addition to averaging in time, the results were spatially averaged along conical surfaces defined by $\eta(r, x) = \frac{r}{x - x_0}$ in the self-similar region $12 \leq x \leq 30$ and normalised by the mean midline velocity v_c. From Figure 7-6b, it can be seen that the mean axial velocities agree with the reference data across the entire self-similarity region. Figures 7-7a and 7-7b show the mean axial and radial velocity fluctuations in the self similar region. Both graphs indicate that the simulation is able to accurately capture velocity fluctuations in agreement with the experimental data. These results demonstrate that the optimal RK schemes combined with a BDF scheme are suitable for simulating turbulent flows, and that this approach can correctly predict mean flow and turbulent quantities.

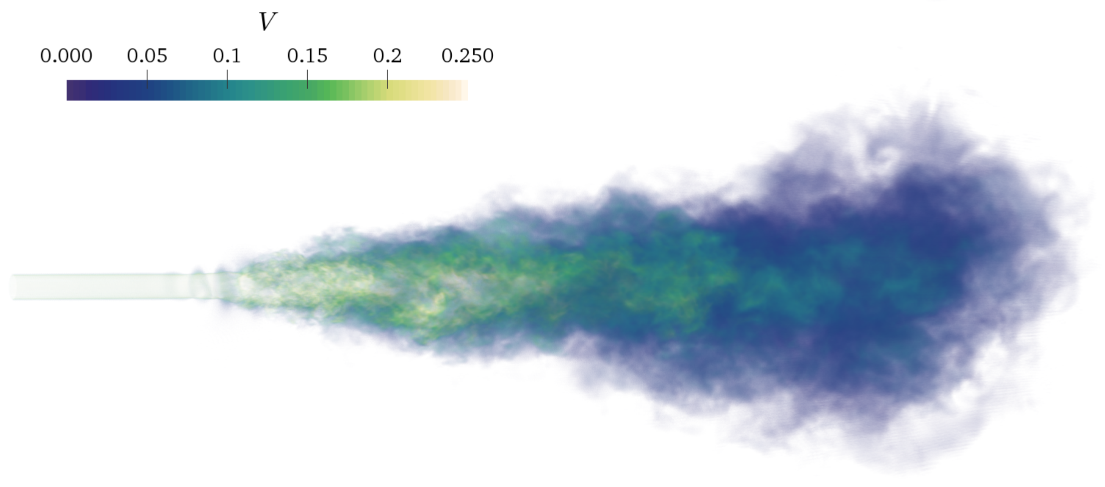
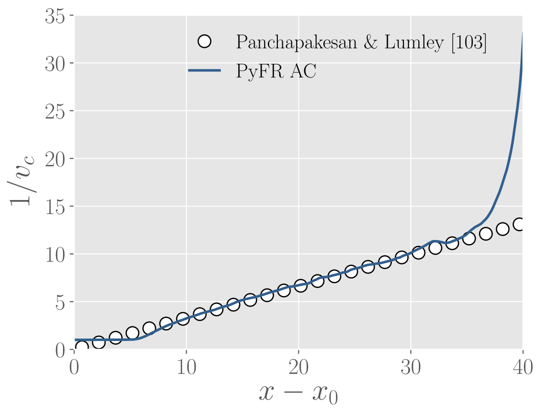
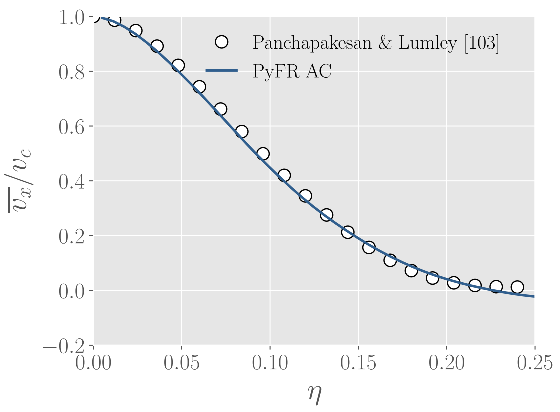


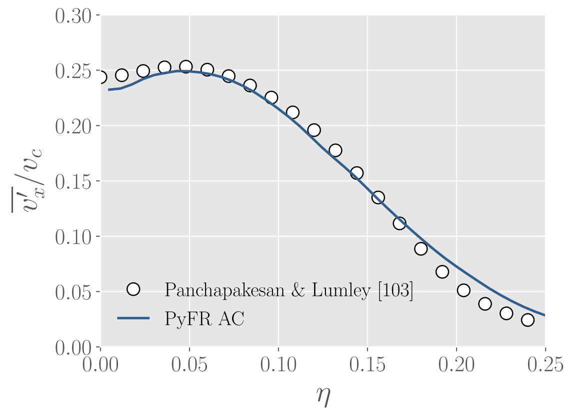
Figure 7-5: Volume rendering of the instantaneous volumetric velocity field at $t = 1220$.

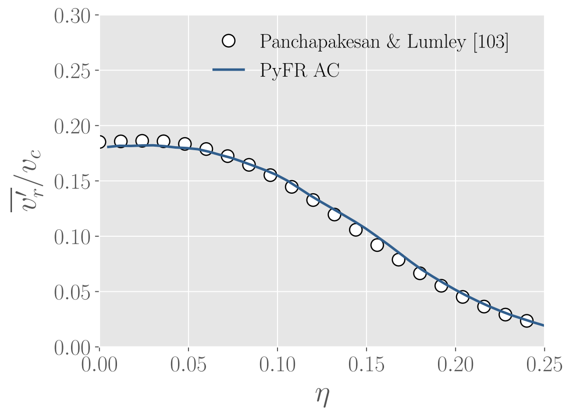




(a) (b)

Figure 7-6: (a) The mean axial mid-line velocity decay rate. (b) The mean axial velocity along the self-similarity coordinate.





(a) (b)

Figure 7-7: (a) Self-similar mean axial root-mean-square velocity fluctuations. (b) Self-similar mean radial root-mean-square velocity fluctuations.

Chapter 8

DARPA SUBOFF

8.1 Problem specification

In this Chapter, P-multigrid, locally adaptive pseudo-time stepping and optimal RK schemes are used together to simulate flow over an idealised version of the DARPA SUBOFF submarine model [107] at $Re = 1.2 \times 10^6$ based on the length of the hull and free-stream velocity. The idealised geometry in this study comprise of the axisymmetric hull, sail and stern appendages. The DARPA SUBOFF case has been previously studied experimentally by Jimenez et al. without stern appendages [108] at $1.1 \times 10^6 \leq Re \leq 67 \times 10^6$ and with stern appendages [109] at $4.9 \times 10^5 \leq Re \leq 1.8 \times 10^6$. Numerically Kumar and Mahesh [110] studied only the hull at $Re = 1.1 \times 10^6$ using LES with a FV discretisation. The version including the sail and appendages has been studied by Posa and Balaras [111] at $Re = 1.2 \times 10^6$ using LES with a FD immersed boundary method, and by Bhushan et al. [112] at $Re = 1.2 \times 10^7$ using hybrid RANS/LES techniques with a FV method.

8.1.1 Meshing

The mesh for the DARPA SUBOFF configuration was generated using Pointwise V18.2R2. Figure 8-1 shows an xz-slice of the mesh at $y = 0$, where the submarine diameter $D = 0.508$ and length $L = 4.356$. The mesh topology consists of structured boundary layer blocks in the vicinity of the model surface and unstructured blocks in the far-field. The total element count is 4,984,353, comprising of 4,772,275 hexahedral elements and 212,260 prismatic elements. The boundary layer blocks were generated such that the first $P = 4$ Gauss-Legendre solution point is at approximately $z^+ = u^\tau z/\mu = 2.5$, with $u^\tau = \sqrt{\tau_w}$ being the wall-shear stress reported in [111].

The mesh was generated by first creating a so-called butterfly topology on the bow (nose) and the tip of the stern (tail), which are illustrated in Figures 8-2a and b. Subsequently, these were extruded as structured blocks of hexahedra all the way to the inlet and outlet to avoid a singularity at the mid-line. Furthermore, unstructured surface meshes of triangles were generated for the tips of the stern appendages and the sail as illustrated in Figures8-2c and d. These were extruded as prism blocks to the far-field. After such extrusions, 2D meshes of quadrilaterals and triangles were generated between the structured blocks at $y = 0$ and revolved 360° over all areas that are axisymmetric. Finally, structured blocks of hexahedra were generated between the stern appendages and around the sail.

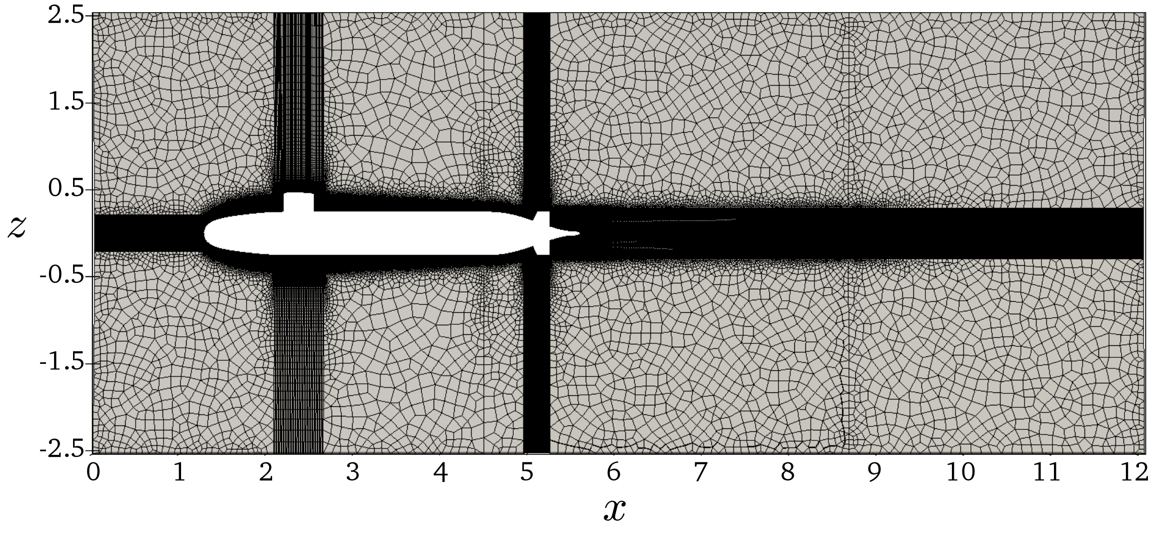


Figure 8-1: Computational grid for the idealised DARPA SUBOFF model sliced on the xz-plane at $y = 0$.

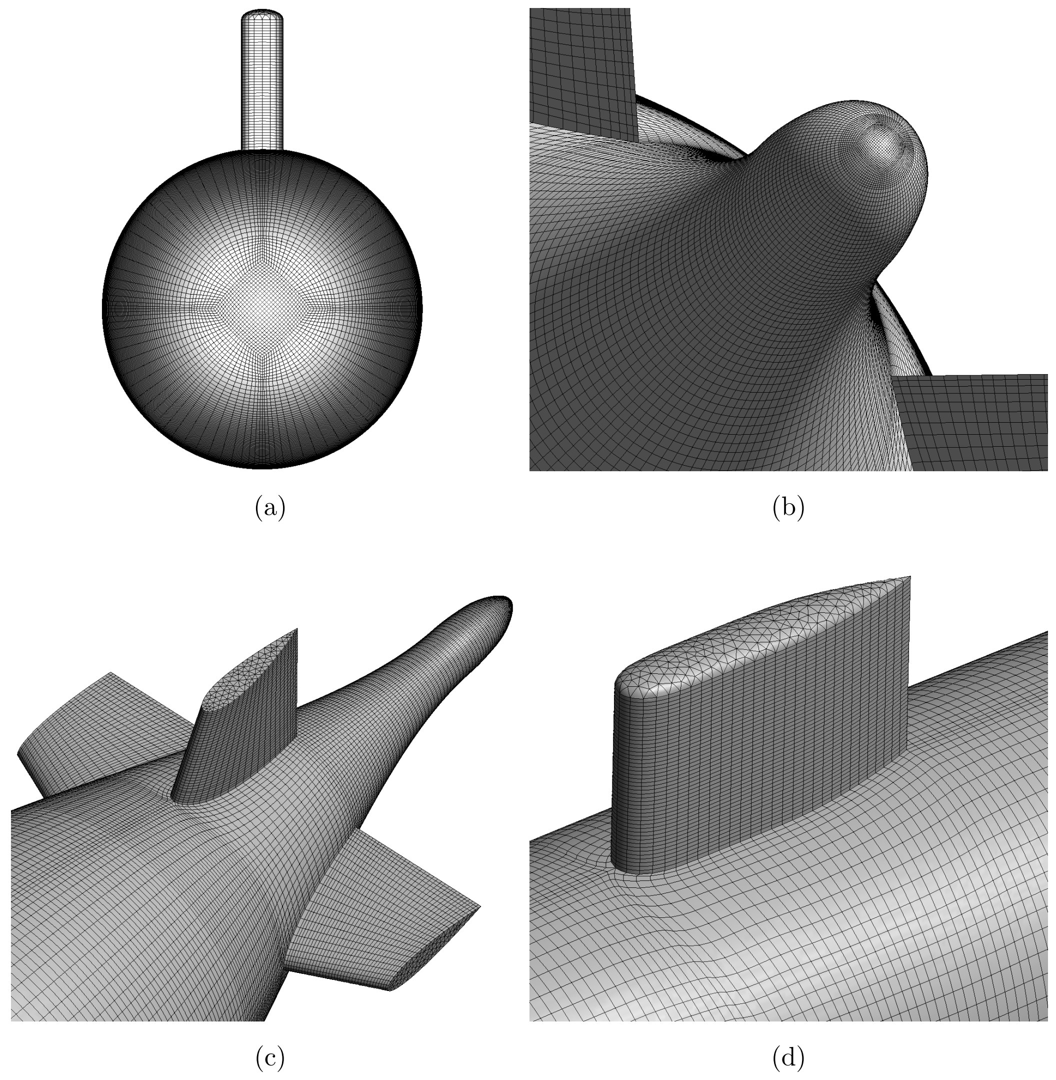


Figure 8-2: Surface mesh close-ups of the (a) bow, (b) stern, (c) stern appendages, and (d) sail.

Generating curved boundary meshes is one of the biggest challenges in the high-order community. With high-order compact discretisations, curving the surface mesh is crucial to represent the geometrical features accurately without requiring excessively small elements. The latest release of Pointwise implements high-order curved element mesh generation. In Pointwise v18.2 [113], the curving approach consist of first elevating the degree of the linear mesh with additional nodes and projecting them on top of a Computer Aided Design (CAD) model. Finally, perturbations, which are defined as the distance between the linear boundary and the projected boundary, are spread to the interior domain using a weighted smoothing method. Pointwise v18.2 performs all these steps automatically during the mesh export step. However, prior to the export step, all linear surfaces nodes have to be restricted on top the CAD model. This can be done by applying the 'project onto database' function to each boundary domain. Furthermore, high-order solver attributes, such as mesh degree and the number of smoothing steps have to be specified. Figure 8-3 illustrates linear and quadratically curved representations of the sail surface.

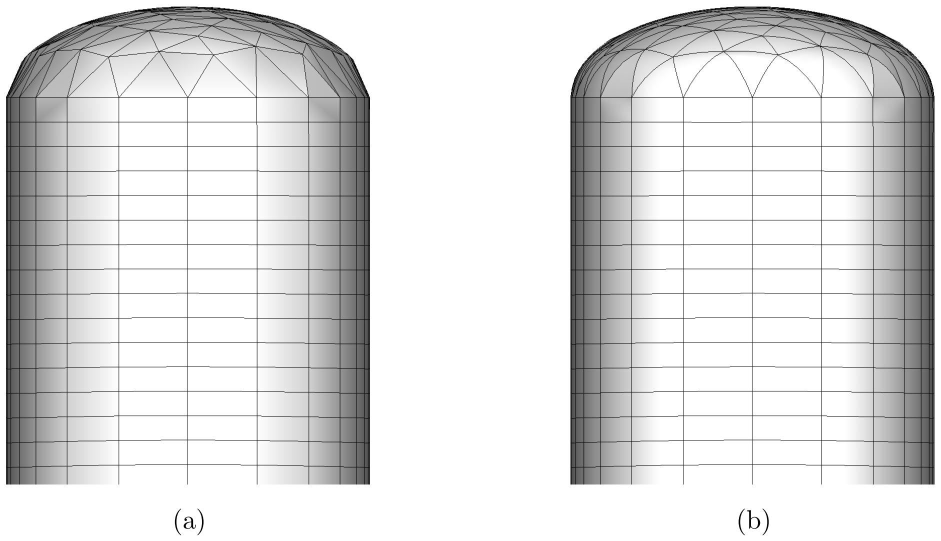


Figure 8-3: A (a) linear and (b) quadratically curved sail surface.

8.1.2 Configuration

One complete simulation with $P = 4$ and $\zeta = 4$ was run on 256 Nvidia P100 GPUs using double precision. The Gauss-Legendre solution point distribution was used for hexahedral elements and the Gauss-Legendre-Williams-Shunn solution point distribution was used for prismatic elements. The Gauss-Legendre flux point distribution was used for quadrilateral interfaces and the Williams-Shunn flux point distribution was used for triangular interfaces. No subgrid-scale turbulence model or spatial filtering was applied, and the simulation can be considered as implicit LES. Physical time was discretised with BDF2, where $\Delta t = 1 \times 10^{-3}$. P-multigrid convergence acceleration and locally adaptive pseudo-time stepping with the FR-DG $P = 4$ optimal

10-stage $\text{RK}_{10,1,1}$ embedded pair scheme was used in pseudo-time with $f_{\max} = 1.01$, $f_{\min} = 0.98$, $\kappa = 5 \times 10^{-7}$, $\Delta\tau_{min} = 4 \times 10^{-5}$, $f_\tau = 2.5$ and $\alpha_\tau = 1.6$. The P-multigrid cycle was prescribed as 1-1-1-1-4-1-1-1-6, where the integers denote the number of iterations corresponding to polynomial levels 4-3-2-1-0-1-2-3-4. Additionally, volume flux anti-aliasing for hexahedral elements was performed with quadrature degrees 11-9-7-5-3-5-7-9-11, and volume flux anti-aliasing for prismatic elements was performed with quadrature degrees 10-8-5-4-2-4-5-8-10. The Riemann-invariant-based boundary condition was prescribed at all far-field boundaries with $p_\infty = 1$, $\mathbf{v}_\infty = \{1\ 0\ 0\}^T$ and $\zeta_\infty = 120$, and a no-slip condition was prescribed on the model surface. The initial condition at $t = 0$ was set as $u_\alpha = \{1\ 1\ 0\ 0\}^T$.

Tripping was found to be necessary to induce transition of the hull boundary layer. This was prescribed as a source term which imposes a wall-normal force 0.5 hull diameters D down-stream from the tip of the bow, mimicking the effect of a trip wire used in the experiments [108, 109]. A similar strategy was used to trip the boundary layer in previous numerical studies [111, 110].

The simulation was run in three parts. First, the flow developed up to $t = 9.002$ with $P = 1$. Second, the simulation was restarted with $P = 4$ and run up to $t = 36.795$. Third, statistics were gathered between $t = 36.795$ and $t = 54.762$. In total, the simulation took 73,216 GPUh.

8.2 Parallel Efficiency

Parallel efficiency of the SUBOFF case between $N = 128$ and $N = 256$ was studied by running the simulation for one characteristic time unit $t_c = V/D$, where V_∞ is the free-stream velocity. The GPU memory loading was approximately 70% at $N = 128$ and 35% at $N = 256$. Figure 8-4 plots the strong scaling between the measurement points. A parallel efficiency of $1.82/2.0 = 0.91$ was observed at $N = 256$ which is in line with $3.48/4 = 0.87$ parallel efficiency obtained for the incompressible jet case with P-multigrid under 25% load at $N = 36$ in Figure 5-7. As locally adaptive pseudo-time stepping and optimal RK schemes do not introduce any load imbalance or inter-element communication, they should have minimal effect on the strong scaling. The high parallel efficiency observed at $N = 256$ justifies it as a cost-effective job size for the full simulation.

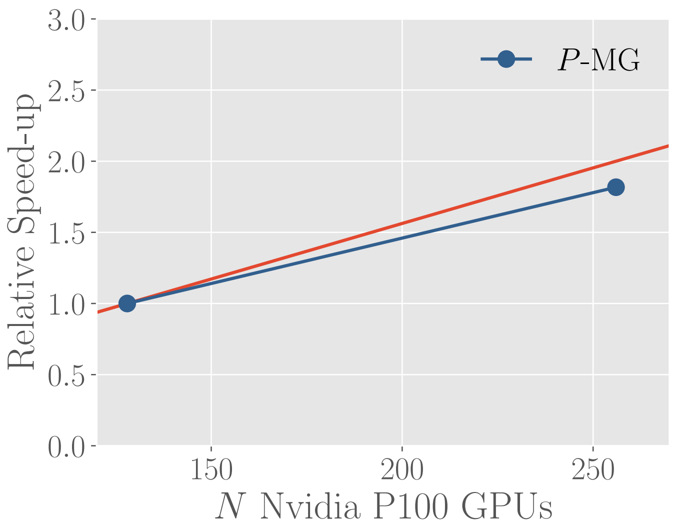


Figure 8-4: Strong scaling of the DARPA SUBOFF case with P-multigrid (P-MG). The red line indicates ideal strong scaling.

8.3 Results

Previous studies have shown that the wake of the SUBOFF configuration is characterised by a bimodal behaviour. This behaviour can be seen as two local maxima in the turbulent stress profiles. It originates from the thickening boundary layer along the contracting stern and is enhanced by the presence of the stern appendages. Figure 8-5 illustrates the flow in this region with iso-surfaces of the Q-criterion coloured by the velocity magnitude V at $t = 54.762$.

Figure 8-6 shows the vorticity magnitude $|\omega|$ on the xz-plane at $y = 0$. It can be seen that the tripping mechanism quickly transitions the laminar boundary layer to turbulence, and the turbulent boundary layer keeps developing over the entire length of the hull. Furthermore, the sail and the stern appendages generate highly turbulent wakes which merge into the stern boundary layer wake downstream. Figures 8-7a-c show the time-averaged turbulent kinetic energy $k = (\overline{v_z'}^2 + \overline{v_y'}^2 + \overline{v_z'}^2)/2$ on the xz-

plane at $y = 0$, on the xy-plane at $z = 0$, and on a diagonal plane that is rotated $45°$ in the x-direction, respectively. The bimodal behaviour can be clearly observed as two streaks of high k in all of the plots.

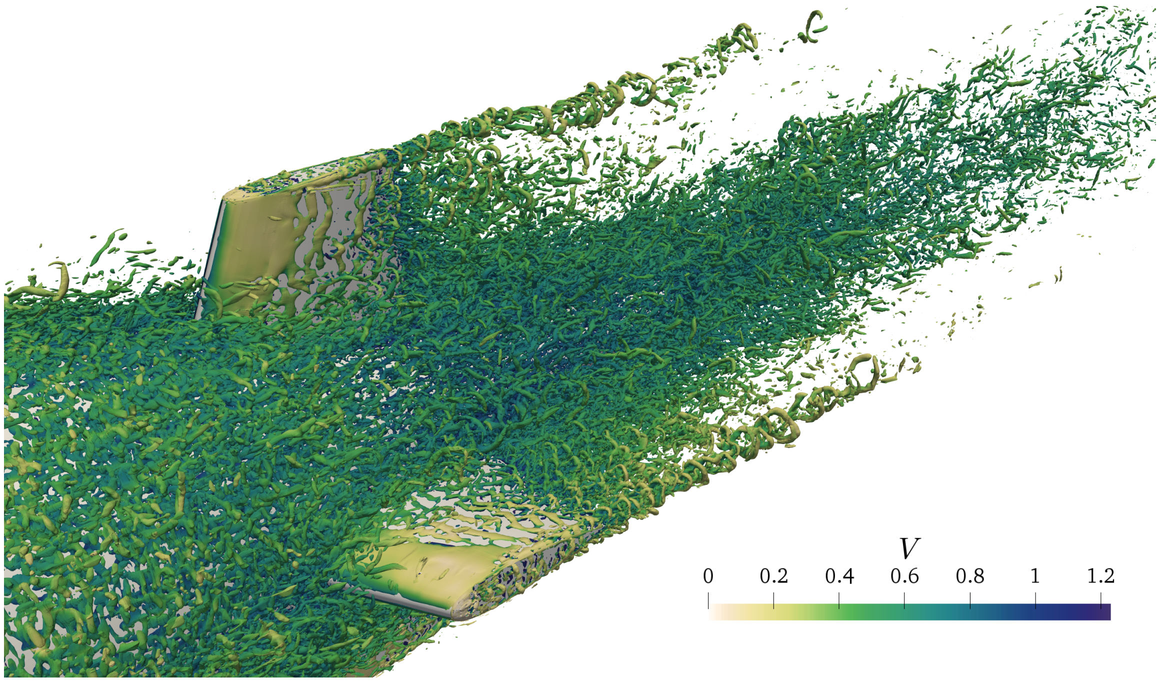


Figure 8-5: Iso-surfaces of the Q-criterion coloured with the velocity magnitude V at $t = 54.762$.

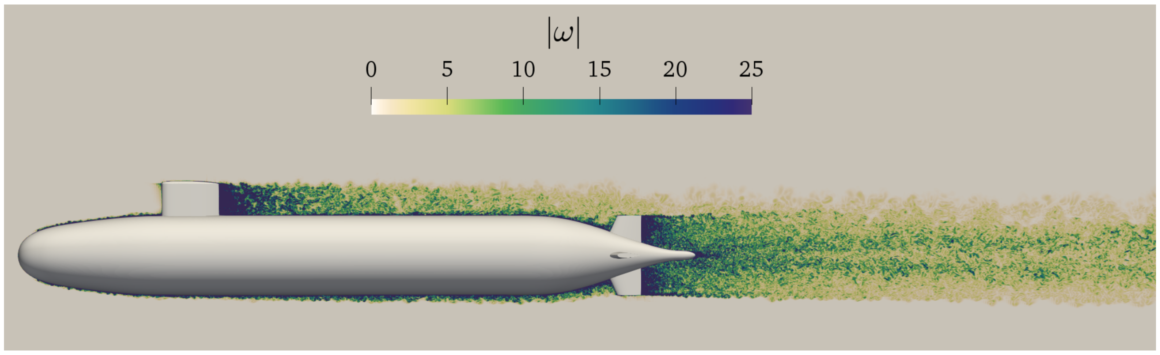
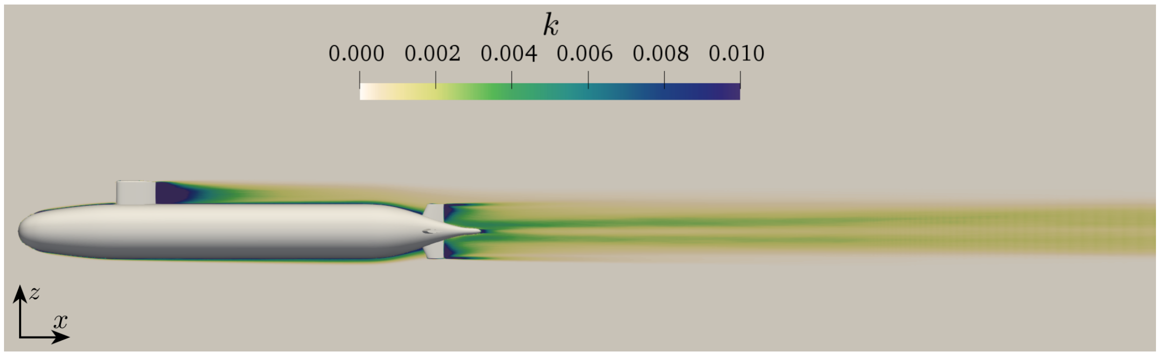
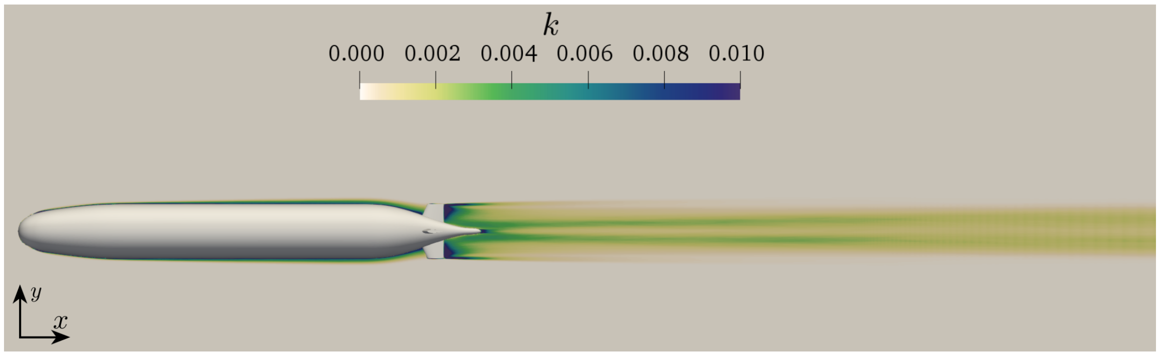


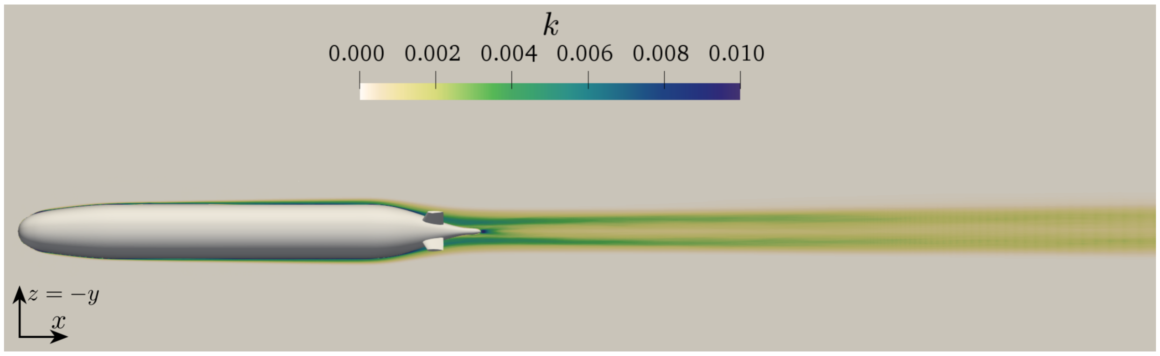
Figure 8-6: Vorticity magnitude $|\omega|$ on xz-plane at $y = 0$ and $t = 54.762$.



(a)



(b)



(c)

Figure 8-7: Turbulent kinetic energy on the (a) xz-plane at $y = 0$, (b) xy-plane at $z = 0$, and (c) diagonal plane that is rotated 45° in x-direction.

Figure 8-8a plots $\overline{v_x}/V_e$ at a measurement location that is $6D$ down-stream from the tip of the stern at $y = 0$, where V_e is the free-stream velocity outside of the wake. Figure 8-8b plots $(V_e - \overline{v_x})/v_0$, where v_0 is the maximum velocity defect, against z/l_0, where l_0 is the half-wake width, $6D$ downstream from the tip of the stern. The experimental data of Jimenez et al. [109] and numerical data of Posa et al. [111] are given as a reference. It can be seen that the quantities are overall well-predicted. The main axial velocity $\overline{v_x}/V_e$ is slightly over-estimated in the vicinity of the mid-line, but excellent agreement with the experimental data is achieved elsewhere.

Figures 8-9a-c plot the time-averaged root-mean-square velocity fluctuations scaled by V_e, $6D$ downstream from the tip of the stern. The turbulent quantities are on the whole in line with previous studies. From Figure 8-9a, it can be seen that $\overline{v_x'}/V_e$ agrees with the experimental data well, apart from in the immediate vicinity of the mid-line. From Figure 8-9b, it can be seen that notably better agreement with the experimental data is observed for $\overline{v_z'}/V_e$, compared to the results of Posa and Balaras. From Figure 8-9b it can be seen that the cross-term $\overline{v_x'v_z'}/V_e$ result also compares favourably with the experimental result 8-9b, especially on the sail side with positive z.

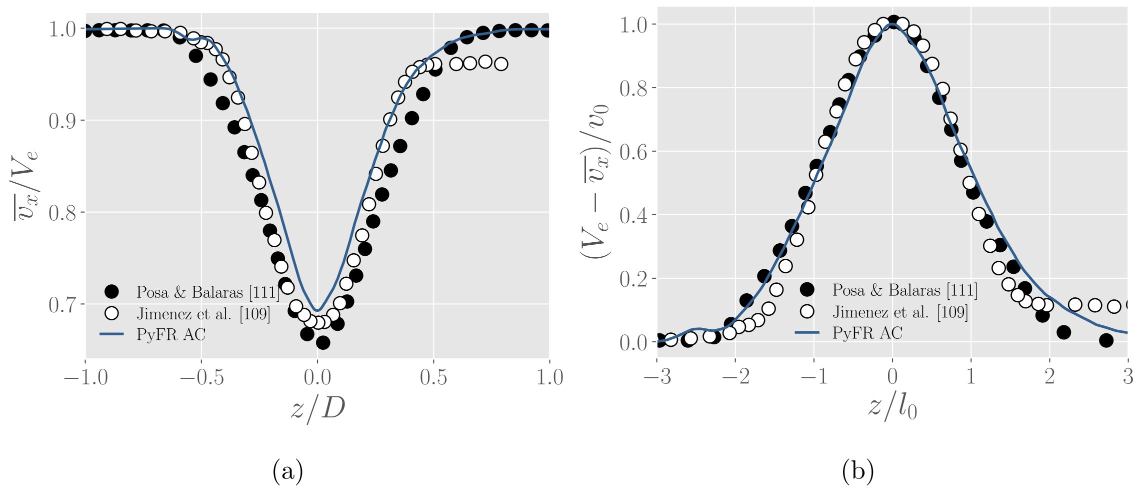
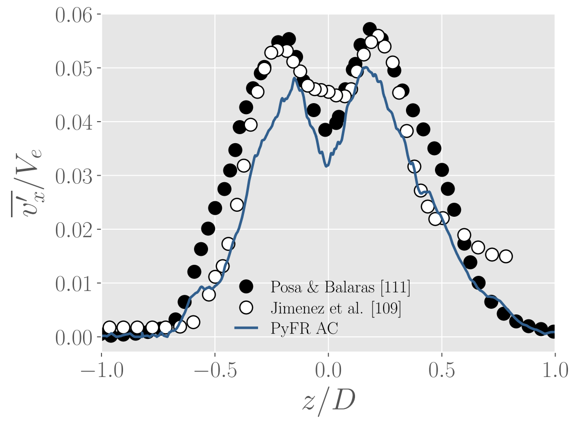
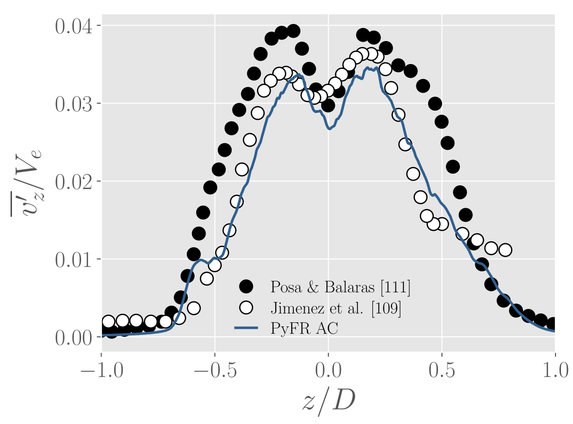


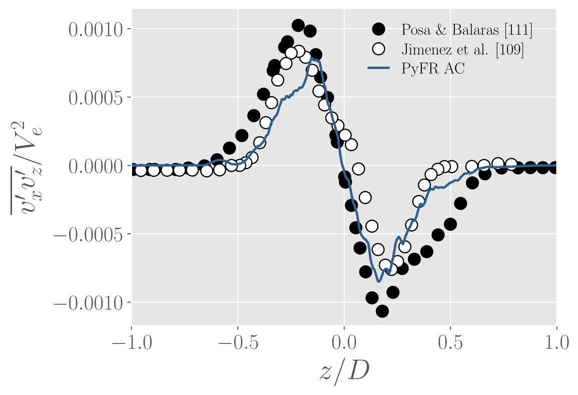
Figure 8-8: (a) $(\overline{v_x}/V_e)$ and (b) $V_e - \overline{v_x}/v_0$ at a measurement location that is $6D$ down-stream from the tip of the stern at $y = 0$.



(a)



(b)



(c)

Figure 8-9: (a) $\overline{v'_x}/V_e$ and (b) $\overline{v'_z}/V_e$ at a measurement location that is 6 hull diameters down-stream from the tip of the stern at $y = 0$.

Figure 8-10 shows the time-averaged pressure field on the xz-plane at $y = 0$. Figure 8-11 shows the time-averaged pressure coefficient $\overline{C_p} = \frac{\bar{p}-p_\infty}{\frac{1}{2}V_\infty^2}$ on the model surface at $y = 0$ together with the result of Bhushan et al. [112]. From Figure 8-10 it can be seen that regions of high pressure are formed on the front face of sail and the stern appendages. They are also observed as peaks of $\overline{C_p}$ in Figure 8-11. Overall, $\overline{C_p}$ is in very good agreement with the reference data. The negative peak of $\overline{C_p}$ in the bow region is due to the tripping term and should be disregarded. The other strong negative peak near $x/L = 0.85$ occurs in a very small region at the leading edge of the stern appendages. The origin of this peak requires further investigation.

On the whole, the obtained results demonstrate that the artificial compressibility solver with P-multigrid, locally adaptive pseudo-time stepping and optimal RK schemes can be used to conduct high-fidelity implicit LES of industrially relevant problems at scale using hundreds of GPUs.

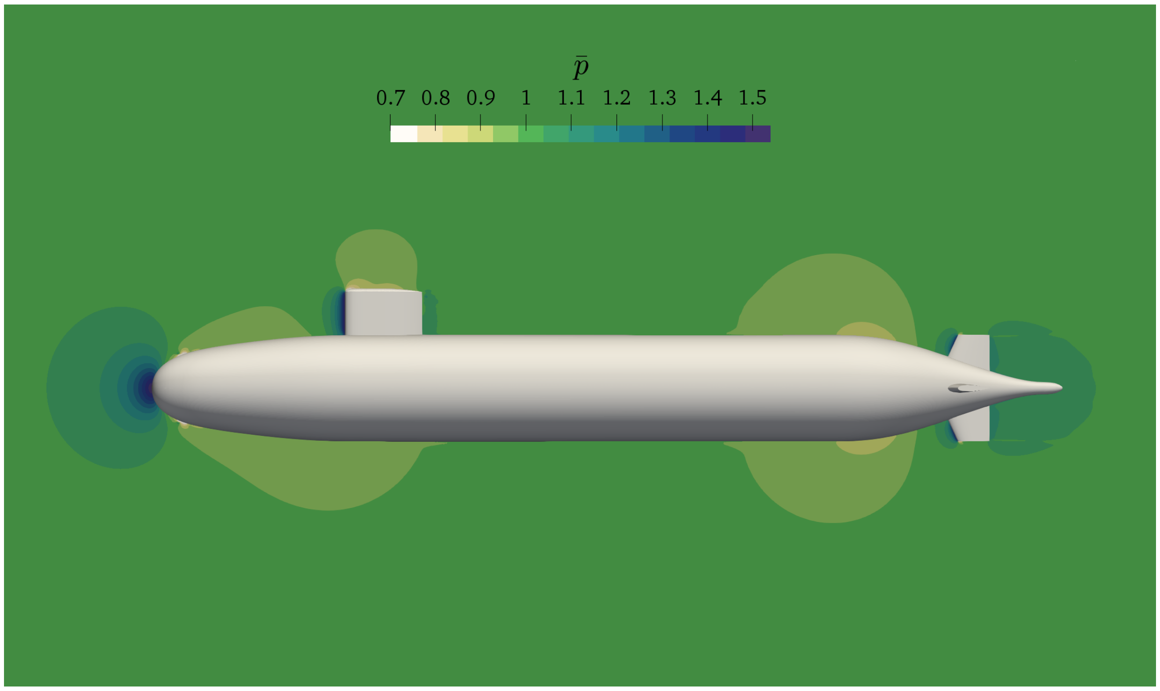


Figure 8-10: Average pressure $\bar{p}$ on the xz-plane at $y = 0$.

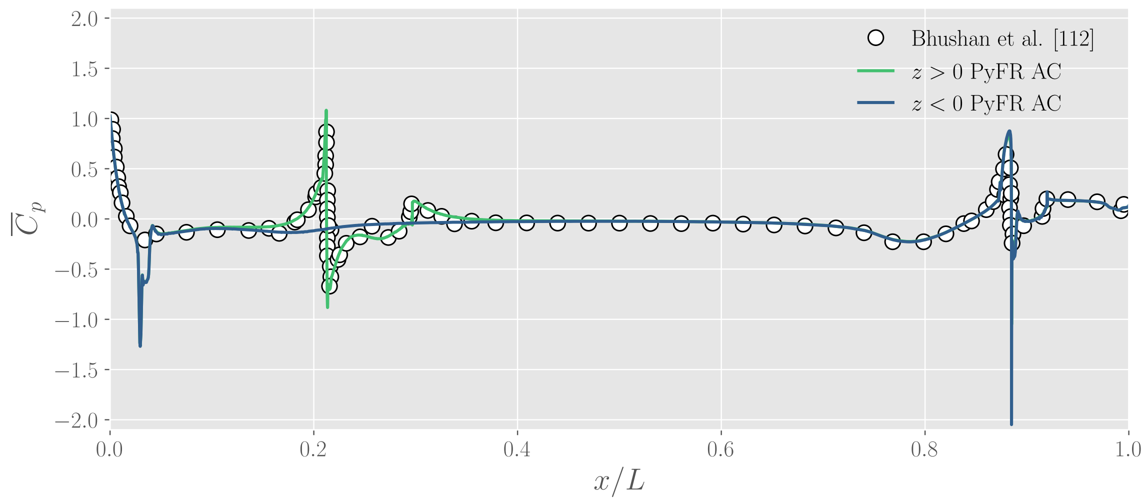


Figure 8-11: Pressure coefficient on the hull surface at $y = 0$

Chapter 9

Conclusions

9.1 Summary

A high-order incompressible Navier-Stokes solver was developed in the Python-based PyFR (www.pyfr.org) [17] framework. The solver was based on the ACM formulation with the FR discretisation in space and explicit dual-time stepping in time. Choices regarding the numerical methods and implementation were motivated as follows. Firstly, high-order FR was selected as the spatial discretisation due to its low dissipation and ability to work with unstructured meshes of complex geometries. Being discontinuous, it also allows the majority of computation to be performed locally. Secondly, convergence acceleration techniques were restricted to explicit methods in order to retain the spatial locality provided by FR, which allows efficient harnessing of the massively parallel compute capability of modern hardware. Thirdly, the solver was implemented in the PyFR framework with cross-platform support such that it can run on modern heterogeneous systems via an MPI + X model, with X being CUDA, OpenCL or OpenMP. As such, it is well-placed to remain relevant in an era of rapidly evolving hardware architectures. The implementation of the artificial compressibility solver was detailed in Chapter 4.

In order to decrease time to solution, three explicit convergence acceleration techniques were developed. Chapter 5 described P-multigrid which was validated for a Taylor-Green vortex test case at $Re = 1,600$. Chapter 6 described locally adaptive pseudo-time stepping which was validated for an SD7003 test case at $Re = 60,000$. Chapter 7 described optimal Runge-Kutta schemes which were validated for a round jet at $Re = 10,000$. The results of all validation cases were found to be in very good agreement with previous experimental and numerical studies.

In addition to the turbulent validation test cases, a performance study with a 2D cylinder test case at $Re = 100$ was undertaken using all possible convergence acceleration combinations. Figure 9-1 summarises the achieved speed-ups relative to classical RK4 pseudo-time stepping. Of the individual techniques, P-multigrid achieved the best performance, leading to a speed-up factor of 6.27. The best performing combination of two techniques was P-multigrid and locally adaptive pseudo-time stepping, leading to a speed-up factor of 15.24. Combining all convergence acceleration techniques lead to a speed-up factor of 21.26.

Finally, the artificial compressibility solver and all of the convergence acceleration techniques was applied to simulate a DARPA SUBOFF submarine model at $Re = 1.2 \times 10^6$ in Chapter 8. Excellent agreement with previous studies was obtained, demonstrating that the artificial compressibility solver with P-multigrid, locally adaptive pseudo-time stepping and optimal RK schemes can be used to conduct high-fidelity implicit LES of industrially relevant problems at scale using hundreds of GPUs.

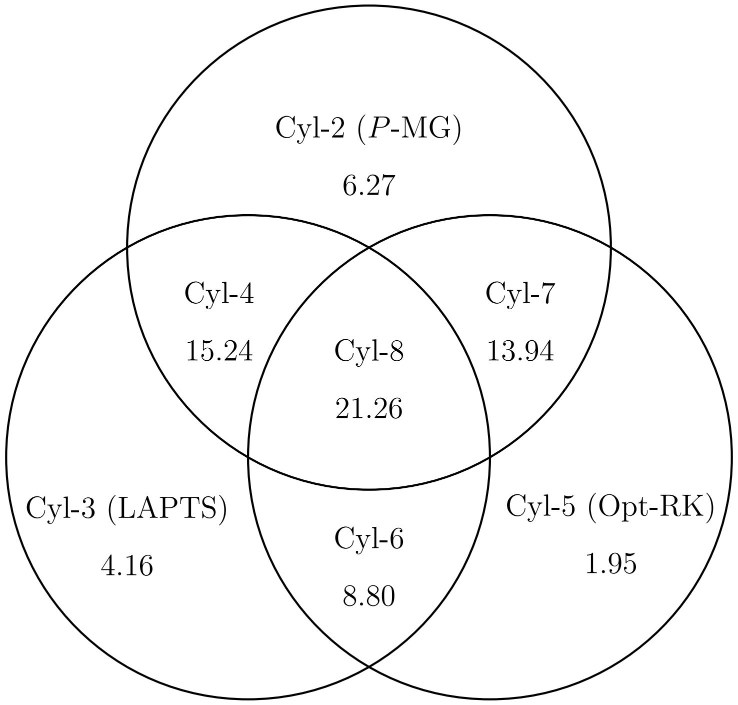


Figure 9-1: A summary of the speed-ups of all convergence acceleration combinations relative to RK4 pseudo-time stepping for the 2D cylinder cases Cyl-1 - Cyl-8.

9.2 Future Work

In the short term, future work should involve investigation of additional steady-state convergence acceleration techniques. For example, residual splitting and explicit residual averaging [114] techniques are likely to be well-suited for the current solver. In residual splitting, the residual is split into convective and diffusive parts and the diffusive part is frozen within a pseudo-time step in order to save computational cost. In residual averaging, the residual at each point is replaced with a weighted average of residuals at neighbouring points to increase maximum allowable pseudo-time step size. Neither technique has so far been applied in the context of high-order FR.

In the medium to long term, future work should focus on extending the physics capabilities of the solver such that it can be applied to a wider range of hydrodynamic problems. This work should include extension two two-phase flows via level sets [115] or the volume-of-fluid method [116] and extension to hydroacoustics via the hydrodynamic/acoustic splitting approach [117].

Bibliography

[1] B. C. Vermeire, F. D. Witherden, and P. E. Vincent, "On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools," *Journal of Computational Physics*, vol. 334, pp. 497–521, 2017.

[2] Z. J. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. T. Huynh, N. Kroll, G. May, P. O. Persson, B. van Leer, and M. Visbal, "High-order CFD methods: Current status and perspective," *International Journal for Numerical Methods in Fluids*, vol. 72, no. 8, pp. 811–845, 2013.

[3] M. Turner, J. Peiró, and D. Moxey, "Computer-aided design curvilinear mesh generation using a variational framework," *Computer-Aided Design*, vol. 103, pp. 73–91, 2018.

[4] D. Gottlieb and S. A. Orszag, *Numerical analysis of spectral methods: theory and applications*, vol. 26. Siam, 1977.

[5] S. K. Lele, "Compact finite difference schemes with spectral-like resolution," *Journal of computational physics*, vol. 103, no. 1, pp. 16–42, 1992.

[6] W. H. Reed and T. R. Hill, "Triangular mesh methods for the neutron transport equation," *Los Alamos Report LA-UR-73-479*, no. 836, p. 10, 1973.

[7] B. Cockburn, S. Hou, and C.-W. Shu, "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II : General framework," *Mathematics of Computation*, vol. 52, no. 186, pp. 411–435, 1989.

[8] B. Cockburn, S. Hou, and C.-W. Shu, "The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV : The multidimensional case," *Mathematics of Computation*, vol. 54, no. 190, pp. 545–581, 1190.

[9] Y. Liu, M. Vinokur, and Z. J. Wang, "Spectral difference method for unstructured grids I: Basic formulation," *Journal of Computational Physics*, vol. 216, no. 2, pp. 780–801, 2006.

[10] Z. J. Wang, Y. Liu, G. May, and A. Jameson, "Spectral difference method for unstructured grids II: Extension to the Euler equations," *Journal of Scientific Computing*, vol. 32, no. 1, pp. 45–71, 2007.

[11] D. A. Kopriva and J. H. Kolias, "A conservative staggered-grid Chebyshev multidomain method for compressible flows," *Journal of Computational Physics*, vol. 125, no. 1, pp. 244–261, 1996.

[12] H. T. Huynh, "A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods," in *18th AIAA Computational Fluid Dynamics Conference*, 2007.

[13] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven, "Nodal discontinuous Galerkin methods on graphics processors," *Journal of Computational Physics*, vol. 228, no. 21, pp. 7863–7882, 2009.

[14] P. Castonguay, D. Williams, P. Vincent, M. López, and A. Jameson, "On the development of a high-order, multi-GPU enabled, compressible viscous flow Solver for mixed unstructured grids," in *20th AIAA Computational Fluid Dynamics Conference*, 2011.

[15] P. Castonguay, *High-Order Energy Stable Flux Reconstruction Schemes for Fluid Flow Simulations on Unstructured Grids.* PhD thesis, Stanford University, 2012.

[16] M. R. López, A. Sheshadri, J. R. Bull, T. D. Economon, J. Romero, J. E. Watkins, D. M. Williams, F. Palacios, A. Jameson, and D. E. Manosalvas, "Verification and validation of HiFiLES: A high-Order LES unstructured solver on multi-GPU platforms," in *32nd AIAA Applied Aerodynamics Conference*, 2014.

[17] F. D. Witherden, A. M. Farrington, and P. E. Vincent, "PyFR: An open source framework for solving advection-diffusion type problems on streaming architectures using the flux reconstruction approach," *Computer Physics Communications*, vol. 185, no. 11, pp. 3028–3040, 2014.

[18] F. D. Witherden, *On the development and implementation of high-order flux reconstruction schemes for computational fluid dynamics.* PhD thesis, Imperial College London, 2015.

[19] C. T. Jacobs, S. P. Jammy, and N. D. Sandham, "OpenSBLI : A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures," *Journal of Computational Science*, vol. 18, pp. 12–23, 2017.

[20] F. D. Witherden, B. C. Vermeire, and P. E. Vincent, "Heterogeneous computing on mixed unstructured grids with PyFR," *Computers and Fluids*, vol. 120, pp. 173–186, 2015.

[21] J. S. Park, F. D. Witherden, and P. E. Vincent, "High-order implicit large-eddy simulations of flow over a NACA0021 aerofoil," *AIAA Journal*, vol. 55, no. 7, 2017.

[22] C. T. Jacobs, M. Zauner, N. D. Tullio, S. P. Jammy, D. J. Lusher, and N. D. Sandham, "An error indicator for finite difference methods using spectral techniques with application to aerofoil simulation," *Computers and Fluids*, vol. 168, pp. 67–72, 2018.

[23] D. J. Lusher, S. P. Jammy, and N. D. Sandham, "Shock-wave / boundary-layer interactions in the automatic source-code generation framework OpenSBLI," *Computers and Fluids*, vol. 173, pp. 17–21, 2018.

[24] P.-O. Persson, "A sparse and high-order accurate line-based discontinuous Galerkin method for unstructured meshes," *Journal of Computational Physics*, vol. 233, pp. 414–429, 2013.

[25] F. Bassi, L. Botti, A. Colombo, A. Crivellini, A. Ghidoni, and F. Massa, "On the development of an implicit high-order discontinuous Galerkin method for DNS and implicit LES of turbulent flows," *European Journal of Mechanics B/Fluids*, vol. 55, pp. 367–379, 2016.

[26] B. C. Vermeire, S. Nadarajah, and P. G. Tucker, "Implicit large eddy simulation using the high-order correction procedure via reconstruction scheme," *International Journal for Numerical Methods in Fluids*, vol. 82, no. 5, pp. 231–260, 2016.

[27] W. Pazner and P.-O. Persson, "High-order DNS and LES simulations using an implicit tensor-product discontinuous Galerkin method," in *23rd AIAA Computational Fluid Dynamics Conference*, 2017.

[28] P. Fernandez, N. C. Nguyen, and J. Peraire, "The hybridized discontinuous Galerkin method for implicit large-eddy simulation of transitional turbulent flows," *Journal of Computational Physics*, vol. 336, pp. 308–329, 2017.

[29] R. Vandenhoeck and A. Lani, "Implicit high-order flux reconstruction positivity preserving LLAV scheme for viscous high-speed flows," in *AIAA Scitech 2019 Forum*, 2019.

[30] J. E. Watkins, J. Romero, and A. Jameson, "Multi-GPU, implicit time stepping for high-order methods on unstructured grids," in *46th AIAA Fluid Dynamics Conference*, 2016.

[31] J. Lou, Y. Xia, L. Luo, H. Luo, J. R. Edwards, and F. Mueller, "OpenACC directive-based GPU acceleration of an implicit reconstructed discontinuous Galerkin method for compressible flows on 3D unstructured grids," in *54th AIAA Aerospace Sciences Meeting*, 2016.

[32] M. Aissa, T. Verstraete, and C. Vuik, "Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes," *Computers and Mathematics with Applications*, vol. 74, no. 1, pp. 201–217, 2017.

[33] F. Rieper and G. Bader, "The influence of cell geometry on the accuracy of upwind schemes in the low mach number regime," *Journal of Computational Physics*, vol. 228, no. 8, pp. 2918–2933, 2009.

[34] H. Guillard and C. Viozat, "On the behaviour of upwind schemes in the low Mach number limit," *Computers & Fluids*, vol. 28, pp. 63–86, 1999.

[35] C. Liang, S. Premasuthan, and A. Jameson, "High-order accurate simulation of low-Mach laminar flow past two side-by-side cylinders using spectral difference method," *Computers and Structures*, vol. 87, no. 11-12, pp. 812–827, 2009.

[36] G. Lodato and A. Jameson, "LES modeling with high-order flux reconstruction and spectral difference schemes," *7th ICCFD*, pp. 1–17, 2012.

[37] J. Bodart, C. Scalo, G. Shelekhov, and L. Joly, "Separation delay via hydro-acoustic control of a NACA-4412 airfoil in pre-stalled conditions," in *55th AIAA Aerospace Sciences Meeting*, 2015.

[38] V. Vatsa and E. Turkel, "Choice of variables and preconditioning for time dependent problems," in *16th AIAA Computational Fluid Dynamics Conference*, 2003.

[39] E. Turkel, "Preconditioned methods for solving the incompressible and low speed compressible equations," *Journal of Computational Physics*, vol. 72, no. 2, pp. 277–298, 1987.

[40] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, R. M. Kirby, and S. J. Sherwin, "Nektar++: An open-source spectral/hp element framework," *Computer Physics Communications*, vol. 192, pp. 205–219, 2015.

[41] J.-E. W. Lombard, D. Moxey, and S. J. Sherwin, "Implicit large-eddy simulation of a wingtip vortex," *AIAA Journal*, vol. 54, no. 2, 2016.

[42] D. Serson, J. R. Meneghini, and S. J. Sherwin, "Direct numerical simulations of the flow around wings with spanwise waviness," *Journal of Fluid Mechanics*, vol. 826, pp. 714–731, 2017.

[43] B. Krank, N. Fehn, W. A. Wall, and M. Kronbichler, "A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow," *Journal of Computational Physics*, vol. 348, pp. 634–659, 2017.

[44] N. Fehn, W. A. Wall, and M. Kronbichler, "On the stability of projection methods for the incompressible Navier Stokes equations based on high-order discontinuous Galerkin discretizations," *Journal of Computational Physics*, vol. 351, pp. 392–421, 2017.

[45] N. Fehn, W. A. Wall, and M. Kronbichler, "Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows," *Journal of Computational Physics*, vol. 372, pp. 667–693, 2018.

[46] B. Cockburn, J. Gopalakrishnan, and R. Lazarov, "Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems," *SIAM Journal on*, vol. 47, no. 2, pp. 1319–1365, 2009.

[47] N. C. Nguyen, J. Peraire, and B. Cockburn, "A hybridizable discontinuous Galerkin method for the incompressible Navier-Stokes equations," in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010.

[48] S. Laizet and E. Lamballais, "High-order compact schemes for incompressible flows : A simple and efficient method with quasi-spectral accuracy," *Journal of Computational Physics*, vol. 228, no. 16, pp. 5989–6015, 2009.

[49] S. Laizet and N. Li, "Incompact3d : A powerful tool to tackle turbulence problems with up to O (10ˆ5) computational cores," *International Journal for Numerical Methods in Fluids*, vol. 67, pp. 1735–1757, 2011.

[50] C. Diaz-Daniel, S. Laizet, and J. C. Vassilicos, "Simulations of a wall-attached cube immersed in laminar and turbulent boundary layers," *International Journal of Heat and Fluid Flow*, vol. 68, pp. 269–280, 2017.

[51] E. P. Francisco, L. F. R. Espath, S. Laizet, and J. H. Silvestrini, "Reynolds number and settling velocity influence for finite-release particle-laden gravity currents in a basin," *Computers and Geosciences*, vol. 110, pp. 1–9, 2017.

[52] X. Roca, N. C. Nguyen, and J. Peraire, "GPU-accelerated sparse matrix-vector product for a hybridizable discontinuous Galerkin method," in *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2011.

[53] A. Karakus, N. Chalmers, K. Swirydowicz, and T. Warburton, "A GPU accelerated discontinuous Galerkin incompressible flow solver," *arXiv preprint arXiv:1801.00246*, pp. 1–33, 2017.

[54] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, and R. Strzodka, "AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods," *SIAM Journal on Scientific Computing*, vol. 37, no. 2, pp. 602–626, 2015.

[55] A. J. Chorin, “A numerical methods for solving incompressible viscous flow problems,” *Journal of Computational Physics*, vol. 135, pp. 118–125, 1997.

[56] A. Jameson, “Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings,” in *10th Computational Fluid Dynamics Conference*, 1991.

[57] F. Bassi, A. Crivellini, D. A. Di Pietro, and S. Rebay, “An artificial compressibility flux for the discontinuous Galerkin solution of the incompressible Navier-Stokes equations,” *Journal of Computational Physics*, vol. 218, no. 2, pp. 794–815, 2006.

[58] C. Liang, A. Chan, X. Liu, and A. Jameson, “An artificial compressibility method for the spectral difference solution of unsteady incompressible Navier-Stokes equations on multiple grids,” *In 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2011.

[59] C. Cox, C. Liang, and M. W. Plesniak, “A high-order solver for unsteady incompressible Navier-Stokes equations using the flux reconstruction method on unstructured grids with implicit dual time stepping,” *Journal of Computational Physics*, vol. 314, pp. 414–435, 2016.

[60] N. A. Loppi, F. D. Witherden, A. Jameson, and P. E. Vincent, “A high-order cross-platform incompressible Navier-Stokes solver via artificial compressibility with application to a turbulent jet,” *Computer Physics Communications*, vol. 233, pp. 193–205, 2018.

[61] F. Witherden and P. Vincent, “An Analysis of Solution Point Coordinates for Flux Reconstruction Schemes on Triangular Elements,” *Journal of Scientific Computing*, vol. 61, pp. 398–423, 2014.

[62] L. Shunn and F. Ham, “Symmetric quadrature rules for tetrahedra based on a cubic close-packed lattice arrangement,” *Journal of Computational and Applied Mathematics*, vol. 236, no. 17, pp. 4348–4364, 2012.

[63] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications.* Springer Science & Business Media, 2007.

[64] P. E. Vincent, P. Castonguay, and A. Jameson, “A new class of high-order energy etable flux reconstruction schemes,” *Journal of Scientific Computing*, vol. 47, no. 1, pp. 50–72, 2011.

[65] P. Castonguay, D. Williams, P. Vincent, and A. Jameson, “Energy stable flux reconstruction schemes for advection-diffusion problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 267, pp. 400–417, 2013.

[66] P. E. Vincent, A. M. Farrington, F. D. Witherden, and A. Jameson, “An extended range of stable-symmetric-conservative flux reconstruction correction

functions," *Computer Methods in Applied Mechanics and Engineering*, vol. 296, pp. 248–272, 2015.

[67] Y. Allaneau and A. Jameson, "Connections between the filtered discontinuous Galerkin method and the flux reconstruction approach to high order discretizations," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, pp. 3628–3636, 2011.

[68] D. M. Williams and A. Jameson, "Energy stable flux reconstruction schemes for advection-diffusion problems on tetrahedra," *Journal of Scientific Computing*, vol. 59, no. 3, pp. 721–759, 2014.

[69] P. Zwanenburg and S. Nadarajah, "Equivalence between the energy stable flux reconstruction and filtered discontinuous Galerkin schemes," *Journal of Computational Physics*, vol. 306, pp. 343–369, 2016.

[70] A. Sheshadri and A. Jameson, "An analysis of stability of the flux reconstruction formulation on quadrilateral elements for the linear advection-diffusion equation," *Journal of Scientific Computing*, vol. 74, no. 3, pp. 1757–1785, 2018.

[71] H. T. Huynh, "High-order methods including discontinuous Galerkin by reconstructions on triangular meshes," in *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2011.

[72] P. Castonguay, P. E. Vincent, and A. Jameson, "A new class of high-order energy stable flux reconstruction schemes for triangular elements," *Journal of Scientific Computing*, vol. 51, no. 1, pp. 224–256, 2012.

[73] D. M. Williams, P. Castonguay, P. E. Vincent, and A. Jameson, "Energy stable flux reconstruction schemes for advection-diffusion problems on triangles," *Journal of Computational Physics*, vol. 250, pp. 53–76, 2013.

[74] P. Raviart and J. Thomas, "Primal hybrid finite element methods for 2nd order elliptic equations," *Mathematics of Computation*, vol. 31, no. 138, pp. 391–413, 1977.

[75] A. R. Winters, R. C. Moura, G. Mengaldo, G. J. Gassner, S. Walch, J. Peiro, and S. J. Sherwin, "A comparative study on polynomial dealiasing and split form discontinuous Galerkin schemes for under-resolved turbulence computations," *Journal of Computational Physics*, vol. 372, pp. 1–21, 2018.

[76] A. Jameson, P. E. Vincent, and P. Castonguay, "On the non-linear stability of flux reconstruction schemes," *Journal of Scientific Computing*, vol. 50, no. 2, pp. 434–445, 2012.

[77] A. Jameson, "A proof of the stability of the spectral difference method for all orders of accuracy," *Journal of Scientific Computing*, vol. 45, pp. 348–358, 2010.

[78] W. Trojak, R. Watson, and P. G. Tucker, "Temporal stabilisation of flux reconstruction on linear problems," *2018 Fluid Dynamics Conference*, 2018.

[79] E. Hairer and H. Wanner, *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems.* Springer, 2 ed., 1996.

[80] D. I. Ketcheson and A. J. Ahmadia, "Optimal stability polynomials for numerical integration of initial value problems," *Communications in Applied Mathematics and Computational Science*, vol. 7, no. 2, 2012.

[81] C. A. Kennedy, M. H. Carpenter, and R. M. Lewis, "Low-storage, explicit Runge-Kutta schemes for the compressible Navier-Stokes equations," *Applied Numerical Mathematics*, vol. 35, no. 3, pp. 177–219, 2000.

[82] Z. Yang and D. Mavriplis, "Unstructured dynamic meshes with higher-order time integration schemes for the unsteady Navier-Stokes equations," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, 2005.

[83] D. T. Toro and E. F. Elsworth, "A numerical investigation of the artificial compressibility method for the solution of the Navier-Stokes equations," in *Cranfield Aeronautics Report 9213*, 1992.

[84] J. M. Hsu, *An implicit-explicit flow solver for complex unsteady flows.* PhD thesis, Stanford University, 2004.

[85] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[86] B. D. Wozniak, F. D. Witherden, F. P. Russell, P. Vincent, and P. H. J. Kelly, "GiMMiK - Generating bespoke matrix multiplication kernels for accelerators: Application to high-order computational fluid dynamics," *Computer Physics Communications*, vol. 202, pp. 12–22, 2016.

[87] A. Heinecke, G. Henry, M. Hutchinson, and H. Pabst, "LIBXSMM : Accelerating small matrix multiplications by runtime code generation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016.

[88] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL : A scripting-based approach to GPU run-time code generation," *Parallel Computing*, vol. 38, pp. 157–174, 2012.

[89] M. Harris, "Optimizing parallel reduction in CUDA," *Nvidia developer technology*, 2007.

[90] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal, "p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations," *Journal of Computational Physics*, vol. 207, no. 1, pp. 92–113, 2005.

[91] W. M. van Rees, A. Leonard, D. I. Pullin, and P. Koumoutsakos, "A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers," *Journal of Computational Physics*, vol. 230, no. 8, pp. 2794–2805, 2011.

[92] A. Jameson, "Transonic flow calculations," *Lectures Presented at Von Karman Institute*, pp. 1–93, 1976.

[93] A. G. Malan, R. W. Lewis, and P. Nithiarasu, "An improved unsteady, unstructured, artificial compressibility, finite volume scheme for viscous incompressible flows: Part I. Theory and implementation," *International Journal for Numerical Methods in Engineering*, vol. 54, no. 5, pp. 695–714, 2002.

[94] C. G. Thomas and P. Nithiarasu, "Influences of element size and variable smoothing on inviscid compressible flow solution," *International Journal of Numerical Methods for Heat and Fluid Flow*, vol. 15, no. 5, pp. 420–428, 2005.

[95] S. Minisini, E. Zhebel, A. Kononov, and W. Mulder, "Local time stepping with the discontinuous Galerkin method for wave propagation in 3D heterogeneous media," *Geophysics*, vol. 78, no. 3, 2013.

[96] P. Nithiarasu, "An efficient artificial compressibility (AC) scheme based on the characteristic based split (CBS) method for incompressible flows," *International Journal for Numerical Methods in Engineering*, vol. 56, no. 13, pp. 1815–1845, 2003.

[97] A. D. Beck, T. Bolemann, D. Flad, H. Frank, G. J. Gassner, F. Hindenlang, and C. D. Munz, "High-order discontinuous Galerkin spectral element methods for transitional and turbulent flow simulations," *International Journal for Numerical Methods in Fluids*, vol. 76, no. 8, pp. 522–548, 2014.

[98] M. S. Selig and B. D. McGranahan, "Summary of Low speed Airfoil Data," *SoarTech*, vol. 1, 1995.

[99] B. C. Vermeire, N. A. Loppi, and P. E. Vincent, "Optimal Runge-Kutta schemes for pseudo time-stepping with high-order unstructured methods," *Journal of Computational Physics*, vol. 383, pp. 55–71, 2019.

[100] P. E. Vincent, P. Castonguay, and A. Jameson, "Insights from von Neumann analysis of high-order flux reconstruction schemes," *Journal of Computational Physics*, vol. 230, no. 22, pp. 8134–8154, 2011.

[101] B. C. Vermeire and P. E. Vincent, "On the behaviour of fully-discrete flux reconstruction schemes," *Computer Methods in Applied Mechanics and Engineering*, vol. 315, pp. 1053–1079, 2017.

[102] B. C. Vermeire and P. E. Vincent, "On the properties of energy stable flux reconstruction schemes for implicit large eddy simulation," *Journal of Computational Physics*, vol. 327, pp. 368–388, 2016.

[103] N. R. Panchapakesan and J. L. Lumley, "Turbulence measurements in axisymmetric jets of air and helium. I - Air jet. II - Helium jet," *Journal of Fluid Mechanics*, vol. 246, pp. 197 – 223, 1993.

[104] G. Lipari and P. K. Stansby, "Review of experimental data on incompressible turbulent round jets," *Flow, Turbulence and Combustion*, vol. 87, no. 1, pp. 79–114, 2011.

[105] B. J. Boersma, "Numerical simulation of the noise generated by a low Mach number, low Reynolds number jet," *Fluid Dynamics Research*, vol. 35, no. 6, pp. 425–447, 2004.

[106] C. Bogey and C. Bailly, "Turbulence and energy budget in a self-preserving round jet: direct evaluation using large eddy simulation," *Journal of Fluid Mechanics*, vol. 627, pp. 129–160, 2009.

[107] N. C. Groves, T. T. Huang, and M. S. Chang, "Geometric characteristics of DARPA suboff models (DTRC Model Nos. 5470 and 5471)," *David Taylor Research Center*, 1989.

[108] J. M. Jiménez, M. Hultmark, and A. J. Smits, "The intermediate wake of a body of revolution at high Reynolds numbers," *Journal of Fluid Mechanics*, vol. 659, pp. 516–539, 2010.

[109] J. M. Jiménez, R. T. Reynolds, and A. J. Smits, "The effects of fins on the intermediate wake of a submarine model," *Journal of Fluids Engineering*, vol. 132, no. 3, 2010.

[110] P. Kumar and K. Mahesh, "Large-eddy simulation of flow over an axisymmetric body of revolution," *Journal of Fluid Mechanics*, vol. 853, pp. 537–563, 2018.

[111] A. Posa and E. Balaras, "A numerical investigation of the wake of an axisymmetric body with appendages," *Journal of Fluid Mechanics*, vol. 792, pp. 470–498, 2016.

[112] S. Bhushan, M. F. Alam, and D. K. Walters, "Evaluation of hybrid RANS/LES models for prediction of flow around surface combatant and Suboff geometries," *Computers and Fluids*, vol. 88, pp. 834–849, 2013.

[113] S. L. Karman, J. T. Erwin, R. S. Glasby, and D. Stefanski, "High-order mesh curving using WCN mesh optimization," in *46th AIAA Fluid Dynamics Conference*, 2016.

[114] A. Jameson, "Origins and further development of the Jameson-Schmidt-Turkel schme," *AIAA Journal*, vol. 55, no. 5, pp. 13–17, 2017.

[115] Z. Wang and Z. J. Wang, "Multi-phase flow computation with semi-lagrangian level," *In 43rd AIAA Aerospace Sciences Meeting and Exhibit*, 2005.

[116] M. Mortezazadeh and K. Hejranfar, "Simulation of incompressible multiphase flows using the artificial compressibility method," in *ASME 2018 5th Joint US-European Fluids Engineering Division Summer Meeting. American Society of Mechanical Engineers*, pp. 1–7, 2019.

[117] C. Schmitt, *A hydrodynamic/acoustic splitting approach for the prediction of combustion induced noise.* PhD thesis, Stanford University, 2013.