Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Industriale

Towards efficient and scalable discontinuous Galerkin methods for unsteady flows

Ph.D. Dissertation of:
Matteo Franciolini

Advisor:
Prof. Andrea Crivellini

Curriculum Supervisor:
Prof. Ferruccio Mandorli

XVIII edition - new series

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria Industriale

Towards efficient and scalable discontinuous Galerkin methods for unsteady flows

Ph.D. Dissertation of:
Matteo Franciolini

Advisor:
Prof. Andrea Crivellini

Curriculum Supervisor:
Prof. Ferruccio Mandorli

XVIII edition - new series

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Facoltà di Ingegneria
Via Brecce Bianche – 60131 Ancona (AN), Italy

Acknowledgments

With my deepest gratitude I would like to thank people that assisted my personal and professional development over these years. First and foremost, I acknowledge my advisor, Prof. Andrea Crivellini, for his never ending efforts to guide my research, and more importantly, for teaching me the passion for this job. Second, Prof. Krzysztof Fidkowski is gratefully acknowledged for accommodating my visit at the University of Michigan and for his menthorship during the months spent in the United States. All the people of the "MIGALE" team, Dr. Francesco Carlo Massa, Dr. Alessandro Colombo, Dr. Lorenzo Botti and Prof. Francesco Bassi from University of Bergamo, Dr. Alessandra Nigro from University of Bozen, Dr. Gianmaria Noventa and Prof. Antonio Ghidoni from University of Brescia are also gratefully acknowledged for the useful discussions and collaboration. In particular, a special thank goes to Lorenzo Botti for his help in introducing the formalism presented in Chapter 2 and 5. My gratitude goes also to Dr. Andrea Da Ronch from the University of Southampton and his research group, for the great work done together over the last years. I would like also to thank Dr. Andrea Cimarelli for the valuable time spent together at Marche Polytechnic University.

The work proposed reports geometries, meshes, experimental data and numerical data kindly provided by other researchers and professionals. To this end, Aki Mäkivirta from Genelec, Timo Lähivaara and Simo-Pekka Simonaho from University of Eastern Finland and Tomi Huttunen from Kuava Oy are acknowledged for providing the Genelec speaker geometry and details of their numerical and experimental tests. Dr. Riccardo Togni, Dr. Andrea Cimarelli and Prof. Elisabetta De Angelis are acknowledged for sharing the computational details and the results of the Rayleigh–Bérnard convection DNS data. Prof. Michele Napolitano (Politecnico di Bari) is also gratefully acknowledged for sharing the experimental data of the T3L test case suite. Dr. Ralf Hartman from DLR, the EU-funded projects "Advanced Turbulence Simulation for Aerodynamic Application Challenges" (ATAAC) and "Towards Industrial LES/DNS in Aeronautics – Paving the Way for Future Accurate CFD" (TILDA) are also acknowledged for sharing the mesh of the Boeing rudimentary landing gear test case.

High-performance computing resources have been granted in this work by CINECA awards, under the ISCRA initiative, for a total of five research grants, in which the author was either Principal Investigator or Collaborator:

- Iscra C HOSTREP, grant number HP10BEE6C4, titled "High order discontinuous Galerkin simulations of the turbulence over a rounded leading edge flat plate";
- Iscra C EDGE, grant number HP10CPSWZ2, titled 'Efficient high-order discontinuous Galerkin simulations of the transition and turbulent reattachment

over separated flows";

- Iscra C PAST, grant number HP10BMA1AP, titled "A posteriori analysis of a new tensorial subgrid viscosity approach for the simulation of turbulent flows";
- Iscra B HIHODG, grant number HP10CKYRYE, titiled "High performance implicit time integration schemes for discontinuous Galerkin solutions of incompressible Turbulent flows";
- Iscra B TORTILLA, grant number HP10CE90VW, titled "Investigation of boundary conditions influence on LES accuracy".

Marche Polytecnic University, through the "Fondazione Cariverona", as well as the Department of Industrial Engineering and Mathematical Sciences is acknowledged for funding this work. The Department of Energy (United States), grant number DE-FG02-13ER26146/DE-SC0010341, is also acknowledged for co-funding the months spent at the University of Michigan.

Ancona, March 2019

Matteo Franciolini

Abstract

In recent years the increasing availability of High Performance Computing (HPC) resources strongly promoted the widespread of high fidelity simulations for industrial research and design. One of the most promising approaches to those kind of simulations is the discontinuous Galerkin (dG) discretization method. Those methods have been applied in the context of Large Eddy Simulation (LES) turbulence modelling approaches. Research on this topic is growing fast and several efforts focused on devising efficient time integration strategies, as well as parallelization algorithms, suited for massively parallel architectures.

The contribution of the thesis is three-fold. First, the work introduces an efficient hybrid MPI/OpenMP parallelisation paradigm to fruitfully exploit large HPC facilities. Second, it reports efficient, scalable and memory saving solution strategies for stiff dG discretisations. Third, it compares those solution strategies, for the first time using the same framework, to hybridizable discontinuous Galerkin (HDG) methods, including a novel implementation of a p-multigrid preconditioning approach, on unsteady flow problems involving the solution of the Navier–Stokes equations.

The improvements in computational efficiency have been evaluated on these cases of growing complexity involving large eddy simulations of turbulent flows. First, the Rayleigh-Bénard convection problem and the turbulent channel flow at moderately high Reynolds numbers is presented. Being the nature of those problems not particularly stiff, the solution strategies proposed result up to five times faster than standard matrix-based methods while allocating the 7% of the memory. A second family of test cases involve the LES simulation of a rounded leading edge flat plate under different levels of free-stream turbulence. Although the increased stiffness of the iteration matrix due to the use of curved and stretched elements, the solver resulted more than three times faster while allocating the 15% of the memory if compared to standard methods. As a final validation of the methods proposed on an industry-relevant test case, the large eddy simulation of the Boeing Rudimentary Landing Gear at $Re = 10^6$, is reported. In all the cases, a remarkable agreement with experimental data as well as previous numerical simulations is documented.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years the increasing availability of High Performance Computing (HPC) resources strongly promoted the widespread of high fidelity simulations for industrial research and design. One of the most promising approaches to those kind of simulations is the discontinuous Galerkin (dG) discretization method. dG methods have been successfully applied to the simulation of turbulent flows by solving the incompressible [1, 2, 3] and compressible [4, 5] Reynolds Averaged Navier–Stokes (RANS) equations. More recently, they have been also applied in the context of Large Eddy Simulation (LES) turbulence modelling approaches. The high potential of dG approximations for the under-resolved simulation of turbulent flows has been demonstrated in the literature for those moderate Reynolds numbers conditions where Reynolds-averaged Navier–Stokes (RANS) approaches are known to fall short, *e.g.*, massively separated flows [6, 7, 8, 9, 10, 11, 12].

The application of dG methods for DNS and LES is appealing for several reasons. Those methods can easily achieve high-order accuracy with a great geometrical flexibility and are perfectly suited to *hp*-adaptation techniques, as well as to parallel computing, due to the compact stencil of the space discretization. Moreover, dG methods show numerical properties suitable to the Implicit LES (ILES) of turbulent flows [13]. In fact, the dissipation of the numerical scheme behaves like a spectral cut-off filter, which mimics the role of subgrid-scale (SGS) models proper of classical LES approaches. However, advantages in favor of this strategy over classical SGS methods exist: having a spectral dissipation confined on the highest wavenumbers at high orders of accuracy, ILES allows for an accurate simulation of laminar-to-turbulent transitional flows where explicit SGS modeling using constant viscosity are known to fail, and variable-viscosity SGS models are typically tuned using ad-hoc prescriptions [14]. In addition, it has been also proved that using high order DG discretizations it is possible to achieve the same spatial resolution with a reduced number of degrees of freedom (DoF) than standard methods [7, 6].

Research on this topic is growing fast and several efforts focused on devising efficient time integration strategies, as well as parallelization algorithms, suited for massively parallel architectures. Indeed, the inherently unsteady nature of LES/ILES and the need to reduce time-to-results pose serious challenges to the achievement of cost effective scale-resolving computations and the ability to fruitfully exploit large computational facilities. In this context high-order implicit time integration schemes are attractive to overcome the strict stability limits of explicit methods (which scale

as $\Delta t \sim h/k^2$, where h is the element size and k is the polynomial order [15]) when dealing with high-degree polynomial approximations, see for example [1, 8, 16]. However, the development of an efficient solution and time integration strategy is still subject of active research. In fact, implicit methods necessarily entail the solution of large non-linear/linear systems of equations [17, 16, 18]. Such large systems are characterised by a sparse iteration matrix that requires a large CPU time and memory to be evaluated and stored, and need to be solved with an iterative solver. To overcome this limitation, several previous studies proposed a matrix-free implementation of the iterative solution strategy. This approach does not need to store the iteration matrix, and its use appear to gain interests from the research community. However, even in a matrix-free context, the preconditioner still needs the evaluation of the Jacobian as well as its storage, and thus the memory footprint can be fairly high when using standard methods such as the incomplete lower-upper factorization with zero filling, ILU(0), of the iteration matrix. For this reason the development of an efficient as well as memory saving preconditioner strategy is mandatory to reduce the overall memory footprint of the application. With the idea of applying those algorithms on large HPC facilities, the strategy has to be scalable, *i.e.* the number of iterations to reach convergence does not change by parallelising the computation using *message passing interface* (MPI) libraries. This is not the case of state-of-the-art preconditioners, which degrade the performances when domain decomposition approaches are employed.

Another interesting aspect is that current trends in HPC facilities promote the use of alternative ways to MPI to fruitfully exploit new computational nodes. For example, the development of many integrated cores (MIC) such as the Xeon Phi. It is typically retained that *open multi-processing* paradigms (OpenMP) allows a performance improvement on those platforms: in fact, the use of MPI within a node can be not optimal since the shared memory can be exploited to reduce the overhead cost of the communications. For this reason, a hybrid MPI/OpenMP implementation is usually suggested as a viable way to improve the parallel efficiency of codes running on clusters of multi-core nodes.

Bearing that in mind, the main objectives of the work are three-fold. First, he thesis addresses the parallelization of the explicit-in-time discontinuous Galerkin solver on new HPC architectures by proposing a novel hybrid MPI/OpenMP paradigm. Second, the thesis aims at the development of efficient, memory saving and scalable solvers for solving linear systems arising from implicit high-order discontinuous Galerkin discretizations applied to scale-resolving simulations. Third, the thesis compares the novel solution strategies in the context of novel space discretizations such as the hybridizable discontinuous Galerkin method.

1.2 Thesis outline

The thesis is organized as follows. Chapter 2 reports the description of the governing partial differential equations as well as the details of the space and time discretizations.

Chapter 3 reports on the hybrid MPI/OpenMP parallelization of the explicit version of the discontinuous Galerkin solver for compressible flows. Target applications

considered linear Euler as well as Euler equations, and the compressible Navier–Stokes equations. The aim is to increase the scalability of the solver on large HPC facilities reducing the amount of intra-node communications by exploiting the shared memory context. To this end, three pure OpenMP parallelization strategies are proposed and compared on different machine architectures and using different compilers. To avoid data races, the first strategy implements a colouring algorithm for the mesh element faces; the second one mimics a pure MPI implementation, while the third method is somehow half way between the previous two. While the parallel efficiency is affected by machines and compilers employed, some general indications and trends are provided. In particular, it is observed that thanks to the compactness of the discretization, all the three strategies perform quite satisfactory and show optimal parallel efficiencies on all the architectures. More importantly, the hybrid MPI/OpenMP parallel performance of the method is also assessed on a turbulent flow computation on three different HPC facilities using the simplest OpenMP strategy considered, *i.e.* the colouring algorithm, which avoids two nested partitioning of the domain. It is demonstrated that such implementation provides high parallel efficiencies on large HPC facilities consisting of multi- and many-core nodes. This is particularly true when the set of NS equations is considered. In fact, the discretization of the viscous terms involve more floating point operations in the loop over faces than EE/LEE. The hybrid approach also showed a clear gain in parallel performance over pure MPI when the CPU is able to handle more than one hardware threads per core.

Chapter 4 introduces the implementation of a GMRES matrix-free framework and provides insights on its computational efficiency. To this end, several solver settings such as the time step size, the relative tolerance for the linear system solution, the lagging of the Jacobian evaluation, the mesh density, the polynomial order and the parallel efficiency are considered. The method is assessed through the solution of a model test case for the incompressible Navier–Stokes equations, *i.e.* the laminar travelling waves. The results prove that, on top of the memory saving, a matrix-free iteration approach performs comparably to a matrix-based one in terms of number of iterations, and shows a competitive CPU time when the computational complexity increases, for example using high-order polynomials. In this chapter, since the space discretization was not considerably stiff, standard preconditioning strategies such as ILU(0) and element-wise block-Jacobi are employed for the iterative solver.

Chapter 5 extends the matrix-free framework to more powerful preconditioning strategies able to deal with stiffer space discretizations. To this end, a multilevel preconditioning strategy based on lower-order discretization of the problem is proposed. The strategy combines a matrix-free implementation of the smoother and the use of memory saving element-wise block-Jacobi preconditioners on the finest level to obtain optimal algorithmic scalability, large speed-ups in CPU time and a considerable reduction in the overall memory footprint of the application. An aspect of novelty consists on the use of a rescaled-inherited approach, as opposed to the standard inheritance of the polynomial spaces to obtain the coarse order matrix operators. The approach reduces the amount of penalization on the coarse levels of the discretization according to the scaling of the stabilization term and improves the convergence rates of the iterative solution strategy. The solution two stiff problems is presented. First,

the incompressible, two-dimensional flow around a circular cylinder; second, the incompressible, three-dimensional laminar flow around a sphere. Significant benefits of the approach respect to standard methods in terms of number of iterations, execution time of the application and memory footprint are documented.

Chapter 6 focuses on comparing the efficiency of the strategies proposed in Chapter 5 to those obtained using novel space discretizations, *i.e.* the hybridizable discontinuous Galerkin (HDG) method in the context of compressible flow simulations. The aim here is both to introduce efficient and scalable preconditioning strategies for this class of discretization methods using multilevel strategies based on coarse-order discretizations of the problem, as well as comparing within a single framework the performance with DG. In HDG the globally coupled degrees of freedom are only those involving faces, while the element-interior degrees of freedom are statically condensed out of the system. For this reason, standard-inheritance would require a larger number of operations if compared to DG. With the idea of minimising the costs of the projections, an approximate-inherited approach is introduced to increase the computational efficiency. It is worth mentioning that this kind of application have never been reported in the literature, as p-multigrid is still an open problem for HDG. The approach is validated in two-dimensional laminar compressible flows of growing complexity. The results show that the strategy is able to reduce considerably the number of iterations if compared to standard methods, but the costs of the static condensation and back-solve which can still be pretty large at high orders prevent this gain to be reflected on the execution time of the application. Part of the scope of the chapter is also to compare HDG with DG in terms of accuracy and efficiency. In particular, the memory-saving strategy introduced in 5 is considered for the comparison. It is observed that, while HDG works with significantly less degrees of freedom, the memory footprint of the solvers are comparable, but the executions time are still in favour to the DG based matrix-free multigrid solver.

Chapter 7 presents applications of the solution strategies hereby developed on incompressible turbulent flows ranging from academic to industry-relevant test cases: the turbulent channel flow up to $Re_\tau = 950$, the Rayleigh–Bénard natural convection up to $Ra = 10^8$, the turbulent flow on a rounded-leading edge flat plate, namely the T3L test case of the ERCOFTAC test case suite under different Reynolds numbers and levels of free-stream turbulence. Finally, preliminary results of the under-resolved DNS of the Boeing Rudimentaring Landing Gear test case at $Re = 10^6$ is presented. A discussion on the physical accuracy of the results will precede insights on the computational efficiency of the solvers, demonstrating reliability and the accuracy of the proposed solution strategies as well as considerable speed-up values if compared to state-of-the-art solution methods.

Conclusions and final remarks are given in Chapter 8.

Chapter 2

The numerical framework

The thesis considers applications of different types involving different flow models and complexity: the compressible Navier–Stokes equations (CNS), Euler equations (EE), Linearized Euler equations (LEE), incompressible Navier–Stokes equations (INS), and INS with Boussinesq approximation of buoyancy effects (BINS). Those set of PDEs are discretized in space and time using the discontinuous Galerkin method coupled with explicit and implicit high order Runge–Kutta schemes. The aim of the chapter is to describe the numerical discretization features and to provide appropriate background on the implementation details.

2.1 Governing PDEs

The general form of the partial differential equations (PDEs) considered in this work can be written as

$$\partial_t \boldsymbol{w} + \nabla \cdot (\boldsymbol{F}^c - \boldsymbol{F}^\nu) + \boldsymbol{s} = 0, \tag{2.1}$$

where $\boldsymbol{w} \in \mathbb{R}^m$ is the vector of working variables, with m equal to the number of variables, $\boldsymbol{F}^c, \boldsymbol{F}^\nu \in \mathbb{R}^d \times \mathbb{R}^m$ are the convective and diffusive flux functions, and $\boldsymbol{s} \in \mathbb{R}^m$ is the source term of the governing equations. Note that the fluxes $\boldsymbol{F}^c, \boldsymbol{F}^\nu$ are function of the state vector $\boldsymbol{w}$ and gradient vector $\nabla \boldsymbol{w}$, while the source term is case-specific. The definition of the working variables, fluxes and source terms is provided in the remaining of the section. In what follows, the quantities are given in non-dimensional form using reference values of length (L_r), velocity (U_r), pressure (p_r), density (ρ_r), temperature (θ_r), dynamic viscosity (μ_r), and thermal conductivity (λ_r).

Compressible Navier–Stokes equations

The set of compressible Navier–Stokes (NS) equations are obtained as

$$\begin{aligned} \boldsymbol{w} &= \begin{pmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho e_0 \end{pmatrix}, \quad \boldsymbol{F}^c = \begin{pmatrix} \rho \boldsymbol{u} \\ \rho \boldsymbol{u} \otimes \boldsymbol{u} + (p/\gamma M^2) \boldsymbol{I} \\ \boldsymbol{u}(\rho e_0 + (\gamma - 1)p) \end{pmatrix}, \\ \boldsymbol{F}^\nu &= \begin{pmatrix} \boldsymbol{0} \\ (1/Re)\boldsymbol{\Pi} \\ (\gamma(\gamma-1)M^2/Re)\boldsymbol{\Pi} \cdot \boldsymbol{u} - (\gamma/PrRe)\boldsymbol{q} \end{pmatrix}, \end{aligned} \tag{2.2}$$

with $\boldsymbol{u} \in \mathbb{R}^d$ and d the number of space dimensions. The non-dimensional groups M, Re and Pr are the Mach number, the Reynolds number and the Prandtl number

defined such that

$$M = \frac{U_r}{\sqrt{\gamma\theta}}, \quad Re = \frac{\rho_r U_r L_r}{\mu_r}, \quad Pr = \frac{\mu_r c_p}{\lambda_r}$$

while $e_0, p \in \mathbb{R}$ are the total energy and pressure, while $\boldsymbol{\Pi} \in \mathbb{R}^{d\times d}$ is the total stress tensor and $\boldsymbol{q} \in \mathbb{R}^d$ is the heat flux vector. Those quantities are given by

$$e_0 = e + (\boldsymbol{u}\cdot\boldsymbol{u})/2, \qquad e = c_v\theta, \qquad p = \rho\theta,$$
$$\boldsymbol{\Pi} = 2\mu\left(\boldsymbol{D} - \frac{1}{3}(\nabla\cdot\boldsymbol{u})\boldsymbol{I}\right), \quad \boldsymbol{q} = -\lambda\nabla\theta, \quad \boldsymbol{D} = \frac{1}{2}\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right).$$

Here e is the internal energy, θ is the temperature, $\gamma = c_p/c_v$ is the ratio of gas specific heats, μ is the non-dimensional viscosity, λ is the non-dimensional conductivity, and $\boldsymbol{D}$ is the non-dimensional mean strain-rate tensor.

Euler equations

Euler equations set is easily obtained from (2.2) by zeroing out the diffusive terms. Therefore,

$$\boldsymbol{w} = \begin{pmatrix} \rho \\ \rho\boldsymbol{u} \\ \rho e_0 \end{pmatrix}, \quad \boldsymbol{F}^c = \begin{pmatrix} \rho\boldsymbol{u} \\ \rho\boldsymbol{u}\otimes\boldsymbol{u} + (p/\gamma M^2)\boldsymbol{I} \\ \boldsymbol{u}(\rho e_0 + (\gamma-1)p) \end{pmatrix}. \tag{2.3}$$

Linearized Euler equations

Linearized Euler equations consider small perturbations of the flow field with respect to a non-uniform, average state and are typically employed for acoustic propagation studies. The working variables are therefore perturbation quantities, while the fluxes are obtained from (2.3) by linearising around a mean flow solution $\overline{\rho}, \overline{\boldsymbol{u}}, \overline{p}$. In the framework employed, the pressure is used instead of the total energy. The set reads as follows:

$$\boldsymbol{w} = \begin{pmatrix} \rho \\ \boldsymbol{u} \\ p \end{pmatrix}, \quad \boldsymbol{F}^c = \begin{pmatrix} \overline{\boldsymbol{u}}\rho + \overline{\rho}\boldsymbol{u} \\ \overline{\boldsymbol{u}}\otimes\boldsymbol{u} + p/(\overline{\rho}\gamma M^2)\boldsymbol{I} \\ \overline{\boldsymbol{u}}p + \gamma\overline{p}\boldsymbol{u} \end{pmatrix}, \tag{2.4}$$

The source term $\boldsymbol{s}$, which represents the effects of the non-uniform mean flow on the perturbation fields, is considered zero for uniform average flow fields.

Incompressible Navier–Stokes equations

The complete set of the Incompressible Navier–Stokes (INS) equations can be written in the following non-dimensional form

$$\boldsymbol{w} = \begin{pmatrix} 0 \\ \boldsymbol{u} \end{pmatrix}, \quad \boldsymbol{F}^c = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{u}\otimes\boldsymbol{u} + p\boldsymbol{I} \end{pmatrix}, \quad \boldsymbol{F}^\nu = \begin{pmatrix} \boldsymbol{0} \\ (1/Re)\nabla\boldsymbol{u} \end{pmatrix}, \tag{2.5}$$

Here, $p = P/\rho$ is the pressure divided by the constant density and Re is the Reynolds number defined as for Eq. (2.2). Note that in the incompressible case the primitive variables $(p, \boldsymbol{u})^T$ are used to solve the equations. However, the state vector is a modified vector since it shows zero instead of the pressure variable.

Incompressible Navier–Stokes with Boussinesq approximation

If the natural convection effects have to be considered, for example in the Rayleigh–Bérnard convection (RBC) problem, the INS equations are properly modified under the Boussinesq approximation and written in non-dimensional form using the following vectors

$$\begin{aligned} \boldsymbol{w} &= \begin{pmatrix} 0 \\ \boldsymbol{u} \\ \theta \end{pmatrix}, \quad \boldsymbol{F}^c = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} \\ \boldsymbol{u}\theta \end{pmatrix}, \\ \boldsymbol{F}^\nu &= \begin{pmatrix} \boldsymbol{0} \\ (\sqrt{Pr/Ra})\nabla \boldsymbol{u} \\ (1/\sqrt{RaPr})\nabla\theta \end{pmatrix}, \quad \boldsymbol{s} = \begin{pmatrix} 0 \\ \theta \boldsymbol{n} \\ 0 \end{pmatrix}, \end{aligned} \tag{2.6}$$

where θ is the temperature and $\boldsymbol{n}$ is the gravitational acceleration direction. Equations (2.6) are non-dimensionalized using a reference length L_r, a representative temperature difference $\Delta\Theta$, a reference "free-fall" velocity $U_R = \sqrt{g\alpha\Delta\Theta L_r}$, where g is the gravitational acceleration and β the thermal expansion coefficient. The Rayleigh number is $Ra = g\beta\Delta\Theta L_r^3/\nu\alpha$, where $\nu = \mu/\rho$ is the kinematic viscosity and $\alpha = \lambda/(\rho c_p)$ is the thermal diffusivity, while Pr is defined as for Eq. (2.2).

2.2 The dG space discretization

The discontinuous Galerkin (dG) method considers the weak formulation of Equation (2.1). Rewriting the system in integral form,

$$\int_\Omega \partial_t \boldsymbol{w} + \int_\Omega \nabla \cdot (\boldsymbol{F}^c - \boldsymbol{F}^\nu) + \int_\Omega \boldsymbol{s} = \boldsymbol{0}. \tag{2.7}$$

the weak formulation of the equations can be obtained by multiplying Equation (2.7) by an arbitrary test function $\boldsymbol{z} \in \mathbb{R}^m$ and integrate over the domain Ω. Integrating by parts the divergence terms, the weak form of equation (2.7) reads: find $\boldsymbol{u} \in \mathbb{R}^m$ such that

$$\begin{aligned} &\int_\Omega \partial_t \boldsymbol{w} \cdot \boldsymbol{z} - \int_\Omega (\boldsymbol{F}^c - \boldsymbol{F}^\nu) : \nabla \boldsymbol{z} \\ &+ \int_{\partial\Omega} \boldsymbol{n} \cdot (\boldsymbol{F}^c - \boldsymbol{F}^\nu) \cdot [\![\boldsymbol{z}]\!] + \int_\Omega \boldsymbol{s} \cdot \boldsymbol{z} = \boldsymbol{0}, \end{aligned} \tag{2.8}$$

$\forall \boldsymbol{z} \in \mathbb{R}^m$. In Eq. (2.8), $\partial\Omega$ is the domain boundary and $\boldsymbol{n}$ is the normal pointing outside the domain.

In order to build a dG discretization of equation (2.8), an approximation Ω_h of Ω, that is the collection of disjoint mesh elements $\kappa \in \mathcal{T}_h$, such that $\bigcup_{\kappa\in\mathcal{T}_h} \overline{\kappa} = \Omega_h$, is considered. The number of non-overlapping elements is set as $n_e = \mathrm{card}(\mathcal{T}_h)$. The mesh skeleton $\mathcal{F}_h$ is the collection of mesh faces σ. Internal faces $\sigma \in \mathcal{F}_h^{\mathrm{i}}$ are defined as the intersection of the boundary of two neighboring elements: $\sigma = \partial\kappa \bigcap \partial\kappa'$ with $\kappa \neq \kappa'$. Boundary faces $\sigma \in \mathcal{F}_h^{\mathrm{b}}$ reads $\sigma = \partial\kappa \bigcap \partial\Omega_h$. Clearly $\mathcal{F}_h = \mathcal{F}_h^{\mathrm{i}} \cup \mathcal{F}_h^{\mathrm{b}}$.

Moreover, it is set

$$\Gamma_h^i = \bigcup_{\sigma \in \mathcal{F}_h^i} \sigma, \qquad \Gamma_h^b = \bigcup_{\sigma \in \mathcal{F}_h^b} \sigma, \qquad \Gamma_h = \Gamma_h^i \cup \Gamma_h^b. \tag{2.9}$$

where σ denotes the generic mesh element face.

Each component of the state vector is sought, for a time interval $t \in [0, t_F]$, in the so called *broken polynomial spaces* defined over $\mathcal{T}_h$

$$\mathbb{P}_d^k(\mathcal{T}_h) = \left\{ v_h \in L^2(\Omega_h) \mid \forall \kappa \in \mathcal{T}_h, v_h|_\kappa \in \mathbb{P}_d^k(\kappa) \right\} \tag{2.10}$$

where $\mathbb{P}_d^k(\kappa)$ is the space of polynomial functions in d variables and total degree k defined over κ. To overcome the ill-conditioning of elemental mass matrices for higher-order polynomials on high aspect ratio and curved elements, a hierarchical and orthogonal set of shape functions defined in physical space is chosen. This set is obtained by using a modified Gram-Schmidt procedure, considering as a starting point a set of monomial functions of the same degree k. Since no continuity requirements are enforced at inter-element boundaries, v_h admits two-valued traces on the partition of mesh skeleton $\mathcal{F}_h^{\mathrm{i}}$. Accordingly, average and jump operators over internal faces are introduced as

$$\textbf{Average} : \{\{v_h\}\} = \frac{1}{2}\left(v_h|_\kappa + v_h|_{\kappa'}\right), \qquad \textbf{Jump} : [\![v_h]\!] = \left(v_h|_\kappa - v_h|_{\kappa'}\right). \tag{2.11}$$

Those definition can be properly extended at the domain boundaries to account for the weak imposition of boundary conditions of the problem.

Accounting for the considerations above, the discrete counterpart of Equation (2.8) reads: find $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such that, for all $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$,

$$\begin{aligned} &\sum_{\kappa \in \mathcal{T}_h} \int_\kappa \partial_t \boldsymbol{w}_h \cdot \boldsymbol{z}_h - \sum_{\kappa \in \mathcal{T}_h} \int_\kappa \left(\boldsymbol{F}_h^c - \widetilde{\boldsymbol{F}}_h^\nu \right) : \nabla_h \boldsymbol{z}_h \\ &+ \sum_{\sigma \in \mathcal{F}_h} \int_\sigma \boldsymbol{n}^\sigma \cdot \left(\widehat{\boldsymbol{F}}_h^c - \widehat{\boldsymbol{F}}_h^\nu \right) \cdot [\![\boldsymbol{z}_h]\!] + \sum_{\kappa \in \mathcal{T}_h} \int_\kappa \boldsymbol{s}_h \cdot \boldsymbol{z}_h = \boldsymbol{0}, \end{aligned} \tag{2.12}$$

where $[\![\boldsymbol{z}_h]\!] = [\![v_{h,i}]\!]\, \boldsymbol{e}_i$, with $\boldsymbol{e}_i$ the unity vector on direction x_i, and $\boldsymbol{n}^\sigma$ is the normal vector with respect to σ. While obtaining (2.12) from (2.1) follows a standard finite-element practice, which involve the element-by-element integration by parts after having multiplied by a suitable test function, the dG method hinges on the definition of suitable numerical fluxes $\widehat{\boldsymbol{F}}_h^c$ and $\widetilde{\boldsymbol{F}}_h^\nu$, $\widehat{\boldsymbol{F}}_h^\nu$ to ensure stability, consistency, local conservation and inter-element coupling of the method.

Consistent and stable inviscid numerical fluxes $\widehat{\boldsymbol{F}}_h^c$ can be obtained though the appropriate solution of a Riemann problem. In this case, either the exact solver, the Van Leer's or the Roe's approximate Riemann solvers are employed for the compressible case [19, 20, 21]. The incompressible Navier–Stokes equations are handled in a similar fashion through the exact solution of local Riemann problems based on an artificial compressibility perturbation of the equations as proposed in [22, 1]. It is worth pointing out that the artificial compressibility term involve the computation of the flux function only, and the approach is fully consistent with the incompressible

Navier–Stokes equations when a time accurate computation is performed.

As regards the viscous numerical fluxes, a consistent gradient is introduced according to the first form of the Bassi and Rebay scheme (BR1) proposed in [23]. For all $\boldsymbol{\pi}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{d\times m}$, $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, the consistent gradient $\boldsymbol{\tau}_h(\boldsymbol{w}_h)$ is defined such that

$$\begin{aligned}\int_\Omega (\boldsymbol{\tau}_h(\boldsymbol{w}_h) - \nabla \boldsymbol{w}_h) : \boldsymbol{\pi}_h = -\sum_{\sigma\in\mathcal{F}_h} \int_\sigma [\![\boldsymbol{w}_h]\!] \cdot \{\{\boldsymbol{\pi}_h\}\} \cdot \boldsymbol{n}^\sigma \\ := \sum_{\sigma\in\mathcal{F}_h} \int_\Omega \boldsymbol{r}^\sigma([\![\boldsymbol{w}_h]\!]) : \boldsymbol{\pi}_h = \sum_{\kappa\in\mathcal{T}_h} \int_\kappa \boldsymbol{R}^\kappa(\boldsymbol{w}_h) : \boldsymbol{\pi}_h\end{aligned} \tag{2.13}$$

where $\boldsymbol{r}^\sigma([\![\boldsymbol{w}_h]\!]) : [\mathbb{P}_d^k(\sigma)]^m \to [\mathbb{P}_d^k(\mathcal{T}_h)]^{d\times m}$ is the local lifting operator, and

$$\boldsymbol{R}^\kappa(\boldsymbol{w}_h) := \sum_{\sigma\in\mathcal{F}_{\partial\kappa}} \boldsymbol{r}^\sigma([\![\boldsymbol{w}_h]\!]) \tag{2.14}$$

is the elemental lifting operator, with $\mathcal{F}_{\partial\kappa}$ is the set of faces belonging to κ. In this work the implementation of the viscous flux functions rely on the second form of the Bassi and Rebay scheme (BR2), introduced to reduce the stencil of the BR1 discretization and analyzed in the context of the Poisson problem by [24] and [25]. The BR2 viscous fluxes are functions of elemental spatial derivatives corrected by suitable lifting operator contributions

$$\begin{aligned}\widetilde{\boldsymbol{F}}_h^\nu &= \boldsymbol{F}^\nu\left(\boldsymbol{w}_h, \nabla_h \boldsymbol{w}_h - \boldsymbol{R}^\kappa(\boldsymbol{w}_h)\right), \\ \widehat{\boldsymbol{F}}_h^\nu &= \{\{\boldsymbol{F}^\nu\left(\boldsymbol{w}_h, \nabla_h \boldsymbol{w}_h - \eta_\sigma \boldsymbol{r}^\sigma([\![\boldsymbol{w}_h]\!])\right)\}\},\end{aligned} \tag{2.15}$$

$\widehat{\boldsymbol{F}}^\nu$ ensures consistency and stability of the scheme and $\widetilde{\boldsymbol{F}}^\nu$ guarantees the symmetry of the formulation. As proved by Brezzi et al. [24], coercivity for the BR2 discretization of the Laplace equation holds provided that η_σ is greater than the maximum number of faces of the elements sharing σ.

As regards the boundary conditions of the problem on the boundary faces $\sigma \in \mathcal{F}_h^{\mathrm{b}}$, a ghost boundary state $\boldsymbol{w}_b$ having support on the interface $\sigma \in \mathcal{F}_h^{\mathrm{b}}$ of a ghost neighboring elements κ_g is properly define. The ghost boundary imposes the conservation of Riemann invariants based on the hyperbolic nature of the governing partial differential equations. Note that in the incompressible case, the hyperbolic nature of the system is recovered by the artificial compressibility flux approach. Accordingly, both the internal state $\boldsymbol{w}$ and the boundary data are involved in the definition of flux functions on the boundary faces of the computational domain. More detailed information on boundary conditions are beyond the scope of this chapter and can be found in Ref. [26, 4, 22] for the EE/LEE, CNS and INS cases.

2.3 Time integration

The accurate time integration of the spatially dG discretized PDEs can be presented in compact form by recalling the state vector $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ and identifying the

unknown vector at time t_n with $\boldsymbol{w}_h^n$. Moreover, the flux functions are introduced as

$$\widetilde{\boldsymbol{F}}_h(\boldsymbol{w}_h) \stackrel{\text{def}}{=} \left(\boldsymbol{F}_h^c - \widetilde{\boldsymbol{F}}_h^\nu\right) \in \mathbb{R}^d \otimes \mathbb{R}^m \tag{2.16}$$

$$\widehat{\boldsymbol{F}}_h(\boldsymbol{w}_h) \stackrel{\text{def}}{=} \left(\widehat{\boldsymbol{F}}_h^c - \widehat{\boldsymbol{F}}_h^\nu\right) \in \mathbb{R}^d \otimes \mathbb{R}^m, \tag{2.17}$$

collecting the viscous and inviscid flux contributions. For all $\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, the residual of the dG spatial discretization in (2.12) is defined as follows

$$\begin{aligned} f_h(\boldsymbol{w}_h, \boldsymbol{z}_h) = &- \sum_{\kappa\in\mathcal{T}_h} \int_\kappa \sum_{i=1}^m \sum_{p=1}^d \widetilde{F}_{p,i}(\boldsymbol{w}_h)\, \frac{\partial z_i}{\partial x_p} \\ &+ \sum_{\sigma\in\mathcal{F}_h} \int_\sigma \sum_{i=1}^m \sum_{p=1}^d n_p^\sigma\, \widehat{F}_{p,i}(\boldsymbol{w}_h)\, [\![z_i]\!]\,, \end{aligned} \tag{2.18}$$

where the mesh step size subscript when working in index notation has been dropped. For all $\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, the Jacobian of the residual reads

$$j_h(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) = \frac{\partial f_h(\boldsymbol{w}_h, \boldsymbol{z}_h)}{\partial \boldsymbol{w}_h} \delta\boldsymbol{w}_h. \tag{2.19}$$

In particular the gradient-free $j_h^{\not\nabla}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h)$ part of the operator, as well as the gradient $j_h^{\nabla}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h)$ contribution, are splitted as

$$\begin{aligned} j_h^{\not\nabla}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) = &- \sum_{\kappa\in\mathcal{T}_h} \int_\kappa \sum_{i,j=1}^m \sum_{p=1}^d \frac{\partial \widetilde{F}_{p,i}(\boldsymbol{w}_h)}{\partial w_j}\, \delta w_j\, \frac{\partial z_i}{\partial x_p} \\ &+ \sum_{\sigma\in\mathcal{F}_h} \int_\sigma \sum_{i,j=1}^m \sum_{p=1}^d n_p^\sigma \frac{\partial \widehat{F}_{p,i}(\boldsymbol{w}_h)}{\partial w_j}\, \delta w_j\, [\![z_i]\!]\,, \end{aligned} \tag{2.20}$$

$$\begin{aligned} &j_h^{\nabla}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) = \\ &- \sum_{\kappa\in\mathcal{T}_h} \int_\kappa \sum_{i,j=1}^m \sum_{p,q=1}^d \frac{\partial \widetilde{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - R_q^\kappa(w_j)\Big)} \left(\frac{\partial(\delta w_j)}{\partial x_q} - R_q^\kappa(\delta w_j)\right) \frac{\partial z_i}{\partial x_p} + \\ &+ \sum_{\sigma\in\mathcal{F}_h} \int_\sigma \sum_{i,j=1}^m \sum_{p,q=1}^d n_p^\sigma \frac{\partial \widehat{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - \eta_\sigma r_q^\sigma(w_j)\Big)} \left\{\!\!\left\{ \frac{\partial(\delta w_j)}{\partial x_q} - \eta_\sigma\, r_q^\sigma\left([\![\delta w_j]\!]\right) \right\}\!\!\right\} [\![z_i]\!]\,. \end{aligned} \tag{2.21}$$

Prior to introducing the formulation for the temporal discretization the following mass bilinear form is defined: for all $\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$

$$m_h(\boldsymbol{w}_h, \boldsymbol{z}_h) = \sum_{\kappa\in\mathcal{T}_h} \int_\kappa \sum_{i=1}^d w_i\, z_i. \tag{2.22}$$

Consequently, the unsteady problem can be re-written as: find $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such

Algorithm 1 Explicit low-storage Runge-Kutta method

set $n_F = \dfrac{t_F}{\delta t}$
do $n = 0, n_F$
  set $\delta \boldsymbol{w}_h^0 = \boldsymbol{w}_h^n$, $\boldsymbol{w}_h^{n+1} = \boldsymbol{w}_h^n$
  do $s = 1, n_s$
    find $\delta \boldsymbol{q}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such that, for all $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$

$$\frac{1}{\delta t} m_h(\delta \boldsymbol{q}_h, \boldsymbol{z}_h) = -f_h(\delta \boldsymbol{w}_h^{s-1}, \boldsymbol{z}_h) \tag{2.24}$$

    set $\delta \boldsymbol{w}_h^s = \delta \boldsymbol{w}_h^0 + \mathsf{a}_s \, \delta \boldsymbol{q}_h$
    do $o = 1, s-1$
      find $\delta \boldsymbol{q}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such that, for all $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$

$$\frac{1}{\delta t} m_h(\delta \boldsymbol{q}_h, \boldsymbol{z}_h) = -f_h(\delta \boldsymbol{w}_h^{o-1}, \boldsymbol{z}_h) \tag{2.25}$$

      update $\delta \boldsymbol{w}_h^s$ += $\mathsf{b}_o \, \delta \boldsymbol{q}_h$
    enddo
  enddo
  do $s = 1, n_s$
    find $\delta \boldsymbol{q}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such that, for all $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$

$$\frac{1}{\delta t} m_h(\delta \boldsymbol{q}_h, \boldsymbol{z}_h) = -f_h(\delta \boldsymbol{w}_h^{s}, \boldsymbol{z}_h) \tag{2.26}$$

    update $\boldsymbol{w}_h^{n+1}$ += $\mathsf{b}_s \delta \boldsymbol{q}_h$
  enddo
enddo

that

$$m_h(\partial_t \boldsymbol{w}_h, \boldsymbol{z}_h) + f_h(\boldsymbol{w}_h, \boldsymbol{z}_h) = 0 \tag{2.23}$$

for all $\boldsymbol{z}_h \in [\mathbb{P}^k(\mathcal{T}_h)]^m$. Appropriate time discretization schemes are applied to Eq. (2.23). Numerical integration of Eq. (2.23) is performed by means of suitable Gauss quadrature rules. Cheap non-product formulae taken from [27] are typically preferred to tensor-product ones for computational efficiency. Numerical integration calls for the evaluation of shape functions at quadrature points.

2.3.1 Explicit Runge–Kutta schemes

To integrate in time Eq. (2.23) given the initial condition $\boldsymbol{w}_h^0 = \boldsymbol{w}_h(t=0) \in [\mathbb{P}^k(\mathcal{T}_h)]^m$, the sequence $\boldsymbol{w}_h^{n+1}$ is defined iteratively by means of the explicit multi-stage Runge–Kutta scheme as described in Algorithm 1, where a_i, b_i are real, scheme-specific coefficients, and $\delta \boldsymbol{w}_h^s$, with $s = \{1, \cdots, n_s\}$, the solutions at each stage of the scheme that are properly combined to compute the solution $\boldsymbol{w}_h^{n+1}$ at the next time level. In this work the time integration scheme employed is the five stage, order four, Runge–Kutta scheme of [28]. This accurate, low-storage, scheme requires only two levels of registers for the data storage, and it is thus very well suited for a memory-lean implementation of a DG method. Defining the mass matrix operator as

$$(\boldsymbol{M}_h \, \delta \boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} = m_h(\delta \boldsymbol{w}_h, \boldsymbol{z}_h) \qquad \forall \, \delta \boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m, \tag{2.27}$$

it is clear that Equations (2.24)-(2.26) calls for the solution of the following linear system

$$\boldsymbol{M}_h \; \delta \boldsymbol{w}_h = \boldsymbol{g}_h \tag{2.28}$$

where $\delta \boldsymbol{w}_h, \boldsymbol{g}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ are the unknown polynomial function and the right-hand side arising from the Runge-Kutta time discretization, respectively. Fortunately, the solution of Eq. (2.28) does not require a global matrix allocation. In fact, the mass matrix arising from the DG discretization of the governing equations is block diagonal thanks to the use of generic hierarchical basis function, *i.e.* in the case of the EE or LEE problems considered herein. This means that it can be locally inverted once using a lower-upper (LU) factorization that can be stored and applied cheaply at every stage of the Runge–Kutta scheme. Differently, for viscous test cases orthonormal basis functions are employed, which make the mass matrix equal to the identity matrix. In this case, the application of the $\boldsymbol{M}_h^{-1}$ matrix is no longer required.

2.3.2 Implicit Runge–Kutta schemes

Despite being high order accurate and computationally cheap, explicit time integration methods show strict stability limits especially using high-order space discretizations. The reason to consider implicit schemes is twofold. First, implicit schemes allow to overcome time step limitations. Several previous studies demonstrated how high order implicit schemes, despite being more memory consuming, allow to increase the computational efficiency of unsteady flow simulations for generally stiff problems involving unstructured meshes. On the other hand, some equation sets such as INS do not allow a straightforward implementation of explicit schemes, since the incompressibility constrain has to be solved in a semi-explicit fashion using velocity correction schemes. Two families of implicit schemes will be hereby considered. The first one involve the multi-stage linearly-implicit Rosenbrock-type Runge–Kutta methods, which involve the solution of one linear system per each stage. A second family is the multi-stage explicit first stage singly diagonally-implicit Runge–Kutta methods, which require to solve a non-linear system per stage. Details of the schemes are given in the remaining of the section.

Linearly-implicit Rosenbrock-type Runge–Kutta schemes

The multi-stage linearly implicit Rosenbrock-type Runge-Kutta method requires the solution of a linear system at each stage $s = \{1, \cdots, n_s\}$. The most appealing feature of the method is that the iteration matrix is the same per each stage, and therefore the Jacobian needs to be assembled only once per time step. This attractive feature allows a good compromise between solution accuracy at large time steps and computational efficiency [1, 16]. Given the initial condition $\boldsymbol{w}_h^0 = \boldsymbol{w}_h(t=0) \in [\mathbb{P}^k(\mathcal{T}_h)]^m$, a sequence $\boldsymbol{w}_h^{n+1}$ is iteratively defined by means of the Rosenbrock scheme as described in Algorithm 2, where γ, a_{ij}, c_{ij} and m_i are real, scheme-specific coefficients and $\delta \boldsymbol{w}_h^s$, with $s = \{1, \cdots, n_s\}$, the solutions at each stage of the scheme that are properly combined to compute the solution $\boldsymbol{w}_h^{n+1}$ at the next time slab. The Rosenbrock time marching strategy in Algorithm 2 advances the solution in time by repeatedly solving the linearized system of equations(2.29), once for each stage of the Runge-Kutta method.

Algorithm 2 Multi-stage linearly implicit (Rosenbrock-type) Runge-Kutta method

1: set $n_F = \dfrac{t_F}{\delta t}$
2: **do** $n = 0, n_F$
3: set $\boldsymbol{w}_h^{n+1} = \boldsymbol{w}_h^n$
4: **do** $s = 1, n_s$
5: set $\delta \boldsymbol{p}_h = \boldsymbol{0} \wedge \delta \boldsymbol{q}_h = \boldsymbol{0}$
6: **do** $o = 1, s-1$
7: $\delta \boldsymbol{p}_h \mathrel{+}= \mathsf{a}_{s,o}\, \delta \boldsymbol{w}_h^o$
8: $\delta \boldsymbol{q}_h \mathrel{+}= \mathsf{c}_{s,o}\, \delta \boldsymbol{w}_h^o$
9: **enddo**
10: find $\delta \boldsymbol{w}_h^s \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such that, for all $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$

$$\begin{aligned} &\frac{1}{\gamma \delta t} m_h(\delta \boldsymbol{w}_h^s, \boldsymbol{z}_h) + j_h(\boldsymbol{w}_h^n, \delta \boldsymbol{w}_h^s, \boldsymbol{z}_h) = \\ &- f_h(\boldsymbol{w}_h^n + \delta \boldsymbol{p}_h, \boldsymbol{z}_h) - \frac{1}{\delta t} m_h(\delta \boldsymbol{q}_h, \boldsymbol{z}_h) \end{aligned} \tag{2.29}$$

11: **enddo**
12: **do** $s = 1, n_s$
13: update $\boldsymbol{w}_h^{n+1} \mathrel{+}= \mathsf{m}_s \delta \boldsymbol{w}_h^s$
14: **enddo**
15: **enddo**

Introducing the Jacobian and mass matrix operators

$$\begin{aligned} (\boldsymbol{J}_h\, \delta \boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= j_h(\boldsymbol{w}_h, \delta \boldsymbol{w}_h, \boldsymbol{z}_h) \quad \forall\, \boldsymbol{w}_h, \delta \boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m, \\ (\boldsymbol{M}_h\, \delta \boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= m_h(\delta \boldsymbol{w}_h, \boldsymbol{z}_h) \quad \forall\, \delta \boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m, \end{aligned} \tag{2.30}$$

the equation system (2.29) can be compactly rewritten as follows:

$$\boldsymbol{G}_h\, \delta \boldsymbol{w}_h = \boldsymbol{g}_h \tag{2.31}$$

where $\boldsymbol{G}_h = (1/\gamma \delta t)\boldsymbol{M}_h + \boldsymbol{J}_h$ is the global matrix operator, and $\delta \boldsymbol{w}_h, \boldsymbol{g}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ are the unknown polynomial function and the right-hand side arising from the linearly-implicit Runge-Kutta time discretization, respectively. It is worth pointing out that, considering the incompressible equation sets, the mass matrix $\boldsymbol{M}_h$ is a modified mass matrix which zero entries corresponding to the pressure DoFs. The Jacobian matrix $\boldsymbol{J}_h$ can be assembled as n_e^2 block sparse matrix, while the rank of each block is mn_v. Thanks to the compactness of the DG discretization, the degrees of freedom of a generic element κ are only coupled with those of the neighbouring elements and therefore the number of nonzero blocks for each (block) row κ of the Jacobian is equal to the number of elements surrounding κ plus one. Thus, the memory footprint scales as $n_e(n_f+1)(mn_v)^2$, where n_f is the number of faces of the mesh elements. Note that, for a broken polynomial space, $n_v = \prod_{i=1}^{d}(k+i)/i$, while for complete polynomials $n_v = \prod_{i=1}^{d}(k+1)$. In this document the Rosenbrock schemes are applied mainly for incompressible flow computations. In particular, the four stages, order three (ROSI2PW) scheme of Rang and Angermann [29] is employed. This scheme preserves its formal accuracy when applied to the system of DAEs arising form the spatial discretization of the INS equations.

Algorithm 3 Explicit first-stage singly-diagonally implicit Runge-Kutta method

set $n_F = \dfrac{t_F}{\delta t}$
do $n = 0, n_F$
  set $\boldsymbol{w}_h^{n+1} = \boldsymbol{w}_h^n$
  do $s = 1, n_s$
    set g_h to zero
    do $o = 1, s-1$
      update $g_h(\boldsymbol{z}_h)$ += $\mathsf{a}_{s,o} f_h(\delta \boldsymbol{w}_h^o, \boldsymbol{z}_h)$
    enddo
    find $\delta \boldsymbol{w}_h^s \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ such that, for all $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$

$$\frac{1}{\mathsf{a}_{s,s}\delta t} m_h(\delta \boldsymbol{w}_h^s, \boldsymbol{z}_h) = -f_h(\delta \boldsymbol{w}_h^s, \boldsymbol{z}_h) - \frac{1}{\mathsf{a}_{s,s}} g_h(\boldsymbol{z}_h) \tag{2.32}$$

  enddo
  do $s = 1, n_s$
    update $\boldsymbol{w}_h^{n+1}$ += $\mathsf{b}_s \delta \boldsymbol{w}_h^s$
  enddo
enddo

Explicit first stage singly-diagonally Runge–Kutta schemes

The multi-stage explicit first stage singly-diagonally implicit scheme (ESDIRK) requires the solution of a nonlinear system at each stage $s = \{1, \cdots, n_s\}$. Given the initial condition $\boldsymbol{w}_h^0 = \boldsymbol{w}_h(t=0) \in [\mathbb{P}^k(\mathcal{T}_h)]^m$, the sequence $\boldsymbol{w}_h^n$, with $n = \{0, \cdots, n_s\}$ can be obtained as described in Algorithm 3, where a_{ij}, b_i are real, scheme-specific coefficients and $\delta \boldsymbol{w}_h^s$, with $s = \{1, \cdots, n_s\}$, the solutions at each stage of the scheme, that are properly combined to compute the solution at the next time slab. The algorithm requires the solution of (2.32). To this end, defining the mass matrix and Jacobian operators as in Eq. (2.30), the nonlinear system can be solved through Newton's method, which requires to solve a set of linear systems. A single Newton's update can be compactly written as

$$\boldsymbol{G}_h \; \delta \boldsymbol{w}_h = \boldsymbol{g}_h \tag{2.33}$$

where $\delta \boldsymbol{w}_h, \boldsymbol{g}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$ are the unknown polynomial function and the right-hand side arising from the Newton's method, respectively, and $\boldsymbol{G}_h = (1/\mathsf{a}_{s,s}\delta t)\boldsymbol{M}_h + \boldsymbol{J}_h$ is the stage-specific global matrix operator. One of the advantages of the ESDIRK schemes in comparison to other implicit strategies is that the first stage of the method is explicit since $\mathsf{a}_{1,1} = 0$. This increase the computational efficiency of the approach in comparison to other time integration strategies. In this work the ESDIRK scheme will be employed in the context of implicit, compressible Navier–Stokes simulations. In particular, an order-three, four stage algorithm presented in [30] has been selected.

Chapter 3

Hybrid MPI/OpenMP parallelization strategies

Nowadays the need of parallel efficiency is no more an exclusive requirement for applications running on High Performance Computing (HPC) systems. In fact, all the up to date computers use multi-core chips having a number of cores significantly large, especially for high-end multi-CPU systems such as workstations and servers. According to this trend, the new HPC platforms are evolving towards configurations with more and more computational cores for each node. Additional hardware resources, such as many-core, the Intel Xeon Phi, or graphics processing unit (GPU) accelerators, can be also connected to the computational platforms in order to improve further their performance. An efficient use of these computational resources is of primary importance for CFD/CAA, as well as for many other scientific applications. However, the design of an efficient parallel implementation suitable for different computer architectures is not an easy task.

The parallelization of DG code can follow two common approaches. First, the use of the Message Passing Interface (MPI) library, which is widely employed by the CFD/CAA community and it can be used on both the distributed and shared memory platforms. Second, the use of the Open Multi-Processing (OpenMP) paradigm, which is known to be a valuable alternative on shared memory systems. However, few papers deal with an OpenMP implementation of a DG solver in the current literature. To the authors' best knowledge only [31, 32, 33] deal, at least partially, with hybrid MPI/OpenMP implementations of a fully unstructured DG solvers. Other authors [34, 35, 36], for example, presented OpenMP parallelization strategies somehow related to the geometric configurations considered and not easily generalizable to general purpose CFD/CAA solvers based on unstructured grids.

Despite being OpenMP confined to shared memory systems, it shows considerable advantages over MPI. The most important one is the small programming effort required for the parallelization of an application. In addition, OpenMP allows a theoretically higher performance improvement on multi- and many-core platforms: the use of MPI within a node can be not optimal since the shared memory can be exploited to reduce the overhead cost of the communications. For this reason, a hybrid MPI/OpenMP implementation is usually suggested to improve the parallel efficiency of codes running on clusters of multi-core nodes. To this end, the use of an efficient OpenMP parallelization is mandatory, and the design of such strategy is part of the scope of the present work.

Nevertheless, several papers state that for a large amount of cores the achievement of high OpenMP parallel efficiencies is not trivial. Moreover, whether or not all the applications and hardware platforms can effectively benefit from a hybrid

MPI/OpenMP approach is not clear, *e.g.* [37, 38]. The reasons for these statements are that *i*) the code should be fully parallelized even in the OpenMP context; *ii*) the use of OpenMP implies overheads due to the initialization of parallel and/or work-sharing regions; *iii*) the overheads due to the explicit and implicit synchronizations of some of the work-sharing constructs; *iv*) the penalization due to false sharing of cache lines between processors; *v*) the memory latency due to the cache coherent Non Uniform Memory Access (ccNUMA) hardware design employed in many recent multi-core architectures; *vi*) the compiler performance in handling the OpenMP directives. In addition, obtaining a reliable indication about the general effectiveness of the implementation is not straightforward due to the large number of factors affecting the OpenMP parallel performance. However, several scalability tests on both multi-core and many-core platforms are reported to provide some general indications and trends.

While MPI makes the use of domain decomposition algorithm almost mandatory, shared memory parallel implementations can consider different alternatives. One of the contributions of this work is to discuss on the parallel efficiency of three OpenMP-based parallelization strategies. The first one relies on an original implementation of a colouring algorithm, properly employed to deal with a DG solver, the second mimics a pure MPI implementation in an OpenMP context, while a third one can be considered as somehow "in the middle" between them. The results show that all the considered implementations perform quite well. The domain decomposition algorithm reaches the highest level of parallel efficiency at low computational loads. However, this approach does not take advantage of some interesting features of OpenMP, such as the advanced options for loops scheduling. In addition, this type of implementation is more complex since it requires a "two-level" partitioninig of the computational domain. Although a bit less performing on small problems, the colouring approach excels when dealing with larger grids and polynomial orders. Colouring can be also considered as the most appealing choice for the implementation in an existing MPI-based code. The same algorithm is particularly well suited to run on hardware accelerators, an opportunity recently given by the OpenMP 4.0 standard. Additionally, the performance gain observed for different clusters of multi-core nodes for a hybrid MPI/OpenMP implementation based on a colouring algorithm is reported in the paper. Preliminary results about the usage of Xeon Phi coprocessor as accelerator, also known as *offload mode*, are also given. The availability of a portable high-level Application Programming Interface (API) as OpenMP, able to deal with a wide range of parallel architectures, clearly represents a great advantage for the HPC practitioners.

It is worth noticing that the applications reported in this chapter are all referred to an explicit-in-time discontinuous Galerkin discretizations. The reason for this choice is related to the fact that for LEE/EE, as well as the set of compressible Navier–Stokes equations without turbulence modelling, an explicit solver can be competitive in terms of memory requirements and efficiency. Moreover, it allows to benchmarck the code focusing on the parallelization of the residual evaluation only, which simplifies the analysis of the results as well as the implementation. See [39, 40, 41] for additional details. Finally, it is worth mentioning that the extension of the approach to an implicit-in-time discretization is subject of current research.

Algorithm 4 Structure of the explicit DG solver

1: **do** $n = 1, \#time\_steps$ ▷ Time steps loop
2: `!$OMP parallel` ▷ Open the Parallel section
3: Compute $\delta t = \min(\delta t_K)$ based on the `CFL` condition
4: $\boldsymbol{Q} = \boldsymbol{Q}^0$
5: **do** $s = 1, n_s$ ▷ Loop for the Runge–Kutta steps
6: $\boldsymbol{R} = \boldsymbol{R}_{\mathcal{F}_h^i}(\boldsymbol{Q})$
7: $\boldsymbol{R} = \boldsymbol{R} + \boldsymbol{R}_{\mathcal{F}_h^b}(\boldsymbol{Q})$
8: $\boldsymbol{R} = \boldsymbol{R} + \boldsymbol{R}_{\mathcal{F}_h^p}(\boldsymbol{Q})$ ▷ Only for Partition & Face-Partition algorithms
9: $\boldsymbol{R} = \boldsymbol{R} + \boldsymbol{R}_{\mathcal{T}_h}(\boldsymbol{Q})$
10: $\boldsymbol{R} = \delta t \boldsymbol{M}_h^{-1} \boldsymbol{R}$
11: **if** $(s \neq n_s)$ **then**
12: $\boldsymbol{Q} = \boldsymbol{Q}^0 - \mathsf{a}_s \boldsymbol{R}$
13: **end if**
14: $\boldsymbol{Q}^0 = \boldsymbol{Q}^0 - \mathsf{b}_s \boldsymbol{R}$
15: **enddo**
16: Compute $||\boldsymbol{R}||_{L_2}$ and $||\boldsymbol{R}||_{L_\infty}$
17: **enddo**

3.1 Parallel OpenMP Implementation

Although this section is mainly referred to LEE and EE, the algorithms and the considerations reported below can be extended to the NS case. In Algorithm 4, the tasks of the explicit-in-time scheme of the DG solver proposed in Algorithm 1 are reported. The implementation of an explicit DG solver calls for the use of two main arrays hereby named solution $\boldsymbol{W}$ and residuals $\boldsymbol{R}$. In a modal framework, it is convenient to arrange those two arrays using two indices pointing to the equation i and the polynomial degree j and an element-index p pointing to the generic element $\kappa_p \in \mathcal{T}_h$, *i.e.* $W(i,j,p)$ and $R(i,j,p)$. According to the previous section and for computational convenience, the evaluation of the residuals vector are split over two main loops. The former, performed over the elements-index p (line 9), collects the elemental contributions to $\boldsymbol{R}_{\mathcal{T}_h}$. The latter, performed over the faces-index q, with $\sigma_q \in \mathcal{F}_h$, collects the numerical fluxes contributions $\boldsymbol{R}_{\mathcal{F}_h}$. This loop can be further split in two tasks: the first covers all the internal faces, $\boldsymbol{R}_{\sigma_i}$ with $\sigma \in \mathcal{F}_h^i$ (line 6); the second considers all the boundaries faces, $\boldsymbol{R}_{\sigma_b}$ with $\sigma \in \mathcal{F}_h^b$ (line 8).

The evaluation of the $\boldsymbol{R}_\kappa$ contribution to the residual vector can be safely and easily parallelized using, for example, the Fortran `!$OMP do` work sharing directive without any data-race condition problem, see Algorithm 5. Such a straightforward parallelization is not possible for the loop over the faces. In fact, for each face σ the residuals of both the κ^- and κ^+ elements are simultaneously updated and two threads can overwrite the same memory address of the $\boldsymbol{R}$ vector.

In the following sub-sections three possible OpenMP parallelization algorithms for the DG solver are described. To simplify the discussion, only to the residuals evaluation is considered. In fact both the solution stage of Algorithm 4, line 10, which relies on an element-wise linear system direct solver, and the Runge–Kutta update steps of Algorithm 4, lines 12 and 14, are handled as $\boldsymbol{R}_\kappa$. In fact, these operations do not

Algorithm 5 Evaluation of $\boldsymbol{R} = \boldsymbol{R} + \boldsymbol{R}_{\mathcal{T}_h}(\boldsymbol{Q})$ by work sharing

```
1: !$OMP do
2: do p = 1, n_e
3:     do j = 1, m
4:         do i = 1, n_v
5:             R(i,j,p) = R(i,j,p) - ∫_{κ_p} Σ_{n=1}^{d} F̃_{j,n}(w_h) ∂z_i/∂x_n
6:         enddo
7:     enddo
8: enddo
```

Table 3.1: Summary of the OpenMP algorithms characteristics

Parallel Task	Colouring	Partition	Face partition
$\boldsymbol{R}_{\mathcal{T}_h}$	work-sharing	by partition	work-sharing
$\boldsymbol{R}_{\mathcal{F}_h^i}$	work-sharing with colouring	by partition	by partition
$\boldsymbol{R}_{\mathcal{F}_h^b}$	work-sharing with colouring	by partition	by partition
$\boldsymbol{R}_{\mathcal{F}_h^p}$	–	work-sharing with colouring	work-sharing with colouring
Runge–Kutta	work-sharing	by partition	work-sharing
$\min(\delta t_K)$	work-sharing	work-sharing	work-sharing
$\|\|\boldsymbol{R}\|\|_{L_2}$ and $\|\|\boldsymbol{R}\|\|_{L_\infty}$	work-sharing	work-sharing	work-sharing

imply any data-race issue and can be arranged in a parallel loop over the elements as in Algorithm 5. It is worth noting that, in the EE/LEE case, the LU (lower-upper) factorization of the block-diagonal mass matrix is computed and stored only once during preprocessing, as it does not change during all the solver execution. Table 3.1 summarizes the employed parallelization strategies for each of the building blocks of the solver.

3.1.1 Colouring algorithm

The colouring procedure, reported in Algorithm 6, is based on a naive algorithm similar to that used by [42, 43]. The main difference here is that the colouring procedure is applied to faces instead of mesh elements. The algorithm can be described as follows. When a colour has to be attributed to a face σ, one must check the colours already assigned to the faces belonging to the elements sharing σ. The scope is to set a different colour to σ; if all the colours of the palette are in use a new colour is added and given to σ. Any mesh element cannot have more than one face per colour. Once that all the faces are coloured the contribution of numerical fluxes to the residual vector can be safely assembled in parallel for all the faces of the same colour (Algorithm 7). An example is reported in Figure 3.1(a), where the $\boldsymbol{R}_{\mathcal{F}_h^i}$ contributions due to the four red internal faces can be summed concurrently (operation represented by an arrow) to the $\boldsymbol{R}$ global vector. The same holds for the

Algorithm 6 Colouring of the faces

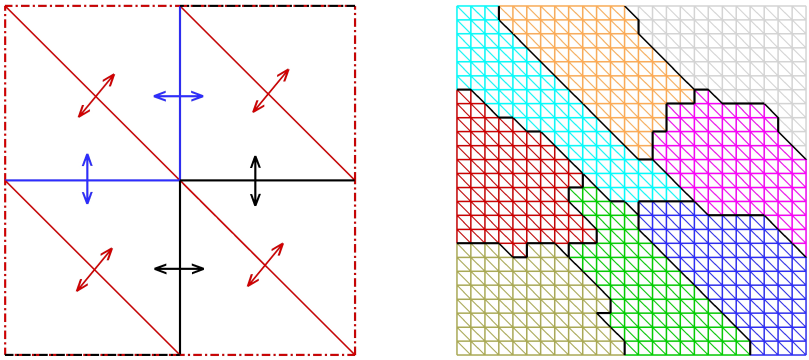
```
1: nc = 0                                                    ▷ nc is the number of colors
2: ifc(1 : nfi) = 0                                          ▷ ifc is the array of the colors of the faces
3: do if = 1, nfi                                            ▷ nfi is the number of internal faces, F ∈ F_h^i
4:     do ic = 1, nc
5:         flag = 0
6:         do k = 1, 2
7:             ie = ife(k, if)  ▷ ife(k, if) points the elements at both the sides of a face if
8:             do j = 1, nfe(ie)  ▷ nfe(ie) is the number of faces for each element ie
9:                 jf = ief(j, ie)                          ▷ ief(j, ie) points to the nfe(ie) faces of the element ie
10:                if (ifc(jf) = ic + 1) then
11:                    flag = 1
12:                end if
13:            enddo
14:        enddo
15:        if (flag = 0) then
16:            go to 19
17:        end if
18:    enddo
19:    nc = max(nc, ic + 1)
20:    ifc(if) = ic + 1
21: enddo
```

Algorithm 7 Evaluation of $\boldsymbol{R} = \boldsymbol{R}_{\mathcal{F}_h^i}(\boldsymbol{Q})$ by colouring

```
1: do ic = 1, nc
2:     !$OMP do
3:     do if = sfc(ic), efc(ic)                              ▷ loop over the faces of the same color ic
4:         q = icf(if)                  ▷ icf(ic, if) points to all F ∈ F_h^i of the same color ic
5:         p1 = ife(1, q)
6:         p2 = ife(2, q)
7:         do j = 1, m
8:             do i = 1, N_dof^K
9:                 R(i, j, p1) = ∫_{σ_q} Σ_{n=1}^{d} n_n^σ F̂_{j,n}(w_h) [[z_i]]
10:                R(i, j, p2) = ∫_{σ_q} Σ_{n=1}^{d} n_n^σ F̂_{j,n}(w_h) [[z_i]]
11:            enddo
12:        enddo
13:    enddo
14: enddo
```

two black and for the two blue faces.

Although this algorithm is clearly not optimized to obtain a group of colours characterized by an equal number of faces, its naive application performs pretty well. This is confirmed by the numerical experiments where for different meshes of triangular elements, where often it has been obtained the minimum number of possible colours,



(a) colouring (b) partition

Figure 3.1: Schematic representation of the element faces treatment on the employed algorithms.

i.e. three. Colours may change with the initial ordering of the faces, as well as with grid reordering. However, numerical experiments revealed that the overall parallel performance does not significantly change when the number of colours is not optimal. Moreover, as pointed out by [43], it is possible to add an *a posteriori* phase to the algorithm to balance the number of faces for each colour.

The same algorithm can be used without any modifications for the boundary faces where boundary conditions are imposed. In practice almost all the boundary faces have the same colour and only if one element owns more than one boundary face more than one colour are needed. As for the internal faces the boundary $\boldsymbol{R}_{F_b}$ terms of the same colour can be safely evaluated and assembled in parallel, see dash-dot lines of Figure 3.1(a).

3.1.2 Partition algorithm

This algorithm mimics the classical MPI parellization paradigms based on domain decomposition. The computational grids are partitioned, here by means of the Metis [44] library, in a set of non-overlapping sub-domains, see Figure 3.1(b). Each OpenMP thread only deals with its subset of elements and faces in a pure Single Program Multiple Data (SPMD) programming fashion. This means that the main loops are performed by specifying explicitly their bounds within the code, see Algorithms 8 and 9 dealing, respectively, with the loops over the volumes and the faces. In this way the partition algorithm uses a coarse-grained parallelization strategy, which is a less common choice than the fine-grained parallelization proper of the work-sharing constructs in a pure OpenMP context. Only in the sub-domains interfaces, *i.e.* the black faces in Figure 3.1(b), the solver takes advantage of the global view of the application memory address space, avoiding an explicit use of buffers and messages. The contributions to the residuals vector related to the numerical flux computation

Algorithm 8 Evaluation of $\boldsymbol{R} = \boldsymbol{R} + \boldsymbol{R}_{\mathcal{T}_h}(\boldsymbol{Q})$ by partitions

1: IO=`OMP_GET_THREAD_NUM() +1`
2: **do** $p = spe(IO), epe(IO)$
3: **do** $j = 1, m$
4: **do** $i = 1, n_v$
5: $R(i,j,p) = R(i,j,p) - \int_{\kappa_p} \sum_{n=1}^{d} \widetilde{F}_{j,n}(\boldsymbol{w}_h) \frac{\partial z_i}{\partial x_n}$
6: **enddo**
7: **enddo**
8: **enddo**

Algorithm 9 Evaluation of $\boldsymbol{R} = \boldsymbol{R}_{\mathcal{F}_h^i}(\boldsymbol{Q})$ by partitions

1: IO = `OMP_GET_THREAD_NUM() + 1`
2: **do** $q = spf(IO), epf(IO)$ ▷ $F \in \mathcal{F}_h^i$ of the partition IO
3: $p1 = ife(1,q)$
4: $p2 = ife(2,q)$
5: **do** $j = 1, m$
6: **do** $i = 1, n_v$
7: $R(i,j,p1) = \int_{\sigma_q} \sum_{n=1}^{d} n_n^\sigma \widehat{F}_{j,n}(\boldsymbol{w}_h) \, [\![z_i]\!]$
8: $R(i,j,p2) = \int_{\sigma_q} \sum_{n=1}^{d} n_n^\sigma \widehat{F}_{j,n}(\boldsymbol{w}_h) \, [\![z_i]\!]$
9: **enddo**
10: **enddo**
11: **enddo**

over these faces, denoted as $\boldsymbol{R}_{\sigma_p}$, are assembled in parallel according to the colouring Algorithm 7 of the previous section.

This is one of the few parts of the code relying on work-sharing directives. Their usage is motivated by the lack in the OpenMP standard of a reduction clause for the `!$OMP parallel` regions. For example, at the beginning of each Runge–Kutta step the solver computes the maximum δt satisfying the `CFL` stability condition for each element $\kappa \in \mathcal{T}_h$, line 5, Algorithm 4, and the L_2 and L_∞ norms of the residuals vector, at line 16. These tasks imply reduction operations over the domains and a loop over the elements which is performed according to a `!$OMP do`. The optimized reduction clause available for this OpenMP directive was preferred to a version that may scale poorly, see [45].

3.1.3 Faces-partition algorithm

In the faces-partition algorithm the residual contributions related to faces are evaluated as in the partition approach (Algorithm 9) while the volume residual is assembled using the work-sharing directives (Algorithm 5). The idea is to use a partition of the grid only to parallelize the loop over the faces. The volume terms are instead handled in a parallel fashion following one of the `!$OMP do` scheduling policies. This means

that while one process always deals with the internal faces of the same partition, there is not a link between the elements of a partition and the elements handled by an OpenMP thread.

3.1.4 Comments on the algorithms

It is worth noting that all the OpenMP implementations proposed initialise the parallel region by using the `!$OMP parallel` directive, see line 2 of Algorithm 4, even those using work-sharing directives. This avoids the overhead associated to the creation of a team of threads at each parallelized do loop. In other words, the use a combined work-sharing directive such as the `!$OMP parallel do` is completely avoided. Thanks to the compactness of the DG methods all the aforementioned implementations can be considered as fully parallelized, *i.e.* no serial part of the code can compromise the scalability. Moreover, the shared memory was fruitfully exploited to perform only once the evaluation of the flux terms at the inter-partitions faces, avoiding any redundant operations. Such level of parallelization is more difficult to be achieved with a pure MPI solver. As reported in [46], an implementation of this type requires twice the number of MPI communications if compared to a strategy in which a redundant evaluation of these numerical flux terms is performed.

Due to a possible not optimal grid partitioning and to the evaluation of the $\boldsymbol{R}_{F_b}$ terms at boundary, the workload among processes can be unbalanced for the partition-based algorithm. In fact, this algorithm admits that a partition does not own any boundary face, since it may be internal to the domain. From this point of view the colouring is superior: the algorithm can be applied on both internal and boundary faces in the same way and it can take advantage of the `!$OMP do` scheduling options. On the other hand, the pure partition strategy allows to avoid several OpenMP synchronizations. This is possible because every thread deals only with the same portion of the shared memory arrays $\boldsymbol{W}$ and $\boldsymbol{R}$. The only instance where synchronization is mandatory is at the beginning and at the end of the parallel loop performed for the evaluation of the $\boldsymbol{R}_{\sigma_p}$ contributions.

All the proposed implementations follow the so called *first touch policy*. This means that at the execution beginning any thread initializes the part of the arrays it is going to access. Most operating systems (OS) assign a memory address physically close, within the same NUMA region, to the core that for first initialise the data. This aspect is particularly relevant when using ccNUMA platforms. If initialization is performed in a serial fashion all the memory data will be placed within only one NUMA region, reducing the overall memory bandwidth to and from the CPUs cores.

Finally, from the point of view of the data locality, the partition algorithm seems preferable. In fact, for $\sigma \in \mathcal{F}_h^i$, the $\boldsymbol{W}$ degrees of freedom (DOFs) of both the κ^- and κ^+ elements can be generally owned by a single NUMA region. The situation is quite different when using the colouring algorithm, being the allocation of the $\boldsymbol{W}$ DOFs of neighbouring elements in the same NUMA region not trivial. For the faces-partition algorithm, it has been pointed out that a strict connection between the faces of a partition and the mesh elements handled by an OpenMP thread does not exist, and thus, even in this case, the data affinity can clearly be not optimal.

3.1.5 Hybrid MPI/OpenMP parallel algorithm

In the hybrid MPI/OpenMP implementation, the OpenMP parallelization relies on the colouring algorithm, which can be very easily integrated within a MPI solver. This algorithm has been preferred since it avoids the difficulties related to the two nested domain decompositions, one for the MPI implementation and the other used at the OpenMP level. With the colouring approach, the communications can be managed only by one OpenMP thread for each MPI process. Moreover, the colouring algorithm is well suited to deal with accelerators, such as GPUs [43] or the Xeon Phi coprocessors in *offload mode*, see Section 3.3.4.

The MPI implementation takes advantage of the non-blocking communications, which are overlapped with computations as much as possible. In other words, the $\boldsymbol{R}_{\sigma_i}$ evaluation and the exchange of messages, required to update DOFs of the ghost elements, are performed simultaneously. Although for hyperbolic governing equations it is possible to overlap the MPI communications with the computation of $\boldsymbol{R}_{\kappa}$, this optimization has been avoided in this work. Therefore, the same algorithm is used for both the EE/LEE and the NS governing equations. It is worth noticing that, using the BR2 scheme, the diffusive part of the volume integrals of Eq. (2.12) collects the contributions related to the lift operator, Eq. (2.13), which involve the jumps of the variables at the mesh interfaces, see Eq. (2.15). In this case the contribution of the lifting operator can be efficiently evaluated while looping over the faces, assembling only later the $\boldsymbol{R}_{\kappa}$ residuals components. This task cannot be overlapped with the MPI communications.

Finally, to limit the MPI overheads, the distributed memory code uses ghost elements. This means that the $\boldsymbol{R}_{\sigma_p}$ contributions to the residuals vector, $\boldsymbol{R}$, are concurrently computed by two processes. The efficiency of the MPI solver will be demonstrated in the Section 3.3.3.

3.2 Test cases description

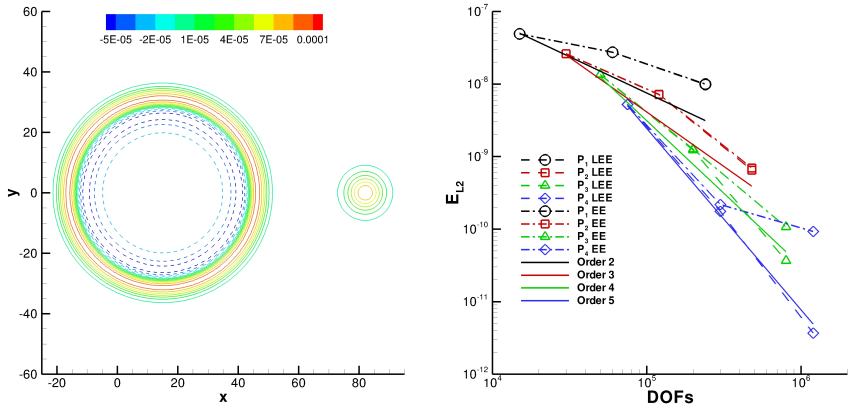
3.2.1 2D inviscid flow problem

This two-dimensional test case, Problem 1 of Category 3 of the first Workshop for Computational Aeroacoustics [47], requires the solution of the propagation of both an acoustic pulse and of an entropic vortex. The computational domain is a square, $-100 \leq x_i \leq 100$, the mean flow is uniform, the Mach number is $M_\infty = 0.5$, and the velocity vector is aligned to the x_1 axis. The initial conditions are defined as follows

$$\begin{aligned} p &= \varepsilon e^{-\alpha_1 r_1^2}, \quad \rho = p + 0.1\varepsilon e^{-\alpha_2 r_2^2}, \\ u_1 &= 0.04\varepsilon x_2 e^{-\alpha_2 r_2^2}, \quad u_2 = -0.04\varepsilon\,(x_1 - 67)\, e^{-\alpha_2 r_2^2}, \end{aligned} \tag{3.1}$$

being $[\rho, p, u_i]^T$ the perturbation quantities and

$$\begin{aligned} &\alpha_1 = \frac{\ln(2)}{9}, \quad \alpha_2 = \frac{\ln(2)}{25}, \quad r_1 = \sqrt{x_1^2 + x_2^2}, \\ &r_2 = \sqrt{(x_1 - 67)^2 + x_2^2}. \end{aligned}$$



(a) $\mathbb{P}_4$, $n_e = 20000$ (b) convergence rate

Figure 3.2: Two-dimensional pulse propagation problem, ρ contour plot and convergence rate at $t = 30$

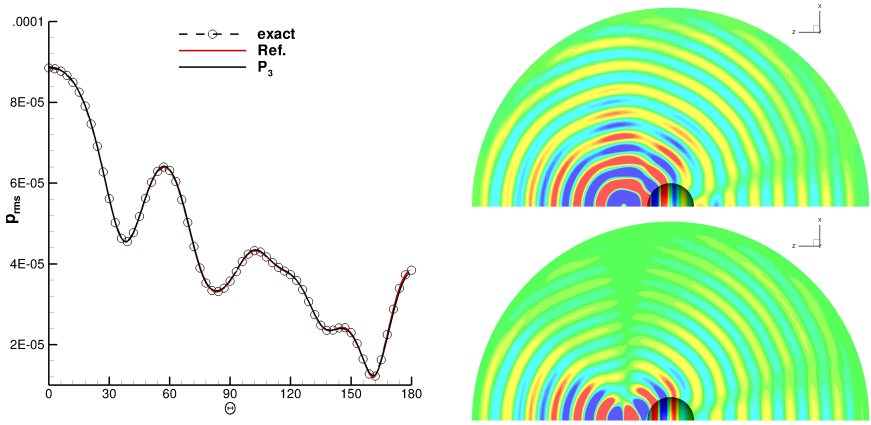
In Eq. (3.1), a numerical perturbation ε, here set to 10^{-3}, has been introduced to multiply the original initial conditions. The intent was to limit, as much as possible, the non-linear effects and use the analytical solution, which is written for the LEE case, for the evaluation of the convergence rate in the EE case also. To this purpose, three grids consisting of $n_e = 1250, 5000, 20000$ triangular elements and polynomial approximations up to $\mathbb{P}_4$ were considered. The E_{L_2} error norms of Figure 3.2(b) are evaluated at the time $t = 30$ when the acoustic and the entropic vortex have not yet reached the outer boundaries, see Figure 3.2(a). This choice was made to avoid all the uncertainties related to the boundaries conditions, which can compromise the convergence rates. The theoretical convergence rates are obtained both for the LEE and EE solutions. In fact, the errors norms are almost identical up to $E_{L_2} \approx 10^{-10}$. Below this value the difference between the linear and the non-linear solution is the main source of error, and E_{L_2} stalls to an almost fixed value. This consideration explains the deviation for EE from the ideal convergence at low error levels. Finally, note the clear advantage in using a high-order approximations for small E_{L_2} values.

3.2.2 3D inviscid flow problems

The first three-dimensional test case concerns the scattering by a perfectly reflecting spherical wall of the perturbations generated by an acoustic source in a null mean velocity. This test case was proposed as Problem 4 of Category 1 in the Second Computational Aeroacoustics (CAA) Workshop on Benchmark Problems [48]. The acoustic source is distributed spatially by $\boldsymbol{s} = [0, s_2, 0, 0, 0]^T$, where

$$s_2 = -A\, e^{-B\ln(2)\left[x_1^2 + x_2^2 + (x_3 - x_{3s})^2\right]} \cos(\omega t)\,, \tag{3.2}$$

which is added to the source term $\boldsymbol{s}$ of Eq. (2.1). The constants in Eq. (3.2) are $A = 0.01$, $B = 16$, $x_{3s} = 2$, $\omega = 2\pi$. The problem is symmetric around the x_3 axis and this allows to reduce the computational domain to a quarter of the complete spherical domain. Symmetry conditions have been imposed on the planes $x_1 = 0$ and $x_2 = 0$. The computational domain has been discretized using a grid consisting of 89628 hexahedral elements. The spherical outer boundary of the domain, including the radial thickness of a sponge layer, has a radius equal to $8.5R$, where R is the radius of the sphere. The sponge layer is a sacrificial region in which the out-coming waves are exponentially damped to avoid spurious reflections at the true boundaries, where characteristics-based boundary conditions are employed. The spherical inner boundary of the sponge layer has a radius equal to $7.5R$, *i.e.* the layer width corresponds to one wavelength. In previous experiments, see [49, 50], and following the indications provided in [51], the sponge layer strength was set to $\eta_{target} = 20$ dB with a quadratic profile. The Figure 3.3(a) shows the root-mean-square of the pressure, computed during one period of the source disturbance, along a circle with radius $5R$. The fourth order accurate, $\mathbb{P}_3$, numerical solutions compare very well with both the analytical ones derived by [52], and previous results obtained with a different non-reflecting boundary treatment [26]. Finally, Figure 3.3(b) shows the p and u_3

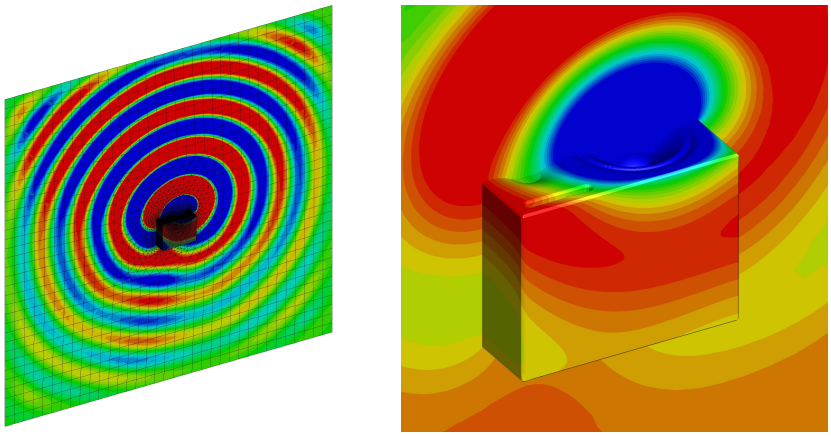


(a) p_{rms} (b) top p, bottom u_3

Figure 3.3: Acoustic scattering from a sphere, $\mathbb{P}_3$ solution. Ref. solution from [26], exact solution from [52]

contours for the $\mathbb{P}_3$ solution.

The second three-dimensional test case solves LEE to compute the directivity from a loudspeaker. This problem, based on an industrial geometry, was first proposed and solved by [53]. The description of this test case will be given using dimensional quantities. The acoustic source is a sinusoidal normal velocity assigned to the surface of the woofer, the low frequency driver, corresponding to $1kHz$. The mean flow velocity is null and the sound velocity was set to $343m/s$. The computational domain is a box centered in the loudspeaker of extension $-1.5m \leq x_i \leq 1.5m$. Taking



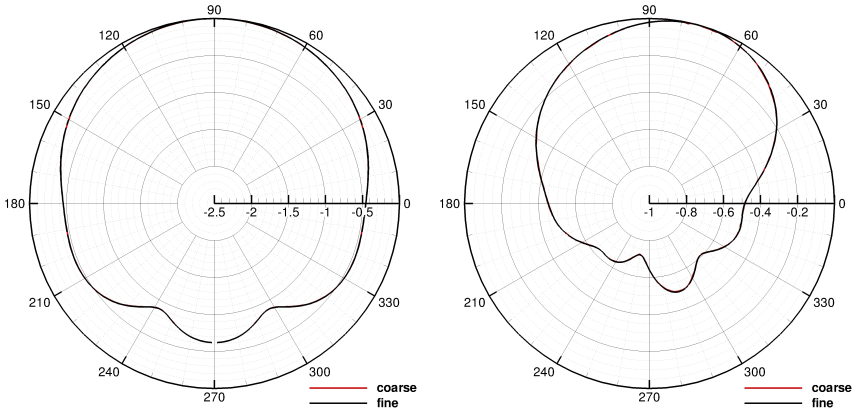
(a) computational mesh (b) detailed view

Figure 3.4: Instantaneous contour of acoustic pressure on the loudspeaker and on the symmetry plane, $\mathbb{P}_3$ coarse grid solution

advantage of symmetry of the problem, the simulation was performed within the half of the domain. To avoid possible spurious reflections due to the boundary conditions, a sponge-layer with the same parameters of the previous test case has been employed.

For this flow problem two computational grids were considered. The fine mesh consists of 576762 tetrahedral elements, while the coarse mesh consists of 109067 hybrid elements (432 pyramids, 9918 prisms, 21186 tetrahedrons, 77531 hexahedrons). The hybrid unstructured coarse mesh was generated using tetrahedral elements near the body and an isotropic distribution of hexahedral elements, with edge length equal to a quarter of the wavelength, in the far-field. The mesh is designed to properly propagate the acoustic signal to the far-field. Figure 3.4 shows an instantaneous contour of pressure, as well as the grid distribution on the body and on the symmetry plane. The accuracy of the results obtained on the coarse hybrid grid confirms the greater suitability of DG methods for the simulation of acoustic propagation around complex geometries if compared to other high-order methods, *e.g.* finite differences, typically used in CAA.

Figure 3.5 displays the computed directivity patterns, evaluated on both the vertical and horizontal planes, on a circumference with radius equal to 1 m and centred in the top of the dust cup of the speaker on the symmetry plane. These results are normalized and expressed in decibels. The fine and coarse grids results are indistinguishable and agree very well with the those reported by [53], where are validated using experimental data. These results were obtained with a constant amplitude vibration set in every point of the woofer and neglecting any geometrical deformation. The reflex tubes and the cabinet volume were not modelled.



(a) Horizontal plane

(b) Vertical plane

Figure 3.5: Normalized directivity patterns of the loudspeaker at $1kHz$, $\mathbb{P}_3$ solutions

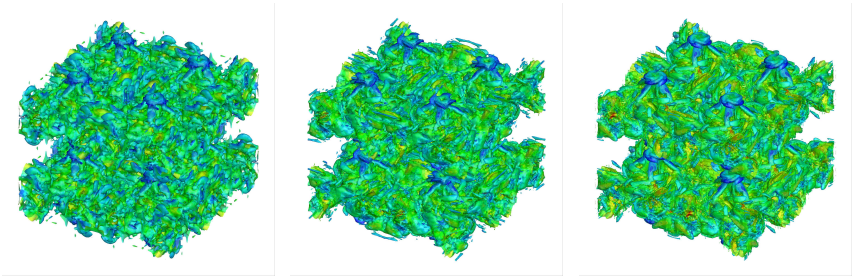
3.2.3 3D viscous flow problem

The viscous test case here proposed is the WS1 of the 5^{th} workshop on high order methods [54], which involves the implicit large eddy simulation of the Taylor Green Vortex problem at Reynolds number $Re = \rho_0 V_0 L/\mu = 1600$ and Mach number $M = 0.1$. This test case, which shows large scale vortices interacting with each other producing transition to turbulence and decay, it has been widely considered in the literature as a benchmark for turbulent flow simulations. The geometry consist on a tri-periodic cube with sides of length $2\pi L$, and the flow is initialized using the following primitive variable distribution

$$
\begin{aligned}
u_1 &= V_0 \sin\left(\frac{x}{L}\right) \cos\left(\frac{y}{L}\right) \cos\left(\frac{z}{L}\right), \\
u_2 &= -V_0 \cos\left(\frac{x}{L}\right) \sin\left(\frac{y}{L}\right) \cos\left(\frac{z}{L}\right), \\
u_3 &= 0, \\
\rho &= \rho_0, \\
p &= p_0 + \frac{1}{16}\left(\cos\left(\frac{x}{L}\right) + \cos\left(\frac{y}{L}\right)\right)\left(\cos\left(\frac{z}{L}\right) + 2\right).
\end{aligned}
$$

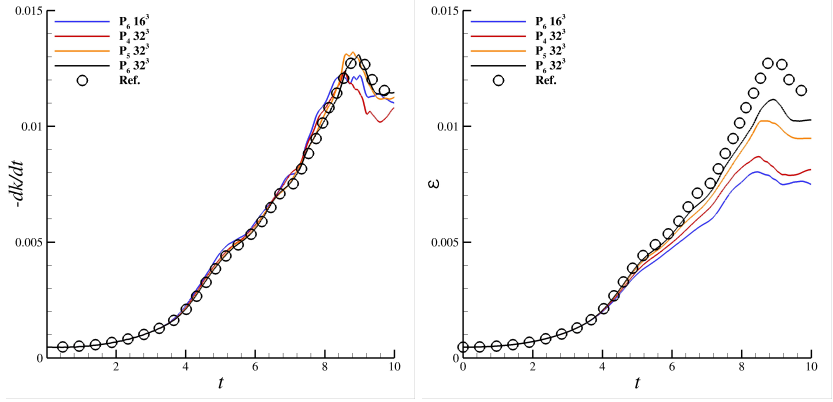
In this work the space dicretization relies on two different grids (16^3 and 32^3) and several orders of polynomial approximation of the solution ($\mathbb{P}_4$, $\mathbb{P}_5$ and $\mathbb{P}_6$) to show a spatial convergence of the problem, which involves the development of the turbulent flow over time. Such transient is recorded by the use of two main quantities. The first one is the time derivative of the kinetic energy, which is defined as

$$
k \overset{\text{def}}{=} \frac{1}{16\pi^3 \rho_0} \int_\Omega \rho u_k u_k \, \mathrm{d}\boldsymbol{x},
$$



(a) 16^3 $\mathbb{P}_6$ (b) 32^3 $\mathbb{P}_4$ (c) 32^3 $\mathbb{P}_6$

Figure 3.6: Taylor–Green vortex at $Re = 1600$. $\lambda_2 = -1.5$ iso–surface at $t = 10$. Iso–surface coloured by velocity magnitude.



(a) Kinetic energy k time derivative (b) Theoretical dissipation rate, ε

Figure 3.7: Taylor Green vortex at $Re = 1600$. Results for different numerical resolutions.

while the second one is the theoretical dissipation rate, evaluated as

$$\varepsilon \stackrel{\text{def}}{=} \frac{2\mu}{8\pi^3\rho_0} \int_\Omega \frac{\partial u_i}{\partial x_j} \frac{\partial u_i}{\partial x_j} \, \mathrm{d}\boldsymbol{x}.$$

which are related with each other, in the incompressible limit, via the following relation

$$-\frac{\mathrm{d}k}{\mathrm{d}t} = \varepsilon.$$

It is worth noting that the difference between these two quantities is connected to the numerical dissipation of the discretization. In fact, the implicit LES involves the solution of the Navier–Stokes equations without the use of any explicit subgrid scale model, but relying only on the diffusion of the numerical scheme. Several papers [9, 11] demonstrated the suitability of discontinuous Galerkin space discretizations for this kind of problems.

Figure 3.6 shows the $\lambda_2 = -1.5$ iso-surface at the final time $t = 10$ for three space discretizations of increasing resolution. An increased smoothness of the solution can be identified. As regards the solution quality, Figure 3.7 reports time derivative of the kinetic energy and the theoretical dissipation rate ε, in comparison with the reference DNS data of [55]. An overall agreement with the DNS can be noticed by increasing the resolution of the space discretization. In particular, the time derivative of the kinetic energy is globally well captured, while the pseudo-dissipation ε, which is reported in Fig. 3.7(b), gets closer to the DNS data by increasing the resolution. The numerical dissipation of the scheme bridges the imbalance of the kinetic energy derivative and the pseudo-dissipation ε.

3.3 Parallel performance

To obtain reliable indications about the parallel performance of all the OpenMP implementations proposed, a comparison on multi-core machines has been performed, including clusters and many-core platforms. Both the hardware and software characteristics of these architectures have a strong influence on the parallelism, and a description of the architectures considered will precede every numerical experiment. For the sake of clarity, Table 3.2 summarizes the main hardware features. With the exception of the AMD cluster, all the systems were/are hosted by the Italian supercomputing centre (CINECA).

To asses the performances of the different OpenMP algorithms the two-dimensional test case was considered, see Section 3.2.1, while three-dimensional problems were used for the hybrid MPI/OpenMP investigation, see Section 3.2.2. Numerical experiments reported consist in measuring the CPU time needed to perform 20 and 10 Runge–Kutta time steps for the 2D and the 3D test cases, respectively. To get indications about the solver usage when applied to real applications, the ancillary operations needed for the user's residuals monitor were also taken into account. These operations involve the computation and the write to a file of the $||\boldsymbol{R}||_{L_2}$ and $||\boldsymbol{R}||_{L_\infty}$ values at each time step. To optimize the use of the platform, the OpenMP capability to bind threads to cores has been employed, while the threads affinity as been left to the OS management. In fact, the way in which threads are assigned to the cores has a small influence on the parallelism. As it can be seen in the next section, the only exception is when a core pair of the AMD 6276 Opteron CPU is used.

3.3.1 Performance of the OpenMP implementations

Multi-Core platform

The first system consists of two $2.3GHz$ Opteron 6276 processors. This sixteen cores CPU is a multi-chip module built by two eight-cores CPUs placed on a single die package interconnected by a HyperTransport link. Each CPU in the package has its own memory controller, which is shared by the 8 cores. This configuration leads to 4 NUMA regions, see [56]. A peculiarity of this hardware is that the cores are organized in core pair, sharing some hardware components such as the floating point units and the L2 cache. Due to this configuration the maximum overall performance of a socket

System name	CPU	sockets per node	cores per socket	threads per core	type of memory	network type	nodes
-	AMD Opteron 6276	2	16 (2.3 GHz)	1	64GB, 2 NUNA per socket	Infiniband	2
GALILEO	Intel Xeon Haswell	2	8 (2.4 GHz)	1	128GB, 1 NUMA zone per socket	Infiniband	516
GALILEO coprocessor	Intel Xeon Phi 7120P (Knights Corner, KNC)	1	61 (1.238 GHz)	4	16GB, interconnecting ring	-	-
FERMI	IBM Power PC A2	1	16 (1.6 GHz)	4	16GB, UMA	5D Torus interconnect	10240
MARCONI A1	Intel Xeon E5-2697 v4	2	18 (2.3 GHz)	1	128GB, 1 NUMA zone per socket	OmniPath	1512
MARCONI A2	Intel Xeon Phi 7250 (Knights Landing, KNL)	1	68 (1.4 Ghz)	4	two-dimensional mesh, 16GB of MCDRAM, 96 GB DRAM	OmniPath	3600

Table 3.2: Summary of the hardware characteristics of the computational platform employed in the study.

cannot be sixteen times the single core performance. Indeed, a significant performance reduction is observed when running two identical instances of the serial DG solver using both the cores of a pair. In this case the execution time is approximately 1.25 times the standard wall clock time. This performance reduction is not observed if cores of different pairs are employed. As a direct consequence, the maximum speed-up achievable using all the 32 cores is assumed to be limited to 25.6, corresponding to an apparent parallel efficiency of about 80%.

Different compilers have also a sensible impact on the performance of the algorithms. In the present investigation, two Fortran compilers have been considered: the GNU `gfortran-4.7.2` and the Open64 `openf90-4.5.2.1`. All the numerical experiments were executed with the `-O3` optimization level and the appropriate options for the specific hardware architecture. The AMD suggestions [57] have been followed, rather than performing an in depth optimization among all the options. Such compiler usage can be considered representative of the typical use of a HPC practitioner.

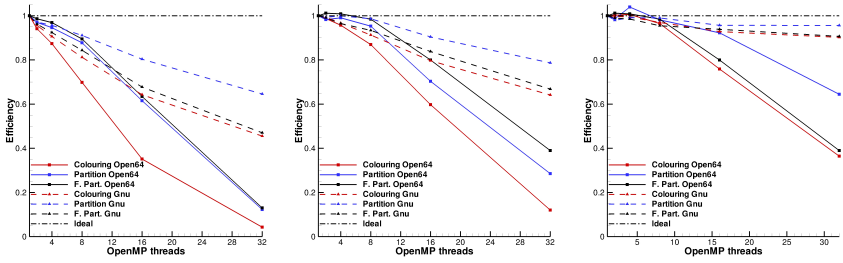
It is worth noting that the Open64 compiler, highly optimized for AMD CPUs, gives a serial performance gain of about 33% over `gfortran`, which is almost independent by grid density and the degree of the polynomial approximation. However, the focus of this paper is not on serial computations and these data are only reported to provide a comprehensive understanding of the parallel performance shown in the remaining of the paper.

Many-Core platforms

Two types of Many Integrated Core (MIC) platforms are considered in this work. They are Intel Xeon Phi chips of two different generations, the older Intel Xeon Phi 7120P (known as Knights Corner, KNC) and the newer Intel Xeon Phi 7250 (formerly Knights Landing, KNL). Both of them are able to handle four hardware threads for each core, but while the first generation is a treated as coprocessor, the second is used as a standalone CPU.

The KNC architecture owns 61, 1.238 GHz cores. As one core was reserved to the OS, the numerical experiments were performed up to 240 threads. The total memory of the system is 16 GB, subdivided in 16 channels supported by controllers. The cores and the memory are placed on a high-speed interconnecting ring, and the theoretical memory bandwidth is 352 GB/s, where cores operate in a symmetric way. In other words the memory access is uniform, see [58]. Computational experiments were performed in *native mode*. This means that the solver is executed on the Phi coprocessor directly without involving the host CPU, here a Xeon Intel Haswell 2.40 GHz. In this case the Intel `ifort-15.0.2` compiler was used, which supports both the *native* and the *offload* modes of the Phi coprocessor.

As regards the KNL, it is made of 68 1.40 GHz cores. The platform is organized in tiles made of two cores, sharing 1 MB of L2 cache and connected to each other by a two-dimensional mesh. A characteristic of this hardware is that it owns two types of memory. Eight high-bandwidth ($\approx$ 450 GB/s) MCDRAM (Multi-Channel Dynamic Random Access Memory) in chip devices, for a total of 16 GB, are coupled with six-channels of standard DRAM for about 96 GB ($\approx$ 90 GB/s). The two-dimensional interconnecting mesh can be configured at boot time into different modes. Here the



(a) $\mathbb{P}_1$ (b) $\mathbb{P}_3$ (c) $\mathbb{P}_6$

Figure 3.8: Two-dimensional OpenMP solver running on two AMD Opteron 6276 CPUs. $n_e = 1250$. Influence of the algorithms and of the compilers on the parallel performance.

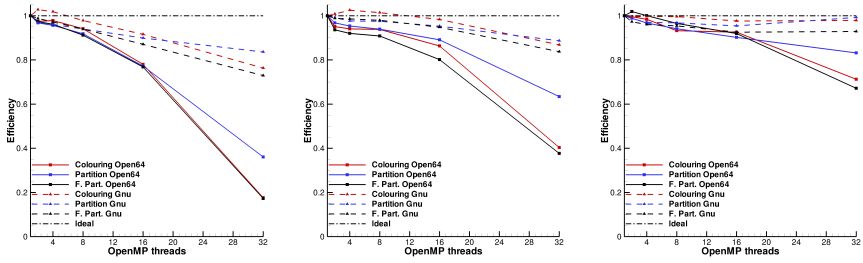
mode called flat/snc2 (Sub-Numa Clustering 2) is employed. In this configuration the memory is exposed as two non-uniform memory access domains to the OS. This is the mode usually suggested for NUMA-aware applications, see [59]. Finally, it is worth noting that, having the solver a limited memory footprint, all the allocations were the MCDRAM only using `numactl`. Here the compiler is the newer Intel `ifort-17.0.1`. For this version Intel reports an improvement of vectorization capabilities which is here relevant since each core KNL exploits two 512 bit vector processing units.

Multi-Core scalability

Figure 3.8 shows that, up to 8 OpenMP threads and for moderate computational loads, the parallel performance are quite high all the compilers and implementations. As a matter of the fact, adopting the coarsest grid and a $\mathbb{P}_6$ approximation the parallel efficiency is always above the 95%, see Figure 3.8(c). At $\mathbb{P}_3$ the efficiency is still above the 85%, Figure 3.8(b), while at $\mathbb{P}_1$ it decreases significantly. In particular, this behaviour holds for the Open64 compiler when coupled with the colouring algorithm, Figure 3.8(a). With this small number of elements, the partition algorithm seems the best choice while the colouring the worst one.

When the number of threads is further increased, 16 or 32 threads, the parallel efficiency drops down. Nevertheless, with GNU and the $\mathbb{P}_6$ approximation the efficiency is still above the 90% for all the algorithms. Note that for `OMP_NUM_THREADS` ≥ 16 the Open64 implementation of the OpenMP directives shows a clearer performance degradation, compared to GNU, which is more evident decreasing the polynomial approximation.

Using the finer grids, Figures 3.9 and 3.10, the parallel performance significantly raises. Moreover, the colouring algorithm overtakes the other approaches in several cases. In this context, only with Open64 and `OMP_NUM_THREADS` $= 32$ the partition algorithm proves a clear efficiency advantage. In some way this approach is able to exploit, even using the less-efficient OpenMP compiler, the peculiar hardware characteristics related to the core pair organization of the AMD CPU. That this behaviour, as well as the optimal performance at the lower computational loads, is ascribed to



(a) $\mathbb{P}_1$ (b) $\mathbb{P}_3$ (c) $\mathbb{P}_6$

Figure 3.9: Two-dimensional OpenMP solver running on two AMD Opteron 6276 CPUs. $n_e = 5000$. Influence of the algorithms and of the compilers on the parallel performance.

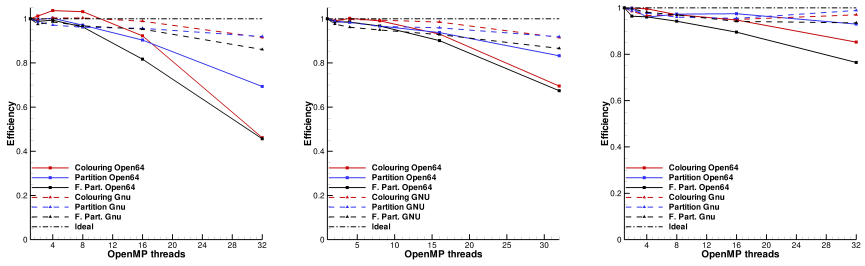
the smaller number of implicit synchronisations of its parallel constructs and to its better data locality. Differently, the good performances of the colouring approach with $n_e \geq 5000$ put in evidence that this algorithm is able to better balance the work loads among the cores. At $n_e = 20000$ the algorithm can take advantage of the `guided` or the `auto` scheduling policies for all the `!$OMP do` work-sharing loops, when available[1], see for example line 2 of Algorithm 7. On the other hand, most of the times the `static` option gives better results when using $n_e = 1250$, confirming that with a small number of elements per core it is preferable to limit as much as possible the overheads associated to the OpenMP instructions. As regards the face partition algorithm, even if the overall scalabilty is quite sattisfactory, it shows a significant advantage only adopting Open64 at the lower computational loads, see Figures 3.8(a) and 3.8(b).

Finally, in all the figures showing the efficiency at 32 cores, the reduced performance measured when using both the cores of a pair are taken into account. As reported in the previous section, the ideal speedup using 32 cores is in fact assumed equal to 25.6. In addition, the results on parallel performance reported in the following are obtained with EE, but identical speedups can be obtained when using the LEE model.

Many-Core scalability

Figure 3.11 displays the parallel speedup measured using two generations of the Intel Xeon Phi many-core CPUs. The results confirm the general trends observed in previous Section. The partition algorithm often performs as the best, in particular for low orders computations, *i.e.* $\mathbb{P}_1$ and $\mathbb{P}_3$ with $n_e = 5000$, while the colouring algorithm suffers at the lowest computational loads, *e.g.* $\mathbb{P}_1$ with $n_e = 5000$. However, the colouring algorithm is able to deliver an excellent speed-up in large simulations, *e.g.* $\mathbb{P}_6$ with $n_e = 20000$ and 240 threads. As shown in Figure 3.11(a), the code scalability up to 60 threads is almost linear and, as expected, leaves the ideal values when more then one thread per core is used. However, differently from what usually

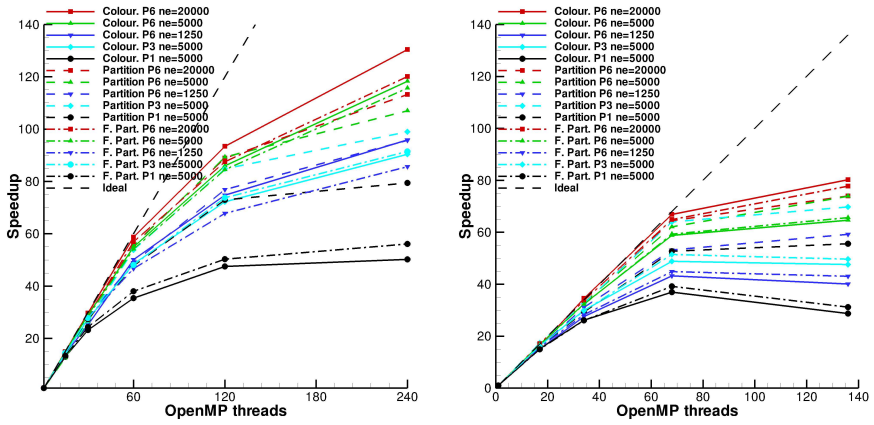
[1] The `auto` option was introduced in the OpenMP standard only with the release 3.0, it delegates the scheduling decision to the compiler.



(a) $\mathbb{P}_1$ (b) $\mathbb{P}_3$ (c) $\mathbb{P}_6$

Figure 3.10: Two-dimensional OpenMP solver running on two AMD Opteron 6276 CPUs. $n_e = 20000$. Influence of the algorithms and of the compilers on the parallel performance.

reported on Hyperthreading technology applied to the Intel CPUs, with Xeon Phi the performance significantly improves when adopting 2 and 4 threads per core. In fact, when using all the available hardware threads, the execution time is almost halved. This results in a maximum speed-up of about 130.



(a) KNC (b) KNL

Figure 3.11: Two-dimensional OpenMP code running on a Xeon Phi CPUs. Influence of different algorithms and computational loads on the parallel performance.

Figure 3.11(b) displays the same numerical experiments on the newer and faster Xeon Phi KNL. Despite being able to deal with a maximum of 272 threads, a decrease in computational performance using four threads per core has been observed, and therefore the scalability was tested only up to 136 OpenMP threads. This behaviour is probably related to the improved vectorization capability of the newer Intel compiler. The results are similar to those obtained with the KNC CPUs, but the higher performance of this platform, which is formally three times faster in terms

of peak performances, makes more difficult to obtain optimal parallel efficiencies. This consideration holds true in particular with the smallest grid size and polynomial orders, *i.e.* the $\mathbb{P}_6$, $n_e = 1250$ and $\mathbb{P}_1$, $n_e = 5000$ simulations. In those cases, the partition algorithm performs as the best. For high computational loads, *i.e.* the $\mathbb{P}_6$, $n_e = 20000$, small differences between the three algorithms are observed, despite being the colouring algorithm slightly more efficient.

3.3.2 Performance comparison with the recent literature

Numerical experiments reveal an almost ideal parallel efficiency for a considerably large number of threads in a pure OpenMP mode. The achievement of such result does not seem to be trivial, in particular with unstructured CFD/CAA solvers.

Brus et al. [33] present a DG discretization is coupled with an explicit Runge-Kutta time integrator to solve the two-dimensional shallow water equations. This set of equations do not involve any diffusive term and it can be assumed similar to EE and LEE. The solver employed by Brus et al. is parallelized using an algorithm similar to the face partition. Finally, their numerical experiments are performed on test cases resulting in computational loads comparable to those reported in this paper. Nevertheless, the OpenMP parallel efficiency reported by the authors is clearly below the MPI scalability shown within the same work (Figures 16-18 in [33]). This happens even with 16 cores only (two Intel Xeon E5, Sandy Bridge, 3.1 GHz CPUs). Differently from Brus et al., all the OpenMP implementations reach the linear behaviour, meaning that they can be assumed comparable to an efficient MPI code.

Reuter et al. [35] show a $\mathbb{P}_1$ DG solution of the baroclinic shallow water equations. Here the authors report a satisfactory parallel performance, *i.e.* a speedup of 50 with 60 threads, on Intel Xeon KNC CPU. Their OpenMP strategy solves the race condition at faces using temporary arrays. The results for the test case 1 (Table 3 in [35]), obtained with an explicit Runge–Kutta scheme and neglecting the diffusion terms of the model, are comparable in terms of parallel performance with what reported here. Figure 3.11(a) of this paper shows that only the coarsest problem with the Face Partition and the Colouring algorithms, *i.e.* $\mathbb{P}_1$ and $n_e = 5000$, is not able to get the same parallel efficiency. It is worth noting the considerable lower number of mesh elements of the coarsest case hereby presented, *i.e.* $n_e = 5000$, when compared with the computations of Reuter et al., *i.e.* 98000. Using Hyperthreading technology, a speedup about 130 against 68 has been observed, but a different level of vectorization within the code probably affected this type of result.

Waltz et al. [60] present an edge-based finite element scheme for linear tetrahedral. A multi-stage explicit time integrator is used to solve the EE. To avoid data race in their OpenMP implementation, they use indirect store operations. Their scalability on Intel Xeon Phi KNC CPU is not completely satisfactory when several cores are used, being the efficiency above the 90% only up to 8 cores. This behaviour is also observed when they consider a very large problem with $2.8 \cdot 10^6$ mesh points.

Sato et al. [42] employs a colouring algorithm to parallelize using OpenMP a Gauss-Seidel based unstructured solver for steady laminar and Reynolds averaged Navier–Stokes equations in the context of finite volumes. Using all the 16 cores of platform

based on two octa-core Intel Xeon CPUs, they achieved an acceptable maximum speedup of about 14, but only when dealing with very large problems, *i.e.* more than 10^7 cells.

3.3.3 Performance of the Hybrid MPI/OpenMP

This sub-section analyses the performance of the hybrid MPI/OpenMP parallel implementation. It is worth pointing out that, when dealing with clusters of multi-core nodes, the impact of the hardware configuration on performance is even larger, since the inter-connecting network characteristics have to be taken into account. There are also many other computational details that can influence the parallel efficiency, such as the polynomial degree of the space discretization and the number and the types of the mesh elements. A complete and definitive study on the hybrid MPI/OpenMP parallelization is behind the scope of this paragraph. However, it is shown that, at least for some cases, the approach guarantees a sensible improvement in the parallel efficiency. Please note that, unless explicitly stated, the results reported in the following refer to the acoustic scattering of a sphere test case.

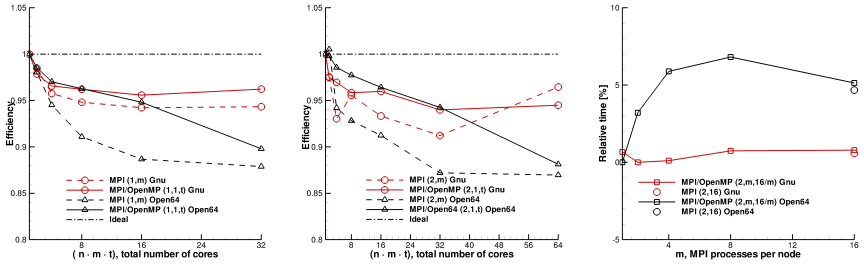
Multi-Core Clusters

Four different computer platforms have been considered: *i*) a small system based on two AMD Opteron 6276 machines, see Section 3.3.1, interconnected by a point to point Infiniband network; *ii*) the IBM-BlueGene/Q system named FERMI; *iii*) the LENOVO NeXtScale Cluster named MARCONI (both A1 and A2 partitions). FERMI is a massively parallel architecture consisting of 10240 nodes equipped by a IBM PowerPC A2 1.6 GHz chip with 16 cores per node; MARCONI-A1 connects 1512 compute nodes made by two 18-cores Intel Xeon E5-2697 v4 (Broadwell) 2.30 GHz; MARCONI-A2 is made of 3600 Intel Xeon Phi 7250 CPUs (KNL) 1.40 GHz computing nodes (described in Section 3.3.1). It is worth noting that both the PowerPC A2 and the KNL can execute concurrently up to four hardware threads, and the best performance are expected when multiple threads per core are used. Differently from Section 3.3.1, the mesh interconnecting cores and memory modules is the standard for MARCONI A2, *i.e.* the *cache mode* in which the faster MCDRAM is used as cache.

For the sake of compactness the use of the following notation to describe the parallel setting of a computation. An hybrid MPI/OpenMP computation is denoted by a triple of positive integers (n, m, t), where n is the number of computing nodes, m is the number of MPI processes running in each of node and t is the number of OpenMP threads for each MPI process. The product of these values is equal to the total number of threads used. For example, $(8, 2, 16)$ means that 8 nodes with 2 MPI processes for each node and 16 OpenMP threads for each MPI process are used for a computation, *i.e.* a total of 256 threads are employed. For a pure MPI run, only (n, m) is used, being $t = 1$.

Hybrid MPI/OpenMP scalability (Linearized Euler equations)

The system based on the AMD Opteron 6276 CPU has been used to assess the parallel performance in cases with a significant load per core. The two compilers presented in



(a) Efficiency, 1 node (b) Efficiency, 2 nodes (c) $(2, m, 16/m)$ relative time

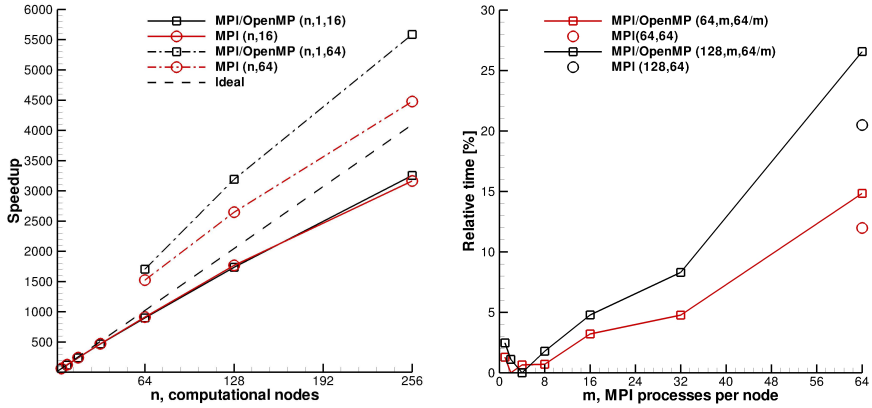
Figure 3.12: Acoustic scattering from a sphere. Opteron 6276 system

Section 3.3.1 have been employed. The first numerical experiments were performed within a single node only. The aim was to compare with each other the performance of the pure OpenMP and the pure MPI implementations. In fact, recent versions of the MPI library such as MVAPICH-2.2.1 use shared memory channels for the inter-node communications, and therefore the message passing between cores sharing the memory is faster then message passing between nodes.

Figure 3.12(a) shows that the parallel efficiency of the hybrid implementation is slightly higher if compared to the pure MPI one for all the compilers. Using both the nodes as $(2, 1, t)$ the single-node results are confirmed. Only when the computation lays on all the available 64 cores, the pure MPI solver compiled with GNU is more efficient than the hybrid one, see Figure 3.12(b). More generally, when both the cores of a pair are used the MPI/OpenMP advantage over the pure MPI slightly decreases. This behaviour deserves further investigations and it seems similar to that reported for the colouring and the face partition strategies for OpenMP.

Figure 3.12(c) shows the effect of varying the number of OpenMP threads per MPI process. The configurations considered are of the type $(2, m, t)$ with $(m \cdot t) = 16$. The relative wall clock time with respect to the best hybrid case, obtained by varying the combination between the MPI processes and the OpenMP threads, is also reported. It is clear that the first-touch policy allows the usage of a small number of MPI processes without any efficiency loss due to the 4 NUMA regions of the hardware. This effect is particularly clear when using the Open64 compiler. In fact, in this case the $m = 1$ choice revealed to be the best one. The pure MPI case $(2, 16)$ is also reported in the figure for comparison purposes. The difference between the pure MPI and the hybrid $(2, 16, 1)$ results represents the overhead cost of OpenMP and of the colouring algorithm when using a thread only. This value can be almost neglected for both the compilers.

Figure 3.13(a) displays the speedup obtained when running the same test case on FERMI. Using a number of threads equal to the number of cores, *i.e.* 16, a good parallel performance with both the pure MPI and the hybrid MPI/OpenMP implementations has been obtained. For example, the parallel efficiency at 4096 cores (22 elements per core) is 83.5% and 77.2% for then MPI/OpenMP and pure MPI implementations, respectively. When using 64 threads per node, corresponding to



(a) Strong scalability

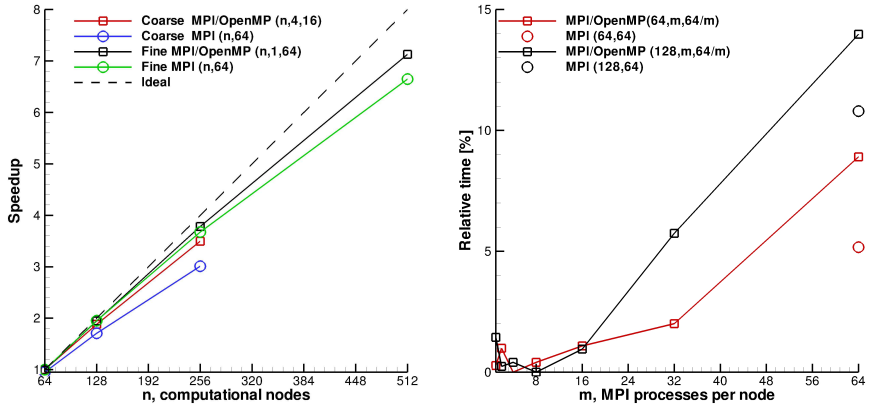
(b) $(n, m, 64/m)$ relative time

Figure 3.13: Acoustic scattering from a sphere. FERMI BlueGene/Q platform

4 hardware threads per core, the parallel performance of the hybrid solver largely outperforms that of the pure MPI one. This gain increases, for a given grid density, together with the number of computational nodes employed. Note that when using 16384 total threads each of them deals with only 5.5 grid elements on average.

Figure 3.13(b) shows the relative wall clock time when using all the 64 hardware threads of a node and varying the combination between the MPI processes and the OpenMP threads. Being this platform is of the UMA type, the best performances have been obtained using a small number of MPI processes, as expected. In this case, the values relative to the pure MPI implementation highlights that the overhead cost of OpenMP and of the colouring algorithm is quite low for the IBM `xl` compiler available on FERMI and it is around 5%. These results with 64 threads per node were confirmed also by computing the second test case, *i.e.* the directivity from a loudspeaker, on two grids consisting of $n_e = 109067$ hybrid elements and 576762 tetrahedral elements respectively, see Figure 3.14.

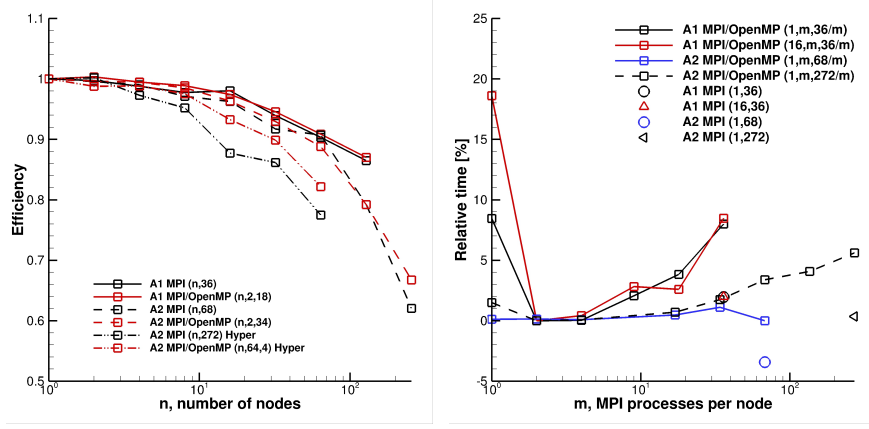
The scalability of the solver is assessed by computing the directivity from a loudspeaker using the finest grid on both MARCONI A1 and A2 systems, see Figure 3.15(a). As far as the single thread per core configuration is employed, the efficiency of the pure MPI solver is comparable to that of the hybrid MPI/OpenMP (solid and dashed lines). When enabling hyper-threading on the A2 system, an improvement in parallel efficiency is observed for the hybrid implementation (red, dash-dotted lines) compared to the pure MPI case (black, dash-dotted lines). As with FERMI, the influence on performance of the number of MPI ranks m within a single node has not been investigated. The influence of this parameter is shown Figure 3.15(b) by reporting the relative percentage gain/loss in terms of CPU time when varying m with a fixed number of threads per node. The pure MPI results reported in Figure 3.15(b) reveal that on A1 there is no OpenMP penalization due to the use of the different organization of the loop over the face in the colouring algorithm. The loss of compu-



(a) Speedup, 64 threads per node

(b) $(n, m, 64/m)$ relative time, coarse grid

Figure 3.14: Acoustic radiation of a speaker, coarse and fine grids. FERMI Blue-Gene/Q platform. The speedup is evaluated according to the 64 nodes, 1024 cores, performance



(a) Code scalability

(b) Choice of the number of MPI processes

Figure 3.15: Efficiency of the LEE solver on the MARCONI platforms, evaluated according to the performance of one node (36 and 68 cores, respectively). Acoustic radiation of a speaker test case.

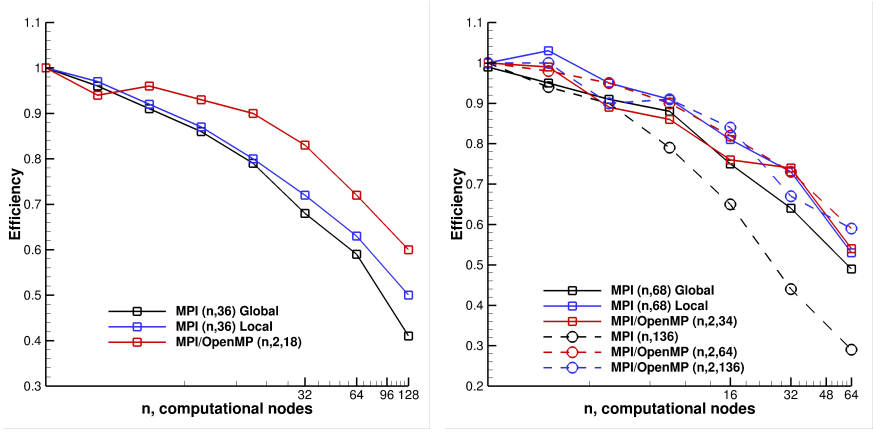
tational efficiency observed on A2 is about the 5% both for the 68 and the 272 cases, which results are similar to FERMI. When using A2 and 256 nodes (33 elements per core) a strong scaling speedup for the hybrid MPI/OpenMP implementation has been measured, roughly $1.2 \cdot 10^4$. This large value is particularly significant because it has been obtained on a large problem that can still be addressed in a serial computation on the same machine.

Hybrid MPI/OpenMP scalability (Navier–Stokes equations)

The suitability of the hybrid implementation proposed here in the context of a Navier–Stokes solver has been demonstrated by computing the parallel performance on both the MARCONI systems. As a representative flow problem the Taylor–Green vortex with a very high-order discretization, *i.e.* $\mathbb{P}_6$ and $n_e = 32^3$, has been considered. In addition, two types of partitioning strategies for the MPI solver are compared. The former, denoted as *global*, minimizes the overall communications between mesh element partitions. The latter, proposed in [61, 62], employs a "two-level" partitioning of the domain. The first-level decomposition is performed according to the number of nodes, while the second-level decompose the node-local partition according to the number of cores per node. In the following, the strategy will be called as *local* partitioning. This approach naturally minimizes extra-node communications, and guarantees that the same amount of MPI messages are exchanged in the pure MPI and hybrid MPI/OpenMP runs.

Figure 3.16(a) reports the results obtained on MARCONI A1. In this case a single thread per core configuration was used. The hybrid MPI/OpenMP solver, employed with two MPI partitions per node, outperforms the pure MPI implementation. The choice of the number of MPI partitions was consistent with the numerical experiments reported in Fig. 3.15(b). It is worth noting that, in case of a pure MPI implementation, using the *local* partitioning strategy improves the parallel efficiency if compared to the *global* one, especially when the largest number of nodes is used.

On the A2 platform, a different behaviour has been observed. The single thread per core configurations (solid lines in Figure 3.16(b)) show very similar behaviour when the MPI, both with the local and global partitioning approach, and the hybrid MPI/OpenMP implementations are used. Differently, when using hyperthreading,



(a) A1 (b) A2

Figure 3.16: Efficiency of the Navier–Stokes solver on the MARCONI platforms, evaluated according to the performance of one node (36 and 68 cores, respectively). Taylor–Green Vortex test case.

larger differences on the results arise (dashed lines). In this case the efficiency of the pure MPI solver drops down considerably, while the use of a hybrid MPI/OpenMP implementation allows to maintain as high parallel efficiencies as for the single thread configuration. This behaviour has been noticed both using a two-thread per core and a four-thread per core configuration. It is worth noting that, when the hyper-threading is employed, the performance of the code grows by a factor of 1.29 considering a single node computation, which makes the higher parallel efficiency of the OpenMP solver particularly attractive. Moreover, the results were found only weakly dependent to the number of MPI ranks used within the node of such platform, see Figure 3.15(b). This fact is due to the different architecture of the KNL node.

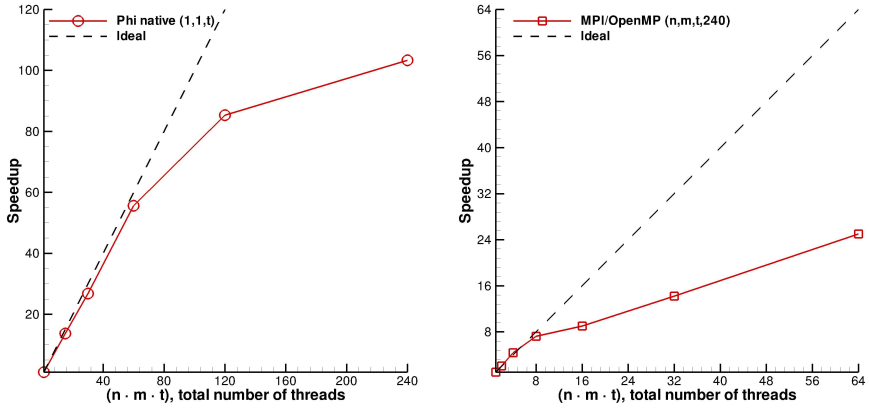
By comparing the the advantages of using a hybrid implementation in the NS or LEE context, a significant advantage for the former case is observed. This gain is due to the larger number of operations performed over each face for the viscous solver, which influence the amount of workload on the duplicated faces on the partition boundaries. Moreover, the measured scaling of the DG NS solver is comparable with that of similar and at the state-of-art codes. For example, Munz et al.in [63] report a strong scaling, performed on the CRAY-XE6 cluster, using form 11 to 64 AMD Opteron 32-core nodes of 61%. Considering that the numerical experiment range from 16 to 128 nodes on MARCONI A1, a strong scaling of 62% in pure MPI mode and 67% in the hybrid mode has been measured. Note that number of DOFs per core is 625 in the most parallelized case, which is a bit larger than that of the present computations, *i.e.* 597.

Finally, it is worth pointing out that a certain dispersion on the measured CPU times on the MARCONI A2 system was observed. Only the best-performing experiments have been reported in the paper. This behaviour explains why the A2 curves are generally less smooth, see for example Figure 3.16(b).

3.3.4 OpenMP & accelerators

Among the new opportunities offered by the new OpenMP 4.0 standard, hardware accelerators such as GPU and the Xeon Phi coprocessor can be easily considered. Here, the first naive attempt of using this opportunity is reported. For those experiments has been performed on the nodes of the GALILEO cluster hosted by CINECA, which owns two Xeon Phi KNC coprocessors per node. See Table 3.2 for the description of the platform. On this machine the scalability of the proposed implementations, for the sake of compactness not reported here, is very similar to that obtained on the MARCONI A1 platform reported in Figure 3.15.

The Xeon Phi hardware is accessed in the so-called *offload mode* trough the `!$OMP target` directives. The computation of the $\boldsymbol{R}_\kappa$ contribution to the residuals vector was assigned to the 240 threads of the Xeon Phi accelerator, see Figure 3.17(a) for the parallel performance of the solver on a single KNC. Using `!$OMP task` directives the latter operation was overlapped to the evaluation of the $\boldsymbol{R}_\sigma$ contributions performed in the host CPU. This parallel configuration has been denoted as $(n, m, t, 240)$, where 240 indicates the Xeon Phi threads employed for each MPI process. Since each node owns two coprocessors, $m = 2$ was used for these numerical experiments. In this



(a) Xeon Phi

(b) Xeon & Xeon Phi

Figure 3.17: Acoustic scattering from a sphere. GALILEO platform. In a) the Xeon Phi CPU is used in *native mode*, in b) as coprocessor in the *offload mode* (240 threads).

case the code uses the MPI library for the communications between nodes and the OpenMP paradigm to assign parts of the execution to the CPU and coprocessor cores. A clear performance gain can be obtained only when few cores of the host CPU are employed. In fact the Xeon Phi CPU shows good parallel performances, see Figure 3.17(b), but a high serial wall clock time. In fact, using the proposed solver, a Phi core is about 24 times slower than one Xeon CPU core, while the 240 Phi threads are about 1.3 times slower than the 8 Xeon cores. Bearing that in mind, a 33% time reduction in using the coprocessor together with a single CPU core has been observed. The scaling follows the ideal law up to 8 host cores. However, using all the sixteen CPUs of a node and the two coprocessors the advantage is limited to the 17% only.

Unfortunately it is not possible to extrapolate this result to multi-nodes configurations. Figure 3.17(b) shows the low parallel performance of the code, with the speedup computed according to the measured serial execution time of the configuration $(1, 1, 1, 240)$. The main reason for this behaviour is the low speed of the data transfer on the PCI express bus, the associated overhead becomes more relevant when decreasing the parallel granularity. This type of data transfer is needed both to copy the current $\boldsymbol{W}$ vector from the host CPU to the accelerator and to get the $\boldsymbol{R}_\kappa$ array in the node main RAM. Further investigation on the efficient usage of accelerators within the OpenMP paradigm is the subject of ongoing work.

3.4 Summary

In this chapter an investigation of OpenMP and hybrid MPI/OpenMP implementations of a high-order explicit-in-time DG solver for CFD and CAA applications is presented. Despite OpenMP is considered to be the most straightforward way to parallelize an application in the context of shared memory systems if compared to

MPI, many issues have to be considered. In fact, the NUMA design of many recent hardware platforms, as well as the effectiveness of the compiler when including the OpenMP directives, can significantly reduce the computational efficiency. While the former problem can be addressed using the first touch policy, which can partially mitigate some inefficiencies by making the code NUMA-aware, the latter can be faced only dealing with OpenMP in an MPI-like manner, using a domain decomposition algorithm. However, some powerful features of OpenMP are lost, together with the simplicity of the parallelization based on work-sharing directives. Nevertheless, the implementation of a simple colouring algorithm shows high parallel efficiencies. When this approach is used within a MPI code, the resulting hybrid strategy improves the scalability on large HPC facilities consisting of multi- and many-core nodes. This is particularly true when the set of NS equations is considered. In fact, the discretization of the viscous terms involve more floating point operations in the loop over faces than EE/LEE. The hybrid approach also showed a clear gain in parallel performance over pure MPI when the CPU is able to handle more than one hardware threads per core.

Other reasons promoting the use of OpenMP exist. In fact, the recently released 4.0 standard can deal with hardware accelerators too. Although preliminary, encouraging results have been obtained by exploiting these new OpenMP features. Moreover, the possibility to improve further the performance of the solver using an asynchronous parallel model can be investigated, for example using MPI's one-sided communication functionalities and OpenMP's pragma directives for task-based parallelism. To this end, the use of OmpSs [64], a variant of OpenMP focused on exploiting irregular and asynchronous parallelism and also able to deal with distributed memory machines and heterogeneous architectures, as well as other alternatives of this type such as FREDDO [65], can be considered in the next future.

Finally, it is worth to remark that an extension of the approach proposed to implicit-in-time discretizations would require the implementation of a hybrid MPI/OpenMP parallelization of the residual's Jacobian evaluation, the preconditioner assembly and application, as well as the iterative solver. Such an implementation is beyond the scope of the thesis, and it will be subject of forthcoming research.

Chapter 4

Matrix-free implicit time integration

This chapter deals with computational efficiency of implicit-in-time DG discretizations. In fact, it can be recognized that the memory footprint of the residual's Jacobian scales at k^{2d}, and it is clear that for high-order of polynomial approximations the high demand of memory and the large CPU times for the matrix assembly limit the applicability of those schemes for industrially-relevant computations. However, the implementation of a matrix-free (MF) generalized minimal residual method (GMRES) solver appears to be a viable way to overcome those limitations. Previous studies considered the possibility of using memory-saving implementations of the iterative solver. For example, in [66], a matrix-free GMRES solver was used to solve steady compressible flows. In [67] a matrix-free approach is employed in the context of several time integration strategies with applications to unsteady, laminar two-dimensional problems. In [68] the use of a matrix-free GMRES is proposed for the solution of both the primal and adjoint problem with applications to compressible Navier–Stokes equations. However, none of those work explicitly provide estimates of the numerical efficiency of a matrix-free iterative solution strategy in comparison to a matrix-based one. Interestingly, despite several authors [11, 8, 7, 69, 70, 71, 72] make use of one those two approaches, it is not clear how the methods compare with each other, particularly for large unsteady problems.

The aim of the chapter is therefore to compare the performance of a matrix-free implementation of the iterative solver to a standard matrix-based one, where the residual's Jacobian is analytically computed and stored in memory. An important part of the Chapter is the preconditioning strategy, which is mandatory to improve the efficiency and reduce time-to-solution of the iterative strategy. In the MB case the use of the iteration matrix itself to build such operator seems the most natural choice, for example as an incomplete lower upper factorization (ILU) or a block Jacobi (BJ), since the Jacobian matrix has to be evaluated explicitly and stored to solve the system. On the other hand, the MF approach does not need the Jacobian to advance the solution in time, and therefore the preconditioner can be computed more cheaply using an approximation of the iteration matrix. For example, it is possible to freeze the evaluation of the Jacobian blocks for some time steps, and/or to use a computationally cheaper element-wise version of the block Jacobi (EWBJ), which reduces also the memory footprint of the application. These properties become particularly attractive in a matrix-free framework because the approximation of the Jacobian is may be an unstable operation within matrix-based contexts, even using Rosenbrock $\mathcal{W}$-methods, which can produce a higher time integration error or loss of the algorithm stability for large time step sizes.

In order to compare MB and MF implementations, in this chapter incompressible flows are considered. The reason for this choice is ascribed to what reported in [16]. In fact, it is demonstrated that in the incompressible case the Newton's linearization of a DG discretization results in a stiffer Jacobian matrix, thus the advantages related to its approximation should be significantly larger in the case of DG discretization of the compressible fluid-dynamics governing equations, widely adopted in literature. It is here retained that the incompressible case is also particularly interesting since even the explicit/semi-explicit methods requires a linear solver for the pressure DoF, see for example [73, 74, 75]. Finally, the author remarks that the use of linearly implicit Rosenbrock schemes calls for linear system solutions only, but this can be considered as a building block for other implicit and high order time integration schemes also, see for example Diagonally Implicit Runge–Kutta (DIRK) or those based on Backward Differentiation Formulae (BDF). With such schemes, the advantages of using MF solvers may be even larger since multiple Jacobian evaluation per time step are usually required.

The test case considered is the two-dimensional incompressible laminar traveling waves. The results in this Chapter, prove that, when adopting the same preconditioner, there exists a wide range of time step sizes for which both the algorithms behave identically. In fact, only when very small values of GMRES relative tolerance (tol_r) are needed the MF approximation of the Jacobian appears resulting in a greater difficulty to reach a target convergence of the linear system solution. Fortunately, such small tolerances are recommended only for very small time step sizes in order to preserve the theoretical accuracy order in time. Nevertheless, with implicit time integration strategies the time step size should always be quite large, otherwise explicit methods are probably more efficient. In these circumstances it is really important to avoid over-solving the linear system from the computational point of view. It is suggested that only with a reasonable tol_r the comparison between the algorithms is acceptable and that the above mentioned limit of the MF algorithm is not a real issue. For this purpose, a criterion for a reliable selection of this parameter is also introduced, which is based on the embedded Runge–Kutta scheme, and the advantages of using such procedure are discussed. See [76, 77] for additional details.

4.1 Matrix-free GMRES algorithm

In this Chapter the restarted GMRES method with right preconditioning is employed to solve the linear systems arising from the implicit time discretization of the equations. Algorithm 10 provides implementation details. Starting from an initial guess $\delta \boldsymbol{w}_h^0$ and given a fixed number of Krylov subspaces s, the method seeks the solution by linearly combining the basis of the spaces through the solution of a minimization problem. The Krylov subspaces are obtained through a standard Arnlodi process, which simply constructs an orthogonal basis of the preconditioned Krylov subspace

$$\text{Span}(\boldsymbol{r}_h^0, \boldsymbol{G}_h \boldsymbol{P}_h^{-1} \boldsymbol{r}_h^0, \ldots, (\boldsymbol{G}_h \boldsymbol{P}_h^{-1})^{s-1} \boldsymbol{r}_h^0) \tag{4.1}$$

by a modified Gram-Schmidt process, in which the new vector to be orthogonalized is obtained from the previous vector process. The last step of Algorithm 10 forms the

Algorithm 10 Restarted GMRES algorithm (right-preconditioned)

1: Choose $\delta \boldsymbol{w}_h^0$ and define s the dimension for the Krylov-Subspaces. Define $\bar{\boldsymbol{H}}_m \in \mathbb{R}^{s+1} \otimes \mathbb{R}^s$ and initialise the components $\mathsf{h}_{i,j}$ to zero;
2: *Arnoldi Process:*
3: Compute $\boldsymbol{r}_h^0 = \boldsymbol{g}_h - \boldsymbol{G}_h \delta \boldsymbol{w}_h^0$, $\beta = \|r_h^0\|_2$ and $\boldsymbol{v}_h^1 = r_h^0/\beta$.
4: **do** $j = 1, s$
5: Compute $\boldsymbol{z}_h^j = \boldsymbol{P}_h^{-1} \boldsymbol{v}_h^j$;
6: Compute $\boldsymbol{w}_h = \boldsymbol{G}_h \boldsymbol{z}_h^j$;
7: **do** $i = 1, j$
8: set $\mathsf{h}_{i,j} = (\boldsymbol{w}_h, \boldsymbol{v}_h^j)$
9: set $\boldsymbol{w}_h = \boldsymbol{w}_h - \mathsf{h}_{i,j} \boldsymbol{v}_h^j$
10: **enddo**
11: Compute $\mathsf{h}_{(j+1),j} = \|\boldsymbol{w}_h\|_2$ and $\boldsymbol{v}_h^{j+1} = \boldsymbol{w}_h / \mathsf{h}_{(j+1),j}$
12: **enddo**
13: Define $\boldsymbol{V}_s = [\boldsymbol{v}_h^1, \boldsymbol{v}_h^2, \ldots, \boldsymbol{v}_h^s]^T$
14: *Form the approximate solution:*
15: Find $\boldsymbol{y}_h^s = min(\beta \boldsymbol{e}_1 - \bar{\boldsymbol{H}}_s \boldsymbol{y}_h)$, with $\boldsymbol{e}_1 = [1, 0, \ldots, 0]^T$
16: Compute $\delta \boldsymbol{w}_h^s = \delta \boldsymbol{w}_h^0 + \boldsymbol{P}_h^{-1} \boldsymbol{V}_s \boldsymbol{y}_h^s$
17: **if** $\|\boldsymbol{g}_h - \boldsymbol{G}_h \delta \boldsymbol{w}_h^s\|_2 \leq \varepsilon$ **then**
18: EXIT
19: **else**
20: set $\delta \boldsymbol{w}_h^0 = \delta \boldsymbol{w}_h^s$
21: **go to** 2
22: **end if**

solution as a linear combination of the preconditioned vectors

$$\boldsymbol{z}_h^i = \boldsymbol{P}_h^{-1} \boldsymbol{v}_h^i, \quad i = 1, \ldots, s. \tag{4.2}$$

by applying the preconditioner operator $\boldsymbol{P}_h^{-1}$ to the matrix $\boldsymbol{V}_s$. The iterative strategy is stopped if the final residual is smaller than $\varepsilon = \boldsymbol{r}_h^0 tol_r$, with $\boldsymbol{r}_h^0$ the residual of the linear system at the beginning of the iterative process and tol_r a user-defined residual drop. It is worth noting that the right preconditioning approach is employed and thus the definition of the residual is not affected by changing the preconditioning definition.

Standard matrix-based (MB) approaches typically employed for steps 5 and 6 of Algorithm 10 consist of calculating and storing both the Jacobian matrix and the preconditioner. Since the Jacobian matrix size scales as n_v^2, and thus as k^{2d}, this approach requires a high amount of RAM memory to be stored as well as a large CPU time for its computation. In addition, lagging the evaluation of the Jacobian evaluation over multiple time steps may be cumbersome, since it can reduce the temporal accuracy and/or provide stability problems.

A matrix-free implementation of the GMRES solver can be obtained by noting that it calls for matrix-vector products of the type $\boldsymbol{G}_h \delta \boldsymbol{w}_h$. Recalling that $\boldsymbol{G}_h = (1/\alpha \delta t)\boldsymbol{M}_h + \boldsymbol{J}_h$ with α a scheme-specific coefficient, using Eq. (2.30) one can approximate the matrix-vector product (2.19) via a double bilinear form evaluation [78]

such that

$$\frac{1}{\alpha\delta t} m_h(\delta \boldsymbol{w}_h, \boldsymbol{z}_h) + j_h(\boldsymbol{w}_h^n, \delta \boldsymbol{w}_h, \boldsymbol{z}_h) = \frac{1}{\Delta}\left(g_h(\boldsymbol{w}_h^n + \Delta\, \delta \boldsymbol{w}_h, \boldsymbol{z}_h) - g_h(\boldsymbol{w}_h^n, \boldsymbol{z}_h)\right) \quad (4.3)$$

$\forall \boldsymbol{z}_h \in [\mathbb{P}_d^{k\ell}(\mathcal{T}_h)]^{d+1}$, where

$$g_h(\boldsymbol{w}_h, \boldsymbol{z}_h) = \frac{1}{\alpha\delta t} m_h(\boldsymbol{w}_h, \boldsymbol{z}_h) + f_h(\boldsymbol{w}_h, \boldsymbol{z}_h). \quad (4.4)$$

Here the Jacobian matrix has been replaced by its first order finite difference approximation involving a numerical perturbation Δ and a double evaluation of the residuals bilinear form. The numerical perturbation appearing in Eq. (4.3) can be computed according to [79],

$$\Delta = \epsilon \frac{\sqrt{1 + \|\boldsymbol{w}_h\|_{L^2(\Omega)}}}{\|\delta \boldsymbol{w}_h\|_{L^2(\Omega)}}, \quad (4.5)$$

where the ϵ is an user defined input. It is known that the choice of this parameter can be a trick between round-off and truncation errors, nevertheless the value of 10^{-9} is assumed for all the computations without any numerical issues. See [78, 80] for complete review of the method and an in depth discussion of numerical and implementation details.

Eq. (4.3) shows pros and cons over standard matrix-based time integration. First of all, the matrix-vector product is avoided and, at least in theory, the Jacobian computation can be neglected. For practical applications its evaluation is still required to build the preconditioner operator which, in this case, can be obtained in-place, reducing the memory footprint of the code as only one matrix is allocated. Moreover, the preconditioner can be reused for some consecutive iterations (skipping the Jacobian evaluation) without loosing the formal accuracy of the time integration scheme. Such property can be conveniently exploited to increase the computational efficiency of the solution strategy in some circumstances, for example those concerning a high order of polynomial approximation and small time steps, as it will be clear in the remaining of the Chapter. It worth noticing that lagging the Jacobian evaluation becomes more beneficial for the solver efficiency as the computational effort for a single evaluation grows. Being the computational cost of the Jacobian evaluation of the order of $\mathcal{O}(m^2 k^{3d})$, lagging its evaluation is particularly attractive for a three-dimensional problem, $d = 3$, and for a large polynomial order k.

On the other hand, the matrix-free solver requires a residual evaluation per GMRES iteration. From the computational time point of view, the matrix-free iteration can be more expensive than the matrix-based counterpart depending on how much the residual evaluation costs if compared to the matrix-vector product. Recalling the asymptotic scalings of the operation counts per element of each case [70], the matrix-vector product scales as n_v^2, while the residual evaluation as $(n_{ge} n_v)$, where n_{ge} is the number of gauss integration points. Since $n_{ge} \sim k^d$, they scale equally with the polynomial order and, as k increases, a similar behaviour for the two strategies is expected. It is worth mentioning that curved mesh elements call for the use of high order quadrature formulas, and thus a higher value of n_{ge}. This peculiarity will be discussed in the following chapters.

4.2 GMRES single-grid preconditioning

The numerical efficiency of implicit schemes is strongly affected by the preconditioner operator. In fact, the more the preconditioner is close to the inverse of the iteration matrix $\boldsymbol{G}_h^{-1}$, the lower the number of GMRES iterations required to reach the target residual drop tol_r. However, it is typically observed that very effective preconditioners require large computing times both for their evaluation and their application, which may increase the solution time. Finally, it is worth mentioning also the memory footprint of the application. In fact, even using a matrix-free implementation of the matrix-vector products, the preconditioner still needs to be stored and the overall memory footprint may be still fairly large. The optimal strategy is necessarily a tradeoff between those aspects. In this chapter common single-grid preconditioning approaches are introduced to precondition the GMRES solver.

Incomplete lower upper factorization and block Jacobi

Three different approaches have been here implemented and compared. The first one is the incomplete lower-upper factorization of the whole iteration matrix $\boldsymbol{G}_h$, which is evaluated analytically, with zero filling (ILU(0)). Thanks to the zero-filling, the preconditioner assumes the same sparsity pattern of $\boldsymbol{G}_h$, and therefore its memory footprint scales as $n_e(n_f+1)(mn_v)^2$. The use of this approach within a matrix-free context reduces at least the 50% of memory allocated for the implicit time integration of the equations, as the factorization can be performed in-place. It is worth niticing that this preconditioner is obtained by definition in a serial computation. When applied in parallel, this strategy will be referred to as block Jacobi (BJ) when used in parallel computations, as the ILU(0) is applied to the diagonal partition-wise block of the Jacobian. Note that the parallel implementation of the ILU(0) algorithm reduces the preconditioner efficiency as the number of MPI ranks increase. The general implementation of ILU allows to control the levels of fill by using the parameter j, and it is generally denoted as ILU(j). This possibility will be explored in Chapter 5.

Additive Schwarz method

A variant which partially compensates this effect is the Additive Schwartz Method (ASM), that extends each domain partition with a number of overlapping layers that are used for the computation of the ILU(0). In this way the preconditioner efficiency typically increases in parallel computations since the number of GMRES iterations are lower than those of a BJ. However, the use of such preconditioner has to be avoided in some cases, *i.e.* when using a small number of elements per partition, because the higher amount of MPI communications as well as the larger memory footprint may result in a less efficient procedure if compared to the BJ.

Element-wise block Jacobi

A third preconditioner that has been considered here is the so called element-wise block-Jacobi (EWBJ). This preconditioner is assembled by performing the LU of the

diagonal, element-wise blocks of the Jacobian neglecting all the off-diagonal contributions. By definition, this operator has only one non-null entry in each block row and thus requires a very low memory allocation if compared to the full matrix. This approach becomes particularly attractive when coupled with the matrix-free GMRES solver. In this case the calculation of the off-diagonal blocks of the Jacobian can be completely avoided as they are not needed to advance the solution in time. It has been verified that for a three-dimensional solution, employing a $\mathbb{P}_6$ discretization and hexahedral mesh elements, the EWBJ preconditioner costs nearly two and a half times less than the ILU(0)/BJ, while only the 7% of the memory footprint is formally allocated to deal with the implicit time stepping. Another advantage of the approach is that its parallelization is trivial, being the LU factorization performed in a local-to-each element fashion. Numerical experiments show that, despite not particularly well performing on two-dimensional cases on small serial computations, this strategy becomes very efficient for moderately stiff three dimensional problems, *i.e.* when the grid is regular and shows moderately anisotropic and stretched elements at the walls. It is important to point out that this simple preconditioner, which provides the minimum assembly times as well as the maximum memory saving within a matrix-free framework, is effective for simple two-dimensional and three-dimensional test cases only. For stiff space discretizations, the solution strategy does not work as the number of iterations to complete the iterative process grow dramatically. This limits the applicability of such memory-saving algorithm to general problems of arbitrary complexity. Finally, important to remark that the BJ algorithm, being applied to the partition-wise block of the iteration matrix, tend to be equal to the EWBJ in the limit of one mesh element per partition.

4.3 Numerical experiments

In this section incompressible problems are considered. To do so, the four-stage L-stable ROSI2PW [81] scheme will be employed to assess the computational performances of the proposed strategy. This scheme is formally designed for partial differential algebraic equations (PDAEs) of index 2 and shows several advantages over other Rosenbrock schemes as the order-three, three-stage ROS3P [29], and the order-four, six-stage RODASP [82] which are formally designed for PDAEs of index 1 (even if they proved to retain the theoretical accuracy order solving NS equations [1, 82, 16]). In fact, as reported in [81], the ROSI2PW is stiffly accurate and it shows improved convergence when inexact Jacobians are employed ($\mathcal{W}$-property). This fact becomes attractive for matrix-based approaches because it allows to formally freeze the Jacobian evaluation for some time steps while maintaining the theoretical convergence order (see Section 4.3). It is important to stress the fact that the matrix-free implementation does not need the $\mathcal{W}$-property, which is exploited in this paper only for comparison purposes.

The numerical features of the proposed strategies have been evaluated by solving the laminar travelling waves (TW) test case. The problem is defined on a full-periodic square domain sized $[0.25, 1.25]\times[0.5, 1.5]$, where an exact solution of the two-dimensional incompressible Navier–Stokes equations can be found, in non-dimensional

form, as

$$\begin{aligned} u(x,y,t) &= 1 + 2\cos(2\pi(x-t))\sin(2\pi(y-t))e^{-8\pi^2 t/Re} \\ v(x,y,t) &= 1 - 2\sin(2\pi(x-t))\cos(2\pi(y-t))e^{-8\pi^2 t/Re} \\ p(x,y,t) &= -(\cos(4\pi(x-t)) + \sin(4\pi(y-t)))e^{-16\pi^2 t/Re} \end{aligned} \tag{4.6}$$

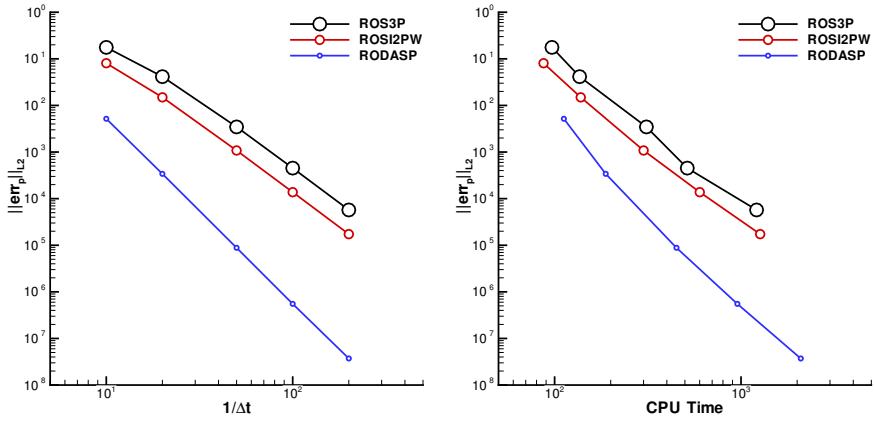
A smooth initial condition for the numerical solver can be conveniently set using $t = 0$ on the system (4.6) and projecting the analytical solution into the modal basis of the DG space. This investigation aims at assessing the performance of the algorithms exploring several solver settings. In particular, the effects of the time step size, the linear system relative tolerance tol_r, the Jacobian lagging, the polynomial order and grid size, and the scalability will be addressed. The numerical experiments were conducted simulating a single convective period, *i.e.* a physical time of $T = 1$. Unless differently specified, all the solutions have been obtained in serial using the ROSI2PW.

4.3.1 On the ROSI2PW performances

The ROSI2PW scheme is here compared to the ROS3P (three-stage, order three) and RODASP (six-stage, order four) schemes. Fig. 4.1(a) shows the L_2-norm of the error on the pressure at the time $t = T$, namely $\|err_p\|_{L_2}$, computed on a 16×16 square grid at $Re = 100$, using a $\mathbb{P}_6$ discretization and different time step sizes ($T/10$, $T/20$, $T/50$, $T/100$, $T/200$). Such discretization ensures that the spatial error is always some orders of magnitude lower than the time discretization error. It is shown that the implementation of all the three schemes converge with their theoretical order, *i.e.* three for the ROS3P and ROSI2PW and four for the RODASP as the δt decreases. In particular, the ROSI2PW scheme operates at a lower $\|err_p\|_{L_2}$ than the ROS3P, despite showing the same convergence order, due to the higher number of stages employed. Fig. 4.1(b) reports the error versus the CPU time for the same settings. For a given error level, the higher the order of the scheme or the number of stages, the lower the CPU time required to integrate the equations. Those considerations, independent of the scheme used neither of the preconditioner employed, have been obtained using an ILU(0)-MB strategy with a tolerance on the linear system of $tol_r = 10^{-8}$.

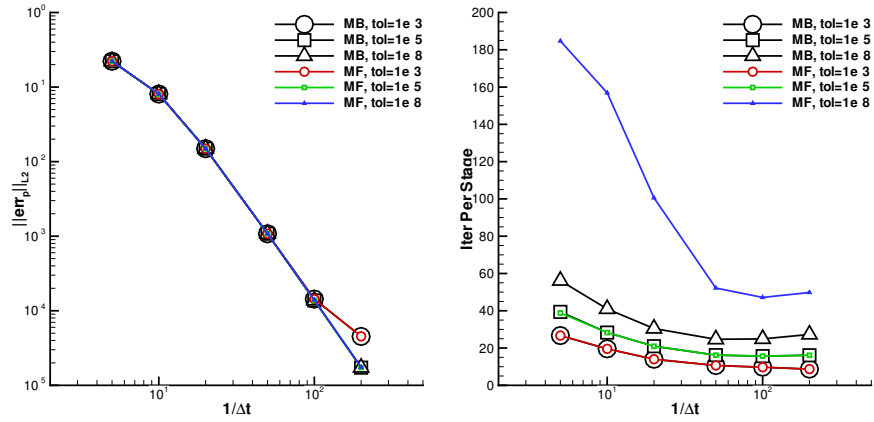
4.3.2 On the GMRES solver tolerance effects

The effects of the relative tolerance used to estimate the convergence level of the GMRES linear system within each stage of the Runge–Kutta scheme on the global solver efficiency, when using matrix-based and matrix-free strategies, are here investigated. For this purpose, a comparison of the performance obtained by using three different tolerance levels, $tol_r = 10^{-3}$, 10^{-5} and 10^{-8} is reported. Fig. 4.2(a) shows that, for all the tolerances employed, the error on the pressure $\|err_p\|_{L_2}$ is consistent between the strategies when using the same tol_r. Fig. 4.2(b) reports the average number of iterations per stage. While for relatively large tolerances ($tol_r = 10^{-3}$, 10^{-5}) the average number of GMRES iterations per stage does not change between the MB



(a) $\|err_p\|_{L_2}$ v.s. $1/\delta t$ (b) $\|err_p\|_{L_2}$ v.s. CPU time

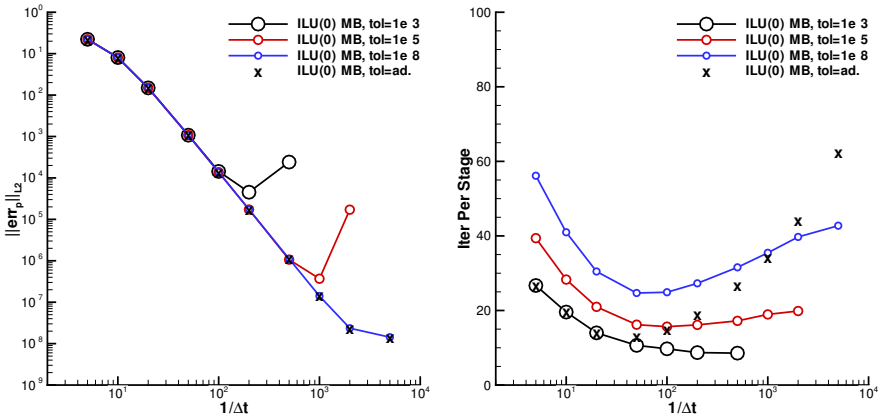
Figure 4.1: TW problem on a 16×16 square grid at $Re = 100$, $\mathbb{P}_6$ discretization, ILU(0)-MB solver, fixed system tolerance $tol_r = 10^{-8}$.



(a) $\|err_p\|_{L_2}$ vs. $1/\delta t$ (b) Iterations per Stage vs. $1/\delta t$

Figure 4.2: TW problem on a 16×16 square grid at $Re = 100$, $\mathbb{P}_6$ discretization, ILU(0)-MB vs. ILU(0)-MF solvers for various fixed system tolerance tol_r.

and MF solvers, for the lowest value ($tol_r = 10^{-8}$) the situation is quite different. In fact, the MF solver requires in this case a considerably higher number of GMRES iterations to converge, which is clearly detrimental for the computational efficiency. This fact may be ascribed to the finite difference approximation of the matrix-vector product. When tol_r is comparable with the ε of Eq. (4.5), the GMRES convergence rate reduces and therefore the number of iterations required to reach convergence increase. For higher values of tol_r, such error is negligible and the solvers behave the



(a) $\|err_p\|_{L_2}$ vs. $1/\delta t$

(b) Iterations per Stage vs. $1/\delta t$

Figure 4.3: TW problem on a 16×16 square grid at $Re = 100$, $\mathbb{P}_6$ discretization, ILU(0)-MB solver, for fixed system tolerances ($tol_r = 10^{-3}$, 10^{-5}, 10^{-8}) and adaptive tolerance ($tol_r = ad.$) given by Eq. (4.7).

same also from the GMRES iterations viewpoint.

Fortunately, not always such small values of tol_r are required to obtain a good solution. Fig. 4.3(a) shows for the ILU(0)-MB setting only, and for a wider range of time step sizes (up to $\delta t = T/5000$), the effects of the linear system tolerance on the output error $\|err_p\|_{L_2}$. If a large tolerance is employed when using a small time step, the system is under-solved and therefore the global error is dominated by the GMRES solution error. In this case the time integration error does not follow anymore the theoretical convergence rate of the scheme. Conversely, comparing Fig. 4.3(a) with Fig. 4.3(b), where the average number of iterations per stage is reported, it can be observed that if a large time step size and a small value of tol_r is used, the GMRES linear system is over-solved and unnecessary iterations are performed. Further details on the relationship between time step size, relative tolerance and solution quality can be found in [83].

In order to avoid both those effects, an adaptive strategy for the evaluation of the largest relative tolerance to solve adequately the GMRES system may be used. If $\boldsymbol{w}_h^n$ is the solution vector at the time step n, and the $\widetilde{\boldsymbol{w}}^n$ is the output of the embedded Runge–Kutta scheme, one can use the formula

$$tol_r^{n+1} = \min(\alpha_n/3, 10^{-3}) \tag{4.7}$$

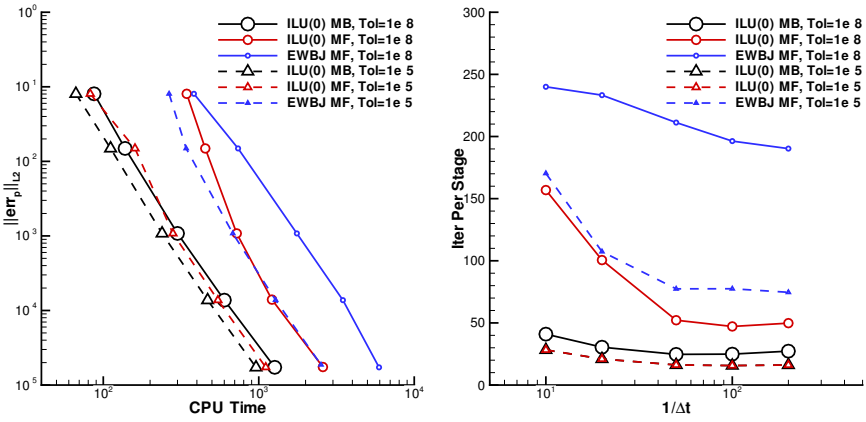
with $\alpha_n = \|\boldsymbol{w}^n - \widetilde{\boldsymbol{w}}^n\|_{L_2}$, to ensure that the GMRES solution error is always below the time discretization error. The proposed strategy employs a 1/3 safety factor on the estimated time discretization error, which has been calibrated on this two-dimensional study, and requires to solve the linear system with a minimum of three orders of magnitude. Such strategy is inspired to what is typically done with adaptive time stepping strategies, see for example [84], while the use of the error estimate at

the time step n to solve the GMRES system at the step $n+1$ is retained to be acceptable for the applications here considered, where the physical time scales are somehow constant over the time steps. Note that the same safety factor has been also employed for three-dimensional turbulent test cases without any numerical issue. Note also that thanks to the properties of the ROSI2PW scheme and its embedded, the error estimation is reliable also when INS equations are considered. In fact, as happens in the context of adaptive time stepping strategies, the error estimate is referred to the local error (evaluated on a single step) of the embedded scheme which is of the same order of accuracy of the full-order Runge–Kutta scheme. In this case, the scheme designed for PDAEs of index 2 (i.e. the INS equations) and therefore it can be verified that $\alpha_n \sim \mathcal{O}(\delta t^3)$. Other embedded Rosenbrock schemes, for example that of the ROS3P, does not show such property and therefore the adaptive strategy for tol_r cannot be formally employed in the context of incompressible flows.

Fig. 4.3 shows the effects of the adaptive tolerance on the solver performance (see black crosses). The use of this strategy produces an $\|err_p\|_{L_2}$ dominated only by the time discretization error, requiring the minimum number of iterations per stage. Note that both having very small and very large time step sizes produce high condition numbers of the iteration matrix, and therefore the linear system requires a higher number of GMRES iterations to reach the target accuracy. This fact may be explained recalling the definition of $\boldsymbol{G}_h = (1/\alpha\delta t)\boldsymbol{M}_h + \boldsymbol{J}_h$: in the former case the dominance of the diagonal part of the iteration matrix is reduced by large δt, while in the latter this behaviour is related to the absence in $\boldsymbol{M}_h$ of the diagonal entries corresponding to the pressure DoF proper of the incompressible discretization. It is worth mentioning that for the last two time steps of Fig. 4.3, $T/2000$ and $T/5000$, the error $\|err_p\|_{L_2}$ is dominated by the space discretization (reducing tol_r does not allow to maintain the asymptotic convergence order).

Considering an interval of time steps where the temporal integration error decreases asymptotically with all the tolerances employed, *i.e.* for $T/10$, $T/20$, $T/50$, $T/100$ and $T/200$, the computational performance of three solver settings (ILU(0)-MB, ILU(0)-MF, EWBJ-MF) are compared on a 16×16, $\mathbb{P}_6$ space discretization for two values of tol_r. Fig. 4.4(a) shows the pressure error $\|err_p\|_{L_2}$ versus the CPU time, while Fig. 4.4(b) reports the average number of iterations per stage for the different time step sizes considered. As already shown previously, the matrix-free approach performs very poorly when solving the GMRES system with very small values of relative tolerance ($tol_r = 10^{-8}$), see solid lines of Fig. 4.4(a). This consideration holds true for both the preconditioners employed, *i.e.* the ILU(0) and the EWBJ, if compared to the MB case. While the reduction of the time step size seems to mitigate those detrimental effects, the ILU(0)-MF results at least 2 times slower than the ILU(0)-MB, while the EWBJ-MF increases the CPU time at least by a factor of six. Therefore, it is not recommended to use the matrix-free approach, in any form, within such operating conditions.

The situation is quite different when a sufficiently large tolerance ($tol_r = 10^{-5}$) is employed to solve the GMRES systems (dashed lines, triangular symbols on Fig. 4.4). As expected, when the same ILU(0) preconditioner is employed, the matrix-free and the matrix-based approaches behave similarly and are able to solve the GMRES



(a) $\|err_p\|_{L_2}$ vs. CPU time

(b) Iterations per Stage vs. $1/\delta t$

Figure 4.4: TW problem on a 16×16 square grid at $Re = 100$, $\mathbb{P}_6$ discretization, ILU(0)-MB, ILU(0)-MF and EWBJ-MF solvers, for various fixed system tolerance tol_r.

system with an identical number of GMRES iterations per stage. It is interesting to notice that for this space discretization (two-dimensional $\mathbb{P}_6$ setting) the costs of a residual evaluation are comparable but slightly higher than the costs of a matrix-vector product, and similarly the CPU times measured between the ILU(0)-MB and ILU(0)-MF solvers. Since the memory saving related to the implicit solution strategy reaches the 50% due to the in-place factorization of the preconditioner, one may consider using this scheme for computational efficiency. The EWBJ-MF (blue curve Fig. 4.4), although reducing the memory requirements by the 90%, is highly penalized by the time spent on the GMRES loop due to the use of a cheaper and less efficient preconditioner, and the penalty factor reaches 2.5 within the smallest time step.

4.3.3 Jacobian lagging

The effects of lagging the Jacobian evaluation for n time steps, on the same TW numerical setup for the three aforementioned solution strategies, using $tol_r = 10^{-5}$, and the ROSI2PW scheme are here investigated. The lagging parameter was set as $n = 10$, which revealed to maximise the performances at the smallest time step sizes. Note that an adaptive strategy to estimate n can be also performed, see for example [85]. In this section two aspects are considered. The first one involves the convergence order of the output error $\|err_p\|_{L_2}$. The second one deals with the computational efficiency.

Fig. 4.5 shows that the lagging does not affect $\|err_p\|_{L_2}$ for all the matrix-free tests (the red and blue dash-dot lines are covered by the solid ones). This is consistent with the fact that, in matrix-free contexts, the lagging operation only affects the computation of the preconditioner and not the iteration matrix. As regards the MB setting, even though a lagging parameter of $n = 10$ was employed, the theoretical

convergence order is maintained thanks to the $\mathcal{W}$-property of the Rosenbrock scheme here used. However, the solution shows a slightly higher $\|err_p\|_{L_2}$ (see black dash-dot line of Fig. 4.5).

Fig. 4.6(a) reports the $\|err_p\|_{L_2}$ versus the CPU time. Using $n = 10$, the ILU(0)-MF solver outperforms the standard ILU(0)-MB, $n = 1$ approach particularly using small time steps, where a speed-up factor of 1.38 has been estimated. The lagging has only a marginal effects on the EWBJ-MF performances, which are still poor in terms of computational time. The reason is here ascribed to the computational cost of the Jacobian. This fact can be quantified with the parameter $r = T_{j_h}/T_{f_h}$, namely the ratio between the CPU time required to evaluate the Jacobian trilinear form j_h and the CPU time required to evaluate the residual bilinear form f_h. For the ILU(0) case a value of $r \approx 67$ has been estimated, while for the EWBJ preconditioner $r \approx 29$, see Tab. 4.1. Such low value of r is consistent with the fact that the EWBJ is cheaper to be computed, and therefore it explains the low amount of CPU time saved. Fig. 4.6(b) reports the average number of iterations per stage. As expected, the preconditioner lagging reduces slightly the efficiency of the GMRES algorithm. However, especially for the highest values of r, the additional iterations are compensated from the CPU time point of view, by the time saved by skipping the preconditioner evaluation. Moreover, the additional GMRES iterations decrease as the time step size decreases since, with the smallest δt, the Jacobian is weakly varying between one time step and another.

Fig. 4.7 reports the computational efficency of the solvers when the adaptive tolerance of Eq. (4.7) and the Jacobian lagging are used together. Clearly the ILU(0)-MF results in the best performing scheme on a wide interval of time step sizes, see Fig. 4.7(a), providing in average a speed-up of 1.3 (in terms of CPU time) if compared to the adaptive ILU(0)-MB, $n = 1$. On the other hand, the performance of the ILU(0)-MB with $n = 10$ are similar to those of the MF, despite showing the slightly higher error. The dashed line, reported as reference, shows the ILU(0)-MB, $n = 1$,

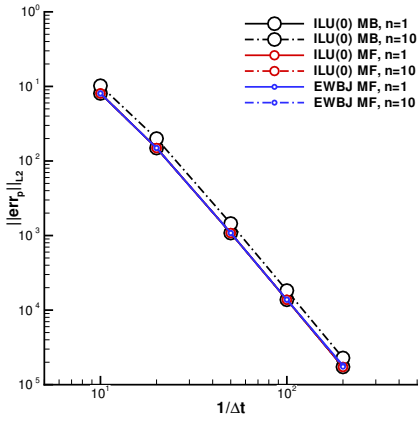
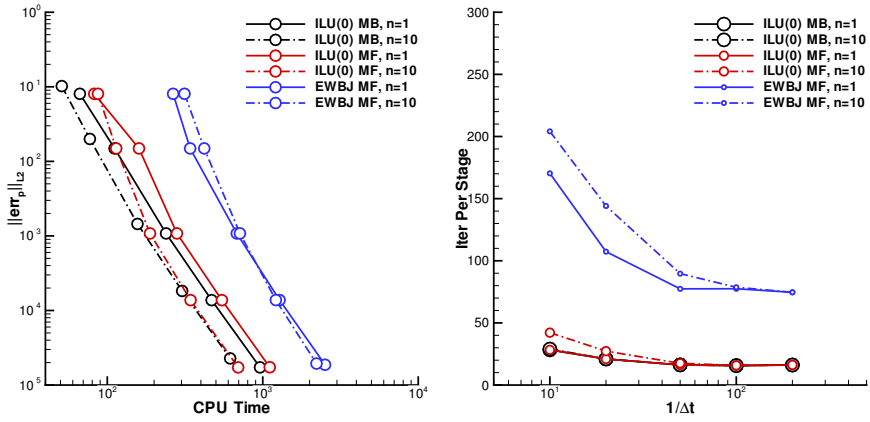


Figure 4.5: Effects of the Jacobian lagging on the TW problem at $Re = 100$ for the three considere strategies.



(a) $\|err_p\|_{L_2}$ vs. CPU time

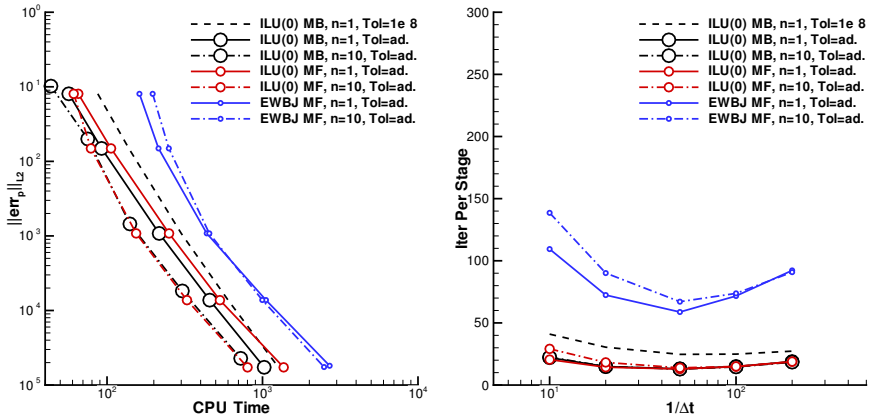
(b) Iterations Per Stage vs. $1/\delta t$

Figure 4.6: Effects of the Jacobian lagging ($n = 10$) on the TW problem at $Re = 100$ using $tol_r = 10^{-5}$.

$tol_r = 10^{-8}$ case, which requires higher CPU time to solve the problem. Despite still poorly performing, the EWBJ-MF penalty is greatly reduced when using those expedients.

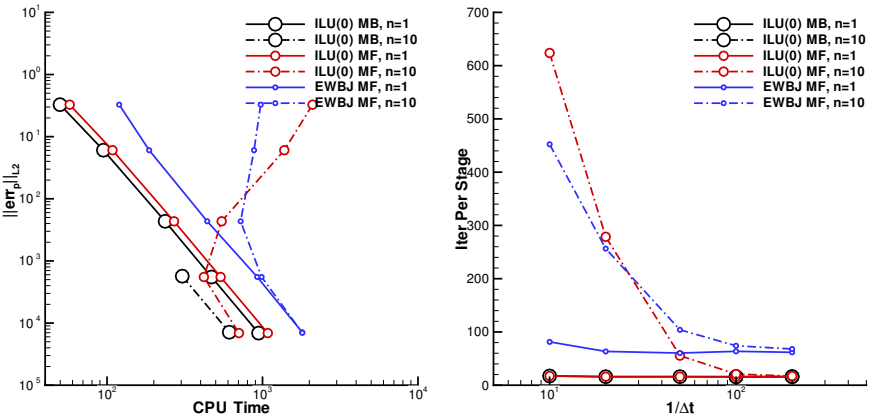
It is worth pointing out that while the ILU(0)-MB setting allows to maintain the theoretical convergence rate for the TW problem at $Re = 100$, which is somehow remarkable for such configuration, the authors experienced a significant decrease of stability, with respect the $n = 1$ case, just by increasing by one order of magnitude the Reynolds number and using moderately high time step sizes. For this purpose Fig. 4.8 has been reported, where only the converged solutions are plotted (black dash-dotted line). Conversely, the matrix-free solver accuracy was unaffected by the lagging operation, which can be employed with confidence even with the largest time step sizes ($T/10$, $T/20$ and $T/50$), where the preconditioner becomes rapidly inefficient and a huge increase in the average iterations per stage is observed. Despite that using $n \neq 1$ is not useful to increase the computational efficiency for such configurations, it confirms further the reliability of the approach.

This strong influence of the Reynolds number on the lagging effectiveness can be understood by noting that, with the incompressible NS, the diffusive terms are linear and therefore their contribution to $\boldsymbol{J}_h$ is exact even when the evaluation of this matrix is frozen. In other words, with the lagging procedure only the convective part of the Jacobian is approximated entailing that the approach is more suited for diffusive dominated regimes. Nevertheless, in the following it is proved that the approach can strongly enhance the computational efficiency of the solver in turbulent three-dimensional problems also. It is finally worth noting that, while the matrix-based approaches formally support the Jacobian lagging only for certain schemes, like the ROSI2PW employed herein, the matrix-free strategy allows to exploit this operation with every scheme.



(a) $\|err_p\|_{L_2}$ vs. CPU time (b) Iterations Per Stage vs. $1/\delta t$

Figure 4.7: Effects of the Jacobian lagging ($n = 10$) together with the adaptive tolerance of Eq. (4.7) on the TW problem at $Re = 100$.



(a) $\|err_p\|_{L_2}$ vs. CPU time (b) Iterations Per Stage vs. $1/\delta t$

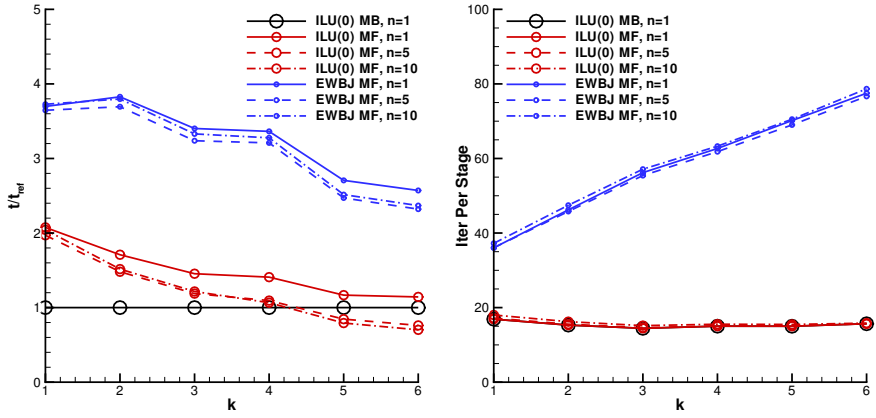
Figure 4.8: Effects of the Jacobian lagging ($n = 10$) with fixed tolerance $tol_r = 10^{-5}$ on the TW problem at $Re = 1000$.

4.3.4 Effects of the space discretization

The effects of space discretization are here reported for the three solution strategies considered. Firstly, the effects of rising the order of polynomial approximation from $\mathbb{P}_1$ to $\mathbb{P}_6$ on the 16×16 grid are reported. Secondly, the results on two grid refinements (32×32 and 64×64), using $\mathbb{P}_6$ polynomials, are compared. The numerical experiments were performed using a time step size of $\delta t = T/100$ and a fixed value of linear system tolerance $tol_r = 10^{-5}$.

$\mathbb{P}$	1	2	3	4	5	6
r (ILU(0))	7	14	23	32	48	67
r (EWBJ)	4	7	11	14	23	29

Table 4.1: Computed $r = T_J/T_R$ for different values of polynomial order on the 2D TW problem.



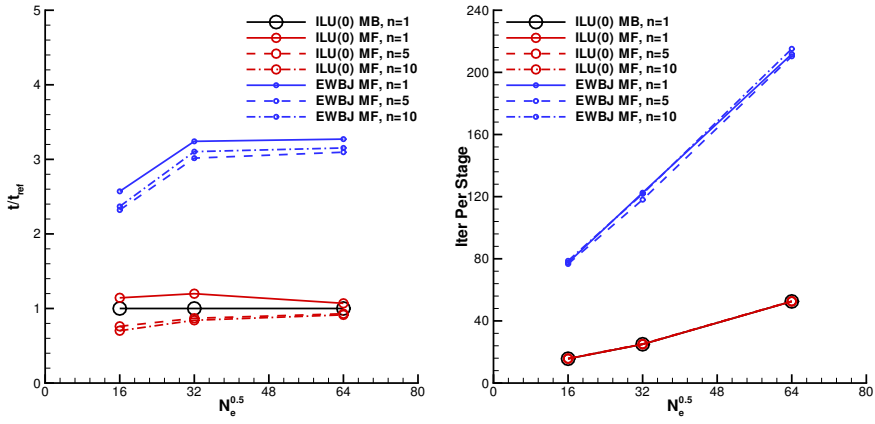
(a) CPU time ratio (t/t_{ref}) vs. polynomial order k

(b) Average iterations per stage vs. polynomial order k

Figure 4.9: Effect of the polynomial order on TW at $Re = 100$, ROSI2PW scheme. ILU(0)-MB, ILU(0)-MF and EWBJ-MF are reported with $n = 1$, 5 and 10. The GMRES system tolerance was set as $tol_r = 10^{-5}$.

Fig. 4.9(a) shows the CPU time of the matrix-free solution strategies non-dimensionalized using the ILU(0)-MB, $n = 1$ case by varying the order of polynomial approximation k. A clear improvement of the matrix-free performance can be noticed as k increases. In particular, the ILU(0)-MF with Jacobian lagging of $n = 5$, 10 outperforms the corresponding MB case for $k \geq 5$. This trend is consistent with the higher CPU time required to evaluate the Jacobian as the polynomial order increases, see Tab. 4.1, which reports the computed r parameter described in Section 4.3.3. The EWBJ-MF, despite poorly performing in all the cases (the number of GMRES iterations per stage increase with k, see blue lines of Fig. 4.9(b)), it reduces its penalty factor from 3.8 at $k = 1$ to 2.5 at $k = 6$.

Fig. 4.10 reports data obtained by increasing the mesh size. While the average number of iterations per stage increase almost linearly with the number of mesh elements, the relative time t/t_{ref} is only weakly varying with n_e, showing that the space discretization affects the results on this two-dimensional case through the polynomial order only.



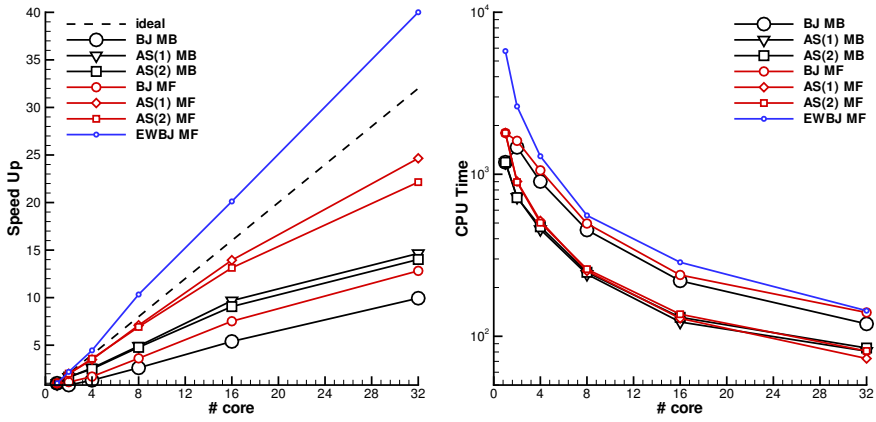
(a) CPU time ratio (t/t_{ref}) vs. $\sqrt{N_e}$ (b) Average iterations per stage vs. $\sqrt{N_e}$

Figure 4.10: Effect of the grid size on TW at $Re = 100$, ROSI2PW scheme. ILU(0)-MB, ILU(0)-MF and EWBJ-MF are reported with $n = 1$, 5 and 10. The GMRES system tolerance was set as $tol_r = 10^{-5}$.

4.3.5 Strong scalability

Numerical experiments performed to evaluate the parallel performance of the algorithm are here reported. The 64×64, $\mathbb{P}_6$ space discretization was employed for this purpose, using a time step size of $\delta t = T/20$. The tests were performed by comparing the CPU time needed by the solver to perform 10 time steps on a computational node based on two sixteen core AMD Opteron 6276 CPUs. The GMRES tolerance tol_r was fixed at 10^{-5}.

Fig. 4.11 shows the Speed-up and the CPU time measured for the BJ-MB, BJ-MF and the EWBJ-MF. The results using the Additive Schwartz Method with 1 (ASM(1)) and 2 (ASM(2)) overlapping layers are also reported for comparison purposes in combination with both the MB and the MF solvers. As regards the Speed-up, is its shown in Fig. 4.11(a) that the BJ scales very poorly. The reason for this can be observed in Fig. 4.12, where the average number of iterations per stage of the GMRES is reported as a function of the number of cores employed. In fact, as the number of MPI ranks increases, the BJ preconditioner decreases its effectiveness and a higher number of iterations are required to reach the target relative tolerance if compared to those required by serial computations. This outlook is only partially mitigated by the AS methods, which in this case increase the computational efficiency. In fact, the number of GMRES iterations grows slightly, *i.e.* the preconditioner effectiveness does not change with the number of domain partitions. However, the parallel efficiency is still far from the ideal scaling due the overhead of MPI communications related to the overlapping blocks. When such strategies are employed within the matrix-free context, the situation is only partially different. In fact, an increase in parallel efficiency (see Fig. 4.11(a)) of the matrix-free strategies is observed, which is only apparent since it is ascribed to a slight different behaviour of the serial computation (see, for



(a) Speed-Up

(b) CPU time

Figure 4.11: Strong scalability analysis of the MB and MF strategies using different preconditioners. TW problem at $Re = 100$, 64×64, $\mathbb{P}_6$ space discretization.

example, the CPU time in Fig 4.11(b)). This fact may be due to cash and memory usage efficiencies varying the size of the local portion of the residuals vector. This behaviour can be also observed for the EWBJ-MF case, which shows large superlinear speed-up values, see blue lines of Fig. 4.11. Note that with this preconditioner the average number of iterations per stage remain constant irrespectively to the number of MPI ranks employed. This happens because the EWBJ is defined locally within each element and therefore it does not change with the number of domain partitions. Interestingly, when using 32 cores it provides a CPU time comparable with the BJ cases. This fact has a huge impact if such preconditioner is employed on highly parallel runs on large HPC systems.

4.4 Remarks

Summarising the two-dimensional numerical experiments performed in this chapter some remarks can be made. In general, it has been shown that the matrix-free solvers are able to provide the same output error levels than standard matrix-based solvers for a very wide ranges of tolerances and time steps. However, only inside an operative range the matrix-free GMRES solver converge with the same number of iterations. This case is typically verified for sufficiently large GMRES tolerances, where the finite difference approximation error does not affect the linear system convergence. For those reasons, an adaptive tolerance strategy that allows to solve the linear system with the minimum number of iterations, without detrimental effect on the accuracy of the solution, is crucial to increase the efficiency of the method. The matrix-free performances may be further improved by lagging the Jacobian evaluation, which does not affect the solution accuracy but increases the computational efficiency when

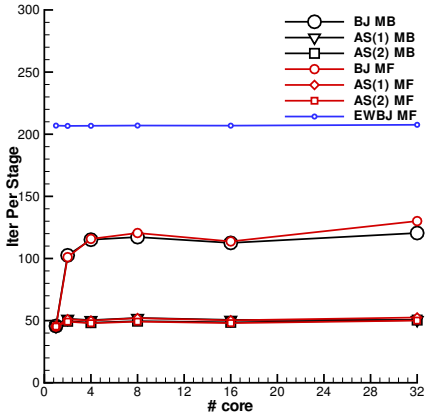


Figure 4.12: Number of iterations per stage as function of the number of domain partitions for different preconditioners. TW problem at $Re = 100$, 64×64, $\mathbb{P}_6$ space discretization.

the costs of the matrix evaluation are high. An estimate of those costs can be obtained via the parameter r, which increases as the order of polynomial approximation rises. Moreover, while MB methods behave better than MF for low values of polynomial orders, using high values of k they behave similarly in terms of CPU time. Finally, when the EWBJ is employed, the MF scales almost optimally and it is able to provide a comparable CPU time than the other strategies on highly parallel runs, despite being highly penalized in serial computations. Such considerations are independent of the mesh size.

Chapter 5

Multigrid preconditioning for stiff discretizations

The chapter is devoted at extending the limitations of what has been reported in Chapter 4 to arbitrary complex problems by proposing an efficient and optimally scalable preconditioning strategy that allows to exploit the memory footprint reduction allowed from a matrix-free strategy.

The key idea here is to exploit multilevel methods. Those methods have been considered in the past as an efficient way to solve both linear and non-linear problems arising from high-order discontinuous Galerkin discretizations, and have been first proposed by Helenbrook et al. [86], Bassi and Rebay [87], Fidkowski et al. [88]. Those authors focused on the analysis of a p-multigrid (p-MG) non-linear solver, proving convergence properties and performance in the context of compressible flows using element- or line-Jacobi smoothing. Several authors also considered multigrid operators built on agglomerated coarse grids, such as h-multigrid, see for example [89, 90, 91]. The possibility of using multigrid operators as a preconditioner was also explored in the context of steady compressible flows, see for example [92, 93]. In these works, the algorithm is reported as the most efficient and scalable if compared to single-grid preconditioners, and a large reduction in the number of iteration to reach convergence was achieved. More recently, an h-multigrid preconditioner was proposed in [94] in the context of steady and unsteady incompressible flows. In this latter work a specific treatment for inherited DG discrezations of viscous terms on coarse levels was introduced, significantly improving the performance of the multigrid iteration.

The aim here is to combine such powerful preconditioning strategies with a matrix-free implementation of the smoothers to reduce as much as possible the memory footprint and increase the computational efficiency on problems of arbitrary complexity. Target applications here require to solve for incompressible flows with linearly implicit Runge–Kutta schemes of the Rosenbrock type. The linear systems solution is based on a matrix-free implementation of the Flexible GMRES (FGMRES) method with p-multigrid preconditioning. In particular, the p-multigrid iteration is built using a matrix-free GMRES smoother on the finest level, and standard matrix-based GMRES smothers on coarser levels. In addition to that, one of the contribution of the chapter is to extend the h- rescaled-inherited approach proposed in [94] to the p version of the multigrid preconditioner. The approach allows to overcome the excess of numerical stabillization obtained on the coarser-level operators through standard subspace-inheritance, reducing the stiffness of the coarser-level problems. The impact of this algorithm on the convergence of the iterative method investigated.

The chapter assesses the behaviour of the rescaled-inherited approach on the solu-

tion of the two-dimensional Poisson problem. Afterwards, two incompressible Navier–Stokes solutions are examinated. The first one deals with the solution of the two-dimensional laminar flow around a circular cylinder, while the second one involve the three-dimensional laminar flow around a sphere. Those two test cases are discretized with curved mesh elements and a large computational domain, which increase the stiffness of the linear systems arising from the time discretization. Large benefits both in terms of the iterative solver convergence rate and of the CPU time while maintaining a favourable memory footprint are documented. See [95] for further details.

5.1 Multigrid preconditioners

This Section considers the possibility to solve the global equation system (2.31) by means of a flexible GMRES (FGMRES) iterative solver preconditioned with p-multigrid. Algorithm 11 reports the FGMRES implementation. As opposed to Algorithm 10, it requires to save the action of the preconditioner in each iteration on the preconditioned vector $\boldsymbol{z}_h^j$, which serves as the basis of the Krylov subspace to find $\delta \boldsymbol{w}_h^s$. This characteristic is required when the action of the preconditioner is not defined as an appropriate constant matrix operator $\boldsymbol{P}_h^{-1}$, but it is a (possibly variable) algorithmic procedure, which requires the approximation of the problem $\boldsymbol{G}_h \, \boldsymbol{z}_h^j = \boldsymbol{v}_h^j$ at step 5 of Algorithm 11. While any iterative solver can be employed to provide the preconditioned vector $\boldsymbol{z}_h^j$, in this chapter it is obtained through a single p-multigrid iteration. It is worth to remark that, considering the definition of $\boldsymbol{v}_h^j$, only an approximation of the solution of the problem $\boldsymbol{G}_h \, \boldsymbol{z}_h^j = \boldsymbol{v}_h^j$ is required to precondition the system, since a numerically exact $\boldsymbol{z}_h^j$ for $j = 1$ would require to solve directly the original problem $\boldsymbol{G}_h \, \delta \boldsymbol{w}_h = \boldsymbol{g}_h$ without preconditioning.

In this Chapter, the preconditioned vector is obtained through a multigrid linear solver. The basic multigrid idea is to exploit iterative solvers to smooth-out the high-frequency component of the error with respect to the unknown exact solution. Indeed, being iterative solvers not effective at damping low-frequency error components, the iterative solution of coarser problems is exploited to circumvent this issue, thus shifting the low-frequency modes towards the upper side of the spectrum. This simple and effective strategy allows to obtain satisfactory rates of convergence all along the iterative process. In p-multigrid the coarse problems are built based on lower-order DG discretizations with respect to the original problem of degree k. L coarse levels are considered, spanned by the index $\ell = 1, ..., L$, which indicate the fine and coarse levels with $\ell = 0$ and $\ell = L$, respectively. The polynomial degree of level ℓ is k_ℓ and the polynomial degrees of the coarse levels are chosen such that $k_\ell < k_{\ell-1}$, $l = 1, ..., L$, with $k_0 = k$. Accordingly the polynomial spaces associated to the coarse levels read $\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)$. The coarse problems corresponding to (2.31) and 2.33 are in the form

$$\boldsymbol{G}_\ell \, \delta \boldsymbol{w}_\ell = \boldsymbol{g}_\ell \tag{5.1}$$

where $\boldsymbol{G}_\ell$ is the global matrix operator on level l and $\delta \boldsymbol{w}_\ell, \boldsymbol{g}_\ell \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m$ are the unknown function and the known right-hand side, respectively.

A crucial aspect for the efficiency of the p-multigrid iteration is related to the

Algorithm 11 Restarted FGMRES algorithm (right-preconditioned)

1: Choose $\delta \boldsymbol{w}_h^0$ and define s the dimension for the Krylov-Subspaces. Define $\bar{\boldsymbol{H}}_m \in \mathbb{R}^{s+1} \otimes \mathbb{R}^s$ and initialise the components $\mathsf{h}_{i,j}$ to zero;
2: *Arnoldi Process:*
3: Compute $\boldsymbol{r}_h^0 = \boldsymbol{g}_h - \boldsymbol{G}_h\ \delta \boldsymbol{w}_h^0$, $\beta = \|r_h^0\|_2$ and $\boldsymbol{v}_h^1 = r_h^0/\beta$.
4: **do** $j = 1, s$
5: Approximate $\boldsymbol{G}_h\ \boldsymbol{z}_h^j = \boldsymbol{v}_h^j$;
6: Compute $\boldsymbol{w}_h = \boldsymbol{G}_h\ \boldsymbol{z}_h^j$;
7: **do** $i = 1, j$
8: set $\mathsf{h}_{i,j} = (\boldsymbol{w}_h, \boldsymbol{v}_h^j)$
9: set $\boldsymbol{w}_h = \boldsymbol{w}_h - \mathsf{h}_{i,j} \boldsymbol{v}_h^j$
10: **enddo**
11: Compute $\mathsf{h}_{(j+1),j} = \|\boldsymbol{w}_h\|_2$ and $\boldsymbol{v}_h^{j+1} = \boldsymbol{w}_h / \mathsf{h}_{(j+1),j}$
12: **enddo**
13: Define $\boldsymbol{Z}_s = [\boldsymbol{z}_h^1, \boldsymbol{z}_h^2, \ldots, \boldsymbol{z}_h^s]^T$
14: *Form the approximate solution:*
15: Find $\boldsymbol{y}_h^s = min(\beta \boldsymbol{e}_1 - \bar{\boldsymbol{H}}_s\ \boldsymbol{y}_h)$, with $\boldsymbol{e}_1 = [1, 0, \ldots, 0]^T$
16: Compute $\delta \boldsymbol{w}_h^s = \delta \boldsymbol{w}_h^0 + \boldsymbol{Z}_s\ \boldsymbol{y}_h^s$
17: **if** $\|\boldsymbol{g}_h - \boldsymbol{G}_h\ \delta \boldsymbol{w}_h^s\|_2 \leq \varepsilon$ **then**
18: EXIT
19: **else**
20: set $\delta \boldsymbol{w}_h^0 = \delta \boldsymbol{w}_h^s$
21: **go to** 2
22: **end if**

computational cost of building coarse grid operators $\boldsymbol{G}_\ell$. While it is possible to assemble the bilinear and trilinear forms j_h, f_h and m_h of Section 2.3 on each level ℓ with the corresponding polynomial functions $\boldsymbol{w}_h, \delta \boldsymbol{w}_h, \boldsymbol{z}_h \in \mathbb{P}_d^{k_\ell}(\mathcal{T}_h)$, significantly better performances are achievable by *restricting* the fine grid operator by means of so called Galerkin projections. The former and the latter strategies are named non-inherited and inherited p-multigrid, respectively. As will be clear in what follows the construction of coarse operators is trivial when polynomial expansions are based on hierarchical orthonormal modal basis functions.

5.1.1 Restriction and prolongation operators

In this section the prolongation and restriction operators required to map polynomial functions on finer and coarser levels, respectively, are described.

Since $\mathbb{P}_d^{k_\ell}(\mathcal{T}_h) \supset \mathbb{P}_d^{k_{\ell+1}}(\mathcal{T}_h)$, the *prolongation* operator $\mathcal{I}_{\ell+1}^{\ell} : \mathbb{P}_d^{k_{\ell+1}}(\mathcal{T}_h) \to \mathbb{P}_d^{k_\ell}(\mathcal{T}_h)$, is the injection operator such that

$$\sum_{\kappa \in \mathcal{T}_h} \int_\kappa (\mathcal{I}_{\ell+1}^{\ell} w_h - w_h) \ = 0, \ \ \forall w_h \in \mathbb{P}_d^{k_{\ell+1}}(\mathcal{T}_h),$$

The prolongation operator from level ℓ to level 0 can be recursively defined by the composition of inter-grid prolongation operators: $\mathcal{I}_\ell^0 = \mathcal{I}_1^0\, \mathcal{I}_2^1 \ldots \mathcal{I}_\ell^{\ell-1}$.

The (L^2 projection) *restriction* operator $\mathcal{I}_\ell^{\ell+1} : \mathbb{P}_d^{k_\ell}(\mathcal{T}_h) \to \mathbb{P}_d^{k}(\mathcal{T}_h)$, is such that

$$\sum_{\kappa \in \mathcal{T}_h} \int_\kappa (\mathcal{I}_\ell^{\ell+1} w_h - w_h)\, z_h = 0, \qquad \forall w_h \in \mathbb{P}_d^{k_\ell}(\mathcal{T}_h),\ \forall z_h \in \mathbb{P}_d^{k_{\ell+1}}(\mathcal{T}_h), \tag{5.2}$$

and the restriction operator from level 0 to level ℓ reads $\mathcal{I}_0^\ell = \mathcal{I}_{\ell-1}^\ell \dots \mathcal{I}_1^2 \mathcal{I}_0^1$.

When applied to vector functions $\boldsymbol{w}_h \in [\mathbb{P}_d^{k_{\ell+1}}(\mathcal{T}_h)]^m$ the interpolation operators act componentwise, *e.g.*, $\boldsymbol{\mathcal{I}}_{\ell+1}^\ell \boldsymbol{w}_h = \sum_{i=1}^m \mathcal{I}_{\ell+1}^\ell w_i$. It is interesting to remark that using hierarchical orthonormal modal basis functions restriction and prolongation operators are trivial, in particular restriction from $\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)$ into $\mathbb{P}_d^{k_{\ell+1}}(\mathcal{T}_h)$ is as simple as keeping the degrees of freedom of the modes of order $k \leq k_{\ell+1}$ and discarding the remaining high-frequency modes.

5.1.2 Fine and coarse grid Jacobian operators

The non-inherited and the inherited version (denoted with superscript $\mathcal{I}$) of the gradient and gradient-free Jacobian operators introduced in (2.20)-(2.21), can be defined as follows for $\ell = 1, ..., L$,

$$\begin{aligned}
(\boldsymbol{J}_\ell^{!\nabla}\, \delta\boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= j_h^{!\nabla}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) \\
(\boldsymbol{J}_\ell^{\nabla}\, \delta\boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= j_h^{\nabla}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) \\
(\boldsymbol{J}_\ell^{!\nabla,\mathcal{I}} \delta\boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= j_h^{!\nabla}(\boldsymbol{\mathcal{I}}_\ell^0 \boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\, \delta\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0 \boldsymbol{z}_h) \\
(\boldsymbol{J}_\ell^{\nabla,\mathcal{I}} \delta\boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= j_h^{\nabla}(\boldsymbol{\mathcal{I}}_\ell^0 \boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\, \delta\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0 \boldsymbol{z}_h)
\end{aligned} \tag{5.3}$$

$\forall\, \boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m$. The main benefit of inherited algorithms is the possibility to efficiently compute coarse grid operators by means of the so called Galerkin projection, avoiding the cost of assembling bilinear and trilinear forms. The procedure is described in what follows, focusing on the benefits of using hierarchical orthonormal basis functions.

The matrix counterpart $\boldsymbol{J}_\ell^{\mathcal{I}}$ of the operator $\boldsymbol{J}_\ell^{\mathcal{I}} = \boldsymbol{J}_\ell^{\nabla,\mathcal{I}} + \boldsymbol{J}_\ell^{!\nabla,\mathcal{I}}$ is a sparse block matrix with block dimension $n_v^\ell = \dim(\mathbb{P}_d^{k_\ell}(\kappa))$ and total dimension $mn_e n_v^\ell$. The matrix is composed of diagonal blocks $\boldsymbol{J}_{\kappa,\kappa}^{\ell,\mathcal{I}}$ and off-diagonal blocks $\boldsymbol{J}_{\kappa,\kappa'}^{\ell,\mathcal{I}}$, the latter taking care of the coupling between neighboring elements κ, κ' sharing a face σ. Once the fine system matrix $\boldsymbol{J}_0$ is assembled, the diagonal and off-diagonal blocks of the Jacobian matrix of coarse levels can be inherited recursively and matrix-free as follows

$$\begin{aligned}
\boldsymbol{J}_{\kappa,\kappa}^{\ell+1,\mathcal{I}} &= \boldsymbol{M}_{\ell+1,\ell}^{\kappa}\ \left(\boldsymbol{J}_{\kappa,\kappa}^{\ell,\mathcal{I}}\right)\ \left(\boldsymbol{M}_{\ell+1,\ell}^{\kappa}\right)^t, \\
\boldsymbol{J}_{\kappa,\kappa'}^{\ell+1,\mathcal{I}} &= \boldsymbol{M}_{\ell+1,\ell}^{\kappa}\ \left(\boldsymbol{J}_{\kappa,\kappa'}^{\ell,\mathcal{I}}\right)\ \left(\boldsymbol{M}_{\ell+1,\ell}^{\kappa'}\right)^t.
\end{aligned} \tag{5.4}$$

The projection matrices read

$$\boldsymbol{M}_{\ell+1,\ell}^{\kappa} = (\boldsymbol{M}_{\ell+1}^{\kappa})^{-1} \int_\kappa \varphi^{\ell+1} \otimes \varphi^{\ell}, \quad \text{where} \quad \boldsymbol{M}_{\ell+1}^{\kappa} = \int_\kappa \varphi^{\ell+1} \otimes \varphi^{\ell+1}, \tag{5.5}$$

and φ^ℓ represents the set of basis functions spanning the space $\mathbb{P}_d^{k_\ell}(\kappa)$. When using

hierarchical orthonormal basis functions, $\boldsymbol{M}^{\kappa}_{\ell+1}$ is the unit diagonal elemental mass matrix and $\boldsymbol{M}^{\kappa}_{\ell+1,\ell} \in \mathbb{R}^{n_v^{\ell} \times n_v^{\ell+1}}$ is a unit diagonal rectangular matrix. Accordingly the Galerkin projection in (5.4) falls back to a trivial and inexpensive sub-block extraction.

Being $\mathbb{P}^{k_0}_d(\mathcal{T}_h) \supset \mathbb{P}^{k_\ell}_d(\mathcal{T}_h)$, it can be demonstrated that inherited and non-inherited p-multigrid algorithms lead to the same inviscid Jacobian operators, that is $\boldsymbol{J}^{!\nabla,\mathcal{I}}_{\ell} = \boldsymbol{J}^{!\nabla}_{\ell}$. As opposite inherited and non-inherited viscous Jacobian differ because of the terms involving lifting operators. Note that inherited lifting operators act on traces of polynomial functions mapped into $\mathbb{P}^{k_0}_d(\mathcal{T}_h)$. In other words one can define

$$\text{inherited lifting operators,} \quad \mathbf{r}^{\sigma}(\llbracket z_h \rrbracket) : \mathbb{P}^{k_0}_d(\sigma) \to [\mathbb{P}^{k_0}_d(\mathcal{T}_h)]^d, \tag{5.6}$$

$$\text{non-inherited lifting operators,} \quad \mathbf{r}^{\sigma}(\llbracket z_h \rrbracket) : \mathbb{P}^{k_\ell}_d(\sigma) \to [\mathbb{P}^{k_\ell}_d(\mathcal{T}_h)]^d. \tag{5.7}$$

Interestingly, using the definitions of the global and local lifting operators, the bilinear form can be rewritten as follows

$$\begin{aligned}
&j^{\nabla}_h(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) = + \\
&- \sum_{\kappa\in\mathcal{T}_h} \int_{\kappa} \sum_{i,j=1}^{m} \sum_{p,q=1}^{d} \frac{\partial \widetilde{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - R^{\kappa}_q(w_j)\Big)} \frac{\partial(\delta w_j)}{\partial x_q} \frac{\partial z_i}{\partial x_p} + \\
&+ \sum_{\sigma\in\mathcal{F}_h} \int_{\sigma} \sum_{i,j=1}^{m} \sum_{p,q=1}^{d} n^{\sigma}_q \frac{\partial \widetilde{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - R^{\kappa}_q(w_j)\Big)} \llbracket \delta w_j \rrbracket \left\{\!\!\left\{ \frac{\partial z_i}{\partial x_p} \right\}\!\!\right\} + \\
&+ \sum_{\sigma\in\mathcal{F}_h} \int_{\sigma} \sum_{i,j=1}^{m} \sum_{p,q=1}^{d} n^{\sigma}_p \frac{\partial \widehat{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - \eta_\sigma r^{\sigma}_q(w_j)\Big)} \left\{\!\!\left\{ \frac{\partial(\delta w_j)}{\partial x_q} \right\}\!\!\right\} \llbracket z_i \rrbracket + \\
&- \sum_{\sigma\in\mathcal{F}_h} \int_{\sigma} \sum_{i,j=1}^{m} \sum_{p,q=1}^{d} \eta_\sigma \frac{\partial \widehat{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - \eta_\sigma r^{\sigma}_q(w_j)\Big)} r^{\sigma}_q\left(\llbracket \delta w_j \rrbracket\right) r^{\sigma}_p\left(\llbracket z_i \rrbracket\right)
\end{aligned} \tag{5.8}$$

showing that only the last term, *i.e.*, the stabilization term, cannot be reformulated lifting-free. In particular, as will be demonstrated in the following section, the inherited stabilization term introduces an excessive amount of stabilization with respect to its non-inherited counterpart, which is detrimental for multigrid algorithm performance. The same behaviour was previously documented in the context of h-multigrid solution strategies, see [94] where the authors consider DG discretizations of the INS equations and [96], where preconditioners for weakly over-penalized symmetric interior penalty DG discretization of elliptic problems are devised.

5.1.3 Scaling of the stabilization term

Following the idea proposed by [94], here a rescaled Galerkin projection of the stabilization term is introduced in order to recover the optimal performances of non-inherited p-multigrid algorithm.

As a first point the following bound on the local lifting operator: let $\phi \in L^2(\sigma)$, for

all $\sigma \in \mathcal{F}_h$

$$\|\mathbf{r}^\sigma(\phi)\|_{[L^2(\Omega)]^d} \leq C_{\mathrm{tr}} h_{\kappa,\kappa'}^{-1/2} \|\phi\|_{L^2(\sigma)}, \tag{5.9}$$

where $h_{\kappa,\kappa'} = \min(h_\kappa, h_{\kappa'})$, see *e.g.*[97, Lemma 2], [98, Lemma 7.2] or [99, Lemma 4.33 and Lemma 5.18] for a proof. The constant C_{tr} depends on d, k and the shape regularity of the elements sharing σ and is inherited from the discrete trace inequality: for all $\kappa \in \mathcal{T}_h$, $\sigma \in \mathcal{F}_h$

$$\|z_h\|_{L^2(\sigma)} \leq C_{\mathrm{tr}} h_{\kappa,\kappa'}^{-1/2} \|z_h\|_{L^2(\kappa)} \tag{5.10}$$

As remarked by Di Pietro and Ern [99, Lemma 1.46] the dependence of C_{tr} on k is a delicate issue that has a precise answer only in specific cases. Here the estimates given by Hesthaven and Warburton [100] are followed, showing that for simplicial meshes C_{tr} scales as $\sqrt{k(k+d)}$ when using complete polynomials of maximum degree k. This choice turns out to be conservative regarding the dependence on k with respect to estimates derived by Schwab [101] based on tensor product polynomials on mesh elements being affine images of the unit hypercube in $\mathbb{R}^d$, which suggest a $\sqrt{k(k+1)}$ scaling.

Using the Cauchy-Schwarz inequality, for all $\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m$ one can get

$$\begin{aligned}
&j_h^{\nabla-\mathrm{STB}}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h)|_{\sigma\in\mathcal{F}_h} = \\
&= \eta_\sigma \int_\Omega \sum_{i,j=1}^m \sum_{p,q=1}^d \frac{\partial \widehat{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - \eta_\sigma r_q^\sigma(w_j)\Big)} \mathbf{r}_q^\sigma([\![\delta w_j]\!]) \mathbf{r}_p^\sigma([\![v_i]\!]) \\
&\leq \eta_\sigma\, C\, k_\ell(k_\ell+d)\, h_{\kappa,\kappa'}^{-1} \,\|\,[\![\delta\boldsymbol{w}_h]\!]\,\|_{L^2(\sigma)}\,\|\,[\![\boldsymbol{v}_h]\!]\,\|_{L^2(\sigma)}
\end{aligned} \tag{5.11}$$

$$\begin{aligned}
&j_h^{\nabla-\mathrm{STB}}(\boldsymbol{\mathcal{I}}_\ell^0\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\delta\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\boldsymbol{z}_h)|_{\sigma\in\mathcal{F}_h} = \\
&= \eta_\sigma \int_\Omega \sum_{i,j=1}^m \sum_{p,q=1}^d \frac{\partial \widehat{F}_{p,i}(\boldsymbol{w}_h)}{\partial\Big(\partial w_j/\partial x_q - \eta_\sigma r_q^\sigma(w_j)\Big)} \mathbf{r}_q^\sigma([\![\delta w_j]\!]) \cdot \mathbf{r}_p^\sigma([\![v_i]\!]) \\
&\leq \eta_\sigma\, C\, k_0(k_0+d)\, h_{\kappa,\kappa'}^{-1} \,\|\,[\![\delta\boldsymbol{w}_h]\!]\,\|_{L^2(\sigma)}\,\|\,[\![\boldsymbol{v}_h]\!]\,\|_{L^2(\sigma)}
\end{aligned} \tag{5.12}$$

where C is independent from h and k. As a result it is possible to introduce the scaling factor $\mathcal{S}_0^\ell = \frac{(k_\ell)(k_\ell+d)}{(k_0)(k_0+d)}$ such that, for all $\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m$, it holds

$$j_h^{\nabla-\mathrm{STB}}(\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) \simeq \mathcal{S}_0^\ell\; j_h^{\nabla-\mathrm{STB}}(\boldsymbol{\mathcal{I}}_\ell^0\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\delta\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\boldsymbol{z}_h). \tag{5.13}$$

The viscous Jacobian stabilization operator reads

$$(\boldsymbol{J}_\ell^{\nabla-\mathrm{STB}}(\delta\boldsymbol{w}_h), \boldsymbol{z}_h)_{L^2(\Omega)} = \mathcal{S}_0^\ell\; j_h^{\nabla-\mathrm{STB}}(\boldsymbol{\mathcal{I}}_\ell^0\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\delta\boldsymbol{w}_h, \boldsymbol{\mathcal{I}}_\ell^0\boldsymbol{z}_h) \tag{5.14}$$

$\forall\, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{T}_h)]^m$ and, accordingly, the Jacobian stabilization diagonal and off-diagonal block contributions $\boldsymbol{J}_{\kappa,\kappa}^{\ell,\nabla-\mathrm{STB},\mathcal{I}}$ and $\boldsymbol{J}_{\kappa,\kappa'}^{\ell,\nabla-\mathrm{STB},\mathcal{I}}$ can be computed recursively and matrix free by means of a rescaled Galerkin projection. The rescaled-inherited

blocks of the Jacobian matrix are computed as follows

$$\begin{aligned}\boldsymbol{J}_{\kappa,\kappa}^{\ell+1,\mathcal{I}} = \boldsymbol{M}_{\ell+1,\ell}^{\kappa}\ \left(\boldsymbol{J}_{\kappa,\kappa}^{\ell,!\nabla,\nabla\setminus\mathrm{STB},\mathcal{I}}\right)\ \left(\boldsymbol{M}_{\ell+1,\ell}^{\kappa'}\right)^{t} + \\ + \mathcal{S}_{\ell}^{\ell+1}\,\boldsymbol{M}_{\ell+1,\ell}^{\kappa}\ \left(\boldsymbol{J}_{\kappa,\kappa}^{\ell,\nabla-\mathrm{STB},\mathcal{I}}\right)\ \left(\boldsymbol{M}_{\ell+1,\ell}^{\kappa'}\right)^{t},\end{aligned} \tag{5.15}$$

$$\begin{aligned}\boldsymbol{J}_{\kappa,\kappa'}^{\ell+1,\mathcal{I}} = \boldsymbol{M}_{\ell+1,\ell}^{\kappa}\ \left(\boldsymbol{J}_{\kappa,\kappa'}^{\ell,!\nabla,\nabla\setminus\mathrm{STB},\mathcal{I}}\right)\ \left(\boldsymbol{M}_{\ell+1,\ell}^{\kappa'}\right)^{t} \\ + \mathcal{S}_{\ell}^{\ell+1}\,\boldsymbol{M}_{\ell+1,\ell}^{\kappa}\ \left(\boldsymbol{J}_{\kappa,\kappa'}^{\ell,\nabla-\mathrm{STB},\mathcal{I}}\right)\ \left(\boldsymbol{M}_{\ell+1,\ell}^{\kappa'}\right)^{t},\end{aligned} \tag{5.16}$$

where $\mathcal{S}_{\ell}^{\ell+1} = \frac{(k_{\ell+1})(k_{\ell+1}+d)}{(k_{\ell})(k_{\ell}+d)}$. Note that $\boldsymbol{J}_{\kappa,\kappa}^{\ell,!\nabla,\nabla\setminus\mathrm{STB}}$ and $\boldsymbol{J}_{\kappa,\kappa'}^{\ell,!\nabla,\nabla\setminus\mathrm{STB}}$ are the Jacobian blocks corresponding to gradient-free contributions plus the gradient contributions without the stabilization terms.

5.1.4 The *p*-multigrid iteration

In this section an overlook of the sequence of operations involved in p-multigrid iterations is provided. The recursive p-multigrid $\mathcal{V}$-cycle and full p-multigrid $\mathcal{V}$-cycle for the problem $\boldsymbol{G}_{\ell}\ \delta\boldsymbol{w}_{\ell} = \boldsymbol{g}_{\ell}$ on level ℓ reads are reported in Algorithms 12 and 13.

To obtain an application of the p-multigrid preconditioner the multilevel iteration is invoked on the problem $\boldsymbol{G}_{h}\ \delta\boldsymbol{w}_{h} = \boldsymbol{g}_{h}$. While one $\mathrm{MG}_{\mathcal{V}}$ iteration requires two applications of the smoother on the finest level (one pre- and one post-smoothing) and one application of the coarse level smoother independently from the number of levels, one $\mathrm{MG}_{\mathrm{full}}$ iteration requires one application of the finest level smoother and L applications of the coarse level smoother.

Here the p-multigrid $\mathcal{V}$ cycle iteration will be applied for the numerical solution of Poisson problems while the full p-multigrid iteration will be applied for the solution of linearized equations systems arising in Rosenbrock time marching strategies

Algorithm 12 $\overline{\boldsymbol{w}}_{\ell} = \mathrm{MG}_{\mathcal{V}}(l, \boldsymbol{g}_{\ell}, \boldsymbol{w}_{\ell})$

1: **if** $(\ell = L)$ **then**
2: $\quad\overline{\boldsymbol{w}}_{\ell} = \mathrm{SOLVE}(\boldsymbol{G}_{\ell}, \boldsymbol{g}_{\ell}, 0)$
3: **end if**
4: **if** $(\ell < L)$ **then**
5: $\quad$*Pre-smoothing:*
6: $\quad\overline{\boldsymbol{w}}_{\ell} = \mathrm{SMOOTH}(\boldsymbol{G}_{\ell}, \boldsymbol{g}_{\ell}, \boldsymbol{w}_{\ell})$
7: $\quad$*Coarse grid correction:*
8: $\quad\boldsymbol{d}_{\ell} = \boldsymbol{g}_{\ell} - \boldsymbol{G}_{\ell}\overline{\boldsymbol{w}}_{\ell}$
9: $\quad\boldsymbol{d}_{\ell+1} = \mathcal{I}_{\ell}^{\ell+1}\boldsymbol{d}_{\ell}$
10: $\quad\boldsymbol{e}_{\ell+1} = \mathrm{MG}_{\mathcal{V}}(\ell+1, \boldsymbol{d}_{\ell+1}, 0)$
11: $\quad\widehat{\boldsymbol{w}}_{\ell} = \overline{\boldsymbol{w}}_{\ell} + \mathcal{I}_{\ell+1}^{\ell}\boldsymbol{e}_{\ell+1}$
12: $\quad$*Post-smoothing:*
13: $\quad\overline{\boldsymbol{w}}_{\ell} = \mathrm{SMOOTH}(\boldsymbol{G}_{\ell}, \boldsymbol{g}_{\ell}, \widehat{\boldsymbol{w}}_{\ell})$
14: **end if**
15: return $\overline{\boldsymbol{w}}_{\ell}$

for DG discretizations of incompressible flow problems. In the context of Poisson problems, the most simple multigrid solution strategy with a single smoothing iteration is applied on all levels except the coarsest for validation purposes. In the context of incompressible flow problem we seek for the best performance employing full p-multigrid and tuning preconditioners and smoothing options.

Matrix-free and matrix-based preconditioned iterative solution strategies

In this chapter several multigrid and single-grid preconditioner strategies for Krylov iterative solvers are considered for the sake of comparison with application to the solution of linear system arising in Rosenbrock-type schemes, see Eq. (2.29). To maintain a compact nomenclature, the solver-preconditioner couple is identify by the following convention:

$$\text{SOLVER(MatVecProdOpt)[PREC(PrecOpt)]}.$$

The MatVecProdOpt describes how matrix-vector products need by Krylov iterative solvers are performed, *i.e.*in a matrix-based or matrix-free fashion, while PrecOpt specifies the type of the preconditioning employed. GMRES and FGMRES solvers are employed in combination with single-grid and p-multigrid preconditioners, respectively. Note that also p-multigrid iterations rely on GMRES preconditioned iterative solver for smoothing purposes. As preconditioners for GMRES iterations, the following options are considered:

1. ASM(i,ILU(j)) - Additive Schwarz domain decomposition Method (ASM) preconditioners with i levels of overlap between sub-domains and a block ILU decomposition for each sub-domain matrix with j levels of fill;

2. BJ or ASM(0,ILU(0)) - ASM preconditioner with no overlap between sub-domains and a block ILU decomposition for each sub-domain matrix with same level of fill of the original matrix;

Algorithm 13 $\overline{\boldsymbol{w}}_\ell = \mathrm{MG}_{\mathrm{full}}(l, \boldsymbol{g}_\ell, \boldsymbol{w}_\ell)$

if $(\ell = L)$ **then**
  $\overline{\boldsymbol{w}}_\ell = \mathrm{SOLVE}(\boldsymbol{G}_\ell, \boldsymbol{g}_\ell, 0)$
end if
if $(\ell < L)$ **then**
  $\boldsymbol{g}_{\ell+1} = \mathcal{I}_\ell^{\ell+1} \boldsymbol{g}_\ell$
  $\widehat{\boldsymbol{w}}_{\ell+1} = \mathrm{MG}_{\mathrm{full}}(\ell+1, \boldsymbol{g}_{\ell+1}, 0)$
  $\mathcal{V}$-cycle correction:
  $\widehat{\boldsymbol{w}}_\ell = \mathcal{I}_{\ell+1}^{\ell} \widehat{\boldsymbol{w}}_{\ell+1}$
  $\boldsymbol{d}_\ell = \boldsymbol{g}_\ell - \boldsymbol{G}_\ell \widehat{\boldsymbol{w}}_\ell$
  $\boldsymbol{e}_\ell = \mathrm{MG}_{\mathcal{V}}(\ell, \boldsymbol{d}_\ell, 0)$
  $\overline{\boldsymbol{w}}_\ell = \widehat{\boldsymbol{w}}_\ell + \boldsymbol{e}_\ell$
end if
return $\overline{\boldsymbol{w}}_\ell$

3. EWBJ - Element Wise Block Jacobi, a BJ preconditioner neglecting off-diagonal blocks, that is an LU factorization of the diagonal blocks.

Note that in serial computations ASM(i,ILU(j)) and BJ fall back to ILU(j) and ILU(0), respectively. ASM and BJ performance differ when the computation is performed in parallel, depending on the number of sub-domains. While efficiency of BJ decreases while increasing the number of sub-domains, ASM seeks to heal the convergence degradation at the expense of an increased memory footprint of the solver as the number of partitions rise (indeed part of the global matrix non-zeros entries are replicated in neighbouring sub-domains). As opposite EWBJ has optimal scalability properties and is particularly well suited for a matrix-free implementation of the iterative solver. Indeed, in this context, it allows to skip the computation of the off-diagonal contributions, thereby reducing the matrix-assembly computation time.

As smoother's preconditioners EWBJ is particularly well suited on the finest level, for the sake of reducing the memory footprint, while ASM makes sense on the coarsest level, to improve the convergence rate of the coarse solver. In this configuration, the off-diagonal blocks needed by the coarsest level operator can be computed at the coarsest polynomial degree for the sake of efficiency. In the rest of the paper several combinations of preconditioners are investigated with particular attention to the quantification of the parallel performance and of the memory savings.

5.1.5 Memory footprint considerations

In this section an estimate of the memory footprint of all the strategies employed in this paper is devised to fully appreciate the memory savings achievable trough a matrix-free solver preconditioned with p-multigrid. One can observe that the memory footprint of the global block matrix scales as $n_e\,(n_f{+}1)\,(m\,n_v)^2$ where $n_f = \overline{\mathrm{card}}(\mathcal{F}_{\partial\kappa})$ is the average number of element's faces, m is the number of variables in d space dimensions and $(m\,n_v)^2$ with $n_v = \dim(\mathbb{P}_d^k)$ is the number of non-zeros in each matrix block. While for a matrix-based implementation both the global system matrix and the preconditioner are assumed to be stored in memory, for a matrix-free approach only the preconditioner is stored. The preconditioner's memory footprint is carefully estimated assuming, respectively, that

1. for EWBJ, only the non-zero entries of a block-diagonal matrix are stored;

2. for BJ, the storage of ILU(0) factorization applied to the domain-wise portion of the iteration matrix, which neglects the off diagonal blocks related to faces residing on a partition boundary.

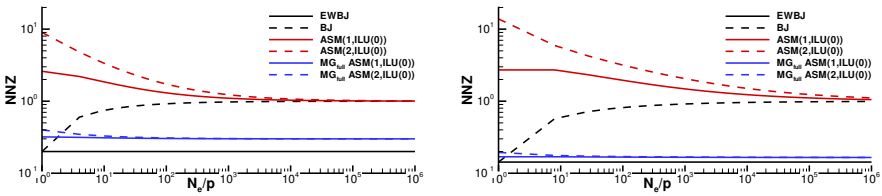
3. For ASM(q,ILU(0)), that the preconditioner applies the ILU(0) decomposition to a larger matrix, bigger than the sub-domain matrix of the BJ precondtioner. The exact number of additional non-zero blocks is difficult to estimate for general unstructured grids since it depends on mesh topology, element types (in case of hybrid grids) and the partitioning strategy. Nevertheless, an estimation can be done based on the following simplifications, *i.e.* assuming a square and cubical domain discretized by uniformly distributed quadrilateral and hexahedral

elements in $2d$ and $3d$, respectively, and considering periodic boundary conditions on the domain boundaries. Accordingly the number of non-zero blocks in each sub-domain matrix can be estimated as follows

$$\begin{aligned}(2d+1)\left(\frac{N_e}{p}+2dq\left(\frac{N_e}{p}\right)^{\frac{d-1}{d}}+2^{d-1}d\sum_{i=1}^{q}(i-1)\right)\\ -2d\left(\left(\left(\frac{N_e}{p}\right)^{\frac{1}{d}}+2q\right)^{d-1}-2^{d-1}(d-2)\sum_{i=0}^{q}(q-i)\right)\end{aligned} \tag{5.17}$$

where q is the number (or depth) of overlapping layers and p is the number of processes. In Eq. (5.17), the first term takes into account that each element of a partition, which is widened with the overlapping elements, contributes to the Jacobian matrix with $(1 + 2d)$ blocks, being $2d$ the number of faces of an element, while the second term subtracts the blocks not considered by the preconditioner, being they due to connection between elements at the boundary faces of the augmented partition. For $q = \{0, 1, 2\}$ an estimation of the number of non-zero blocks corresponding to BJ, ASM(1,ILU(0)) and ASM(2,ILU(0)) preconditioners, respectively, can be obtained.

Figures 5.1(a) and 5.1(b) report the ratio between the estimated number of non-zeros of the preconditioner and the system matrix with respect to the number of sub-domains. For $(N_e/p) = 1$, corresponding to one element per partition, BJ reduces to EWBJ, which provides a $1/(2d + 1)$ decrease of the number of non-zeros. On the other hand, for both ASM(1,ILU(0)) and ASM(2,ILU(0)) the number of non-zero entries grows as (N_e/p) approaches one. In the same manner the memory footprint of p-multigrid preconditioners can be estimated.As reference, the three-level p-multigrid strategy is reported and the specs, also reported on top of Table 5.8, reads: $k = 6$, FGMRES(MF) outer solver, GMRES(MB)[ASM(1,ILU(0)] smoothing on the coarsest level ($k = 1$), GMRES(MF)[EWBJ] on the finest level ($k = 6$) and GMRES(MB)[EWBJ] on the intermedite level ($k = 2$). It is worth noticing that the memory allocation of coarse levels preconditioners has as a small impact on the total



(a) $d = 2$ (b) $d = 3$

Figure 5.1: Estimated relative number of non-zeros (NNZ) of the preconditioner with respect to the non-zeros of the Jacobian as a function of the number of elements per partition for a two dimensional ($d = 2$) and three-dimensional case ($d = 3$). See text for details.

number of non-zeros, due the reduction of block size. For instance, in three space dimensions, $\boldsymbol{G}_2$ and $\boldsymbol{G}_1$ have 440 and 70 time less non-zeros that the $\boldsymbol{G}_0$ matrix, respectively.

5.2 Numerical results

5.2.1 Poisson Problem

Since the stabilization scaling influences only the elliptic part of coarse grid operators, it makes sense to assess its effectiveness tackling the numerical solution of a Poisson problem. In particular the performance of a p-multigrid preconditioned FMGRES solver is considered, applied to a high-order $k = 6$ BR2 DG discretization over three h-refined mesh sequences of the bi-unit square $\Omega = [-1, 1]^2$: i) a regular Delaunay triangular mesh sequence (reg-tri), ii) a distorted quadrilateral mesh sequence (dist-quad) obtained by randomly perturbing the nodes of a Cartesian grid, iii) a distorted and graded triangular mesh sequence (grad-tri) where the elements shrink close to the domain boundaries mimicking the end-points clustering of one-dimensional Gaussian quadrature rules in each Cartesian direction. Dirichlet boundary conditions and the forcing term are imposed according to the smooth exact solution $u = e^{-2.5\left((x-1)^2+(y-1)^2\right)}$. The potential field rapidly varies in the proximity of the upper-right corner of the square in order to replicate the presence of a boundary layer.

The p-multigrid preconditioner options are as follows: three-levels ($L = 2$) and a six-levels ($L = 5$) $\mathcal{V}$-cycle iteration, respectively, with ILU(0) right-preconditioned GMRES smoothers on each level. On all levels but the coarsest (that is for $\ell < L$) a single smoothing iteration is performed. On the coarse level, the relative residual tolerance is set to 10^{-3} and a maximum number of iterations of 40 or 400 is imposed. Polynomial degree coarsening on six-levels is achieved by recursively reducing the polynomial degree by one, that is k_ℓ=6−ℓ. On three the coarsening strategy is more aggressive: k=3 is used on the first level while a second-order k=1 DG discretization is employed on the coarsest level. Interestingly, this latter setup seeks to replicate the four-fold degrees of freedom decrease of h-multigrid strategies in two space dimensions.

Table 5.1 and Table 5.2 consider the three- and the six-levels $\mathcal{V}$-cycle iterations, respectively, and assess the benefits of stabilization scaling (scaling on) with respect to standard inherited-p-multigrid coarse grid operators (scaling off). Execution time gains are remarkable on regular triangular and distorted quadrilateral mesh sequences (solution time speedup of 2.4 and 2.2 on average, respectively) but still present on the graded triangular mesh sequence (50% faster solution process on average).

Performance of iterative solver can be evaluated in terms of convergence factor, that is the average residual decrease per iteration, which can be computed as follows

$$\rho = \exp\left(\frac{1}{N_{\text{it}}} \ln \frac{\|d_{N_{\text{it}}}\|}{\|d_0\|}\right),$$

where N_{it} is the number of iterations required to reach the prescribed residual drop, and d_i is the defect (or residual) of the linear system solution at the i-th iteration.

Solver				ℓ	k_ℓ	tol_r	ITs	Smoother		
				0	6	–	1			
FGMRES[MG$_\mathcal{V}$], $L=2$				1	3	–	1	GMRES[ILU(0)]		
				2	1	10^{-3}	400			
grid	scaling off				scaling on				speedup	
reg-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$39^2\cdot 2$	0.0822	0.957	10	157	0.112	0.833	11	38	1.3	1.7
$79^2\cdot 2$	0.0711	0.969	9	223	0.108	0.929	11	95	1.7	2.3
$158^2\cdot 2$	0.0847	0.99	10	399	0.117	0.946	11	125	2	2.5
$311^2\cdot 2$	0.133	0.997	12	399	0.107	0.982	11	385	1.2	1.2
dist-quad	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
32^2	0.0718	0.913	9	76	0.0409	0.764	8	26	1.2	1.5
64^2	0.0694	0.956	9	155	0.0405	0.865	8	48	1.4	2
128^2	0.0641	0.966	9	200	0.0369	0.881	7	55	1.9	3.3
256^2	0.0606	0.989	9	399	0.0279	0.957	7	159	2.1	3
grad-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$32^2\cdot 2$	0.141	0.909	12	73	0.165	0.747	13	24	1.1	1.1
$64^2\cdot 2$	0.204	0.929	15	94	0.214	0.774	15	27	1.2	1.4
$128^2\cdot 2$	0.285	0.96	19	170	0.315	0.925	20	89	1.5	1.8
$256^2\cdot 2$	0.359	0.965	23	194	0.436	0.944	28	120	1.3	1.4

Solver				ℓ	k_ℓ	tol_r	ITs	Smoother		
				0	6	–	1			
FGMRES[MG$_\mathcal{V}$], $L=2$				1	3	–	1	GMRES[ILU(0)]		
				2	1	10^{-3}	40			
grid	scaling off				scaling on				speedup	
reg-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$39^2\cdot 2$	0.194	0.962	15	39	0.112	0.909	11	39	1.1	1.3
$79^2\cdot 2$	0.356	0.983	23	39	0.142	0.954	12	39	1.4	1.8
$158^2\cdot 2$	0.665	0.993	56	39	0.226	0.985	16	39	2.2	3.2
$311^2\cdot 2$	0.819	0.994	113	39	0.408	0.986	26	39	3.1	4.1
dist-quad	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
32^2	0.0939	0.962	10	39	0.0409	0.72	8	21	1.1	1.3
64^2	0.162	0.982	13	39	0.0406	0.877	8	39	1.2	1.5
128^2	0.351	0.976	23	39	0.0669	0.958	9	39	1.5	2.3
256^2	0.621	0.993	48	39	0.148	0.958	13	39	2.2	3.3
grad-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$32^2\cdot 2$	0.146	0.898	12	39	0.164	0.765	13	26	0.99	0.99
$64^2\cdot 2$	0.215	0.971	15	39	0.214	0.867	15	39	1	1
$128^2\cdot 2$	0.381	0.988	24	39	0.315	0.911	20	39	1.1	1.2
$256^2\cdot 2$	0.579	0.989	42	39	0.437	0.933	28	39	1.3	1.5

Table 5.1: $k=6$ BR2 discretization of the Laplace equation, three-levels p-multigrid preconditioner performance on three h-refined mesh sequences, with and without stabilization scaling. Comparison of convergence rates of the outer solver and the coarse smoother (ρ and ρ_c, respectively), comparison of the number of iterations of the outer solver and the coarse smoother (ITs and ITs$_c$, respectively), and evaluation of the speedup $\left(\frac{\text{wall clock time scaling off}}{\text{wall clock time scaling on}}\right)$ considering solution CPU time and solution plus assembly CPU time (Sol and Tot, respectively).

It is interesting to remark that stabilization scaling always improves the convergence factor of the coarse grid solver having a positive impact on the performance of the algorithm, in particular one of two following situations might occur.

1. The prescribed residual drop of 10^{-3} is attained in a smaller number of coarse solver iterations. This is typically observed when the maximum number of iterations is set to 400.

2. The prescribed maximum number of iteration of the coarse solver is attained leading to a tinier defect for the rescaled stabilization algorithm. Accordingly, convergence of the outer solver is improved and a smaller number of FGMRES iterations is required to solve the linear system. This is typically observed when the maximum number of iterations of the coarse solver is set to 40.

	Solver			ℓ	k_ℓ	tol_r	ITs	Smoother		
	FGMRES[MG$_\mathcal{V}$], $L=5$			0,...,4 5	$6-\ell$ 1	$-$ 10^{-3}	1 400	GMRES[ILU(0)]		
grid	scaling off				scaling on				speedup	
reg-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$39^2\cdot 2$	0.0288	0.965	7	196	0.0176	0.885	6	57	1.5	2
$79^2\cdot 2$	0.0281	0.982	7	389	0.0163	0.913	6	76	1.8	2.5
$158^2\cdot 2$	0.0502	0.996	8	399	0.0186	0.962	6	177	1.8	2.2
$311^2\cdot 2$	0.102	0.997	11	399	0.0161	0.99	6	399	1.7	1.9
dist-quad	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
32^2	0.0274	0.908	7	72	0.00659	0.786	5	29	1.2	1.6
64^2	0.0267	0.963	7	186	0.0063	0.876	5	53	1.4	2
128^2	0.0215	0.98	6	350	0.00623	0.927	5	92	1.7	2.4
256^2	0.0325	0.995	7	399	0.00403	0.967	5	206	1.8	2.4
grad-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$32^2\cdot 2$	0.0684	0.887	9	58	0.0443	0.79	8	30	1.1	1.3
$64^2\cdot 2$	0.0907	0.957	10	157	0.0728	0.84	9	40	1.4	1.8
$128^2\cdot 2$	0.146	0.958	12	163	0.117	0.93	11	96	1.4	1.6
$256^2\cdot 2$	0.208	0.982	15	371	0.168	0.955	13	150	1.8	2.1
	Solver			ℓ	k_ℓ	tol_r	ITs	Smoother		
	FGMRES[MG$_\mathcal{V}$], $L=5$			0,...,4 5	$6-\ell$ 1	$-$ 10^{-3}	1 40	GMRES[ILU(0)]		
grid	scaling off				scaling on				speedup	
reg-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$39^2\cdot 2$	0.166	0.972	14	39	0.0279	0.916	7	39	1.4	1.8
$79^2\cdot 2$	0.339	0.985	22	39	0.0664	0.94	9	39	1.6	2.2
$158^2\cdot 2$	0.661	0.991	55	39	0.206	0.983	15	39	2.5	3.3
$311^2\cdot 2$	0.829	0.988	122	39	0.391	0.979	25	39	3.7	4.6
dist-quad	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
32^2	0.0581	0.956	9	39	0.00658	0.794	5	31	1.2	1.6
64^2	0.139	0.979	12	39	0.00959	0.928	5	39	1.4	2
128^2	0.337	0.982	22	39	0.0442	0.964	8	39	1.7	2.4
256^2	0.613	0.985	47	39	0.137	0.959	12	39	2.5	3.5
grad-tri	ρ	ρ_c	ITs	ITs$_c$	ρ	ρ_c	ITs	ITs$_c$	Tot	Sol
$32^2\cdot 2$	0.0743	0.924	9	39	0.0443	0.82	8	35	1.1	1.1
$64^2\cdot 2$	0.143	0.975	12	39	0.0726	0.89	9	39	1.1	1.3
$128^2\cdot 2$	0.343	0.989	22	39	0.118	0.973	11	39	1.5	1.8
$256^2\cdot 2$	0.564	0.991	40	39	0.215	0.99	15	39	2	2.5

Table 5.2: $k = 6$ BR2 discretization of the Laplace equation, six-levels p-multigrid preconditioner performance on three h-refined mesh sequences, with and without stabilization scaling. Comparison of convergence rates of the outer solver and the coarse smoother (ρ and ρ_c, respectively), comparison of the number of iterations of the outer solver and the coarse smoother (ITs and ITs$_c$, respectively), and evaluation of the speedup $\left(\frac{\text{wall clock time scaling off}}{\text{wall clock time scaling on}}\right)$ considering solution CPU time and solution plus assembly CPU time (Sol and Tot, respectively).

Note that uniform convergence with respect to the mesh density is obtained on regular triangular and distorted quadrilateral mesh sequences when employing a sufficiently high number of GMRES iterations on the coarse level. On the distorted and graded triangular mesh sequence the number of FGMRES iterations increases with the mesh density due to the presence of increasingly stretched elements close to the domain boundaries. Note that the number of iteration increase is less pronounced when employing six-levels instead of three-levels for the $\mathcal{V}$-cycle iteration.

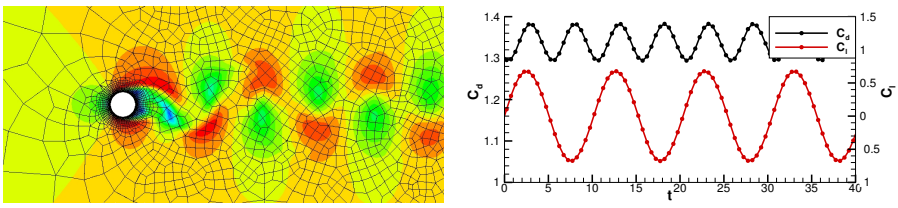
To conclude, it is worth pointing out that the number of iterations of rescaled-inherited and non-inherited multigrid has been checked to be equal on all but the finest grids of the distorted and graded triangular mesh sequence, where the former is slightly sub-optimal as compared to the latter (by at most 20%). This confirms that stabilization terms scaling is almost able to recover the convergence rates of non-inherited multigrid while also cutting down assembly costs.

5.2.2 Incompressible Navier–Stokes equations

In this section the performance of the p-multigrid preconditioner, possibly coupled with the matrix-free solver, is compared to state-of-the-art single-grid strategies in the context of unsteady flow simulations. Two incompressible unsteady flow problems of increasing complexity are here considered: i) the two-dimensional laminar flow around a circular cylinder at $Re = 200$; and ii) the three-dimensional laminar flow around a sphere at $Re = 300$.

Laminar flow past a two-dimensional circular cylinder

The laminar flow around a circular cylinder at $Re = 200$ is solved with $k = 6$ on a computational grid made of 4710 elements with curved edges represented by cubic Lagrange polynomials. The domain extension is $[-50, 100]\times[-50, 50]$ in terms of non-dimensional units. Note that the grid was deliberately generated with a severe grid refinement in the wake region, as well as large elements at the far-field, in order to challenge the solution strategy. Dirichlet and Neumann boundary conditions are imposed at the inflow and outflow boundaries, respectively, while symmetry flow conditions are employed at the top and bottom boundaries. The no-slip Dirichlet boundary condition is imposed on the cylinder wall. A snapshot of the mesh and the velocity magnitude contours is shown in Fig. 5.2(a). Time marching is performed by means of the linearly-implicit four-stage, order three ROSI2PW Runge-Kutta method [29]. The scheme is specifically designed to accurately deal with PDAEs of index 2, like the INS equations. A non-dimensional fixed time step $\delta t = 0.25$, corresponding to 1/20 of the shedding period, is found to be adequate to describe the flow physics and large enough tostress the solution strategy. Fig. 5.2(b) reports lift and drag coefficients history. The drag coefficient $C_d = 1.335$ and the Strouhal number $\mathrm{St} = 0.1959$ are in good agreement with [102] and references therein. Even if, for the sake of efficiency, it is possible to adaptively estimate the relative defect drop tol_r that the linear solver should attain [77], a fix value $tol_r = 10^{-5}$ is set to compare different preconditioners with the same forcing term. The resulting time discretization error, estimated using the embedded Runge–Kutta scheme, is small enough not to affect the overall solution accuracy. Moreover, the defect tolerance is large enough to ensure that the matrix-free approximation error does not affect GMRES convergence, see [77], in consistency



(a) Velocity magnitude iso-contours

(b) C_d and C_l coefficients history

Figure 5.2: Laminar flow around a circular cylinder at $Re = 200$. Velocity magnitude iso-contour.

with what reported in Chapter 4. The right preconditioning approach is employed throughout all the numerical experiments of this paper to reach convergence levels independent of the preconditioner.

Performance assessment The p-multigrid preconditioner approach seeks to minimize the number of GMRES iterations on the fine grid by means of a full p-multigrid solution strategy. A full $\mathcal{V}$-cycle p-multigrid iteration (see Algorithm 13) has many parameters to tune in order to get the best performance: among the others, the most important appear to be i) the choice of the smoother and its preconditioner on each level, ii) the number of smoothing iteration on each level, iii) the forcing term (controlling the exit condition based on the relative defect drop) of the coarse solver and its maximum number of iterations. Accordingly the combination of these parameters lead to a multitude of different configurations that is hard to explore comprehensively. Nevertheless, this Section provides some useful indications that can be directly applied to the simulation of realistic flow problems. In general, with respect to the test cases proposed in Section 5.2.1, the use of a full p-multigrid strategy together with an increased number of smoothing iterations proved to deal more efficiently with the nonlinearity of the governing equations. The experiments are devoted to show the benefits of i) using a rescaled-inherited approach for the coarse space operators to improve the convergence rates of the iterative solver; ii) apply a matrix-free approximation to the fine space to reduce the memory footprint and increase the computational efficiency. Code profiling is applied for 10 time steps starting from a fully developed flow solution obtained with the same polynomial degree, same time step size and solving linear system up to the same tolerance. In practice, the performance of the preconditioners is evaluated on the solution of 40 linear systems. The efficiency of each setting is monitored in parallel, to assess the behavior of different preconditioners in a realistic setup for this kind of computations. The runs are performed on a computational node made by two sixteen-core AMD Opteron CPUs. First, Table 5.3 reports the results obtained using single-grid preconditioners. As expected, the numerical experiments show a sub-optimal parallel efficiency for the BJ preconditioner, indeed the average number of linear iterations increases while increasing the number of subdomains. The iterations number increase tops at 62% when comparing the simulation on 16 cores against the serial one. The ASM(1,ILU(0)) preconditioner partially heals the performance degradation providing a 10% increase of the iterations number at the expenses of a higher memory requirement, as explained in Section 5.1.5. The

Solver	GMRES(MB)		GMRES(MB)		GMRES(MF)		GMRES(MF)	
Prec	BJ		ASM(1,ILU(0))		EWBJ		BJ	
nProcs	ITs	TotTime	ITs	TotTime	ITs	TotTime	ITs	TotTime
1	123.5	3805	123.5	3805	542.8	36700	112.2	9860
2	108.0	1756	121.3	1917	538.7	17980	102.4	4547
4	105.3	859	123.9	982	543.9	9281	103.5	2325
8	138.0	543	120.4	515	542.2	4615	121.4	1333
16	199.7	497	134.7	378	554.4	2934	171.3	995

Table 5.3: Two-dimensional cylinder test case. Single-grid parallel performances, matrix-based and matrix-free implementations. Comparison of the average number of GMRES iterations (ITs) and the whole elapsed CPU time (solution plus assembly) TotTime.

matrix-based and the matrix-free versions of GMRES provide a similar number of iterations with a CPU time that is in favour of the former. This can be explained by the high quadrature cost associated to non-affine mesh elements, see *e.g.*[103]. In fact, while in the previous Chapter it has been demonstrated that the residual computation and a matrix-vector product has similar costs when dealing with high-order discretizations on affine elements, see also [77], in this case the numerical quadrature expense has a higher relative cost on residual evaluation. It is worth pointing out that no attempt was made to optimize numerical quadrature, in particular elements located far from curved boundaries are still treated as high-order non-affine elements for the sake of simplicity. Even if matrix-free iterations can be further improved in term of efficiency in realistic applications, this is beyond the scope of the present work. Table 5.4 allows to evaluate the impact of the fine smoother preconditioner on the computational efficiency. Two parameters of interest are reported, the average number of FGMRES iterations (ITs) and the speed-up with respect to the best performing single-grid preconditioner, $\mathrm{SU_{MB}} = \mathrm{TotTime}/\mathrm{TotTime_{ref}}$, where $\mathrm{TotTime_{ref}}$ is the total CPU time of the GMRES(MB)[ASM(1,ILU(0))] approach. The specs of the p-multigrid iteration setup are reported in the top of the table. The Table also compares the standard-inherited approach (*scaling off*) with the rescaled-inherited one (*scaling on*).

FGMRES[$\mathrm{MG_{full}}$] with 3 GMRES(MB)[BJ] smoothing iterations provides a speed-up of about 2 in serial computations with respect to the reference strategy. Although the solver is still faster than the reference one, the parallel performance is not satisfactory being an increase in the number of iterations observed. As expected a better scalability can be obtained with 3 GMRES(MB)[ASM(1,ILU(0))] smoothing iterations. The numerical experiment revealed that to increase the number of iterations from 3 to 8 is mandatory to maintain the smoothing efficiency of GMRES(MB)[EWBJ],

Solver	ℓ	k_ℓ	tol_r	ITs	Smoother	Prec
FGMRES[$\mathrm{MG_{full}}$]	0,1	6, 2	–	*	GMRES(MB)	‡
	2	1	–	40	GMRES(MB)	ASM(1,ILU(0))
scaling off	*3 ‡BJ		*3 ‡ASM(1,ILU(0))		*8 ‡BJ	*8 ‡EWBJ
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	ITs	SU_{MB}	ITs	SU_{MB}
1	4.10	2.02	4.10	1.98	2.98	1.76	5.48	1.56
2	4.63	1.82	4.05	1.90	3.10	1.65	5.48	1.49
4	5.73	1.65	4.05	1.90	3.63	1.53	5.63	1.48
8	7.20	1.39	4.35	1.74	3.85	1.40	5.63	1.40
16	8.63	1.37	5.38	1.71	5.05	1.28	5.70	1.53
scaling on	*3 ‡BJ		*3 ‡ASM(1,ILU(0))		*8 ‡BJ		*8 ‡EWBJ	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	ITs	SU_{MB}	ITs	SU_{MB}
1	3.43	2.11	3.43	2.11	2.48	1.88	3.50	1.92
2	3.68	1.99	3.55	1.97	2.80	1.72	3.53	1.84
4	4.78	1.79	3.60	1.95	2.55	1.80	3.83	1.79
8	5.13	1.51	3.60	1.74	2.58	1.57	3.68	1.60
16	7.65	1.20	4.10	1.67	2.85	1.49	3.50	1.68

Table 5.4: Two-dimensional cylinder test case. Effects of the smoother type and the rescaled-inherited coarse spaces on parallel performance. Comparison of the average number of FGMRES iterations (ITs) and the speed-up ($\mathrm{SU_{MB}}$) of the p-multigrid preconditioner with respect to the best performing single-grid preconditioner. The asterisk and the double dagger symbols in the solver specs row are placeholders for the number of iterations (ITs) and coarse solver type of each column, respectively.

and to achieve a satisfactory performance in parallel. Indeed, despite being less performing in serial runs, the number of iterations is almost independent from the number of processes. It is worth noting that increasing the number of iterations of GMRES(MB)[BJ] does not pay off in terms of speedup. The number of FGMRES iterations is significantly reduced in all the cases when considering rescaled-inherited coarse grid operators. Although the strategy does not always pay off in terms of speedup because of the increased expense of building coarse grid operators, the GMRES(MB)[EWBJ] smoother applied to rescaled-inherited coarse operator is competitive with more expensive preconditioners in parallel computations. Interestingly, the EWBJ preconditioner is the cheapest from the memory footprint viewpoint.

Table 5.5 compares the computational efficiency when varying the preconditioner on the coarsest level. The cheapest and efficient GMRES(MB)[EWBJ] smoother is employed on all levels but the coarsest. The top and bottom of the table include results for a matrix-based and a matrix-free approach, respectively. Note that only on the finest level matrix-vector products are performed matrix free, both within the outer FGMRES iteration and the fine GMRES smoother. Indeed, since the coarse

Solver	ℓ	k_ℓ	tol_r	ITs	Smoother	Prec
	0	6	–	8	*	EWBJ
FGMRES[MG_{full}]	1	2	–	8	GMRES(MB)	EWBJ
	2	1	–	40	GMRES(MB)	‡
scaling off	*GMRES(MB) ‡BJ		*GMRES(MB) ‡ASM(1,ILU(0))		*GMRES(MB) ‡ASM(1,ILU(1))	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	ITs	SU_{MB}
1	5.48	1.56	5.48	1.56	4.73	1.64
2	5.63	1.47	5.48	1.49	4.73	1.56
4	5.43	1.52	5.63	1.48	4.73	1.57
8	5.90	1.38	5.63	1.40	4.75	1.47
16	6.85	1.37	5.70	1.53	4.95	1.62
scaling on	*GMRES(MB) ‡BJ		*GMRES(MB) ‡ASM(1,ILU(0))		*GMRES(MB) ‡ASM(1,ILU(1))	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	ITs	SU_{MB}
1	3.50	1.92	3.50	1.92	3.15	1.96
2	3.55	1.85	3.53	1.84	3.15	1.88
4	3.35	1.91	3.83	1.79	3.15	1.89
8	3.33	1.70	3.68	1.60	3.15	1.67
16	3.48	1.70	3.50	1.68	3.18	1.73
scaling off	*GMRES(MF) ‡BJ		*GMRES(MF) ‡ASM(1,ILU(0))		*GMRES(MF) ‡ASM(1,ILU(1))	
nProcs	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}
1	0.61	1.59	0.61	1.59	0.69	1.80
2	0.60	1.41	0.61	1.45	0.69	1.63
4	0.60	1.42	0.59	1.41	0.69	1.63
8	0.55	1.43	0.57	1.48	0.66	1.70
16	0.61	1.61	0.73	1.92	0.77	2.03
scaling on	*GMRES(MF) ‡ BJ		*GMRES(MF) ‡ASM(1,ILU(0))		*GMRES(MF) ‡ASM(1,ILU(1))	
nProcs	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}
1	0.89	2.31	0.89	2.30	0.98	2.54
2	0.92	2.18	0.89	2.11	0.98	2.32
4	0.93	2.21	0.87	2.06	0.97	2.30
8	0.86	2.23	0.85	2.20	0.92	2.39
16	1.06	2.80	1.07	2.81	1.16	3.05

Table 5.5: Two-dimensional cylinder test case. Effects of the coarse level solver on parallel performance. Comparison of the average number of FGMRES iterations (ITs) and the speed-up of the p-multigrid preconditioner with respect to the best performing single-grid preconditioner in its matrix-based and matrix-free implementation (SU_{MB} and SU_{MF}, respectively). The asterisk and the double dagger symbols in the solver specs row are placeholders for the smoother and coarse solver types of each column, respectively.

levels operators are fairly inexpensive to store in memory, the moderate memory savings of a matrix-free implementation would not justify the increased computational costs. The results highlight that a further improvement in computational efficiency is achieved by means of a [ASM(1,ILU(1))] preconditioner for the coarsest smoother: the number of FGMRES iterations decreases while maintain optimal scalability and the speedup values for the matrix-free approach increase when considering more subdomains. Due to low polynomial degree of the coarsest level, the additional level of fill of the ILU factorization is not significant from the memory footprint viewpoint. Interestingly, the increased robustness of the rescaled-inherited p-multigrid approach results in similar speedups for all the coarse level solver options. For the matrix-free implementation two different speedup values are reported: i) $\mathrm{SU_{MB}}$ considers as a reference the GMRES(MB)[ASM(1,ILU(0))] solver, that is the best performing matrix-based single-grid strategy, ii) $\mathrm{SU_{MF}}$ considers as a reference the GMRES(MF)[BJ] solver, that is the best performing matrix free single-grid strategy. Note that the inefficient GMRES(MF)[EWBJ] configuration is discarded despite being the less demanding solver from the memory viewpoint. Since the relative cost of the solution with respect to matrix assembly is higher in a matrix-free implementation, reducing the number of FGMRES iterations does pay off. Accordingly the benefits of rescaled-inherited coarse grid operator are more evident: the total execution time is comparable with GMRES(MB)[ASM(1,(ILU(0))] and almost three times faster than GMRES(MF)[BJ].

To conclude Table 5.6 compares three- and four-levels p-multigrid preconditioners. The additional level significantly reduces the number of FMGRES iterations at the expense of storing a fourth degree coarse grid operator. Once again the benefits in terms of speedup are most significant in the matrix-free framework, where solution times dominates assembly times. Matrix-free rescaled-inherited p-multigrid is 30% faster then the best performing matrix-based single-grid solver and 3.5 times faster than the best performing matrix-free single-grid solver.

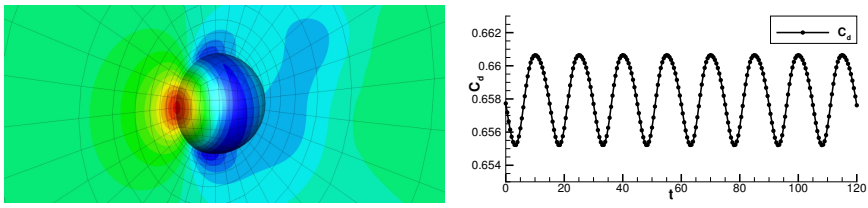
Solver	ℓ	k_ℓ	tol_r	ITs	Smoother		Prec	
	0	6	-	8	*		EWBJ	
FGMRES[$\mathrm{MG_{full}}$]	1,...,L-1	‡	-	8	GMRES(MB)		EWBJ	
	L	1	-	40	GMRES(MB)		ASM(1,ILU(1))	
	GMRES(MB)*		GMRES(MB)*		GMRES(MF)*		GMRES(MF)*	
scaling off	$2^‡$ (L=2)		$4,2^‡$ (L=3)		$2^‡$ (L=2)		$4,2^‡$ (L=3)	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}
1	4.73	1.64	3.00	1.62	0.69	1.80	0.88	2.29
2	4.73	1.56	3.00	1.53	0.69	1.63	0.87	2.06
4	4.73	1.57	3.00	1.54	0.69	1.63	0.87	2.05
8	4.75	1.47	3.10	1.43	0.66	1.70	0.78	2.02
16	4.95	1.62	3.33	1.55	0.77	2.03	0.97	2.54
	GMRES(MB)*		GMRES(MB)*		GMRES(MF)*		GMRES(MF)*	
scaling on	$2^‡$ (L=2)		$4,2^‡$ (L=3)		$2^‡$ (L=2)		$4,2^‡$ (L=3)	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}
1	3.15	1.96	2.00	1.91	0.98	2.54	1.19	3.09
2	3.15	1.88	2.00	1.82	0.98	2.32	1.18	2.80
4	3.15	1.89	2.00	1.83	0.97	2.30	1.19	2.82
8	3.15	1.67	2.00	1.63	0.92	2.39	1.11	2.87
16	3.18	1.73	2.00	1.72	1.16	3.05	1.33	3.50

Table 5.6: Two-dimensional cylinder test case. Comparison of a three-level and four-level p-multigrid strategy in optimal settings for matrix-based and matrix-free fine level options.

Three-dimensional laminar flow past a sphere

As a three-dimensional validation test case, the unsteady laminar flow past a sphere at $Re = 300$ [104, 105, 106, 83] is reported. The solution is characterised by a perfectly periodic behaviour, with the flow maintaining a plane of symmetry. In the present computations, the symmetry plane was enforced by defining a proper boundary condition. The mesh is made of 3560 elements with a bi-quadratic geometrical representation of the wall boundary, see Fig. 5.3(a). The computational domain is obtained via extrusion of the wall surface discretization. While the no-slip condition is set at the wall, velocity inflow and pressure outflow boundary conditions are imposed on the spherical farfield located at 50 diameters. A $k = 6$ representation of the solution was employed for all computations presented hereafter. Note that the small number of mesh elements together with the lack of a refined region in the wake of the sphere reduce the stiffness of the problem. The parallel performance is evaluated running on the Marconi-A1 HPC platform hosted by CINECA, the italian supercomputing center. Scalability is assessed on a single-node base, as the CPU time of the serial computation exceeded the maximum wall-clock time allowed by CINECA. The number of mesh elements is optimised to ensure that all the solution strategies fit the memory of a single node (118 GB). Despite the small size of the problem the following numerical experiments aim at providing reliable indications on the parallel performance that can be extended to real-size production runs.

The solution is accurately integrated in time with a fixed non-dimensional time step $\delta t = 0.5$ and a relative tolerance on the linear system defect drop of $tol_r = 10^{-5}$. The drag coefficient time history is shown in Fig. 5.3(b), its mean value reads 0.659, and the Strouhal number is $\mathrm{St} = 0.133$, in agreement with the published literature, see [83]. Despite the geometry being represented with second degree polynomial spaces, the degree of exactness of quadrature rules does not consider the degree of mappings from reference to physical mesh elements. Accordingly bilinear forms are exactly integrated only over affine mesh elements, located far away from the sphere boundaries. It has been verified that, for this test case, this practice does not compromise accuracy while significantly improving the matrix-free computational time, see [77]. For the sake of efficiency of parallel runs, the mesh has been partitioned using the *local* two-level partitioning strategy described in [107]. The first-level decomposition is performed according to the number of nodes, thus, the second-level



(a) C_p iso-contours (b) C_D

Figure 5.3: Laminar flow around a Sphere at $Re = 300$. Pressure coefficient iso-contours (top) and drag coefficient history (bottom).

decomposition further decompose each node-local partition according to the number of cores per node, such that the extra-node MPI communications are minimized.

Performance assessment Table 5.7 reports the parallel performance of the single-grid matrix-based and matrix-free solvers running in parallel up to 576 cores (6 elements per partition on average). Increasing the number of sub-domains from 36 to 576 leads to an increased number of GMRES iterations: 42% and 20% up when employing a BJ and an ASM(1,ILU(0)) preconditioner, respectively. Thanks to the use of quadrature rules suited for affine meshes, the CPU time of the matrix-free solver is similar to the matrix-based one.

Results reported in Table 5.8 for a three- and four-levels p-multigrid strategy and the exact same setup of two-dimensional computations confirm the efficacy of the multigrid preconditioner: the number of iterations stays the same up to 576 cores and

Solver	GMRES(MB)		GMRES(MB)		GMRES(MF)		GMRES(MF)	
Prec	BJ		ASM(1,ILU(0))		BJ		ASM(1,ILU(0))	
nProcs	ITs	TotTime	ITs	TotTime	ITs	TotTime	ITs	TotTime
36	78.5	1448.1	34.5	1245.9	77.1	1365.9	34.7	1220.7
72	86.7	774.0	35.0	675.2	87.0	743.9	35.0	671.9
144	84.9	380.1	38.2	386.7	85.2	370.3	38.2	381.9
288	102.7	226.7	40.2	233.1	102.4	221.3	40.3	228.5
576	111.8	126.0	41.3	150.6	113.6	129.2	41.3	133.8

Table 5.7: Three dimensional incompressible flow around a sphere. Single-grid parallel performances, matrix-based and matrix-free implementations. Comparison of the average number of GMRES iterations (ITs) and the whole elapsed CPU time (solution plus assembly) TotTime. Computations performed on Marconi-A1@CINECA.

Solver	ℓ	k_ℓ	tol_r	ITs	Smoother		Prec	
	0	6	-	8	*		EWBJ	
FGMRES[MG_{full}]	1,...,L-1	‡	-	8	GMRES(MB)		EWBJ	
	L	1	-	40	GMRES(MB)		ASM(1,ILU(0))	
	*GMRES(MB)		*GMRES(MB)		*GMRES(MF)		*GMRES(MF)	
scaling off	‡2 (L=2)		‡4,2 (L=3)		‡2 (L=2)		‡4,2 (L=3)	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}
36	4.00	1.29	2.00	1.61	1.37	1.29	2.28	2.15
72	4.00	1.37	2.00	1.68	1.52	1.46	2.38	2.29
144	4.00	1.29	2.00	1.43	1.45	1.41	2.07	2.02
288	4.00	1.37	2.00	1.67	1.56	1.52	2.14	2.09
576	4.00	1.28	2.00	1.55	1.46	1.50	1.55	1.59
	GMRES(MB)*		*GMRES(MB)		*GMRES(MF)		*GMRES(MF)	
scaling on	‡2 (L=2)		‡4,2 (L=3)		‡2 (L=2)		‡4,2 (L=3)	
nProcs	ITs	SU_{MB}	ITs	SU_{MB}	SU_{MB}	SU_{MF}	SU_{MB}	SU_{MF}
36	3.0	1.43	2.00	1.51	1.53	1.44	2.09	1.97
72	3.0	1.52	2.00	1.61	1.62	1.56	2.18	2.09
144	3.0	1.36	2.00	1.50	1.55	1.51	1.93	1.88
288	3.0	1.56	2.00	1.60	1.71	1.67	2.02	1.97
576	3.0	1.45	2.00	1.43	1.63	1.67	1.73	1.78

Table 5.8: Three dimensional incompressible flow around a sphere. Efficiency of a three and four level p-multigrid strategy varying the fine level smoother. Comparison of the average number of FGMRES iterations (ITs) and the speed-up of the p-multigrid preconditioner with respect to the best performing single-grid preconditioner in its matrix-based and matrix-free implementation ($SU_{MB}$and SU_{MF},respectively). The asterisk and the double dagger symbols in the solver specs row are placeholders for the smoother and coarse solver types of each column, respectively. Computations performed on Marconi-A1@CINECA.

the speedup are maintained in this largely-parallelized scenario. Stabilization scaling provides slight improvements in terms of FGMRES iterations and computation time, despite the small number of mesh elements. The four-level p-multigrid preconditioner is almost two times faster than the best single-grid setup.

Chapter 6

Preconditioning novel space discretizations

In Chapter 4 and 5 it has been shown that the application of implicit time integration strategies to DG discretizations is hindered by computational time and memory expenses associated with the assembly and storage of the residual Jacobian matrix. Although the Jacobian is sparse, the number of non-zero entries scales as k^{2d}, where k is the approximation order and d is the spatial dimension. Thus, the costs grow rapidly with approximation order, particularly in three-dimensional problems. Motivated by this scaling, the chapters consider the opportunity of a reduced matrix storage implementation of the iterative solver, labelled as matrix-free. This implementation avoids the allocation of the Jacobian matrix but still requires the allocation of a preconditioner operator which in some cases may still be quite large. In this context, the use of multilevel matrix-free strategies with cheap element-wise block-Jacobi preconditioners on the finest level appears to balance computational efficiency and memory considerations with iterative performance for stiff systems. The latter is relevant to solvers applied to DG discretizations, for which the condition number scales as $\mathcal{O}(h^{-2})$, where h is the mesh dimension [108].

The size of the DG system can be further reduced through hybridizable discontinuous Galerkin (HDG) methods, which have been recently considered as an alternative to the standard discontinous Galerkin discretization [109, 110]. HDG methods introduce an additional trace variable on the mesh faces but can reduce the number of globally-coupled degrees of freedom relative to DG, when a high order of polynomial approximation is employed. The reduction occurs through a static condensation of the element-interior degrees of freedom, exploiting the block structure of the HDG Jacobian matrix. Thanks to this operation, the memory footprint of the solver scales as $k^{2(d-1)}$. Additionally, HDG methods exhibit superconvergence properties of the gradient variable in diffusion-dominated regimes. On the other hand, they increase the number of operations local to each element, both before and after the linear system solution. While several works have compared the accuracy and cost of HDG versus continuous [111, 112] and discontinuous [113, 110] Galerkin methods, a comparison of iterative solvers is missing in this context.

Whereas HDG reduces the number of globally-coupled degrees of freedom relative to DG, at high approximation orders, its element-local operation count is non-trivial. This is particularly the case for viscous problems, in which the state gradient is approximated as a separate variable. An alternative approach is to only approximate the state, and to obtain the gradient when needed by differentiating the state. This leads to the *primal* HDG formulation, which is also considered here. The advantage of *primal* HDG relative to HDG lies mainly in the reduction of element-local operations,

which translates into improved computational performance, at the expenses of the loss of gradient variable superconvergence.

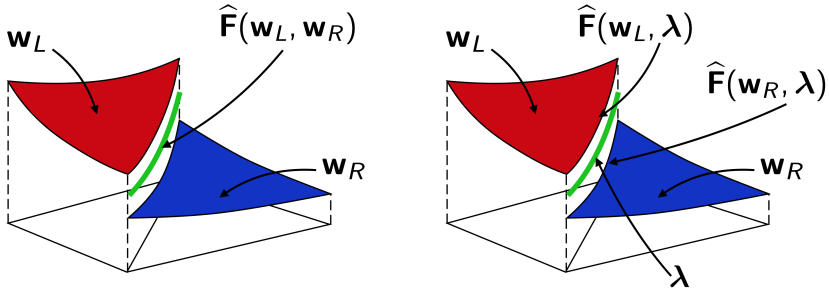
While for DG the use of multilevel strategies to deal with ill-conditioned systems has been previously studied, there use in HDG contexts appears not to have yet been explored, especially for unsteady flow problems. A multilevel technique has been introduced in the context of an h-multigrid strategy built using the trace variable projection on a continuous finite element space [108]. Similarly, [114] propose the use of an algebraic multigrid method applied to a linear finite element space obtained by the projection of the trace variable. In [115] a similar idea is also considered to speed-up the iterative solution process in HDG.

The present chapter focuses on the comparison of different preconditioning strategies to deal with the solution of stiff linear systems arising from a high-order time discretization in the context of DG and HDG spatial discretizations. The scalability of the linear solution process is also considered and compared to standard single-grid preconditioners like ILU(0). The efficiency of the different solution strategies is assessed on two-dimensional laminar compressible flow problems. The results of such cases suggest that (i) similar error levels can be obtained by the two solvers, (ii) the use of a multilevel strategy reduces considerably the number of linear iterations, and (iii) only for the DG discretizations this advantage is reflected in the CPU time.

It is important to remark that this chapter mainly focuses on compressible, low-mach number flows, and the computations were performed using a different solver than that employed in the previous chapters. In fact, both DG and HDG are handled within the same framework called XFlow, developed by Fidkowski and co-workers, see [110] for further details.

6.1 Hybridizable discontinuous Galerkin discretization

The hybridizable discontinuous Galerkin (HDG) method introduces an additional set of variables defined on the mesh element interfaces to reduce the globally coupled degrees of freedom compared to DG, see Fig. 6.1. The main difference between the two discretizations is that, while for DG the numerical flux at the mesh element



(a) DG (b) HDG

Figure 6.1: Comparison of the DG vs HDG discretization

interfaces is unique and it is computed using the left and right states, in HDG the flux is local to each element, since it depends on the internal state as well as the trace variable, and it is required to be continuous by using an additional set of equations. In this chapter two different implementations of the hybridizable DG method are considered and compared to the standard DG methods. The first approach make use of a mixed form for the gradient states (also known as *dual variable*), *i.e.* the gradients are used as an additional element-wise variable, and increases the accuracy of the gradient evaluation. An alternative approach, hereby regarded as *primal* HDG, or *p*HDG [116], reduces the computational costs of the solver by removing the dual variable from the equations and adding a dual-consistency term on the residuals. This approach is observed favourable when an increased accuracy of the gradient variable is not strictly necessary or achievable for the problem. In all cases, the discretization is based on an approximation Ω_h of the domain Ω and a triangulation $\mathcal{T}_h = \{K\}$ of Ω_h. The same nomenclature reported in 2 is here employed. However, the use of tensor-product basis functions will be also explored and compared to those of maximum degree k, exploited in the remaining of the paper. For brevity, the former will be labelled as `QuadLagrange`, while the latter as `TriLagrange` basis functions.

6.1.1 Mixed form

The *m*HDG discretization consider a system of first-order partial differential equations, that can be obtained from (2.7) by introducing $\boldsymbol{\tau}_h \in \mathbb{R}^{m\times d}$ such that

$$\begin{aligned} \int_\Omega \boldsymbol{\tau}_h - \int_\Omega \nabla \boldsymbol{w}_h &= \mathbf{0}, \\ \int_\Omega \partial_t \boldsymbol{w} + \int_\Omega \nabla\cdot(\boldsymbol{F}^c - \boldsymbol{F}^\nu) + \int_\Omega \boldsymbol{s} &= \mathbf{0}. \end{aligned} \tag{6.1}$$

The HDG discretization approximates the variables $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\tau}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, with $\mathbb{P}_d^k(\mathcal{T}_h)$ defined in Chapter 2. An additional trace variable $\boldsymbol{\lambda}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$ is defined in the space

$$\mathbb{P}_d^k(\mathcal{F}_h) = \left\{ \mu_h \in L^2(\Gamma_h) \mid \forall \sigma \in \mathcal{F}_h, \mu_h|_\kappa \in \mathbb{P}_d^k(\sigma) \right\} \tag{6.2}$$

where $\mathbb{P}_d^k(\sigma)$ is the space of polynomials of order k on face σ. Note that the trace variable is defined in the internal faces only, while a properly defined boundary value is used for the flux computation on $\mathcal{F}_h^b$.

The weak form is obtained in this case by weighting the equations in (6.1) with appropriate test functions, integrating by parts, and using the interface variable $\boldsymbol{\lambda}_h$ for the face state. Consistent and stable numerical fluxes are required at the mesh element interfaces. The variational formulation reads: find $\boldsymbol{\tau}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\lambda}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$ such that

$$\sum_{\kappa\in\mathcal{T}_h}\int_\kappa \boldsymbol{\tau}_h : \boldsymbol{\pi}_h + \sum_{\kappa\in\mathcal{T}_h}\int_\kappa (\nabla\cdot\boldsymbol{\pi}_h)\cdot\boldsymbol{w}_h - \sum_{\kappa\in\mathcal{T}_h}\sum_{\sigma\in\mathcal{F}_{\partial\kappa}}\int_\sigma \boldsymbol{\lambda}_h\cdot(\boldsymbol{\pi}_h\cdot\boldsymbol{n}^\sigma) = \mathbf{0}, \tag{6.3}$$

$$\sum_{\kappa\in\mathcal{T}_h}\int_\kappa \partial_t \boldsymbol{w}_h\cdot\boldsymbol{z}_h - \sum_{\kappa\in\mathcal{T}_h}\int_\kappa (\boldsymbol{F}_h^c-\boldsymbol{F}_h^\nu):\nabla_h\boldsymbol{z}_h + \\ + \sum_{\kappa\in\mathcal{T}_h}\sum_{\sigma\in\mathcal{F}_{\partial\kappa}}\int_\sigma \boldsymbol{n}^\sigma\cdot\left(\widehat{\boldsymbol{F}}_h^c-\widehat{\boldsymbol{F}}_h^\nu\right)\cdot[\![\boldsymbol{z}_h]\!]=\boldsymbol{0}, \tag{6.4}$$

$$\sum_{\sigma\in\mathcal{F}_h^i}\int_\sigma \left\{\left\{\boldsymbol{n}^\sigma\cdot\left(\widehat{\boldsymbol{F}}_h^c-\widehat{\boldsymbol{F}}_h^\nu\right)\right\}\right\}\cdot\boldsymbol{\mu}_h=\boldsymbol{0}, \tag{6.5}$$

for all $\boldsymbol{\pi}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\mu}_h\in[\mathbb{P}_d^k(\mathcal{F}_h)]^m$, where $\mathcal{F}_{\partial\kappa}$ is the set of faces $\partial\kappa$ of the element κ. Note that, instead of using the modified flux function $\widetilde{\boldsymbol{F}}_h$, here the viscous fluxes $\boldsymbol{F}_h^\nu$ are evaluated in the element terms using the gradient $\boldsymbol{\tau}_h$. The third equation, which weakly imposes the continuity across interfaces, is required to close the system. Note that the use of the mixed form allows the same theoretical convergence rates between the state variable $\boldsymbol{w}_h$ and the gradient variable $\boldsymbol{\tau}_h$, which in diffusion-dominated regimes allows for a local post-processing of the state to a higher order.

In HDG, the numerical flux functions $\boldsymbol{F}_h^c$ and $\boldsymbol{F}_h^\nu$ are defined defined as

$$\begin{aligned}\boldsymbol{n}^\sigma\cdot\widehat{\boldsymbol{F}}_h^c &= \boldsymbol{n}^\sigma\cdot\boldsymbol{F}_h^c+\left|\boldsymbol{n}^\sigma\cdot\boldsymbol{F}_c'(\boldsymbol{\lambda}_h)\right|(\boldsymbol{w}_h-\boldsymbol{\lambda}_h),\\ \boldsymbol{n}^\sigma\cdot\widehat{\boldsymbol{F}}_h^\nu &= \boldsymbol{n}^\sigma\cdot\boldsymbol{F}_h^\nu+\eta_\sigma\boldsymbol{n}^\sigma\cdot\boldsymbol{r}^\sigma(\boldsymbol{w}_h-\boldsymbol{\lambda}_h),\end{aligned} \tag{6.6}$$

where the first one is the sum of the convective flux and a Roe-like stabilization term, while the second mimics the BR2 implementation being $\boldsymbol{r}^\sigma$ the lifting operator applied to the jump $(\boldsymbol{w}_h-\boldsymbol{\lambda}_h)$, and η_σ the stabilization factor.

Similarly to what is done for DG, at the boundary of the domain, the numerical flux function is made consistent with the boundary conditions of the problem through the definition of a boundary state which accounts for the boundary data and, together with the internal state, allows for the computation of numerical fluxes and the lifting operator on the portion Γ_h^b of the boundary Γ_h.

To obtain a compact form for the time integration of the equations, it is convenient to define the flux functions

$$\boldsymbol{F}_h(\boldsymbol{\tau}_h,\boldsymbol{w}_h)\overset{\text{def}}{=}(\boldsymbol{F}_h^c-\boldsymbol{F}_h^\nu)\in\mathbb{R}^{m\times d} \tag{6.7}$$

$$\widehat{\boldsymbol{F}}_h(\boldsymbol{\tau}_h,\boldsymbol{w}_h,\boldsymbol{\lambda}_h)\overset{\text{def}}{=}\left(\widehat{\boldsymbol{F}}_h^c-\widehat{\boldsymbol{F}}_h^\nu\right)\in\mathbb{R}^{m\times d}, \tag{6.8}$$

as well as the residuals forms q, f and l from Equations (6.3)-(6.5). For all $\boldsymbol{\tau}_h,\boldsymbol{\pi}_h\in$

$[\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$, the forms read

$$\begin{aligned} q_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\pi}_h) = &\sum_{\kappa\in\mathcal{T}_h}\int_\kappa \sum_{i=1}^m\sum_{p=1}^d \tau_{p,i}\pi_{p,i} + \\ &+ \sum_{\kappa\in\mathcal{T}_h}\int_\kappa \sum_{i=1}^m\sum_{p=1}^d w_i \frac{\partial \pi_{p,i}}{\partial x_p} + \\ &- \sum_{\kappa\in\mathcal{T}_h}\sum_{\sigma\in\mathcal{F}_{\partial\kappa}}\int_\sigma \sum_{i=1}^m\sum_{p=1}^d \lambda_i n_p^\sigma \left[\!\left[\pi_{p,i}\right]\!\right] \end{aligned} \tag{6.9}$$

$$\begin{aligned} f_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h) = &- \sum_{\kappa\in\mathcal{T}_h}\int_\kappa \sum_{i=1}^m\sum_{p=1}^d F_{p,i}(\boldsymbol{\tau}_h, \boldsymbol{w}_h) \frac{\partial z_i}{\partial x_p} + \\ &+ \sum_{\kappa\in\mathcal{T}_h}\sum_{\sigma\in\mathcal{F}_{\partial\kappa}}\int_\sigma \sum_{i=1}^m\sum_{p=1}^d n_p^\sigma \widehat{F}_{p,i}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h) \left[\!\left[z_i\right]\!\right], \end{aligned} \tag{6.10}$$

$$l_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h) = \sum_{\sigma\in\mathcal{F}_h^i}\int_\sigma \sum_{i=1}^m\sum_{p=1}^d n_p^\sigma \left\{\!\!\left\{ \widehat{F}_{p,i}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h) \right\}\!\!\right\} \mu_i. \tag{6.11}$$

Recalling the mass matrix bilinear form (2.22), the unsteady HDG problem can be rewritten as follows: find $\boldsymbol{\tau}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\lambda}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$ such that

$$\begin{aligned} &q_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\pi}_h) = 0 \\ &m_h(\boldsymbol{w}_h, \boldsymbol{z}_h) + f_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h) = 0 \\ &l_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h) = 0 \end{aligned} \tag{6.12}$$

for all $\boldsymbol{\pi}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\mu}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$.

6.1.2 Primal form

A variant of the mixed hybridizable discontinuous Galerkin presented in Section 6.1.1 is here denoted as the *primal* HDG method (pHDG) and follows the work proposed in [116]. Here the dual variable is eliminated by introducing the definition of the gradient in Eq. (6.4). The pHDG approximates the variable $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$. The trace variable $\boldsymbol{\lambda}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$ is still employed for hybridization, and the variational formulation reads: find $\boldsymbol{\tau}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\lambda}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$ such that

$$\begin{aligned} \sum_{\kappa\in\mathcal{T}_h}\int_\kappa \partial_t \boldsymbol{w}_h \cdot \boldsymbol{z}_h - &\sum_{\kappa\in\mathcal{T}_h}\int_\kappa \left(\boldsymbol{F}_h^c - \widetilde{\boldsymbol{F}}_h^\nu\right) : \nabla_h \boldsymbol{z}_h + \\ &+ \sum_{\kappa\in\mathcal{T}_h}\sum_{\sigma\in\mathcal{F}_{\partial\kappa}}\int_\sigma \boldsymbol{n}^\sigma \cdot \left(\widehat{\boldsymbol{F}}_h^c - \widehat{\boldsymbol{F}}_h^\nu\right) \cdot \left[\!\left[\boldsymbol{z}_h\right]\!\right] = \boldsymbol{0}, \end{aligned} \tag{6.13}$$

$$\sum_{\sigma\in\mathcal{F}_h^i}\int_\sigma \left\{\left\{\boldsymbol{n}^\sigma\cdot\left(\widehat{\boldsymbol{F}}_h^c-\widehat{\boldsymbol{F}}_h^\nu\right)\right\}\right\}\cdot\boldsymbol{\mu}_h=\boldsymbol{0}, \tag{6.14}$$

for all $\boldsymbol{\pi}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\mu}_h\in[\mathbb{P}_d^k(\mathcal{F}_h)]^m$, where $\mathcal{F}_{\partial\kappa}$ is the set of faces $\partial\kappa$ of the element κ. It is worth pointing out that Equations (6.13)–(6.14) are not obtained by just substituting $\boldsymbol{\tau}_h=\nabla_h\boldsymbol{w}_h$ from Equations (6.3)–(6.5). In fact, the flux function $\widetilde{\boldsymbol{F}}_h^\nu$ defined in Eq. (2.15) is employed to ensure symmetry of the discretization. On the other hand, the same flux functions $\widehat{\boldsymbol{F}}_h^c$ and $\widehat{\boldsymbol{F}}_h^\nu$ reported in Eq. (6.6) are employed. It is important to remark that, involving a lower number of element-wise degrees of freedom than mHDG, this space discretization does not suffer significantly from overhead costs of dealing with the gradients: eliminating the dual variable and adding the adjoint-consistency term $\widetilde{\boldsymbol{F}}_h^\nu$ typically results in a faster solver. Note that in this case the gradients are one order less accurate than the state variable $\boldsymbol{w}_h$. Numerical flux functions, stabilizing terms, and boundary condition enforcement are defined similarly to mHDG.

Defining the flux functions as in Eq. (6.6), and

$$\widetilde{\boldsymbol{F}}_h(\boldsymbol{w}_h)\overset{\text{def}}{=}\left(\boldsymbol{F}_h^c-\widetilde{\boldsymbol{F}}_h^\nu\right)\in\mathbb{R}^{m\times d} \tag{6.15}$$

$$\widehat{\boldsymbol{F}}_h(\boldsymbol{w}_h,\boldsymbol{\lambda}_h)\overset{\text{def}}{=}\left(\widehat{\boldsymbol{F}}_h^c-\widehat{\boldsymbol{F}}_h^\nu\right)\in\mathbb{R}^{m\times d}, \tag{6.16}$$

the following bilinear forms arise from pHDG discretization. For all $\boldsymbol{w}_h,\boldsymbol{z}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\lambda}_h,\boldsymbol{\mu}_h\in[\mathbb{P}_d^k(\mathcal{F}_h)]^m$, the forms read

$$\begin{aligned} f_h(\boldsymbol{w}_h,\boldsymbol{\lambda}_h,\boldsymbol{z}_h)=&-\sum_{\kappa\in\mathcal{T}_h}\int_\kappa\sum_{i=1}^m\sum_{p=1}^d\widetilde{F}_{p,i}(\boldsymbol{w}_h)\,\frac{\partial z_i}{\partial x_p}+\\ &+\sum_{\kappa\in\mathcal{T}_h}\sum_{\sigma\in\mathcal{F}_{\partial\kappa}}\int_\sigma\sum_{i=1}^m\sum_{p=1}^d n_p^\sigma\,\widehat{F}_{p,i}(\boldsymbol{w}_h,\boldsymbol{\lambda}_h)\,[\![z_i]\!], \end{aligned} \tag{6.17}$$

$$l_h(\boldsymbol{w}_h,\boldsymbol{\lambda}_h,\boldsymbol{\mu}_h)=\sum_{\sigma\in\mathcal{F}_h^i}\int_\sigma\sum_{i=1}^m\sum_{p=1}^d n_p^\sigma\left\{\left\{\widehat{F}_{p,i}(\boldsymbol{w}_h,\boldsymbol{\lambda}_h)\right\}\right\}\mu_i. \tag{6.18}$$

Recalling the mass matrix bilinear form (2.22), the unsteady pHDG problem can be rewritten as follows: find $\boldsymbol{\tau}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{w}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\lambda}_h\in[\mathbb{P}_d^k(\mathcal{F}_h)]^m$ such that

$$\begin{aligned} &m_h(\boldsymbol{w}_h,\boldsymbol{z}_h)+f_h(\boldsymbol{w}_h,\boldsymbol{\lambda}_h,\boldsymbol{z}_h)=0\\ &l_h(\boldsymbol{w}_h,\boldsymbol{\lambda}_h,\boldsymbol{\mu}_h)=0 \end{aligned} \tag{6.19}$$

for all $\boldsymbol{\pi}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$, $\boldsymbol{z}_h\in[\mathbb{P}_d^k(\mathcal{T}_h)]^m$, $\boldsymbol{\mu}_h\in[\mathbb{P}_d^k(\mathcal{F}_h)]^m$.

6.2 Time integration

The ESDIRK3 scheme is here employed for the time discretization. As reported in Chapter 2, Algorithm 3, the scheme is third-order accurate and involves four Runge–Kutta stages. Being the first one explicit, it thus calls for the solution of three nonlinear systems. The nonlinear stage is solved using the Newton's method, which is an iterative method that calls for the solution of multiple linear systems. Since one of the objective of HDG is to reduce the globally coupled degrees of freedom arising from the implicit time discretizations, the block-structure of the iteration matrix can be conveniently exploited to obtain a smaller system of ODE. Details of this procedure are given in the following.

Residual's Jacobian – mixed form

The discretization produce the following residual's Jacobian forms:

$$a_h^{q\tau}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{\pi}_h) = \frac{\partial q_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\pi}_h)}{\partial \boldsymbol{\tau}_h}\delta\boldsymbol{\tau}_h, \tag{6.20}$$

$$a_h^{qw}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{\pi}_h) = \frac{\partial q_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\pi}_h)}{\partial \boldsymbol{w}_h}\delta\boldsymbol{w}_h, \tag{6.21}$$

$$b_h^{q\lambda}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\pi}_h) = \frac{\partial q_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\pi}_h)}{\partial \boldsymbol{\lambda}_h}\delta\boldsymbol{\lambda}_h, \tag{6.22}$$

$$a_h^{f\tau}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{z}_h) = \frac{\partial f_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h)}{\partial \boldsymbol{\tau}_h}\delta\boldsymbol{\tau}_h, \tag{6.23}$$

$$a_h^{fw}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h) = \frac{\partial f_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h)}{\partial \boldsymbol{w}_h}\delta\boldsymbol{w}_h, \tag{6.24}$$

$$b_h^{f\lambda}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{z}_h) = \frac{\partial f_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h)}{\partial \boldsymbol{\lambda}_h}\delta\boldsymbol{\lambda}_h, \tag{6.25}$$

$$c_h^{l\tau}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{\mu}_h) = \frac{\partial l_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)}{\partial \boldsymbol{\tau}_h}\delta\boldsymbol{\tau}_h, \tag{6.26}$$

$$c_h^{lw}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{\mu}_h) = \frac{\partial l_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)}{\partial \boldsymbol{w}_h}\delta\boldsymbol{w}_h, \tag{6.27}$$

$$d_h^{l\lambda}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h) = \frac{\partial l_h(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)}{\partial \boldsymbol{\lambda}_h}\delta\boldsymbol{\lambda}_h. \tag{6.28}$$

The derivatives appearing in Equations (6.31)-(6.34) can be computed in analogy to what has been done for the DG discretization, see Equations (2.20) and (2.21). The

block-matrix operators of the discretization are obtained as follows

$$\begin{aligned}
(\boldsymbol{A}_h^{q\tau}\ \delta\boldsymbol{\tau}_h, \boldsymbol{\pi}_h)_{L^2(\Omega)} &= a_h^{q\tau}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{\pi}_h),\\
(\boldsymbol{A}_h^{qw}\ \delta\boldsymbol{w}_h, \boldsymbol{\pi}_h)_{L^2(\Omega)} &= a_h^{qw}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{\pi}_h),\\
(\boldsymbol{B}_h^{q\lambda}\ \delta\boldsymbol{\lambda}_h, \boldsymbol{\pi}_h)_{L^2(\Omega)} &= b_h^{q\lambda}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\pi}_h),\\
(\boldsymbol{A}_h^{f\tau}\ \delta\boldsymbol{\tau}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= a_h^{f\tau}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{z}_h),\\
(\boldsymbol{A}_h^{f\lambda}\ \delta\boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= a_h^{f\lambda}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h),\\
(\boldsymbol{B}_h^{fw}\ \delta\boldsymbol{\lambda}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= b_h^{fw}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{z}_h),\\
(\boldsymbol{C}_h^{l\tau}\ \delta\boldsymbol{\tau}_h, \boldsymbol{\mu}_h)_{L^2(\Omega)} &= c_h^{l\tau}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{\mu}_h),\\
(\boldsymbol{C}_h^{lw}\ \delta\boldsymbol{w}_h, \boldsymbol{\mu}_h)_{L^2(\Omega)} &= c_h^{lw}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{\mu}_h),\\
(\boldsymbol{D}_h^{l\lambda}\ \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)_{L^2(\Omega)} &= d_h^{l\lambda}(\boldsymbol{\tau}_h, \boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h),
\end{aligned} \tag{6.29}$$

for all $\boldsymbol{\tau}_h, \delta\boldsymbol{\tau}_h, \boldsymbol{\pi}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^{m\times d}$; $\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$; $\boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$. Note that the operators $\boldsymbol{A}_h^{ij}$ involve contributions due to element-interior DoFs, $\boldsymbol{B}_h^{ij}$, $\boldsymbol{C}_h^{ij}$ involve mixed element-interior and face contributions, while $\boldsymbol{D}_h^{l\lambda}$ is obtained through face DoFs only. The Newton's method requires to solve multiple linear systems that can be complactly written as

$$\begin{bmatrix} \boldsymbol{A}_h^{q\tau} & \boldsymbol{A}_h^{qw} & \boldsymbol{B}_h^{q\lambda} \\ \boldsymbol{A}_h^{f\tau} & \frac{1}{\alpha\delta t}\boldsymbol{M}_h + \boldsymbol{A}_h^{fw} & \boldsymbol{B}_h^{fw} \\ \boldsymbol{C}_h^{l\tau} & \boldsymbol{C}_h^{lw} & \boldsymbol{D}_h^{l\lambda} \end{bmatrix} \begin{pmatrix} \delta\boldsymbol{\tau}_h \\ \delta\boldsymbol{w}_h \\ \delta\boldsymbol{\lambda}_h \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_h^q \\ \boldsymbol{f}_h^f \\ \boldsymbol{b}_h^l \end{pmatrix}, \tag{6.30}$$

where the vector $(\boldsymbol{f}_h^q,\ \boldsymbol{f}_h^f,\ \boldsymbol{b}_h^l)^T$ is obtained using the residuals forms in consistency with the time discretization.

Residual's Jacobian – primal form

In the primal formulation the contributions of the gradient variable disappear, and thus the only element-interior variable $\boldsymbol{w}_h$ is considered. The residual's Jacobian form become

$$a_h^{fw}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}, \boldsymbol{z}_h) = \frac{\partial f_h(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h)}{\partial \boldsymbol{w}_h}\delta\boldsymbol{w}_h, \tag{6.31}$$

$$b_h^{f\lambda}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{z}_h) = \frac{\partial f_h(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{z}_h)}{\partial \boldsymbol{\lambda}_h}\delta\boldsymbol{\lambda}_h, \tag{6.32}$$

$$c_h^{lw}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}, \boldsymbol{\mu}_h) = \frac{\partial l_h(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)}{\partial \boldsymbol{w}_h}\delta\boldsymbol{w}_h, \tag{6.33}$$

$$d_h^{l\lambda}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h) = \frac{\partial l_h(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)}{\partial \boldsymbol{\lambda}_h}\delta\boldsymbol{\lambda}_h. \tag{6.34}$$

and, in consistency with those equations, the block-matrix operators of the discretization are obtained as follows

$$\begin{aligned}
(\boldsymbol{A}_h^{fw}\ \delta\boldsymbol{w}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= a_h^{fw}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h),\\
(\boldsymbol{B}_h^{f\lambda}\ \delta\boldsymbol{\lambda}_h, \boldsymbol{z}_h)_{L^2(\Omega)} &= b_h^{f\lambda}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{z}_h),\\
(\boldsymbol{C}_h^{lw}\ \delta\boldsymbol{w}_h, \boldsymbol{\mu}_h)_{L^2(\Omega)} &= c_h^{lw}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{w}_h, \boldsymbol{\mu}_h),\\
(\boldsymbol{D}_h^{l\lambda}\ \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h)_{L^2(\Omega)} &= d_h^{l\lambda}(\boldsymbol{w}_h, \boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h),
\end{aligned} \tag{6.35}$$

for all $\boldsymbol{w}_h, \delta\boldsymbol{w}_h, \boldsymbol{z}_h \in [\mathbb{P}_d^k(\mathcal{T}_h)]^m$; $\boldsymbol{\lambda}_h, \delta\boldsymbol{\lambda}_h, \boldsymbol{\mu}_h \in [\mathbb{P}_d^k(\mathcal{F}_h)]^m$. In analogy with Eq. (6.30), one can write the pHDG linear system as

$$\begin{bmatrix} \frac{1}{\alpha\delta t}\boldsymbol{M}_h + \boldsymbol{A}_h^{fw} & \boldsymbol{B}_h^{f\lambda} \\ \boldsymbol{C}_h^{lw} & \boldsymbol{D}_h^{l\lambda} \end{bmatrix} \begin{pmatrix} \delta\boldsymbol{w}_h \\ \delta\boldsymbol{\lambda}_h \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_h^f \\ \boldsymbol{b}_h^l \end{pmatrix}, \tag{6.36}$$

where $(\boldsymbol{f}_h^f,\, \boldsymbol{b}_h^l)^T$ is again obtained using the residuals forms in consistency with the time discretization.

Static condensation and back-solve

Considering the block structure of the matrices appearing in (6.30) and (6.36), the solution of the system can be obtained for a smaller number of DoFs by statically condensing out the element-interior variables. Partitioning the matrix into element-interior and face components, $[\boldsymbol{A}_h,\, \boldsymbol{B}_h;\, \boldsymbol{C}_h,\, \boldsymbol{D}_h]$, and similarly for the right-hand side vector, $[\boldsymbol{f}_h; \boldsymbol{b}_h]$, the Schur-complement linear system reads

$$\underbrace{(\boldsymbol{D}_h - \boldsymbol{C}_h\boldsymbol{A}_h^{-1}\boldsymbol{B}_h)}_{\boldsymbol{G}_h}\,\delta\boldsymbol{\lambda}_h = \underbrace{(\boldsymbol{b}_h - \boldsymbol{C}_h\boldsymbol{A}_h^{-1}[\boldsymbol{f}_h])}_{\boldsymbol{g}_h}, \tag{6.37}$$

which assumes the form $\boldsymbol{G}_h\,\delta\boldsymbol{\lambda}_h = \boldsymbol{g}_h$, that can be solved using a GMRES algorithm. The definition of each block can be found by comparison with Equations (6.30) and (6.36). The static condensation is an operation that involves matrix-matrix products for the iteration matrix, as well as matrix-vector products for the right-hand side. Fortunately, the compact structure of the residual Jacobian prevents us from having to allocate global matrices for the computation of the condensed matrix, *i.e.* the operations described in Eq. (6.37) are local to each element. In addition, the computation of $\boldsymbol{A}_h^{-1}$ can be performed in place. By doing so, the memory footprint of the HDG implementation is not increased during the solve.

After the solution of (6.37), the interior states have to be recovered for the residual evaluation in the next time step. This operation is commonly referred to as the *back solve* and assumes the following form

$$\delta\boldsymbol{w}_h = -\boldsymbol{A}_h^{-1}\left(\boldsymbol{f}_h + \boldsymbol{B}_h\delta\boldsymbol{\lambda}_h\right). \tag{6.38}$$

The implementation choice of assembling the condensed matrix on-the-fly requires re-evaluation of the inverse of the matrix $\boldsymbol{A}_h$ in an element-wise fashion during the back-solve.

As a final remark for the two solvers, it is worth to point out that for both the

mixed and primal form of HDG, memory allocation and time spent on the global solve are lower than that of a DG solver due to the smaller number of globally-coupled degrees of freedom at high orders. On the other hand, the inversion of the $\boldsymbol{A}_h$ block-structured matrix of equation (6.37), although being local to each element, increases the amount of element-wise operations.

6.3 Preconditioning

To compare the performance of HDG discretizations, the use of single-grid as well as multigrid preconditioning approaches are here employed. As regards single-grid preconditioners, the EWBJ and EWBJ approaches are used, see Chapter 4, Section 4.2. For multigrid, the general framework is that of a p-multigrid of Chapter 5, Section 5.1.

In this chapter the additional integration of the residuals and Jacobians on the coarse level is avoided using subspace inheritance of the matrix operators assembled on the finest space. This choice involves projections of the matrix operators and right hand sides, which are computed only once on the finest level. Despite standard inheritance has been demonstrated to be less efficient than a rescaled-inherited approach in incompressible flow problems, only the implementation of the first method is employed hereby for DG, as these operators seems to be sufficiently efficient for target problems involving the compressible NS equations. In fact, the compressible nature of the equations produce a better conditioned system as demonstrated in [16].

In the context of standard discontinuous Galerkin discretizations, see for example [88, 93, 92, 94], the multigrid idea has been thoroughly investigated and exploited in several ways, *e.g.* h-, p-, and hp-strategies. On the other hand, the use of p-multigrid for HDG has not been as widely studied. In this case, the definition of the restriction and prolongation operators, as well as the coarse grid operators and right hand sides, is not straightforward when considering the statically condensed system. Having introduced the concept of subspace inheritance for a standard DG solver in Chapter 4, it is here extended to HDG. Also in this case the full multigrid (FMG) $\mathcal{V}$-cycle solver, outlined in Chapter 5, Section 5.1.4, Algorithm 13, is still employed. The smoothers employed here consist of preconditioned GMRES solvers. Regarding the choice of the multigrid levels and preconditioners, the results reported for the incompressible Navier–Stokes equations have proved to be still valid: an optimal and scalable solver can be obtained using an aggressive preconditioner on the coarsest space discretization, where the factorization of the matrix can be performed at a low computational cost, and the system has to be solved with a higher accuracy. On the other hand, cheaper operators can be used on the finest levels of the discretization, where the systems need not be solved to a high degree of accuracy. Therefore, similar settings to those of Chapter 5 are employed. The following subsections provide the details on the smoothers, as well as how the matrices and vectors are restricted and prolongated between levels.

Single-grid preconditioners

This chapter exploits single-grid preconditioners for the purpose of benchmarking, as well as to precondition the smoothers of the multigrid solver. To simplify the

analysis, only two operators are considered. The first one is the element-wise block-Jacobi (EWBJ). Both in DG and HDG, this approach works with the block-diagonal portion of the iteration matrix and factorize it, in a local-to-each element fashion, using the PLU factorization, and it is applied in the same way for serial and parallel computations.

The second preconditioner is the incomplete lower-upper factorization with zero-fill, ILU(0), labelled as block-Jacobi (BJ) when applied in parallel computations. As observed in Chapter 4, the factorization works with the partition-wise block of the iteration matrix and thus looses its effectiveness in parallel runs. The Additive Schwarz method, which compensates the performance degradation, is not considered in this chapter dealing with compressible flow problems. In fact, as reported in [16], such problems are less stiff than those arising from the discretization of the incompressible Navier–Stokes equations.

It is worth pointing out that, when employed in a matrix-free context, the use of EWBJ allows to reduce the matrix-assembly costs, as only the on-diagonal blocks of the iteration matrix have to be computed and stored.

A multigrid approach for HDG

In HDG, the globally-coupled unknowns are those related to the face DoFs, and the iteration matrix is obtained through static condensation, see Equation (6.37), which allows us to solve the system for the face unknowns only. Theoretically speaking, for element-interior degrees of freedom, the same operators of Section 5.1.1, Chapter 5 can be employed. On the other hand, the operation for faces degrees of freedom can be obtained through similar considerations. In this case, the sequence of approximation spaces on the interior mesh element interfaces, built using lower-order polynomials k_ℓ, with $\ell = 1, ...L$ and $k_0 = k$, are named $\mathbb{P}_d^{k_\ell}(\mathcal{F}_h^i)$. The prolongation operator is now defined as $\mathcal{J}_{\ell+1}^{\ell} : \mathbb{P}_d^{k_{\ell+1}}(\mathcal{F}_h^i) \to \mathbb{P}_d^{k_\ell}(\mathcal{F}_h^i)$ such that

$$\sum_{\sigma \in \mathcal{F}_h^i} \int_\sigma \left(\mathcal{J}_{\ell+1}^{\ell} \lambda_{\ell+1} - \lambda_{\ell+1} \right) = 0, \qquad \forall \lambda_{\ell+1} \in \mathbb{P}_d^{k_{\ell+1}}(\mathcal{F}_h^i). \tag{6.39}$$

On the other hand, the restriction is defined as $\mathcal{J}_{\ell}^{\ell+1} : \mathbb{P}_d^{k_\ell}(\mathcal{F}_h^i) \to \mathbb{P}_d^{k_{\ell+1}}(\mathcal{F}_h^i)$ such that

$$\sum_{\sigma \in \mathcal{F}_h^i} \int_\sigma \left(\mathcal{J}_{\ell}^{\ell+1} \lambda_{\ell} - \lambda_{\ell} \right) \mu_{\ell+1} = 0, \qquad \forall (\lambda_\ell, \mu_{\ell+1}) \in \mathbb{P}_d^{k_\ell}(\mathcal{F}_h^i) \times \mathbb{P}_d^{k_{\ell+1}}(\mathcal{F}_h^i). \tag{6.40}$$

These definitions can also be extended to operate on vector functions $\boldsymbol{\lambda}_h \in [\mathbb{P}_d^{k_\ell}(\mathcal{F}_h^i)]^m$ and are assumed to act component-wise, *i.e.* $\boldsymbol{\mathcal{J}}_{\ell}^{\ell+1} \boldsymbol{\lambda}_h = \sum_i^m \mathcal{J}_{\ell}^{\ell+1} \lambda_i$.

Applying same subspace-inheritance idea reported for DG, one can obtain the coarse space condensed HDG matrix and right hand side through the application of element-interior and face DoFs projections. This involve the definition of matrix operators obtained through the integration on a coarser order space of the forms (6.29) and (6.35) computed using prolongated quantities, in analogy to Eq. (5.3). In this way, a standard-inherited coarse space would require to statically condense out the

system and to obtain the matrix as

$$\boldsymbol{G}_\ell = \boldsymbol{D}_\ell - \boldsymbol{C}_\ell \boldsymbol{A}_\ell^{-1} \boldsymbol{B}_\ell \tag{6.41}$$

with $\boldsymbol{A}_\ell$, $\boldsymbol{B}_\ell$, $\boldsymbol{C}_\ell$ and $\boldsymbol{D}_\ell$ obtained using block-by-block projection of $\boldsymbol{A}_h$, $\boldsymbol{B}_h$, $\boldsymbol{C}_h$ and $\boldsymbol{D}_h$, respectively. This operation would involve the application of mixed element-interior and face degrees of freedom Galerkin projections prior the static condensation of the system. Thus, it comes with an increased operation count compared to DG subspace inheritance.

With the idea of minimizing the number of operations required to assemble the coarse spaces, a further approximation is here proposed. In fact, it is observed that a coarse space matrix can be obtained by applying the block-by-block projection to the statically condensed matrix $\boldsymbol{G}_h$ directly, where the size of each block is $(mn_v^\ell)^2$, with $n_v^\ell = \dim(\mathbb{P}_d^{k_\ell}(\sigma))$. Therefore, each block of the recursively projected matrix assumes the form

$$\boldsymbol{G}_{\sigma,\sigma}^{\ell+1,\mathcal{A}} = \boldsymbol{N}_{\ell+1,\ell}^{\sigma} \boldsymbol{G}_{\sigma,\sigma}^{\ell,\mathcal{A}} \left(\boldsymbol{N}_{\ell+1,\ell}^{\sigma}\right)^T, \tag{6.42}$$

where

$$\left(\boldsymbol{N}_{\ell+1,\ell}^{\sigma}\right) = \left(\boldsymbol{N}_{\ell+1}^{\sigma}\right)^{-1} \int_\sigma \mu^{\ell+1} \otimes \mu^{\ell}, \qquad \boldsymbol{N}_{\ell+1}^{\sigma} = \int_\sigma \mu^{\ell+1} \otimes \mu^{\ell+1}, \tag{6.43}$$

and $\mu^{\ell} \in \mathbb{P}_d^{k_\ell}(\sigma)$. This approach will be referred to as the *approximate*-inherited method.

Similar considerations are valid for the right hand side, which have to be obtained from the residuals of the equations in each level of the discretization. A formally exact definition of the residual would require to

1. project in every level of the multigrid cycle the element-interior and face components of the residuals $\boldsymbol{f}_h, \boldsymbol{b}_h$;

2. *back-solve* for interior degrees of freedom in each level;

3. compute the residuals of the equations using the matrix operators $\boldsymbol{A}_\ell$, $\boldsymbol{B}_\ell$, $\boldsymbol{C}_\ell$ and $\boldsymbol{D}_\ell$;

4. statically condense out the element-interior DoFs to obtain the residuals of the equations.

This operation, which is formally equivalent to what proposed in a DG context, involves a larger amount of operations that would make the multigrid cycle inconvenient in terms of performance. Within the approximate-inherited idea, the right hand sides can be conveniently obtained recursively by face-projection after static condensation of the right hand side on the fine space, see Eq. (6.37), and thus $\boldsymbol{g}_{\ell+1} = \boldsymbol{\mathcal{J}}_\ell^{\ell+1} \boldsymbol{g}_\ell$. The residual vector for the face degrees of freedom is then simply obtained as $\boldsymbol{d}_\ell = \boldsymbol{g}_\ell - \boldsymbol{G}_\ell \, \delta\boldsymbol{\lambda}_\ell$, with $\delta\boldsymbol{\lambda}_\ell$ the solution vector at level ℓ. Despite these additional approximations compared to DG subspace inheritance, such a strategy exhibits very good performance in HDG solutions of the compressible Navier–Stokes equations, as will be demonstrated in the results section.

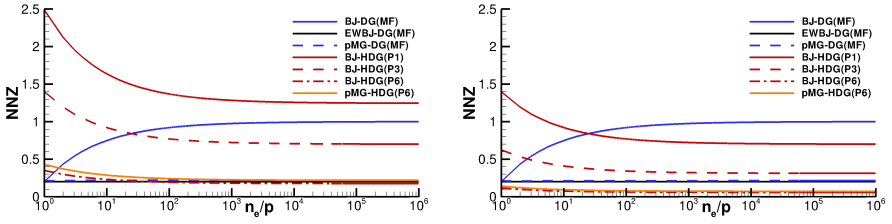
6.3.1 Preconditioning options and memory footprint

The same single-grid preconditioners of Chapter 5 are here employed both for benchmarking as well as to precondition the smoothers of the multigrid strategy. In particular, BJ and EWBJ are considered, while the use of the Additive Schwarz method is neglected being the study related to the compressible case, which generally provides better conditioned problems. Bearing that in mind, it is useful to provide estimates of the memory footprint of the solver as well as the computational time spent on evaluating the matrix operators. It is trivial to observe that the matrix assembly time, as well as its operation count, is a function of the number of non-zeros of the matrix itself. This number depends on the iteration matrix and preconditioner, and it is function of the type of discretization (DG or HDG) as well as the type of the solver (MF or MB) in the DG case. Considering a squared, two-dimensional and bi-periodic domain made by quadrangular elements, it is possible to compute the curves in Figure 6.2, where the number of non-zeros (NNZ), non-dimensionalised by the NNZ of the Jacobian arising from the DG discretization, is reported as a function of the number of elements per partition, n_e/p. It can be observed that

1. the memory footprint of DG, matrix-based as well as HDG solvers is always equal to that of a Jacobian matrix, and this value is function of the polynomial order for HDG. This is motivated by the fact that the preconditioner is always and evaluated using the same memory of the Jacobian matrix;

2. for a matrix-free, DG discretization, the allocation involves only the preconditioner operator. When EWBJ is considered, NNZ reduces by the 80% with respect to the allocation of a full DG Jacobian. As $n_e/p \to 1$, the NNZ of BJ solver approaches that of the element-wise block Jacobi, while for $n_e/p >> 1$ it tends to be that of a Jacobian matrix. This is due to the fact that as the domain is partitioned, the ILU(0) factorization is performed in the squared, partition-wise block of the iteration matrix and therefore, in a matrix-free fashion, the off-partition blocks can be neglected during the assembly phase;

3. the p-multigrid (pMG) matrix-free preconditioning approach applied to a DG



(a) TriLagrange basis functions

(b) QuadLagrange basis functions

Figure 6.2: Allocated number of non-zeros (NNZ) non-dimensionalised by the memory allocation of the DG Jacobian matrix. DG matrix-free solvers compared to HDG for different values of polynomial orders and preconditioning type. p-multigrid preconditioning (pMG) assumed to be that of Table 6.4.

discretization, here assumed to be that of Table 6.4, requires a memory footprint in line with that of an element-wise block Jacobi method, as already observed in [95]. In fact, when using lower-order polynomial spaces with $k_\ell << k$, the size of those matrices is considerably smaller than that of the finest space since they scale with k^6;

4. for HDG, only for high order polynomials NNZ is reduced with respect to the iteration matrix of a DG method. For $k = 6$, a memory footprint in line with that of a EWBJ, matrix-free approach is observed, while for $k = 1, 3$ the memory is considerably larger. It is worth pointing out that the use of TriLagrange basis functions (Figure 6.2(a)) and QuadLagrange basis functions (Figure 6.2(b)) only affects NNZ ratio in the HDG case: in particular, the memory footprint reduction when using QuadLagrange basis is larger due to the higher amount of element-wise unknowns if compared to that of the internal faces;

5. interestingly, the NNZ ratio for HDG when $n_e/p \to 1$. The reason for this is that the face-to-elements ratio within the computational mesh of the domain partition changes.

Finally, it is important to remark that for a matrix-free iterative solver employed in DG contexts, it is possible to optimize the matrix assembly evaluation to compute only the blocks required by the preconditioner. For example, for EWBJ, the evaluation of the off-diagonal blocks of the Jacobian can be neglected, while for pMG matrix-free with EWBJ on the finest space, the off-diagonal blocks could be computed at a reduced polynomial order consistent with that of the coarser spaces.

6.4 Numerical results on a model test case

Numerical experiments to assess the performance of the HDG discretizations in comparison to DG are here presented. First, NS solutions of a vortex transported by uniform flow at $M = 0.05$ and $Re = 100$ are reported. The objective is to i) to show the convergence rates of the solver both in space and time; ii) to investigate the effects of grid refinement for the approximate-inherited approach proposed for HDG, providing mesh-independent convergence rates; and iii) compare the effects of polynomial order, time step size and space discretization on the parallel performance of the solution strategy.

6.4.1 Test case description

The test case is a modified version of the VI1 case studied in the 5^{th} International Workshop on High Order CFD Methods [54], and consists of a two-dimensional mesh on the domain $(x, y) \in [0, 0.1]\times[0, 0.1]$ with periodic boundary conditions on each

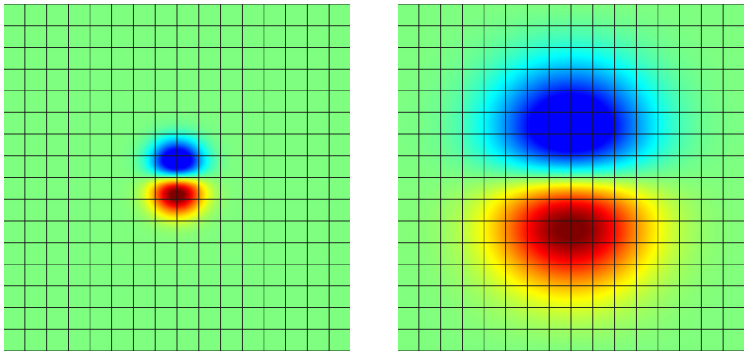


Figure 6.3: Convected vortex at $Re = 100$, $M = 0.05$. Mach number contours. Solution at $t = 0$ (left) and $t = T$ (right).

side. The flow initialisation involves the definition of the following state

$$\begin{aligned} u &= U_\infty \left(1 - \beta \left(\frac{y - Y_c}{R}\right) e^{-r^2/2}\right) \\ v &= U_\infty \beta \left(\frac{x - X_c}{R}\right) e^{-r^2/2} \\ T &= T_\infty - \left(\frac{U_\infty^2 \beta^2}{2C_p}\right) e^{-r^2} \end{aligned} \tag{6.44}$$

with the heat capacity at constant pressure being $C_p = R_{gas}\gamma/(\gamma - 1)$, the non dimensional distance to the initial vortex core position $r = \sqrt{(x - X_c)^2 + (y - Y_c)^2}/R$, and the free stream velocity being $U_\infty = M_\infty \sqrt{\gamma R_{gas} T_\infty}$. The fluid pressure p, the temperature T and density ρ are prescribed to ensure a steady solution of the the problem without uniform flow transport, *i.e.*$\rho_\infty = p_\infty / R_{gas} T_\infty$, $\rho = \rho_\infty (T/T_\infty)^{1/(\gamma-1)}$, $p = \rho R_{gas} T$. The parameters where chosen such that $M_\infty = 0.05$, $\beta = 1/50$ and $R = 0.005$. Differently from VI1, here the set of NS equations are solved instead of the Euler equations, and the Reynolds number, based on the domain extension, was $Re = 100$.

6.4.2 Assessment of the solution accuracy

Numerical experiments have been performed to assess the output error, both in space and time. The meshes here employed were obtained using regular quadrilaterals. The mesh density ranges from 2×2 to 64×64, while the polynomial order range $k \in \{1, 2, 3, 4, 5, 6\}$. The L_2 state error was computed relative to the solution on a 128×128, $\mathbb{P}_6$ space discretization, after one convective period T. The contour plot of the solutions at the initial and final states are shown in Figure 6.3. Table 6.1 reports space discretization errors. The tests were performed using a very small time step size, $T/\Delta t = 4000$, and the ESDIRK3 scheme to ensure a negligible time discretization error, with an absolute tolerance on the non-linear system of 10^{-10}, and a relative tolerance of 10^{-5} on the GMRES. Even though DG suffers less than HDG of

		DG		mHDG		pHDG	
order	grid	$\|err\|_{L_2}$	k	$\|err\|_{L_2}$	k	$\|err\|_{L_2}$	k
$\mathbb{P}_1$	8	4.0938E-07		4.20E-07		4.1798E-07	
	16	1.4796E-07	1.468	1.4514E-07	1.533	1.4406E-07	1.537
	32	3.0795E-08	2.264	2.8355E-08	2.356	2.8020E-08	2.362
	64	5.5247E-09	2.479	4.7822E-09	2.568	4.6898E-09	2.579
$\mathbb{P}_2$	2	5.9266E-07		6.22E-07		6.2124E-07	
	4	2.7443E-07	1.111	2.7344E-07	1.186	2.7306E-07	1.186
	8	3.2505E-08	3.078	3.2777E-08	3.060	3.2344E-08	3.078
	16	2.2983E-09	3.822	2.7457E-09	3.577	2.6311E-09	3.620
	32	2.3510E-10	3.289	3.4213E-10	3.005	3.1517E-10	3.061
	64	3.4853E-11	2.754	6.3410E-11	2.432	5.5951E-11	2.494
$\mathbb{P}_3$	2	3.3987E-07		3.31E-07		3.3149E-07	
	4	3.2850E-08	3.371	3.4696E-08	3.255	3.4275E-08	3.274
	8	1.5714E-09	4.386	1.7405E-09	4.317	1.7008E-09	4.333
	16	8.1804E-11	4.264	7.9918E-11	4.445	7.8935E-11	4.429
	32	7.2397E-12	3.498	7.2689E-12	3.459	7.2173E-12	3.451
$\mathbb{P}_4$	2	1.1210E-07		1.18E-07		1.1741E-07	
	4	5.2542E-09	4.415	5.4176E-09	4.446	5.3668E-09	4.451
	8	1.1772E-10	5.480	1.9915E-10	4.766	1.9838E-10	4.758
	16	6.5563E-12	4.166	7.3117E-12	4.768	7.1101E-12	4.802
$\mathbb{P}_5$	2	4.2677E-08		4.50E-08		4.4730E-08	
	4	6.6523E-10	6.003	7.8204E-10	5.848	7.7398E-10	5.853
	8	1.1519E-11	5.852	9.6028E-11	3.026	9.6035E-11	3.011
	16	5.0750E-12	1.183	4.8543E-12	4.306	4.8326E-12	4.313
$\mathbb{P}_6$	2	1.4170E-08		1.35E-08		1.3510E-08	
	4	9.7090E-11	7.189	3.4373E-10	5.298	3.4301E-10	5.300
	8	5.1249E-12	4.244	2.4622E-11	3.803	2.4821E-11	3.789
	16	5.0800E-12	0.013	5.1008E-12	2.271	5.0943E-12	2.285

Table 6.1: L_2 solution error. Laminar vortex test case at $Re = 100$, $M = 0.05$. Convergence rates for the DG and HDG discretizations.

pre-asympthotic behaviour on such a smooth solution, all the three implementations show comparable error levels and converge with the theoretical convergence rates for every polynomial approximation shown. As a consequence of such analysis, and considering that both the DG and HDG implementations share the same code base, only the CPU time will be considered as a measure of the time-to-solution efficiency.

As regards the time integration scheme, the convergence rates of the ESDIRK3 time integration scheme are also reported in Table 6.2, and have been obtained using the 16×16 grid, $\mathbb{P}_6$ polynomials for the three space discretization strategies. The

	DG		mHDG		pHDG	
$T/\Delta t$	$\|err\|_{L_2}$	k	$\|err\|_{L_2}$	k	$\|err\|_{L_2}$	k
4	5.3895E-07		5.3898E-07		5.3898E-07	
10	1.3656E-07	1.498	1.3657E-07	1.498	1.3657E-07	1.498
20	2.5830E-08	2.402	2.5788E-08	2.405	2.5788E-08	2.405
40	3.7729E-09	2.775	3.7684E-09	2.775	3.7684E-09	2.775
100	2.6181E-10	2.912	2.6049E-10	2.916	2.6050E-10	2.916

Table 6.2: Laminar vortex test case at $Re = 100$, $M = 0.05$. Time convergence rates for the DG, mHDG and pHDG discretizations, using the ESDIRK3 scheme.

theoretical third order convergence rate of the ESDIRK3 time integration scheme can be observed for all the three space discretizations with comparable error levels.

6.4.3 Assessment of the approximate-inherited multigrid approach for HDG

To prove the efficicacy of the multigrid approach proposed herein for HDG, numerical experiments obtained by reducing the mesh element size for different element types are here reported. A total of four mesh sequences are considered in the study obtained as i) regular elements; ii) randomly distorted elements; iii) regular and clustered elements; and iv) clustered-distorted elements. The study was performed on both triangular and quadrangular elements, see Figure 6.4. Note that the random pertubation was limited to the 10% of the minimum dimension of the element, and that the clustering has been obtained by placing the mesh element nodes using gauss-lobatto rules. A full multigrid strategy has been employed for the study. The strategy combines three multigrid levels, and for each of them a BJ-preconditioned GMRES smoother was considered. To avoid the strategy being influenced by the domain decomposition, all the computations are here performed in serial. Three smoothing iterations have been performed in each levels, while 400 iterations were employed on the coarsest level to ensure the results not being polluted by a lack of coarse level resolution. This configuration was found to optimize the solver to deal with serial computations and it is coherent to what has been previously reported in the literature, see [95].

Table 6.3 report the results on triangular and quadrangular mesh elements. The numerical experiment consisted on one time step such that $T/\Delta t = 10$ using the ESDIRK3 scheme, being T the convective period of the problem. The average number

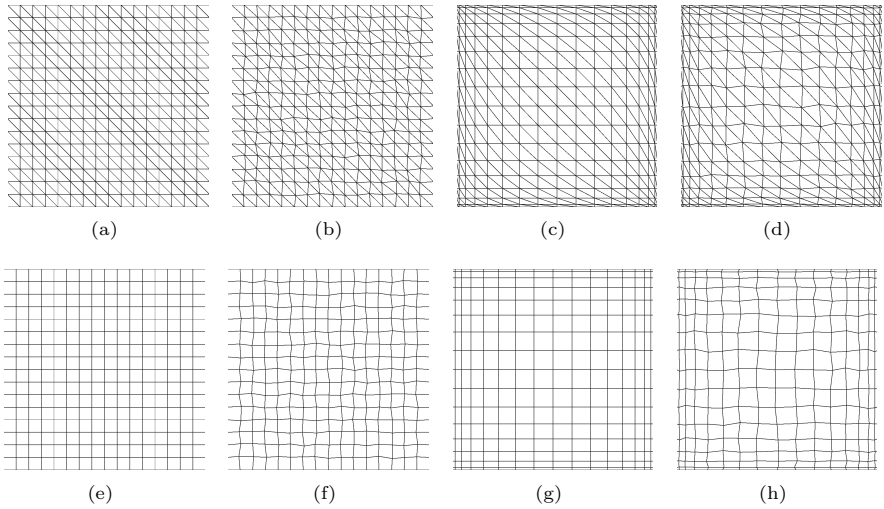


Figure 6.4: Convected vortex at $Re = 100$, $M = 0.05$. Example of computatinal meshes involving, for the tri- and quad-, i) regular elements; ii) randomly distorted elements; iii) regular and clustered elements; and iv) clustered-distorted elements.

	Reg Tri		Dis Tri		Reg Grad Tri		Dis Grad Tri	
n_e	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a
$16^2 \cdot 2$	3.000	0.0060	2.833	0.0056	3.000	0.0120	3.000	0.0131
$32^2 \cdot 2$	3.000	0.0123	3.000	0.0119	3.500	0.0245	3.333	0.0195
$64^2 \cdot 2$	2.500	0.0058	3.000	0.0132	3.667	0.0315	4.000	0.0422
$128^2 \cdot 2$	2.333	0.0047	2.667	0.0086	3.500	0.0290	4.167	0.0560
	Reg Quad		Dis Quad		Reg Grad Quad		Dis Grad Quad	
n_e	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a
16^2	2.833	0.0053	2.333	0.0033	3.000	0.0069	3.000	0.0057
32^2	2.833	0.0101	2.833	0.0081	3.000	0.0087	3.000	0.0096
64^2	3.167	0.0183	3.000	0.0153	3.833	0.0386	3.667	0.0336
128^2	3.333	0.0246	3.500	0.0273	4.333	0.0637	4.500	0.0659

Table 6.3: Laminar vortex test case at $Re = 100$, $M = 0.05$. h-independence test on tri-element (top) and quad-element (bottom) meshes. of a FGMRES(MG_{full}) solver built on three levels. GMRES(BJ) smoothers with 400 iterations on the coarsest level $\ell = 2$. 3 smoothing iterations were employed for $\ell = \{0, 1\}$.

of iterations as well as the average convergence rate (CR) ρ are reported. The CR is defined as $\rho = (r_{IT}/r_0)^{1/IT}$ with IT the number of iteratons, r_0 and r_{IT} the residuals at the first and IT^{th} iteration respectively. In all the numerical experiments the p-multigrid strategy shows to work optimally as the number of iterations slightly grow by increasing the number of mesh elements, even for distorted and graded mesh sequences.

Level	Order	Solver	Preconditioner	Iterations
1	6	GMRES	EWBJ	10
2	2	GMRES	EWBJ	10
3	1	GMRES	BJ	30

Table 6.4: Computational settings for the p-multigrid preconditioning strategy employed within the paper.

6.4.4 Evaluation of the solver efficiency

Numerical experiments in this Section are devoted to assess the performance of p-multigrid preconditioning strategy for HDG in comparison to other operators, as well as with state-of-the-art preconditioned DG discretizations. In particular, the reference here is the matrix-based, BJ preconditioned DG discretization, while the aim is to report insights on the computational time of the solution also, in order to provide an overall idea of the computational efficiency of the solver. The space discretization relies on the 16×16 mesh made by regular quads and two polynomial orders, *i.e.* $k = \{3; 6\}$. As for the time discretization, a non dimensional time step size of $\Delta t = \{1; 0.1\}$ is employed. In both the cases, a single time step is computed, which corresponds to three non-linear problems. It has been observed that the Newton solver converges in two iterations, which means that the CPU time and the average number of iterations is evaluated considering a total of six linear system solutions.

Special attention is hereby given to the parallel efficiency of the computations, both in terms of CPU time and average number of GMRES iterations. To this extent, the ideas reported in Chapter 5 regarding the choice of the number of levels and the smoothing type have been proposed. In fact, despite being that work focused on incompressible flow problems, a very similar behaviour of the solver has been hereby observed: a scalable multigrid strategy to precondition systems arising from the Navier–Stokes equations can be obtained by the use of simple EWBJ-preconditioned GMRES smoothers for all the levels except the coarse one, where more powerful preconditioners can be cheaply introduced on the smoothers due to the low computational cost of the coarse matrix factorization. If compared to Chapter 5, the use of Additive Schwarz preconditioning on the coarsest level is avoided since the compressible NS equations generally provide better conditioned problems. In some cases, where the stiffness is increased by the use of very large time step sizes, an optimal efficiency is achieved just by a slight increase in the number of smoothing iterations even for large parallel runs. Despite this choice proved to be less efficient for incompressible Navier–Stokes equations, its performance were found to be acceptable in the current study.

In the numerical experiments a three-level multigrid strategy is reported. In both the cases, $k_{1,2} = \{2, 1\}$ have been used on the coarser spaces smoothed with EWBJ- and BJ-preconditioned GMRES solvers, respectively. $\{10, 30\}$ smoothing iterations have been employed, which were found to be sufficient to provide optimal results both in serial and in parallel computations for HDG and DG as well. On the finest space, 10 smoothing iterations of GMRES(EWBJ) were used. The computational settings are summarized in Table 6.4. Note that those settings have been found to be enough computationally efficient for all the numerical experiments reported throughout the

paper.

Table 6.5 compares the performance of the multigrid preconditioner to those of BJ for $k = 3$ using $\Delta t = 1$ (left) and $\Delta t = 0.1$ (right), for three space discretizations, *i.e.* DG, mHDG and pHDG. For all the three space discretization, the performance degradation of the BJ preconditioner shows up clearly in view of the considerable increase in number of GMRES iterations moving from serial to parallel runs. On the other hand, the multigrid preconditioning strategy shows to provide higher parallel efficiencies in all the cases, being the increase in number of iterations eiter very low (for the largest time step) or null (for the smallest time step). This fact is ascribed to both the use of local-to-each element smoothers and projections.

As regards the CPU time, the speed-up values non-dimensionalized by the CPU time of the DG, matrix-based and BJ-preconditioned computation is reported. For the largest time step size, switching from single-grid BJ to multigrid provides consistent benefits on all the three space discretizations, being the speed-up values above one. In particular, due to the higher parallel efficiency of the approach, SU_{MB} reaches its peak for 32 cores. The strategy provides a speedup of 2.09 for DG, 2.21 and 3.05 for mHDG and pHDG, respectively.

For the smaller time step the situation is slightly different. In fact, having those kind of problems a lower conditioning of the iteration matrix, the speed-up value obtained from the use of multigrid preconditioning is reduced and approaches 1. As regards HDG, the reduced conditioning of the matrix makes the solution time higher than the reference for the *mixed* formulation, being the speed-up value below one. Conversely, the *primal* formulation still outperforms the reference providing a speed-up factor in the range $[1.29, 1.75]$ due to the smaller amount of operations and Jacobian evaluations with respect to the *mixed* form. However, as opposed to what happens for DG, where an improvement in computational efficiency is still observed, the use of a p-multigrid does not benefit the efficiency of the solver.

Table 6.6 shows the same results for a $k = 6$ space discretization. Similar considerations to those reported for Table 6.5 can be done on the overall parallel behaviour of the preconditioning strategies here considered, *i.e.* the increase in the number of iteration of the preconditioner reflect the performance degradation of the solution strategy when applied in parallel. However, in this case, the advantages arising from the use of a multigrid preconditioning strategy appear more evident. In fact, for the largest time step size the speed-up values are higher than those reported in Table 6.5 involving 3^{rd} order polynomials. In particular, when using 32 cores, it reaches 2.65, 1.87 and 3.84 for DG, mHDG and pHDG respectively. Differently to what observed previously, when reducing the time step size, the advantages of using a multigrid strategy still appear evident for DG, which provide speed-ups in the range $[1.86, 2.29]$. While for higher order polynomials and smaller time steps a *mixed* HDG implementation provides performance in line to that of a matrix-based, BJ-DG solver, the *primal* implementation shows to be better performing overall, even if shows an overall lower parallel efficiency. In particular, BJ-pHDG is the best performing solution strategy providing speed-up values in the range $[1.73, 2.36]$. In this case, p-multigrid performs similarly to BJ.

Table 6.7 reports for comparison the speed-up values obtained using a matrix-free

Discr.	$k=3$, $\Delta t=1$					
Solver	BJ-DG			MG_{full}-DG		
n_p	Time	E	IT_a	SU_{MB}	E	IT_a
1	38.52		81.00	1.03		3.50
2	27.53	0.70	137.33	1.43	0.97	3.50
4	15.10	0.64	154.83	1.51	0.94	3.50
8	8.72	0.55	181.17	1.58	0.85	3.67
16	6.03	0.40	229.50	1.78	0.69	4.00
32	5.82	0.21	284.50	2.09	0.42	4.67
Solver	BJ-mHDG			MG_{full}-mHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	1.49		45.17	1.23		2.50
2	1.75	0.82	77.17	1.54	0.88	2.50
4	1.63	0.70	95.50	1.50	0.78	2.50
8	1.67	0.62	113.17	1.56	0.70	2.50
16	1.76	0.47	138.67	1.75	0.57	2.83
32	1.93	0.27	183.50	2.21	0.37	3.50
Solver	BJ-pHDG			MG_{full}-pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	2.45		44.33	1.95		2.00
2	2.83	0.81	75.17	2.48	0.89	2.00
4	2.60	0.68	94.17	2.26	0.74	2.50
8	2.57	0.58	112.33	2.34	0.67	2.50
16	2.67	0.44	136.83	2.60	0.53	2.67
32	2.59	0.22	182.83	3.05	0.32	3.50
Discr.	$k=3$, $\Delta t=0.1$					
Solver	BJ-DG			MG_{full}-DG		
n_p	Time	E	IT_a	SU_{MB}	E	IT_a
1	23.51		27.83	0.88		2.17
2	15.12	0.78	53.17	1.08	0.96	2.17
4	7.68	0.77	55.33	1.06	0.92	2.17
8	4.15	0.71	65.50	1.04	0.84	2.17
16	2.23	0.66	72.33	0.99	0.74	2.17
32	1.57	0.47	81.83	0.96	0.51	2.17
Solver	BJ-mHDG			MG_{full}-mHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	0.96		20.33	0.79		2.00
2	1.03	0.83	36.00	0.88	0.87	2.00
4	0.91	0.72	41.00	0.80	0.78	2.00
8	0.90	0.66	46.83	0.78	0.70	2.00
16	0.79	0.54	52.67	0.68	0.56	2.00
32	0.81	0.39	64.50	0.70	0.42	2.00
Solver	BJ-pHDG			MG_{full}-pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	1.62		44.33	1.19		2.00
2	1.75	0.84	75.17	1.36	0.89	2.00
4	1.55	0.73	94.17	1.22	0.79	2.00
8	1.51	0.66	112.33	1.19	0.71	2.00
16	1.29	0.52	136.83	1.05	0.58	2.00
32	1.38	0.40	182.83	1.05	0.41	2.00

Table 6.5: Computational efficiency comparison using DG and HDG discretizations. Laminar vortex test case at $Re = 100$, $M = 0.05$, discretized using 16×16 mesh with $\mathbb{P}_3$ polynomials. Two time steps, $\Delta t = \{1; 0.1\}$ using ESDIRK3 scheme are reported. SU_{MB} stands for the speed-up factor referred to the DG, matrix-based, BJ-preconditioned computation, E is the parallel efficiency and IT_a the average number of GMRES iterations.

Discr.	$k=6$, $\Delta t=1$					
Solver	BJ-DG			MG_{full}-DG		
n_p	Time	E	IT_a	SU_{MB}	E	IT_a
1	533.69		87.33	1.73		5.67
2	381.18	0.70	190.83	2.37	0.96	5.83
4	198.53	0.67	213.50	2.42	0.94	5.83
8	109.00	0.61	257.50	2.69	0.95	5.33
16	63.14	0.53	313.17	2.55	0.78	6.33
32	47.91	0.35	392.33	2.65	0.53	7.17
Solver	BJ-mHDG			MG_{full}-mHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	1.46		46.50	1.45		2.67
2	1.75	0.84	112.50	1.76	0.85	2.67
4	1.58	0.73	139.50	1.61	0.75	2.67
8	1.58	0.66	155.83	1.61	0.68	2.83
16	1.52	0.55	203.83	1.59	0.58	3.67
32	1.74	0.42	268.50	1.87	0.45	5.17
Solver	BJ-pHDG			MG_{full}-pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	3.15		45.83	3.11		2.50
2	3.71	0.83	106.50	3.81	0.86	2.50
4	3.35	0.71	133.00	3.46	0.75	2.67
8	3.35	0.65	149.33	3.46	0.68	2.67
16	3.19	0.54	193.33	3.39	0.58	3.50
32	3.38	0.37	257.00	3.84	0.43	5.17
Discr.	$k=6$, $\Delta t=0.1$					
Solver	BJ-DG			MG_{full}-DG		
n_p	Time	E	IT_a	SU_{MB}	E	IT_a
1	390.37		32.00	1.90		3.00
2	247.56	0.79	80.83	2.29	0.95	3.00
4	123.33	0.79	85.33	2.25	0.94	3.00
8	62.84	0.78	99.17	2.24	0.92	3.00
16	32.33	0.75	105.83	2.04	0.81	3.00
32	19.61	0.62	110.67	1.86	0.61	3.00
Solver	BJ-mHDG			MG_{full}-mHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	1.08		23.17	1.07		2.17
2	1.16	0.85	50.67	1.16	0.85	2.17
4	1.01	0.74	54.50	1.01	0.74	2.17
8	0.94	0.68	62.17	0.93	0.68	2.17
16	0.83	0.58	71.83	0.83	0.58	2.17
32	0.80	0.46	82.50	0.80	0.46	2.17
Solver	BJ-pHDG			MG_{full}-pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	2.36		22.17	2.30		2.00
2	2.52	0.84	48.50	2.50	0.86	2.00
4	2.20	0.74	52.50	2.18	0.75	2.00
8	2.04	0.67	59.50	2.03	0.69	2.00
16	1.80	0.58	69.17	1.78	0.58	2.00
32	1.73	0.46	79.67	1.71	0.46	2.00

Table 6.6: Computational efficiency comparison using DG and HDG discretizations. Laminar vortex test case at $Re = 100$, $M = 0.05$, discretized using 16×16 mesh with $\mathbb{P}_6$ polynomials. Two time steps, $\Delta t = \{1; 0.1\}$ using ESDIRK3 scheme are reported. SU_{MB} stands for the speed-up factor referred to the DG, matrix-based, BJ-preconditioned computation, E is the parallel efficiency and IT_a the average number of GMRES iterations.

		$\Delta t = 1$				$\Delta t = 1/10$			
		BJ-DG		MG_{full}-DG		BJ-DG		MG_{full}-DG	
Case	n_p	MF	MFL	MF	MFL	MF	MFL	MF	MFL
$k = 3$	1	0.73	0.97	0.76	0.89	0.84	1.53	0.67	0.85
	2	0.68	0.80	1.02	1.21	0.74	1.09	0.81	1.04
	4	0.65	0.72	1.07	1.28	0.72	1.03	0.78	1.02
	8	0.61	0.73	1.13	1.32	0.68	0.91	0.80	1.03
	16	0.56	0.63	1.20	1.36	0.61	0.79	0.72	0.90
	32	0.59	0.62	1.41	1.59	0.58	0.71	0.69	0.86
$k = 6$	1	1.03	2.10	1.84	2.56	1.00	3.10	1.97	3.36
	2	0.99	1.47	2.42	3.18	0.99	2.03	2.33	3.98
	4	0.96	1.37	2.41	3.32	0.98	1.92	2.23	3.78
	8	0.93	1.25	2.69	3.78	0.95	1.75	2.18	3.65
	16	0.85	1.06	2.35	3.11	0.90	1.56	1.90	3.15
	32	0.78	0.98	2.37	3.03	0.86	1.41	1.70	2.77

Table 6.7: Computational efficiency of a matrix-free DG strategy. Laminar vortex test case at $Re = 100$, $M = 0.05$, discretized using 16×16 mesh with $\mathbb{P}_3$ and $\mathbb{P}_6$ polynomials. Two time steps, $\Delta t = \{1; 0.1\}$ using ESDIRK3 scheme are reported. SU_{MB} stands for the speed-up factor referred to the matrix-based, BJ-DG computation reported in Tables 6.5 and 6.6, SU_{MF} is the speedup computed considering the matrix-free BJ-DG settings. Results obtained by lagging the preconditioner evaluation (MFL) for the entire solution process, *i.e.* six linear systems, are also reported.

implementation of the iterative solver for the DG space discretization. In particular, the matrix-based speedup SU_{MB} using as a reference the matrix-based, BJ-DG solver. CPU time to show the performance compared to the reference algorithm. The performance are evaluated using the same preconditioners reported in Tables 6.5 and 6.6. Considering the single-grid matrix-free, BJ-DG numerical experiments, one can summarize that

1. for $k = 3$, switching MB to MF penalizes the CPU time. The reason for this is two-fold: first, the serial computation is 35% slower than the MB one. Second, when the solver is applied in parallel, the matrix-free implementation seems to be less parallel efficient, since the speed-up factor decreases by increasing the number of processors;

2. for $k = 6$, the penalization reduces. In fact, in serial computation the same computational time has been recorded, meaning that a matrix-free iteration performs similarly to a matrix-vector product. However, a lower parallel efficiency is still observed;

3. when a small time step is employed, higher speed-up values are achieved if compared to MB. In other words, the lower the number of GMRES iterations, the higher the performance. A slightly higher parallel efficiency is also observed.

Those observation are in line with what reported in the context of incompressible flow problems, see [77].

The possibility of lagging the preconditioner evaluation is also explored and the results reported in Table 6.7 (MFL). In particular, the recomputation of the preconditioner for six consecutive iterations is skipped, which means that the Jacobian

matrix is evaluated only for the first non-linear iteration of the first stage of the ES-DIRK3 scheme. By doing so, it is observed that the linear system converges with the same number of iterations which are not reported for brevity. This shows, at least for this kind of problem, that the preconditioner does not loose its efficiency throughout the stages of the same time step. Moreover, it confirms the powerful properties of the matrix-free iterative strategy, which allow to skip the Jacobian evaluation without degrading the convergence of the Newton's method.

From the CPU time point of view, lagging the preconditioner improves the computational efficiency, since the matrix is evaluated only once. This is reflected by the speed-up values, see the MFL result columns. Note that that the maximum speed-up values are obtained for high orders, when the impact of the Jacobian assembly is large on the overall cpu time, and for small time step sizes: in this case the conditioning of the matrix and the CPU time spent on the iterative solution process reduce, and so the computational time saved by skipping the Jacobian assembly reflect on the overall efficiency of the method. It is worth pointing out that by doing so the matrix-free penalization at low orders is reduced, while speed-up values in the range $[1.41, 3.10]$ are achieved for the $k = 6$, $\Delta t = 0.1$ case.

Similar considerations hold true when the matrix-free approximation is employed within a multigrid strategy. Also in this case the problem has been solved using the same number of GMRES iterations with respect to the non-lagging computation. However, from the CPU time of view, a higher speed-up value is achieved both in serial and in parallel runs thanks to the optimal multigrid scalability with respect to single-grid BJ preconditioning. The speed-up factor for the largest polynomial order is now in the range $[2.56, 3.78]$ and $[2.77, 3.98]$ for the large and small time step, respectively. Those speed-up values are larger than the one obtained for a fully matrix-based implementation of the DG discretization, see Table 6.6. In comparison to HDG, it can be seen that by using the Jacobian lagging option larger speed-up values are generally achieved for small time step sizes at high order. However, using MG_{full}-pHDG outperforms MG_{full}-DG even using matrix-free and preconditioner lagging for the largest time step employed, showing a better suitability to deal with very stiff and high order discretizations.

It is worth noting that, for computational efficiency, the matrix-free implementation of the iterative solver is employed only on the finest space of the multilevel iterative solution, similarly to what has been done in [95]. This choice is coherent with the results obtained for the single-grid preconditioner in Table 6.7. In fact, the matrix-free iteration costs similarly to a matrix-based one only for high orders, while it has an increased cost for lower order polynomials, it seems appropriate to employ matrix-based smoothers on the coarse levels of the strategy. When discretizing the equations using coarse meshes at relatively high orders, this idea seems to work optimally. Note also that the overall memory footprint of the application is dominated by the allocation of the block-Jacobi preconditioner for the finest space smoother, see Figure 6.2.

6.4.5 Remarks

The main points of the previous section are here summarized. First, it is proved that having large time step sizes maximise the advantages of coupling HDG and p-multigrid strategy, since the expenses of using static condensation together with a more powerful and expensive preconditioning strategy produce a faster solver only for high conditioning of the system, while the higher scalability of the multigrid algorithm as opposed to single-grid reflect on the overall scalability of the solver. On the other hand, for small time step sizes, the computational time is dominated by the Jacobian assembly and condensation for HDG, thus the CPU time as well as the parallel efficiency is dominated by the matrix assembly.

For HDG, it is shown that the *mixed* and *primal* formulations provide comparable results in terms of error levels both by refining in space and time, despite the *primal* form provides one-order lower convergence rate for the gradient variable. From the algorithmic point of view, the statically condensed system is typically solved using roughly the same number of iterations. However, the computational time is considerably lower for the pHDG formulation, since it does not deal with the Jacobian entries related to the state gradient. For those reason, in the remaining of the work only the primal formulation will be employed for benchmarking. Note that even for small time step size single-grid preconditioned pHDG solver performs fairly well in comparison to the other solution strategies, showing the benefits of the approach for unsteady flow computations.

In addition, the current implementation of the parallelization strategy makes HDG to be less parallel efficient than DG. This fact is ascribed to the higher amount of duplicate operations due to the integration over halo mesh elements and faces created by the domain decomposition required for the static condensation of the interior degrees of freedom of the Jacobian matrix. On the other hand, in DG the duplicate the work involve only partition faces. Other implementation choices, such as those devoting to the minimisation of the duplicate work on the partition boundaries, may be considered in future works.

Finally, it is worth pointing out that, despite being appealing, a matrix-free implementation of the hybridizable discontinuous Galerkin method is not at all straightforward to obtain satisfactory performance in CPU time [114], and thus its development is beyond the scope of the present work.

6.5 Results on complex test cases

The second family of numerical experiments deal with the solution of two test cases, dealing with i) the laminar flow over a two-dimensional circular cylinder at $Re = 100$ and $M = 0.2$ [102]; ii) the solution of the plunging motion of a NACA 0012 airfoil at $Re = 1000$ and Mach number $M = 0.2$. The latter case is solved by using the ALE mesh motion formulation introduced in [110]. Those results are devoted to extend the comparison to more complex unsteady flow problems. In all the cases reported herein, the right-preconditioning approach is still employed such that the convergence test of the linear solver is not affected by changing the preconditioning operator, and therefore all the numerical experiments are performed using a similar accuracy.

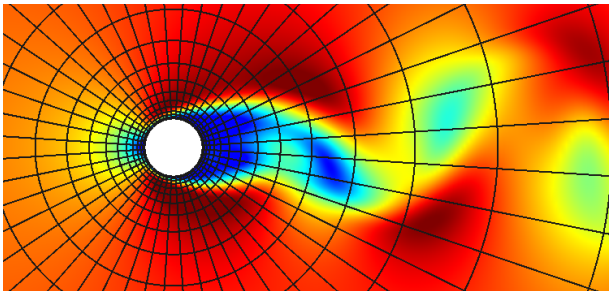


Figure 6.5: Laminar flow around a circular cylinder at $Re = 100$, $M = 0.2$. Mach number contours.

$(U/L)\Delta t$	C_d	C_l	St
0.5	1.3468	3.383e-03	0.16327
0.25	1.3519	-1.441e-03	0.16410
0.125	1.3527	-1.400e-04	0.16410
0.05	1.3528	-1.718e-04	0.16410
0.025	1.3528	-6.353e-06	0.16410

Table 6.8: Laminar vortex test case at $Re = 100$, $M = 0.05$. Time convergence rates for the DG and HDG discretizations and ESDIRK3 scheme.

6.5.1 Circular cylinder

The laminar flow around a circular cylinder at Mach number $M = 0.2$ and Reynolds number $Re = 100$ has been solved on a grid made by $n_e = 960$ mesh elements using a $\mathbb{P}_6$ space discretization. Figure 6.5 shows a snapshot of the computed Mach number contours. To integrate the governing PDEs in time, the four-stage, order-three Explicit-first-stage diagonally-implicit Runge–Kutta method (ESDIRK3) was employed. The solution accuracy was assessed in comparison with literature data [102]. To this end, Table 6.8 shows the averaged drag and lift coefficients, as well as the Strouhal number of the body forces (C_d, C_l, St) for several temporal refinements on the same grid. The coefficients were obtained by averaging a fully developed solution over ten shedding periods. An overall good agreement has been found, while a temporal convergence can be observed by using a non-dimensional time step of $\Delta t \leq 0.25$. It is well known [77] that the time step size deeply affects the iterative solution process, and therefore it has to be taken into account to evaluate the performance. Note that the largest time step size that allows to have converged body forces and Strouhal numbers and to maximise the efficiency of the solution strategy has been employed. Considering the results in Table 6.8, we choose $\Delta t = 0.25$. The parallel performance of the solution strategies introduced in the previous sections are here also assessed by considering the effects of domain decomposition. To do so, a fully-developed flow field is integrated in time for 10 time steps to compute the average number of GMRES iterations and the convergence rates during the non-linear solution. The computations are performed from 1 to 64 cores (n_p) on a platform based on two 16-core AMD Opteron processors arranged in a two-processor per-node fashion, for a total of 32

Prec.	BJ							
Case	DG-MB			DG-MF	DG-MFL	pHDG		
n_p	Time	E	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	E	IT_a
1	17425.77		19.43	1.00	3.90	2.18		17.40
2	11321.70	0.77	60.65	1.00	1.99	2.68	0.94	29.14
4	5907.54	0.74	70.39	0.99	1.67	2.65	0.89	34.16
8	2948.54	0.74	70.15	0.98	1.65	2.50	0.84	35.30
16	1551.90	0.70	85.41	0.94	1.49	2.40	0.77	39.15
32	815.46	0.67	98.24	0.90	1.35	2.19	0.67	45.11
64	692.93	0.39	117.33	0.86	1.19	2.09	0.38	50.53
Prec.	MG_{full}							
Case	DG-MB			DG-MF	DG-MFL	pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	E	IT_a
1	1.70		4.00	1.75	2.24	2.13		3.03
2	2.12	0.96	4.00	2.17	2.77	2.62	0.95	3.03
4	2.20	0.95	4.00	2.20	3.15	2.58	0.89	3.03
8	2.14	0.93	4.00	2.10	2.96	2.45	0.85	3.03
16	2.15	0.89	4.00	2.04	2.92	2.35	0.77	3.03
32	2.12	0.83	4.00	1.91	2.69	2.14	0.67	3.03
64	2.07	0.48	4.00	1.76	2.43	2.00	0.37	3.03

Table 6.9: Circular cylinder test case at $Re = 100$, $M = 0.2$, discretized using 960 mesh elements with $\mathbb{P}_6$ polynomials. Computational efficiency comparison of DG and pHDG solvers using BJ and p-multigrid, respectively. SU_{MB} stands for the speed-up factor referred to the DG[GMRES(BJ);MB] computation, E is the parallel efficiency and IT_a the average number of GMRES iterations.

cores per node. A fixed relative tolerance of 10^{-6} to stop the GMRES solver is used, as well as an absolute tolerance of 10^{-5} for the Newton-Raphson method.

Table 6.9 report the results of the computations. As observed for the convected vortex test case, the BJ preconditioner shows an increase in the number of GMRES iterations by increasing the number of processes in both the DG and pHDG space discretizations. This behavior is attributed to the way the incomplete lower-upper factorization is performed, as it deals with the squared, partition-wise block of the iteration matrix. Therefore, the preconditioner effectiveness naturally decreases as n_p grows, since the amount of off-diagonal blocks neglected by the ILU increases with n_p. By switching from MB to MF, the number of iterations remains the same and it is not reported for brevity. On serial computations, the CPU time remains in the same line since a speed-up value (SU_{MB}) of about one is obtained. When the solver is appied in parallel, a slightly lower parallel efficiency is observed, as the speed-up value decreases by increasing n_p. The possibility of lagging the Jacobian evaluation during the nonlinear solution process, as well as within a single time step through multiple stages of the scheme, is also explored and the results reported in the column labelled as MFL. In this case a large speed-up value is achieved especially for serial computations, where the ILU(0) works remarkably well. On the other hand, when this solution strategy is applied in parallel, a loss in parallel efficiency is observed and the solver performs in line with the reference, matrix-based method.

Considering pHDG with BJ preconditioning, the iterative solution process still suffers of algorithmic degradation by parallelizing the computation, despite being the increase in the number of iterations smaller than that observed for DG. Nevertheless, the solver appears to be faster being the speed-up value in the range [2.09, 2.68].

The parallel efficiency is also higher than that of the referece DG method: pHDG involves expensive local-to-each element operations that affect the computational time of the solver, but they scale ideally on multicore systems. In addition, a lower IT_a is generally observed when employing this space discretization.

On the lower part of the table numerical experiments performed using multigrid preconditioning are reported. Considering the matrix-based DG solver, it is possible to achieve an ideal parallel efficiency of the solver in terms of number of iterations by using multigrid, since they remain constant to 4.00 in each case. The speed-up in this case is in the range [1.7, 2.2], which is consistent with what has been reported previously. By using a matrix-free implementation of the finest level smoother and linear solver, similar performance for the non-lagged preconditioner are observed. As opposed to this, lagging the multigrid linear solver operator produce a speed-up value in the range [2.24, 3.15], which suggests this strategy to be the best performing of all the numerical experiments. In fact, the use of multigrid preconditioning in pHDG does not improve the computational efficiency of the solver despite reducing considerably the number of iterations and providing optimal algorithmic scalability. Observing the pHDG results, one can understand that with such time step size the costs of statically condense out the interior DoFs, as well as the back-solve to recover them from the trace DoFs, dominate the percentage of the overall computational time. It is worth mentioning that in such a practical application the p-multigrid preconditioning approach for HDG shows to work very well from the number of iterations point of view, and that the same numerical set-up than that of a DG solver reported in Table 6.4 has been employed to precondition the system.

As a final comparison, Table 6.10 reports the same test case solved using a grid obtained by splitting the quad elements into triangles. TriLagrange basis functions were employed. Only the computation with $n_p = 64$ is reported. It is worth pointing out that this space discretization reduces the total number of DoFs by the 23.8% with respect to the previous one. By comparing the computational time and average number of GMRES iterations, similar speed-up values are obtained using the DG-MB solver as well as pHDG. On the other hand, the DG-MF solver seems to be slightly penalised with respect to the MB one, as the speed-up values of the computations drop by a factor between 7 to 15 percent. It is worth pointing out that the matrix-free penalization that arise in this case is in line to what reported in previous studies [77, 95] using broken polynomial spaces, which reduce by increasing the number of DoFs per element, for example in three-dimensional computations. On the other hand, a slightly larger speedup for pHDG computations have been observed, which make this solution strategy to be the most convenient from the CPU time point of view.

6.5.2 Laminar flow around a heaving and pitching NACA 0012 airfoil

The test case is the CL1 (heaving and pitching airfoil) proposed on the 5^{th} international workshop on high-order CFD methods (HiOCFD5), see [54]. The simulation involve the set of compressible Navier–Stokes equations, with $\gamma = 1.4$, $Pr = 0.72$ and constant viscosity, to simulate flow over a moving NACA 0012 airfoil. The airfoil was

Preconditioner	BJ					
Case	DG-MB		DG-MF	DG-MFL	pHDG	
n_p	Time	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	322.26	129.050	0.80	1.00	2.40	49.750
Preconditioner	MG_{full}					
Case	DG-MB		DG-MF	DG-MFL	pHDG	
n_p	SU_{MB}	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	1.66	4.525	1.43	1.77	2.05	3.388

Table 6.10: Circular cylinder test case at $Re = 100$, $M = 0.2$, discretized using 1920 mesh elements with $\mathbb{P}_6$ TriLagrange basis functions. Computational efficiency comparison of DG and pHDG solvers using BJ and p-multigrid, respectively. SU_{MB} stands for the speed-up factor referred to the matrix-based, BJ-DG computation and IT_a for the average number of GMRES iterations.

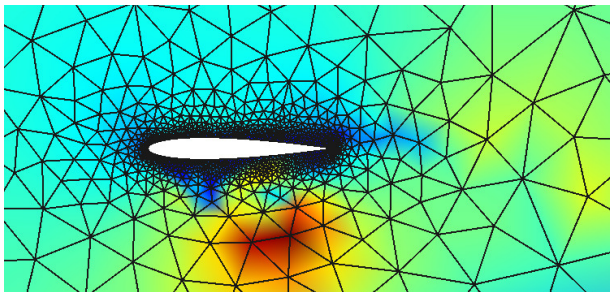


Figure 6.6: Laminar flow around a heaving NACA 0012 airfoil. $Re = 1000$, $M = 0.2$. Mach number contours at the final time $T = 2$.

modified to obtain a closed trailing edge via the equation

$$y(x) = \pm 0.6 \left(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1036x^4\right), \qquad (6.45)$$

with $x \in [0, 1]$. The initial condition was a steady state solution at a free-stream Mach number of $M = 0.2$ and a Reynolds number $Re = 1000$. The motion is a pure plunging motion governed by the following function

$$h(t) = t^2(3 - t)/4, \qquad (6.46)$$

and the output of interest is the total energy and vertical impulse exchanged between the airfoil and the fluid, that can be computed as

$$W = \int_T F_y(t)\dot{h}(t)dt \qquad I = \int_T F_y(t)dt \qquad (6.47)$$

being $F_y(t)$ the vertical force computed on the airfoil surface, and $\dot{h}(t)$ the time derivative of Eq. (6.46). In particular, the output is evaluated at a non-dimensional time $T = 2$. The mesh involves a triangulation of the domain made by $n_e = 2137$ elements, reported in Figure 6.6, and $\mathbb{P}_6$ polynomials.

The time step size of the simulation have been evaluated through a time-convergence

$T/\Delta t$	W	I
20	-1.3860	-2.3667
40	-1.3839	-2.3444
80	-1.3837	-2.3391
160	-1.3837	-2.3378

Table 6.11: Time convergence analysis of the laminar flow around a plunging NACA 0012 airfoil. A satisfactory accuracy is observed for $T/\Delta t = 2$, where $T = 2$ is the length of the simulation and Δt is the time step size.

Preconditioner	BJ					
Case	DG-MB		DG-MF	DG-MFL	pHDG	
n_p	Time	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	900.19	136.119	0.69	0.81	2.35	49.750
Preconditioner	MG_{full}					
Case	DG-MB		DG-MF	DG-MFL	pHDG	
n_p	SU_{MB}	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	1.49	4.477	1.06	1.13	2.13	2.754

Table 6.12: Circular cylinder test case at $Re = 100$, $M = 0.2$, discretized using 2137 mesh elements with $\mathbb{P}_6$ TriLagrange basis functions. Computational efficiency comparison of DG and pHDG solvers using BJ and p-multigrid, respectively. SU_{MB} stands for the speed-up factor referred to the matrix-based, BJ-DG computation and IT_a for the average number of GMRES iterations.

analysis on the output quantities, reported in Table 6.11. Table 6.12 compares the computational efficiency of the solution strategies in terms of speed-up factor (SU_{MB}) and number of iterations IT_a. Only the computation with the larger number of cores is reported to show the efficiency on large and practical computations. The reference to compute the speed-up is the computational time of the matrix-based, BJ-DG solver. By switching from a matrix-based to a matrix-free implementation, the computational strategy is penalised by higher computational cost of a single iterations, see the MF column. By employing the Jacobian lagging (MFL), this penalization is reduced only slightly. On the other hand, the pHDG implementation provides a solver which is more than twice as much as fast, while the system require a considerably lower number of GMRES iterations with respect to the reference. The use of p-multigrid preconditinoing in a DG context provides a speed-up factor of about 1.49 for a fully matrix-based implementation, and the number of iterations drops from 136 to around 4.4 in average. The use of matrix-free in this case still penalizes the solver, which result in about 12% faster for the MFL case. Multigrid preconditioning shows to be robust enough to precondition the linear system arising from the pHDG discretization, since it converges using an average of 2.754 iterations. However, this gain is not reflected on the CPU time, which is slightly higher.

6.6 Final remarks

The Chapter compares, within the same framework, the computational efficiency of different high order discontinuous Galerkin implementations. The first involves a modal discontinuous Galekin method coupled with matrix based and matrix-free it-

erative solvers, while the second one considers a hybridizable discontinuous Galerkin implementation both in the *mixed* form, which allocates the components of the Jacobians related to the gradient variable, and in the *primal* form, which simmetrizes the discretization by adding an additional adjoint-consistency term. The efficiency of the solution strategies is assessed by comparing different single-level preconditioners as well as multilevel ones such as p-multigrid, on a variety of two-dimensional test cases involving laminar viscous flows, including mesh motions, on meshes made by triangular and quadrangular elements. The effects of parallel efficiency and the use of different basis functions have also been considered. The results show that the use of a matrix-free implementation of the iterative solver in the context of implicit discontinuous Galerkin discretizations provides a memory footprint which is in line to that of an HDG method if a block-diagonal preconditioner is employed within the smoother of the finest space. The *primal* HDG method reveals to be more efficient than the *mixed* one, having a lower number of Jacobian entries to be condensed out of the system, and it provides comparable error levels. If compared to pHDG, the p-multigrid matrix-free solver is competitive in terms of CPU time when the problems involve time marching with small time steps, since the preconditioner evaluation can be lagged. On the other hand, HDG methods require expensive element-wise operations that reveal to be a bottleneck in those conditions. Finally, a novel approximate-inherited p-multigrid strategy has been also introduced for HDG. Such strategy showed to be robust and efficient for different test cases and mesh types, being able to reduce considerably the number of iterations of the solution process. However, this gain in number of iterations is not reflected on the CPU time of the solver. Future works will be devoted to the validation of those strategies on stiff three-dimensional cases involving laminar and turbulent flows, and possibly hybrid RANS-LES models.

Chapter 7

Applications to incompressible turbulent flows

The aim of this chapter is two-fold. The first objective is to apply and validate the implicit matrix-free numerical framework on stiff three-dimensional test cases to show reliability and accuracy of the method. Second, the aim is to compare the numerical framework with state-of-the-art methods involving matrix based iterative solvers to summarize the improvements in computational efficiency. The test cases, hereby presented with a growing complexity, involve implicit Large Eddy Simulations (ILES), *i.e.* where the dissipation of the numerical scheme act as a proper subgrid-scale model (SGS). First, the Rayleigh–Bénard natural convection problem at Prandtl number equal to 0.7 for Rayleigh numbers (Ra) up to 10^8 is presented. Second, the incompressible turbulent channel flow up to $Re_\tau = 950$ will be considered. Third, the complete ERCOFTAC T3L test case suite, involving incompressible transitional turbulent flow around a rounded leading edge flat plate for different Rayleigh numbers and free stream turbulence intensities, is presented. Finally, preliminary results of the ILES of the Boeing Rudimentary Landing Gear at $Re = 10^6$ are reported. In all the applications, a discussion on the solution quality will precede the analysis of the computational efficiency of the strategies. The results corroborate that, when coupled with appropriate memory-saving and scalable preconditioners, the use of matrix-free solvers allows a clear improvement in computational efficiency in large HPC facilities.

7.1 Rayleigh–Bénard natural convection

The Rayleigh–Bérnard Convection (RBC) has ben solved using a Prandtl number equal to 0.7 and four different Rayleigh numbers, 10^5, 10^6, 10^7 and 10^8. The computational domain employed is a box characterised by an aspect ratio $L/H = 8$, where L is the wall length scale and H is the distance between the two walls. The variables $x = x_1$ and $y = x_2$ are employed for the wall-parallel directions, while $z = x_3$ for the wall-normal direction. The same has been done for the velocity field components $u = u_1$, $v = u_2$ and $w = u_3$. The RBC equations are here non-dimensionalized using the distance between the walls H, the temperature difference of the two walls $\delta\Theta = T_h - T_c$ and the free-fall velocity $u_R = \sqrt{g\alpha\delta\Theta H}$.

Physical discussion

Table 7.1 shows the computational details of various solutions obtained using different space discretizations and time step sizes. The number of DoF employed is at least one order of magnitude lower than that employed within the reference DNS data [117], therefore the computations should be considered as ILES, apart from Case 1 and

Case	Ra	$\mathbb{P}$	Grid	δt	$\delta t/\delta t_R$	DoF	$\text{DoF}_\text{R}/\text{DoF}$	Nu_1	Nu_2	Nu_3	Nu_4	Nu_R	Nu_{exp}
1	10^5	5	24×24×8	0.1	–	258048	–	4.25	4.25	4.24	4.25	–	3.74
2	10^6	3	16×16×8	0.1	8.33	40960	206.40	5.09	6.59	7.25	7.39	8.17	7.52
3		4	16×16×8	0.1	8.33	71680	117.94	6.07	7.18	7.71	7.56		
4		5	16×16×8	0.1	8.33	114688	73.71	6.79	7.57	7.98	7.87		
5		6	16×16×8	0.1	8.33	172032	49.14	7.30	7.84	8.15	8.10		
6		3	20×20×10	0.05	4.16	80000	105.68	5.75	7.00	7.57	7.62		
7		6	20×20×10	0.05	4.16	336000	25.16	7.65	7.97	8.17	8.16		
8		6	20×20×10	0.1	8.33	336000	25.16	7.65	7.97	8.17	8.16		
9		6	20×20×10	0.2	16.66	336000	25.16	7.64	7.98	8.18	8.17		
10		6	20×20×10	0.4	33.33	336000	25.16	7.56	7.96	8.16	8.15		
11		3	32×32×16	0.05	4.16	327680	25.80	6.95	7.66	8.03	7.95		
12		6	40×40×20	0.1	8.33	2688000	3.15	8.11	8.14	8.17	8.17		
13	10^7	3	32×32×16	0.05	17.86	327680	228.70	8.39	12.18	13.77	14.05	15.55	15.11
14		4	32×32×16	0.05	17.86	573440	130.69	10.53	13.47	14.85	14.49		
15		5	32×32×16	0.05	17.86	917504	81.68	12.09	14.22	15.34	15.14		
16		6	32×32×16	0.05	17.86	1376256	54.45	13.17	14.66	15.46	15.50		
17		6	40×40×20	0.1	35.71	2688000	27.88	13.92	14.95	15.53	15.52		
18	10^8	3	72×72×36	0.1	–	3732480	–	15.46	23.94	27.57	27.64	–	30.35
19		4	72×72×36	0.1	–	6531840	–	19.27	26.10	29.13	28.59	–	
20		5	72×72×36	0.05	–	10450944	–	22.99	27.70	30.04	29.89	–	

Table 7.1: Computational details and computed Nusselt (Nu) numbers on various numerical setup of the DG-ILES solver. $\delta t = \delta \tilde{t} U_R/H$ where $\tilde{t}$ is the dimensional time. δt_R is the average time step size (private communication) while DoF_R is the number of DoF of the reference DNS [117]. Nu_1, Nu_2, Nu_3, Nu_4 are defined in Eq (7.3). Nu_R is the reference Nusselt from [117]; $Nu_{exp} = 0.125 Ra^{0.303} Pr^{0.25}$ from [118]. The ROSI2PW scheme with adaptive GMRES tolerance of Eq. (4.7) was used for all the computations.

12, which have a resolution comparable to that of a DNS. Consistency relations, widely reported in literature [117, 119], have been calculated on the entire domain and are summarized in the table. Those relations are based on the definition of the pseudo-dissipation of kinetic energy, defined as

$$\epsilon = \sqrt{\frac{Pr}{Ra}} \left(\frac{\partial u_i}{\partial x_j} \frac{\partial u_i}{\partial x_j} \right) \tag{7.1}$$

and the dissipation of the temperature squared, defined as

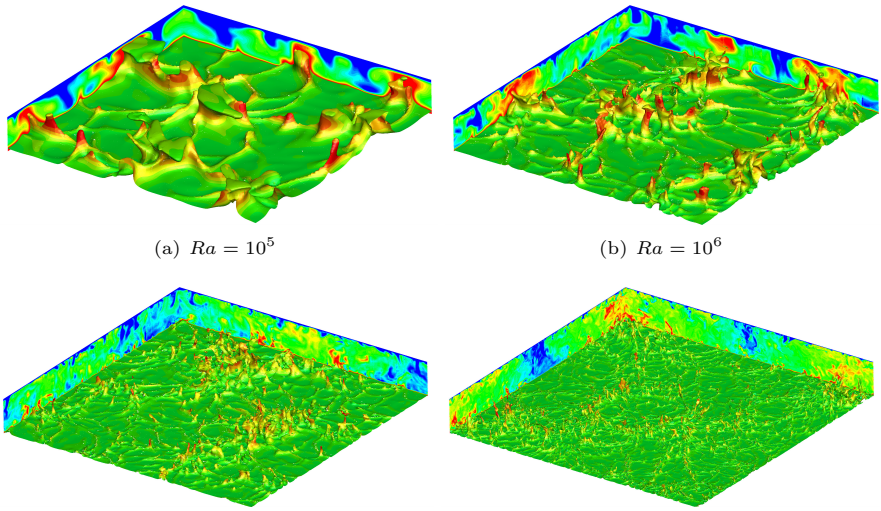
$$\chi = \frac{1}{\sqrt{RaPr}} \left(\frac{\partial \theta}{\partial x_i} \frac{\partial \theta}{\partial x_i} \right). \tag{7.2}$$

Introducing the volumetric average operator $\langle \cdot \rangle_V$, which refers to the time-average over the whole fluid domain, and the wall average operator $\langle \cdot \rangle_w$ which denotes the time-average over the wall surfaces, the following identity must be satisfied

$$\begin{aligned} \langle Nu \rangle_V &= \underbrace{1 + \sqrt{RaPr} \langle \epsilon \rangle_V}_{Nu_1} = \underbrace{\sqrt{RaPr} \langle \chi \rangle_V}_{Nu_2} \\ &= \underbrace{1 + \sqrt{RaPr} \langle w\theta \rangle_V}_{Nu_3} = \underbrace{\langle \frac{\partial \theta}{\partial z} \rangle_w}_{Nu_4} \end{aligned} \tag{7.3}$$

It can be seen that as the resolution increases, enhancing the order of polynomial expansion or the grid size, the values become consistent with the numerical DNS reference Nu_R from [117]. An experimental correlation, $Nu_{exp} = 0.125 Ra^{0.303} Pr^{0.25}$ from [118], has been also evaluated to confirm further the quality of the Implicit LES solution where the DNS data were not available. However, it is more convenient, for what regards the number of DoF, to raise the polynomial order instead of the grid size (see, for instance, Cases 7 and 11 of Tab. 7.1). Note that for a well resolved simulation all the different definitions of the Nusselt number should be equivalent, but Nu_1, Nu_2 and Nu_4 involve the components of the gradients, which are one order less accurate, and therefore they are slower than Nu_3 to reach convergence.

Fig. 7.1 shows the computed iso-surface of temperature $\theta = 0.15$ for different Rayleigh numbers at a representative time far enough from the initial transient phase. The space distribution of the thermal plumes, the coherent structures of the RBC convection, and their reduction in size can be observed as the Rayleigh number increases. Fig. 7.2 shows the computed turbulent statistics using the time-space average operator $\langle \cdot \rangle_\pi$, which deals with ensemble averages of z-averaged quantities. The DG-ILES solution fits well the DNS data despite having a number of DoF orders of magnitude lower than the reference, $\mathrm{DoF_R}$. In particular, as the order k of the polynomial approximation $\mathbb{P}_k$ increases, the amount of numerical dissipation decreases, and the peaks of fluctuation are described with a higher accuracy thanks also to the growth of the cut-off frequency of the scheme. This behaviour can be observed in Fig. 7.3, which shows the power spectral density of the kinetic energy and temperature for the cases at $Ra = 10^6$. The spectra have been obtained using the time series of the variables in a probe point at $z = H/2$, inside the bulk region. The resolved maximum



(a) $Ra = 10^5$ (b) $Ra = 10^6$

(c) $Ra = 10^7$ (d) $Ra = 10^8$

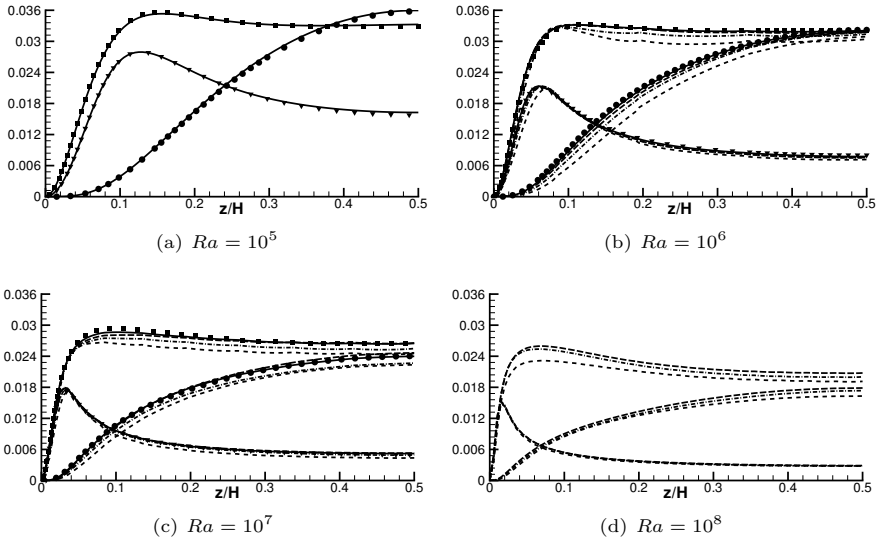
Figure 7.1: Ra effect on the Rayleigh–Bérnard convection. Iso-surfaces of temperature $\theta = 0.15$ coloured by the vertical velocity magnitude. The lateral boundaries report temperature contours.

frequency becomes broader by increasing the order of polynomial approximation (see, for instance, the blue and black solid lines of Fig. 7.3). However, due to the favourable spectral properties of the scheme, the resolved part of the inertial range remains more or less unaffected by the numerical resolution and fits very well the $-5/3$ (Kolmogorov scaling) and the $-7/5$ (Bolgiano scaling) laws for the kinetic energy and the temperature, respectively, as reported in [119]. Moreover, those spectra are in very good agreement with those obtained on a fine grid, reported in green (Case 12 of Tab. 7.1).

A comparison between the solutions obtained using different time step sizes was performed raising eight times the δt value (red and black lines of Fig. 7.3). The resulting spectra are nearly superimposed both in the inertial range and in the dissipative range, where an additional numerical dissipation, although being very small, can be noticed. This result proves the computational benefits of using a high-order implicit time integration strategy (see, in addition, the computed Nusselt number of Cases 7, 8, 9 and 10 of Tab. 7.1). The $\delta t = 0.4$ value is, in fact, almost 35 times larger than that used to compute the reference DNS data.

Assessment of the computational efficiency

As regards the computational efficiency, the comparison between matrix-based and matrix-free solvers is here discussed. A representative setting of the typical use of the solver for turbulent flows is exploited for the RBC problem using up to 6^{th} order polynomials. The grid size selected in this case was made of $20\times20\times10$ mesh elements, and the calculation was run on a four AMD Opteron 6276 CPUs consisting of 64 cores. The numerical experiments were performed by measuring the CPU time



(a) $Ra = 10^5$ (b) $Ra = 10^6$

(c) $Ra = 10^7$ (d) $Ra = 10^8$

Figure 7.2: Turbulent statistics of the Rayleigh–Bénard convection: $Ra = 10^5$ (24×24×8 mesh elements), $Ra = 10^6$ (16×16×8 mesh elements), $Ra = 10^7$ (32×32×16 mesh elements) and $Ra = 10^8$ (72×72×36 mesh elements). Reference DNS data symbols: kinetic energy fluctuations $\langle k' \rangle_\pi$ (squares), vertical velocity fluctuations $\langle w'^2 \rangle_\pi$ (circles) and temperature fluctuations $\langle \theta'^2 \rangle_\pi$ (gradients), from [117]. DG-ILES solutions at $\mathbb{P}_3$ (dashed lines), $\mathbb{P}_4$ (dash dotted lines), $\mathbb{P}_5$ (long dashed lines) and $\mathbb{P}_6$ (solid lines).

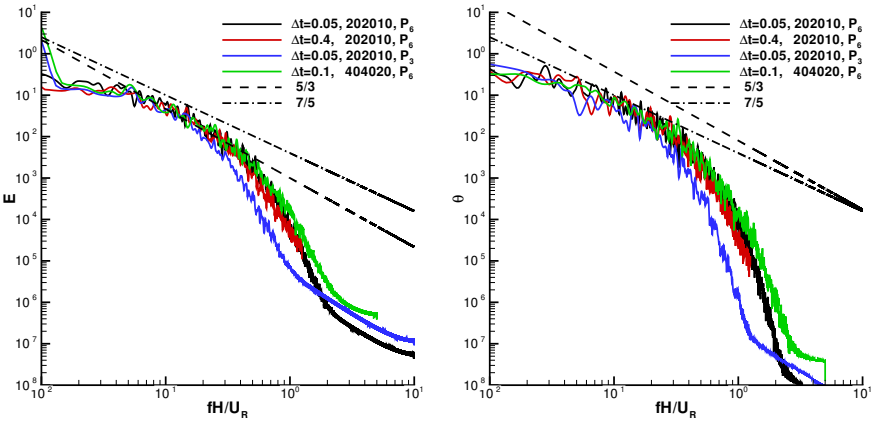
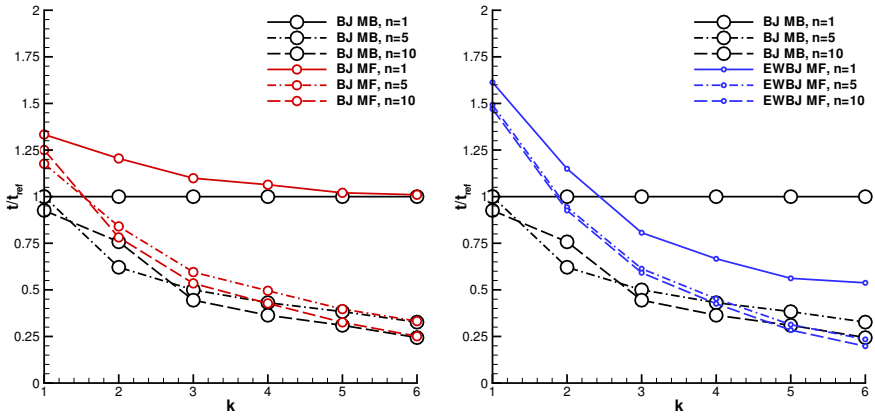


Figure 7.3: Power Spectral Density of the Rayleigh–Bénard convection: $Ra = 10^6$, kinetic energy E (left) and temperature θ (right).

required to advance the solution in time for 10 time steps. As observed for two-dimensional cases, an appropriate solution tolerance of the linear system resulting from the implicit in time DG discretization have to be considered to evaluate the



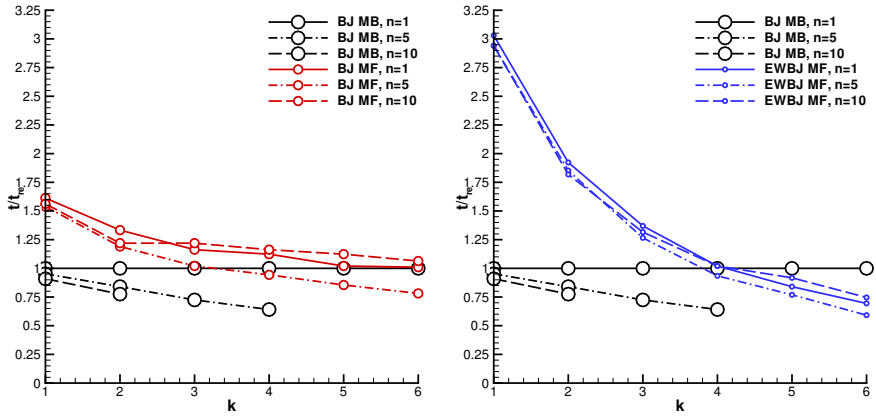
(a) BJ (b) EWBJ

Figure 7.4: Effect of polynomial approximation on computational performances. RBC problem at $Ra = 10^6$, 20×20×10, $\mathbb{P}_6$ space discretization and $\delta t = 0.05$.

computational efficiency. For production runs, the implicit scheme allows the use of large δt, which typically produce time discretization errors far from the machine precision, even using very high order accurate time integration schemes. In consistency to what has been pointed out in Chapter 4 for the two-dimensional case, in such conditions the use of a very small system tolerance does not increase the solution quality, but degrades the computational efficiency. This is particularly true in the context of an implicit LES solution, where the space discretization errors due to the under-resolved flow field dominate the global output error.

Fig. 7.4 shows the CPU time, non dimensionalized using the BJ-MB case, for the numerical experiments performed using $\delta t = 0.05$. In this situation a small time step and a large tolerance on the linear system is employed, which condition is the most favourable to highlight the benefits of a matrix-free approach over the matrix-based.

Three lagging combinations $n = 1, 5, 10$ are reported. As already noticed for the two-dimensional case, all the matrix-free strategies behave increasingly better as the order of polynomial approximation k increases. Considering the comparison BJ-MB and BJ-MF with $n = 1$, one may notice that the matrix-free has a Speed-up value asymptotically constant for high order polynomials, which confirms the theoretical estimates of the operation counts per elements regarding matrix-vector products and residual evaluations reported in Chapter 4. If the Jacobian lagging is employed, the CPU time is reduced considerably, especially for the highest order considered ($k = 6$). The effect is similar to that of Section 4.3.4, but the gain is greater due to the higher values of the r parameter, which is reported in Tab. 7.2. In particular, the BJ-MF with $n = 10$ provides a CPU time saving of about 75% for a $\mathbb{P}_6$ space discretization (red, dashed lines), while the EWBJ-MF (blue, dashed line) setting outperforms all the the others reaching a saving of 80%. The convenience of using the EWBJ preconditioner is due to the fact that the large computational costs of the



(a) BJ (b) EWBJ

Figure 7.5: Effect of polynomial approximation on computational performances. RBC problem at $Ra = 10^6$, 20×20×10, $\mathbb{P}_6$ space discretization and $\delta t = 0.4$.

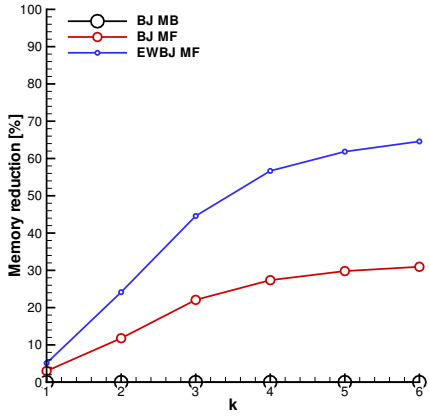


Figure 7.6: Percentage of memory footprint reduction obtained by using the MF approach on the Rayleigh–Benard convection problem for different values of polynomial order.

full Jacobian evaluation and its factorization, which grow considerably with the order of polynomial approximation, are avoided. The efficiency of the BJ preconditioner is in this case not large enough, if compared to the cheap EWBJ, to make this procedure convenient from the CPU time point of view. Finally, while the EWBJ scales almost linearly over a wide range of MPI ranks, the BJ preconditioner scales poorly, as it has been seen in the two-dimensional case.

Fig. 7.5 shows the same numerical experiments, for different order polynomials, using a numerical setup more consistent with a typical use of present DG solver, *i.e.*

$\mathbb{P}$	1	2	3	4	5	6
r (BJ)	12	33	70	120	235	560
r (EWBJ)	7	13	25	40	90	225

Table 7.2: Computed $r = T_J/T_R$ for different values of polynomial order on the 3D RBC problem.

large time step size ($\delta t = 0.4$) and the adaptive GMRES tolerance of Eq. (4.7) (Case 10 of Table 7.1). In this case, the lagged matrix-free solvers decrease the CPU time, with respect to the MB case, only for $k > 4$. In particular, the setting EWBJ-MF with $n = 5$ shows the highest efficiency with a speed-up factor of about 1.8 when using $\mathbb{P}_6$ space discretizations. In this case the large time step size increases the stiffness of the iteration matrix, which therefore calls for a higher number of GMRES iterations to reach the target tol_r. Despite this situation is the worst, as it has been seen on the two-dimensional numerical experiments, the EWBJ preconditioner coupled with the matrix-free still outperforms the other approaches. It is worth noting that the test done using $n = 10$ showed higher CPU times than $n = 1$, because the preconditioner must be updated more frequently during the time integration to maintain its efficiency. As a matter of facts, the Jacobian lagging for the BJ-MB, although reducing the CPU time, revealed to be stable only using low polynomial orders. Only the results for which the calculation was stable have been reported, see black circular symbols on Fig. 7.5. The lagged matrix-based solver has been here used only to evaluate the CPU time and the solution accuracy, which may be less accurate, will not be discussed. Note that for $\delta t > 0.4$ a significant reduction of the stability of both the MB and MF schemes has been observed.

Fig. 7.6 shows the memory saving percentage, independent of the time step size, for the three schemes considered as a function of k. As expected, all the matrix-free versions save a significant amount of memory. This saving is referred to the total memory footprint of the solver, estimated through the PETSc library, which considers also the storage of the shape functions and their derivatives at the gauss integration points, and the memory dynamically allocated for the Krylov subspaces during the GMRES system solution. Within the solution process, the highest percentage of saved memory reaches the 30% (see the black square symbols) for the BJ-MF setting. If the EWBJ-MF preconditioner is used, the memory saving reaches the 66% at $\mathbb{P}_6$.

In conclusion, the EWBJ-MF setting revealed to be the best choice to deal with the solution of such incompressible turbulent flow, proving to be efficient and to scale optimally even on highly parallel systems. For example, it is worth to report that the full (starting from a trivial initial condition and up to the non dimensional time $t = 200$) $Ra = 10^8$ computation, involving more than $5.2 \cdot 10^7$ DoF, is accomplished in less than one day using 68 Intel Xeon processors (1224 cores).

7.2 Channel flow

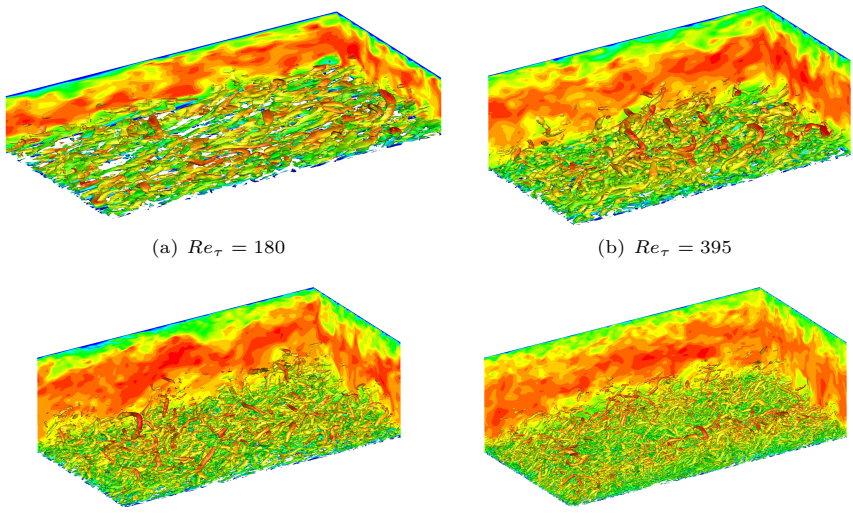
The turbulent channel flow has been computed using four different reynolds numbers, $Re_\tau = 180$, 395, 590 and 950, see Fig. 7.7. The computational details of those calculations are reported in Tab. 7.3. Those Re_τ were chosen in consistency with the reference DNS data in [120, 121]. While for the first three cases the same domain sizes of the DNS data were used, a smaller domain has been employed for the latter case. The computational domains were built using at least a number of DoF one order of magnitude lower than the reference DNS data evaluated using the same domain size. In order to allow an adequate resolution of the viscous sublayer, the mesh elements were stretched at the walls using the following mapping [122]:

$$z = \frac{(2a+b)[(b+1)/(b-1)]^{(h-a)/(1-a)} + (2a-b)}{(2a+1)\{1+[(b+1)/(b-1)]^{(h-a)/(1-a)}\}} \tag{7.4}$$

where h is a linear mapping between the two walls and a, b are coefficients that can be tuned to obtain a suitable wall normal resolution. The fully developed flow field was finally generated using a fixed pressure gradient by adding a constant source term on the momentum equation along the x axis.

Physical discussion

Figures 7.8, 7.9 and 7.10 show the results of the implicit Large Eddy simulation here performed. Due to the x, y homogeneity of the flow, the average velocities and the fluctuations have been obtained using similar procedures of those used for



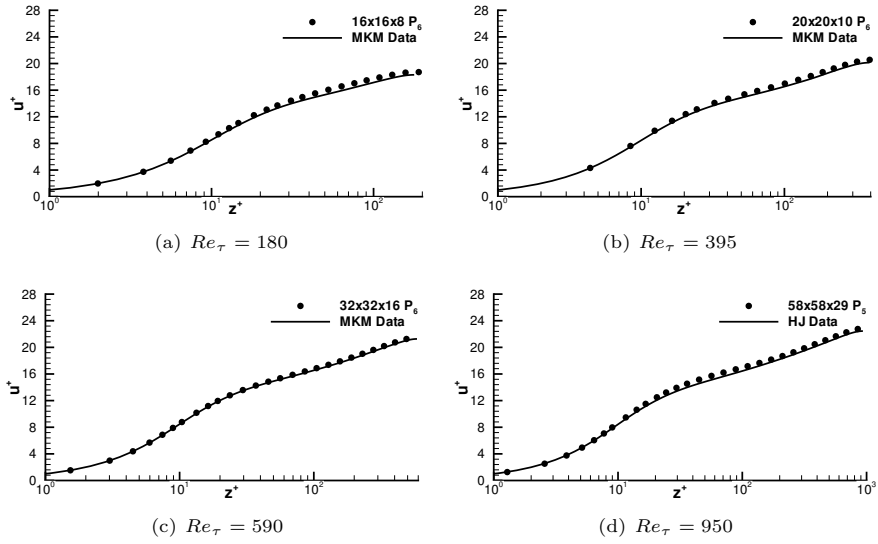
(a) $Re_\tau = 180$ (b) $Re_\tau = 395$

(c) $Re_\tau = 590$ (d) $Re_\tau = 950$

Figure 7.7: Turbulent channel flow at different Re_τ. λ_2 iso-surfaces coloured by the streamwise velocity magnitude. The lateral boundaries report streamwise velocity contours.

Re	Domain	$\mathbb{P}$	Grid	a	b	δx^+	δy^+	δz^+	δt^+	$\delta t/\delta t_R$	DoF	$\mathrm{DoF_R}/\mathrm{DoF}$
180	$4\pi\times2\pi\times2$	6	16×16×8	0.5	1.050	141.37	70.69	12.96	0.90	13.33	$1.72\cdot10^5$	12.43
395	$2\pi\times\pi\times2$	6	20×20×10	0.5	1.020	124.09	62.05	11.69	0.99	–	$3.36\cdot10^5$	28.23
590	$2\pi\times\pi\times2$	6	32×32×16	0.5	1.025	115.85	57.92	10.56	0.59	–	$1.38\cdot10^6$	27.46
950	$2\pi\times\pi\times2$	5	58×58×29	0.5	1.030	102.91	51.46	9.50	0.95	–	$5.46\cdot10^6$	19.22

Table 7.3: Computational details of the turbulent channel flow test case. The coefficient a, b are those used for the wall grid refinement within the Eq. (7.4). The $\delta(\cdot)^+ = u_\tau\delta(\cdot)/\nu$ are the non-dimensional minimum element spacings employed in all the domain, while $\delta t^+ = u_\tau^2\delta t/\nu$ is the non-dimensional time step size employed for the simulation. δt_R is the time step size employed for the DNS [123], while $\mathrm{DoF_R}$ are the reference DNS Degrees of Freedom [120, 121]. Note that for a given mesh and polynomial order the equivalent spatial resolution can be obtained by dividing the mesh size by $(n_v)^{1/3}$.



(a) $Re_\tau = 180$ (b) $Re_\tau = 395$

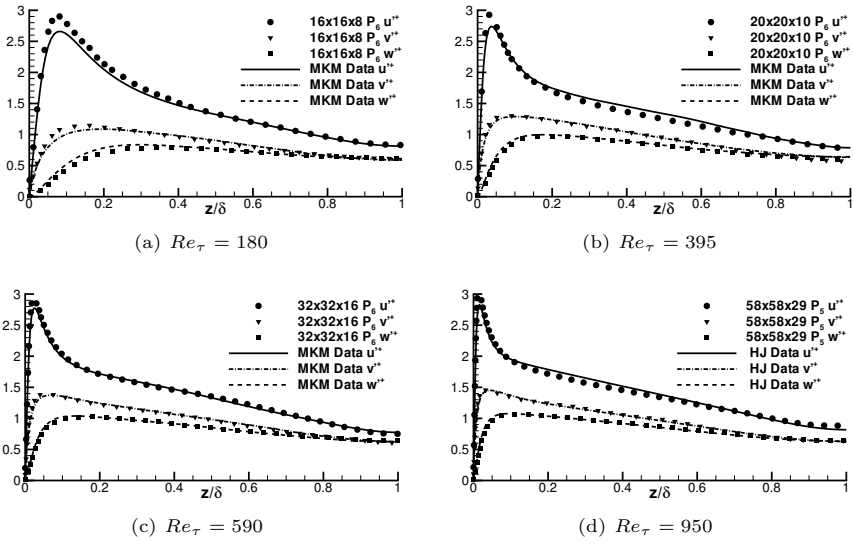
(c) $Re_\tau = 590$ (d) $Re_\tau = 950$

Figure 7.8: $\langle u^+(z^+)\rangle$ mean profiles for the turbulent channel flow at different Reynolds numbers in comparison with reference DNS data from [120] and [121].

the Rayleigh–Bénard convection, *i.e.* using the same $\langle\cdot\rangle_\pi$ operator. A very good agreement with the reference DNS data [120, 121] can be obtained independently of the Reynolds number using a considerable lower number of DoF. In the author's experience, to achieve this solution quality, the grid spacing at the wall should entail a first cell resolution lower than 3 wall units. Such number can be estimated, in a DG context, by dividing the minimum first cell height δz^+, reported in Table 7.3, by the directional number of DoF per equation per element $n_v^{1/3}$. As regards the time step size, that thanks to the implicit scheme here used, it can be increased with confidence without spoiling the accuracy of the statistics, and it can be taken as the largest allowed, for stability reasons, by the numerical scheme. It is worth noticing that time step sizes reported in Tab. 7.3 are up to 45 times larger than those used in [75], in the context of a DG discretization and a semi-explicit time integration scheme.

Performance assessment

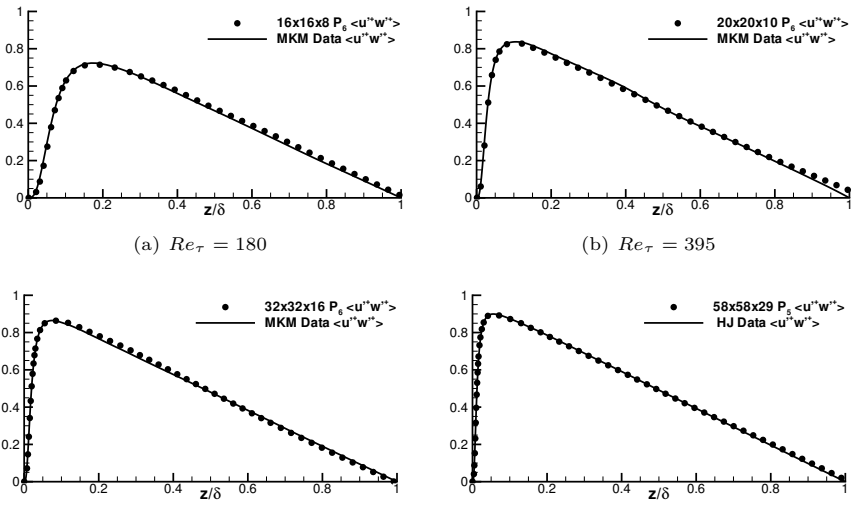
The influence of using the MB or MF approaches on the computational efficiency has been tested for the $Re_\tau = 180$ case using a 16×16×8, $\mathbb{P}_6$ space configuration, varying the time step size, on the four AMD Opteron CPUs. The effects of the adaptive GMRES tolerance have been included in the comparison, *i.e.* as the time step size decreases, the GMRES system is solved with a higher resolution within each stage to maintain the system solution error below the time integration error. Fig. 7.11(a) shows the relative CPU time computed in analogy with those of the RBC problem. The EWBJ-MF reaches the highest efficiencies (a CPU time reduction up to 70%) for all the time steps employed despite the slightly higher number of GMRES iterations



(a) $Re_\tau = 180$ (b) $Re_\tau = 395$

(c) $Re_\tau = 590$ (d) $Re_\tau = 950$

Figure 7.9: $\sqrt{\langle u'^{+2}\rangle}$, $\sqrt{\langle v'^{+2}\rangle}$ and $\sqrt{\langle w'^{+2}\rangle}$ profiles for the turbulent channel flow at different Reynolds numbers in comparison with reference DNS data from [120] and [121]. δ is the semi-channel height.

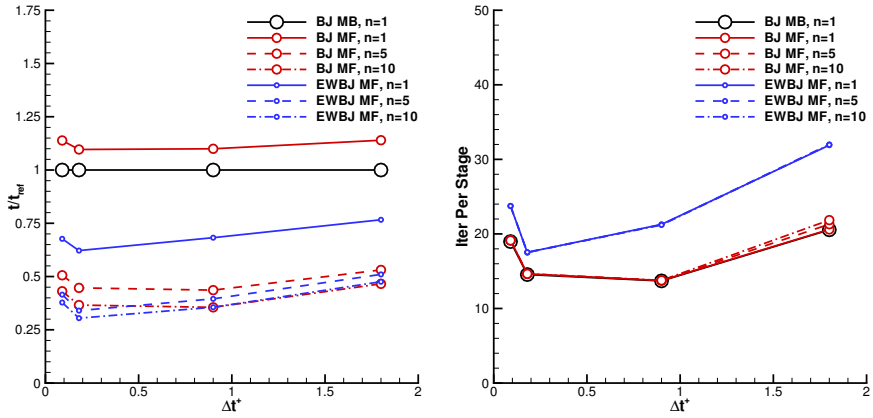


(a) $Re_\tau = 180$ (b) $Re_\tau = 395$

(c) $Re_\tau = 590$ (d) $Re_\tau = 950$

Figure 7.10: $\langle u'^{+}w'^{+}\rangle$ profiles for the turbulent channel flow at different Reynolds numbers in comparison with reference DNS data from [120] and [121]. δ is the semi-channel height.

performed within each stage, see Fig. 7.11(b). As expected, the use of the Jacobian lagging on the BJ-MB setting resulted in an unstable time integration strategy despite



(a) Speed-Up

(b) Iterations per stage

Figure 7.11: Effect of the time step size ($\delta t^+ = u_\tau^2 \delta t/\nu$ is the non-dimensional time step size) on computational performances using the adaptive GMRES tolerance of Eq. (4.7). Turbulent channel flow problem at $Re_\tau = 180$, $16\times16\times8$, $\mathbb{P}_6$ space discretization and different solution strategies.

the $\mathcal{W}$-property of the Runge–Kutta scheme employed, and therefore nothing can be reported for comparison. Lagging the preconditioner, in this case, increases only slightly the average number of GMRES iterations per stage.

As regards the memory saving the BJ-MF method reduces by the 25% the total memory footprint of the application, while the saving of the EWBJ-MF method reaches the 48%. The difference with the Rayleigh–Bénard test case can be ascribed to the number of equations employed to model the physics, *i.e.* $m = 5$ for the Rayleigh–Bénard and $m = 4$ for the channel flow, while the storage of the shape functions does not change with the number of equations m.

7.3 ERCOFTAC T3L test case suite

The section reports the implicit LES of the ERCOFTACT T3L test case suite. The problem considers the transitional flow on a flat plate with a semi-circular leading edge of diameter D. The problem is reported to be very sensitive to the free-stream turbulence intensity and Reynolds number of the computation. The experimental measurements considered six combinations of Reynolds numbers and free stream turbulence intensities, namely the T3L1 to T3L6 test cases. In this chapter report the numerical solution of each case, and considers in addition the zero-free stream case (T3L0), impossible to evaluate experimentally, as well as that obtained on the upper-half of the domain, using a symmetry boundary condition (T3L0S). Finally, the T3L5R and T3L6R cases are computed by suppressing the synthetic turbulent generator and verify the the behaviour of the solution. Therefore, a set of 10 different working conditions are reported. Table 7.4 reports details of the computational settings of the numerical simulations, and reports the bubble length computed by averaging the solution over time, which shows to be very sensitive to the Re, Tu and even the boundary conditions. In all the cases, the solution exhibits a leading edge laminar separation bubble and, downstream the transition, an attached turbulent boundary layer. Those complex flow features are perfectly suited to evaluate the efficiency of the solver and highlight the advantages of using a DG-based ILES approach. ILES naturally resolves all the flow scales (in a DNS-fashion) if the numerical resolution is enough to do so, while the numerical dissipation plays the role of a sub-grid scale model for the spatially under-resolved regions of the domain. In this test case, the laminar region is fully resolved, while in the turbulent region the dissipation of the numerical scheme dumps the under-resolved scales.

The simulations were performed on a hybrid mesh of 38320 elements with curved edges. The unstructured grid is strongly coarsened moving away from the plate, while a structured-like boundary layer is used at the wall. The first cell height is $10^{-2}D$ and the mesh is refined near the reattachment region, where the minimum dimension along x axis is $2 \cdot 10^{-2}D$, see Fig. 7.12. Using post-processed skin friction data, the value corresponds to a first cell height of $\delta y^+ = 11$ for the highest Reynolds number

Case	Tu	Re	l/D
T3L0	0.00%	3450	3.90
T3L0S	0.00%	3450	4.42
T3L1	0.20%	3450	2.69
T3L2	0.65%	3450	2.00
T3L3	2.30%	3450	1.49
T3L4	5.60%	3450	1.08
T3L5	2.30%	1725	2.74
T3L5R	0.00%	1725	6.57
T3L6	2.30%	6900	1.00
T3L6R	0.00%	6900	2.39

Table 7.4: Numerical setup of the T3L test case suite and the computed bubble length representative of the solution.

case, allowing for a spectral resolution of about $y^+ \leq 2.5$ for all the computations, which can be estimated by dividing the first cell height by $n_v^{1/3}$. The domain extension on the $x-y$ plane is taken from [124], *i.e.*, $28D \times 17D$, and it is extruded using 10 elements along the span-wise direction z for a length of $2D$, as in [125]. To the author's best knowledge, only those two works report a LES simulation of this ERCOFTAC test case. In both the cases, the numerical method was based on a standard second-order scheme and a dynamic subgrid scale model. A direct comparison of the present computations with previously published works in terms of DoFs is not trivial due to the differences of the computational domains. In [125] the DoFs count per variable is on the order of $1.88 \cdot 10^6$ and the domain extension in the $x-y$ plane is 1.9 times smaller. In [124] the Dofs count is on the order of $4.39 \cdot 10^6$ and the domain is 4 times larger in the span-wise direction. Note that the results with the lowest resolution presented in this work, which has about $1.39 \cdot 10^6$ DOFs, seems to be in a better agreement for $x/d \gtrapprox 4.5$ if compared to other literature data.

Physical discussion

This type of flow problem is reported to be very sensitive to the free-stream turbulence at the inlet (Tu). The free-stream flow was numerically manipulated to reproduce the free-stream intensities employed by ERCOFTAC for the experiments, as well as previous numerical computations [125, 124], where a white-noise random perturbation was added at the inflow velocity to mimic the low experimental turbulence level, *i.e.*, $Tu < 0.2\%$. In the present work, due to a severe mesh coarsening in the far-field regions, the generation of a free-stream turbulence at inlet is unfeasible. In fact, the coarse spatial discretization at far-field would rapidly damp any random perturbation introduced upstream. Accordingly, the turbulent fluctuations were synthetically injected, via a spatially-supported random forcing term, in those regions of the domain where the mesh density is enough not to dissipate small scales. The random forcing analytic expression assumes a Gaussian distribution in the x direction and is

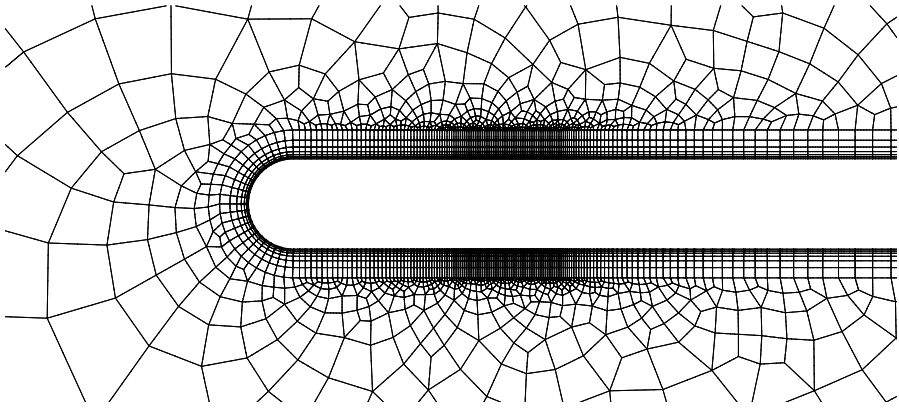


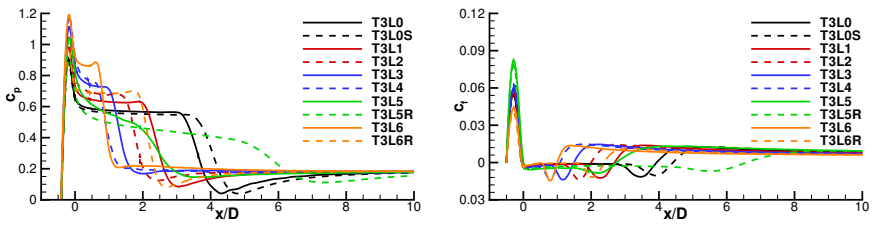
Figure 7.12: T3L1 test case. Near-wall detail of the computational grid.

homogeneous in $y-z$, such that

$$s_i = A \exp\left(-\frac{x-\overline{x}}{2\mu}\right) r_i \tag{7.5}$$

with $i = 1, ..., d$, where A, $\overline{x}$, μ and r_i are, respectively, the amplitude coefficient, the location of the forcing plane, the amplitude of the Gaussian support and a normalised random vector component such that $\sqrt{r_i r_i} = 1$. The Gaussian function was centered in $\overline{x}/D = -3$, and the constants A, μ were adjusted, via trial and error approach, to meet the experimental Tu levels. A fine control algorithm of the turbulent length-scale is here avoided, since the reattachment length is pretty insensitive to this value, see [126, 127, 128]. In the present configuration the expected turbulence intensity is met setting $\mu = 0.01$, while A was adjusted by test and trials to meet the free-stream turbulence intensity requirement near the leading edge. Note that perturbing the velocity field through a forcing term in the momentum equations guarantees a divergence-free perturbation which is consistent with the incompressible nature of the governing equations.

As reported in previous studies, the bubble length is found to be very sensitive to the inlet turbulence intensity, see for example [129]. In particular, when increasing the Tu the bubble length reduces. To this end, the pressure and skin friction coefficients are reported in Figure 7.13. The effect, which is consistent to what has been published previously, is very well captured by the numerical results reported. On the other hand, an increase in Reynolds number also promotes a reduction in the bubble length. It is worth pointing out that the T3L0S case, which one may consider equivalent to the T3L0, shows a larger recirculating region and a pretty different pressure and skin friction coefficient behaviour. Moreover, as opposed to the behaviour documented in [125], no hysteresis effects are observed in the present solutions. Accordingly, if the random source term generating small turbulent perturbations is suddenly suppressed in a fully developed flow field of one of the T3L1-6 cases, the solution of a zero free-stream turbulence case (T3L0) is quickly recovered. Similar observations can be done for the T3L5 and T3L6 cases. By setting the synthetic turbulence generation to zero, the transition mechanism changes and the reattachment length grows, see T3L5R and T3L6R cases. This observation, which happens for all the three Reynolds numbers



(a) C_p (b) C_f

Figure 7.13: T3L1 test case. Effect of the Tu level on the pressure coefficient c_p and the skin friction coefficient c_f.

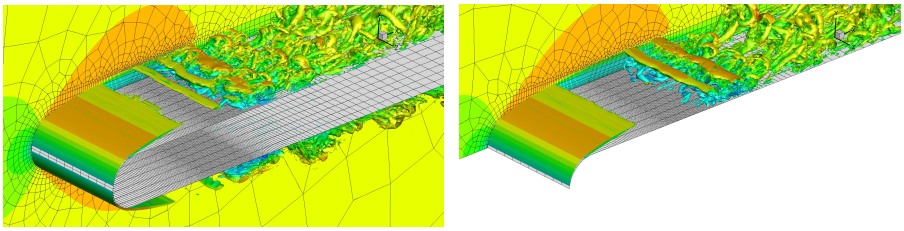
considered in the study, is in contrast to what observed by [129], where the bubble length at the largest Reynolds number is reported to be self-sustained when removing the free stream turbulence.

To corroborate the quality of the simulations, the predicted bubble length for the T3L1 case is reported to be in a better agreement with the experimental data ($l/D = 2.75$) than other numerical computations [125, 124] (2.59 and 3.00, respectively). Moreover, it has been verified by lowering the polynomial degree of the dG discretization that the statistical average l/D is almost converged with respect to the spatial resolution: for $k = 5$ and $k = 4$, the values of 2.70 and 2.73 are obtained, respectively. Convergence of the statistics is also confirmed by polynomial degree independence observed for the skin friction coefficients.

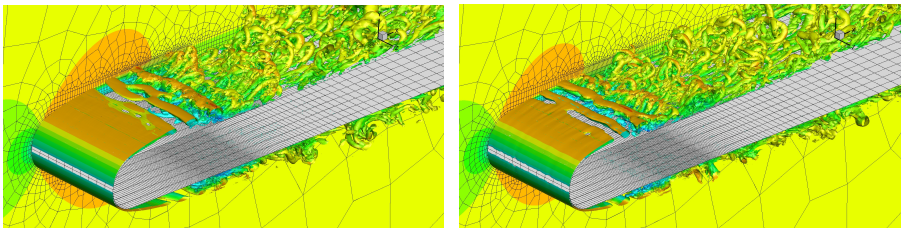
The sensitivity of the laminar separation bubble with respect to Tu and Re can also be observed in Figure 7.14, which shows the $\lambda_2 = -1$ iso-contours of the instantaneous flow field for each of the cases. The quasi two-dimensional Kelvin-Helmholtz instabilities in the shear-layer region above the separation bubble and their convection downstream are observed. As expected, low free-stream turbulence intensities promote the instability of the quasi two-dimensional structures arising from the upstream flow separation. Hairpin vortices developing after flow reattachment and the breakdown to turbulence are similar in all the cases as well. Distortion along the spanwise direction is anticipated upstream in the $Tu = 0.2\%$ case. Increasing Tu, streaky like structures stream-wise oriented are visible close to the leading edge, while for $Tu = 5.6\%$, the Kelvin-Helmholtz instability stage is bypassed anticipating the reattachment and the formation of the hairpin vortices. The development of a turbulent boundary layer is observed downstream the reattachment point. It is interesting to notice the effects of the Re. In fact, it is directly connected to the size of the smallest scales as well as the bubble length, which is affected similarly to Tu.

Figure 7.15 shows the time and span-wise averaged stream-wise velocity field. Figure 7.16 report also the kinetic energy contours for all the cases. It is worth to point out that the kinetic energy, whose maximum is located downstream the separation bubble, also varies with the Re and Tu. The presence of a free stream turbulence intensity is reflected on the contours of the kinetic energy especially for the highest values of Tu. This happens because of the generation of synthetic turbulence on an unstructured mesh, not rigorously uniform in the y direction, which produce a weak dependence of the kinetic energy with y.

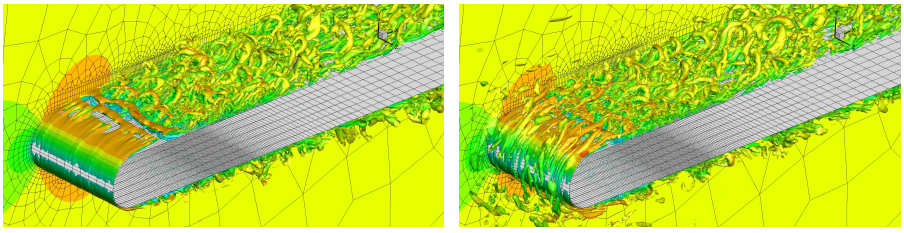
Figures 7.17 and 7.18 compare velocity profiles with the experimental ones, when available. The mean stream-wise velocity $<u>$, and the velocity fluctuation (or velocity RMS), $<u'u'>$ are reported as a function of the normal direction for different stations. Velocity is normalized by the local maximum velocity u_{max}, computed independently for each of the stations. The random forcing efficacy is demonstrated by the very good agreement with experimental data close to the plate stagnation point. The T3L0 and T3L0S cases, which have been obtained using a zero-free stream turbulence intensity, are still compared with the experimental data of the T3L1 case. With $Tu = 0\%$, the size of the separated region is larger than that observed experimentally. Just by using a very low free stream, the transition to turbulence and reattachment of the flow is anticipated and the velocity profiles tend towards the experimental one.



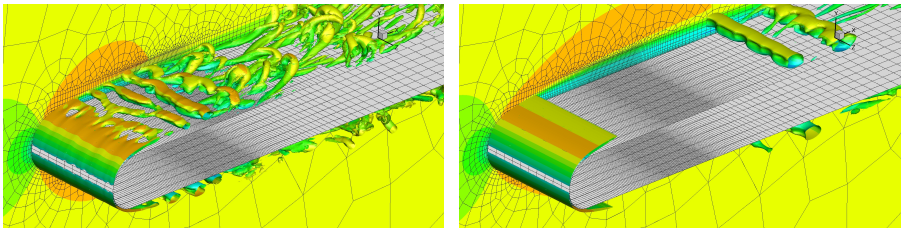
(a) T3L0 (b) T3L0S



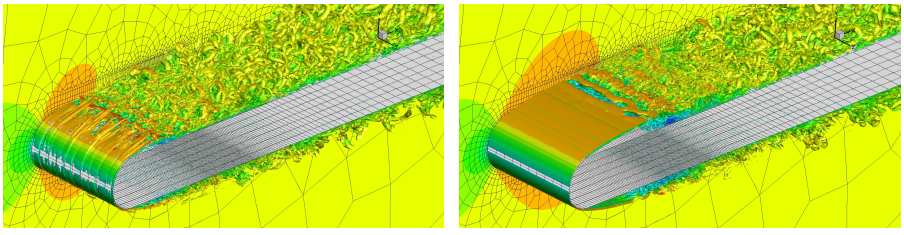
(c) T3L1 (d) T3L2



(e) T3L3 (f) T3L4

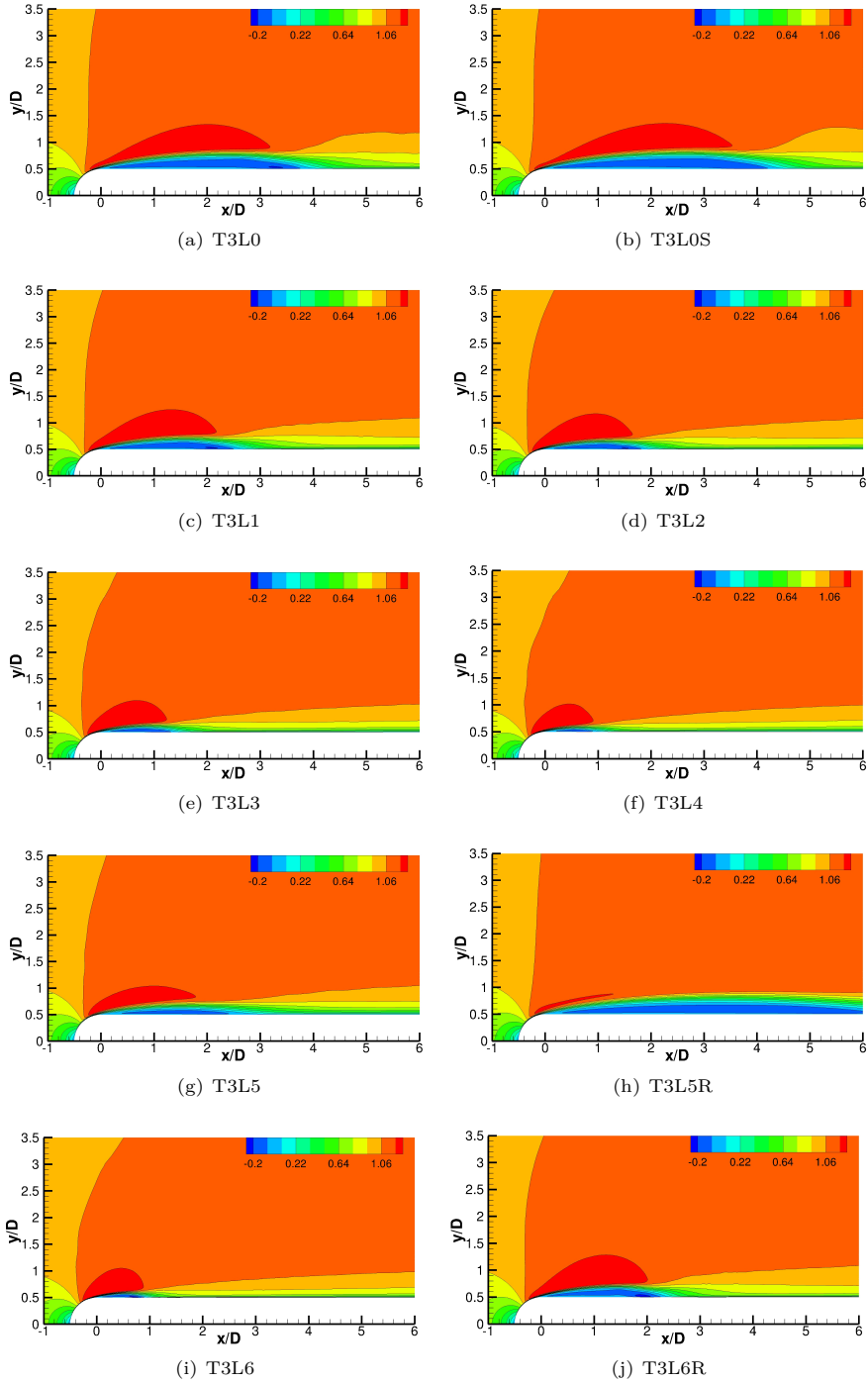


(g) T3L5 (h) T3L5R



(i) T3L6 (j) T3L6R

Figure 7.14: T3L test case, $k = 6$ solutions for different Tu levels. $\lambda_2 = -1$ iso-contour and periodic plane coloured by the streamwise velocity.



(a) T3L0 (b) T3L0S

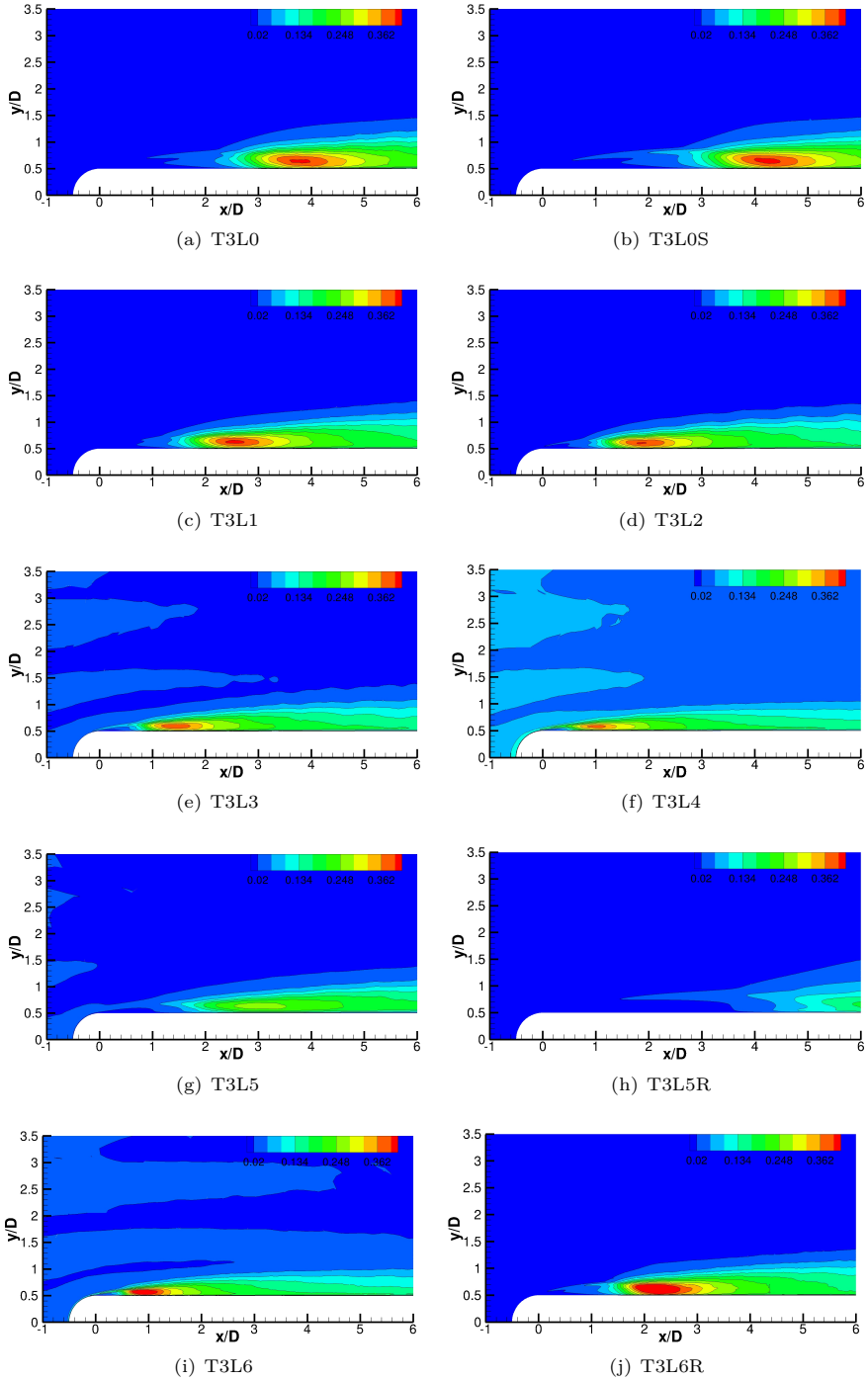
(c) T3L1 (d) T3L2

(e) T3L3 (f) T3L4

(g) T3L5 (h) T3L5R

(i) T3L6 (j) T3L6R

Figure 7.15: T3L test case. Effect of the Tu level. Average velocity magnitude isocontours, $k = 6$ solutions.



(a) T3L0 (b) T3L0S

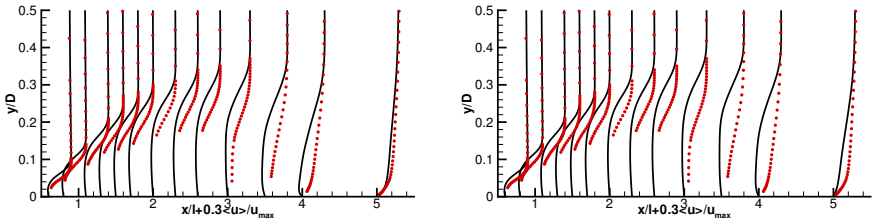
(c) T3L1 (d) T3L2

(e) T3L3 (f) T3L4

(g) T3L5 (h) T3L5R

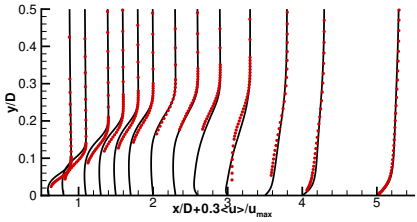
(i) T3L6 (j) T3L6R

Figure 7.16: T3L test case. Effect of the Tu level. Average velocity magnitude isocontours, $k = 6$ solutions. Top: Tu = 0.0%; Bottom: Tu = 0.2%.

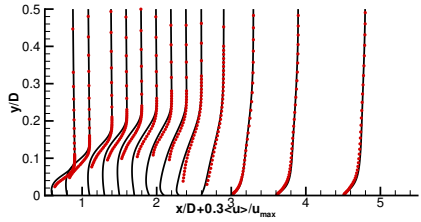


(a) T3L0 vs. Exp.(T3L1)

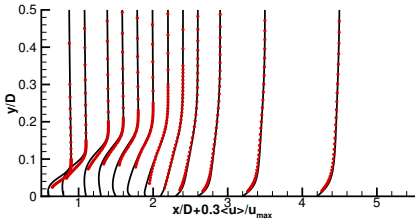
(b) T3L0S vs. Exp.(T3L1)



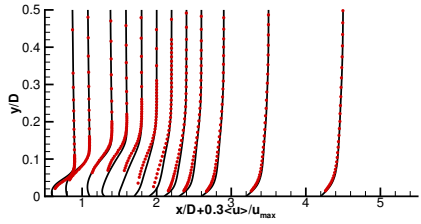
(c) T3L1



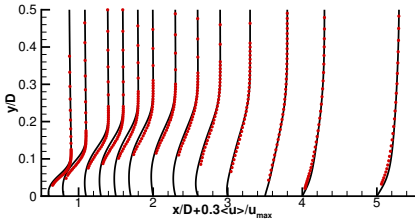
(d) T3L2



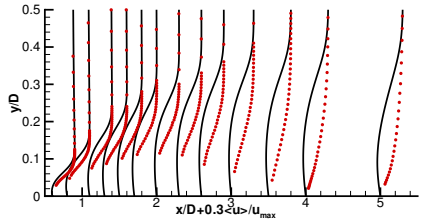
(e) T3L3



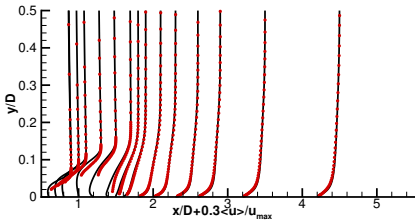
(f) T3L4



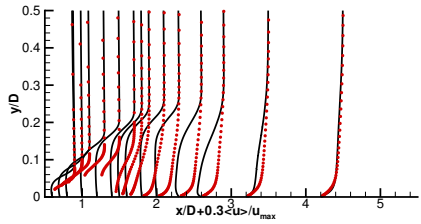
(g) T3L5



(h) T3L5R vs. Exp.(T3L5)

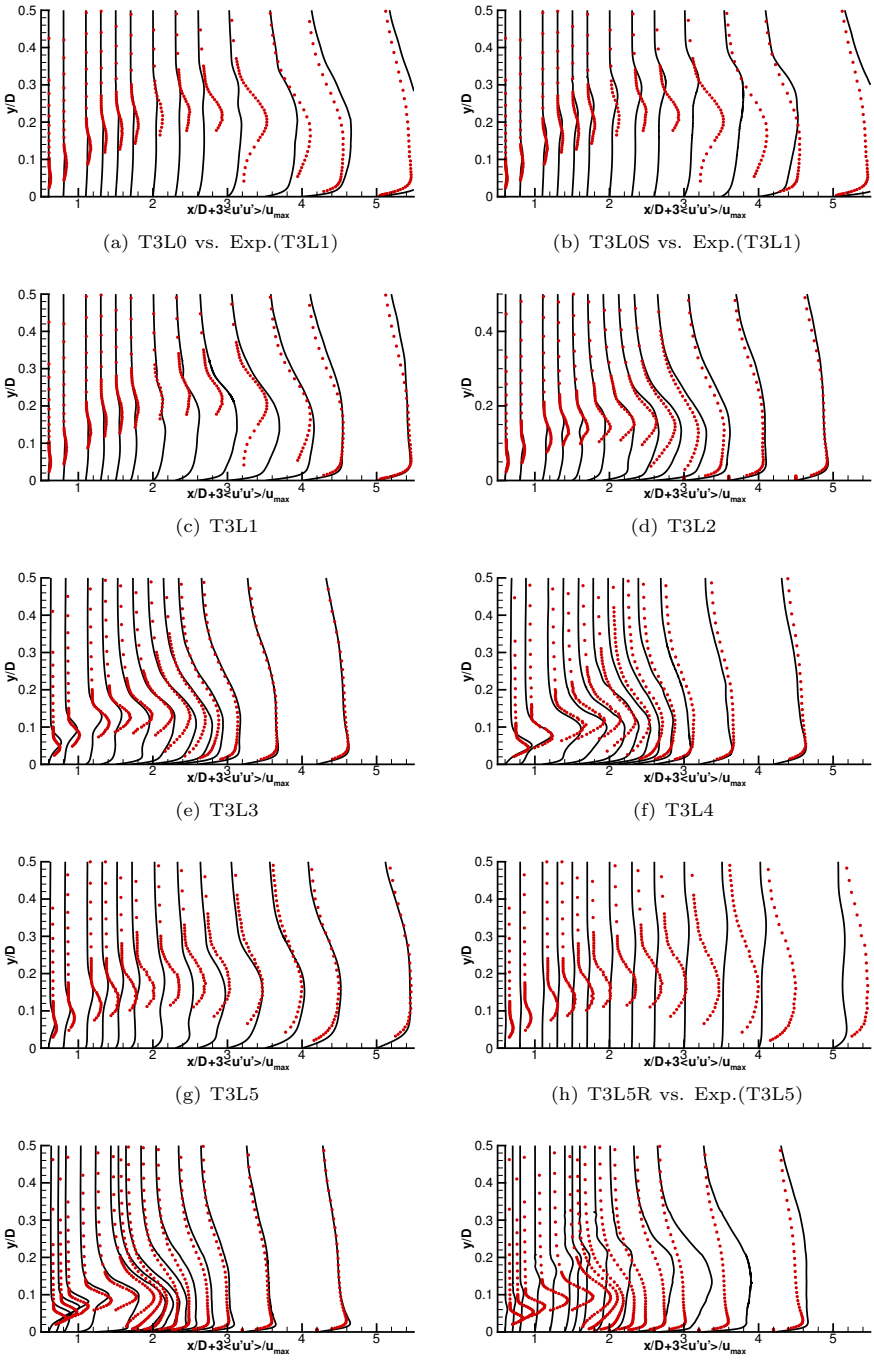


(i) T3L6



(j) T3L6R vs. Exp.(T3L6)

Figure 7.17: T3L test case suite, $k = 6$ solution. Time- and spanwise-averaged stream-wise velocity profiles (black lines) in comparison with experimental data [130] (red dots).



(a) T3L0 vs. Exp.(T3L1)

(b) T3L0S vs. Exp.(T3L1)

(c) T3L1

(d) T3L2

(e) T3L3

(f) T3L4

(g) T3L5

(h) T3L5R vs. Exp.(T3L5)

(i) T3L6

(j) T3L6R vs. Exp.(T3L6)

Figure 7.18: T3L test case suite, $k = 6$ solution. Time- and spanwise-averaged stream-wise velocity fluctuation profiles (black lines) in comparison with experimental data [130] (red dots).

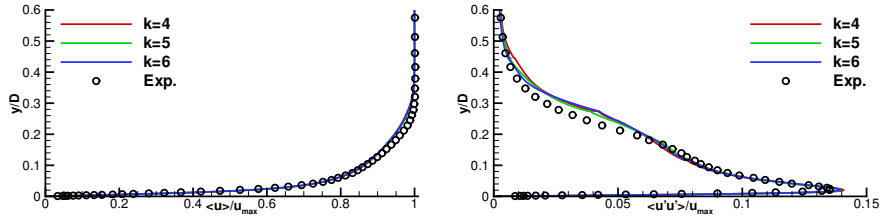


Figure 7.19: T3L1 test case, $k = 6$ solution for $Tu = 0.2\%$. Time- and spanwise-averaged velocity profiles in comparison with experimental data [130] at $x/D = 3.45$. Mean (left) and RMS (right) velocity.

The agreement with experiments on the T3L1-6 cases follows even downstream. The T3L5R and T3L6R cases, compared respectively with the T3L5 and T3L6 experimental data, corroborate the difference on the transition mechanism which is clearly not self-sustained when removing the free stream disturbances.

A comparison with other numerical computations published in the literature is very difficult. In fact, only numerical results of the T3L1 and T3L4 cases are reported in the literature with a different numerical framework and turbulence modelling strategy. As for the T3L1 case [124], it is difficult to observe improvements with respect to previous computational investigations for $x/D < 4.5$. On the other hand, for $x/D > 4.5$ the results obtained here still compare favourably with the experimental data, while the matching is less evident in [124]. The author stresses that the velocity fluctuations compare favourably with the experiments up to $10D$, and the comparison with this section was omitted in previous works. To this end, Figure 7.19 report the solution of the T3L1 test case for $x/D = 3.45$ computed with various polynomial orders, $k = 4, 5, 6$. Both the average velocity profile and RMS compare very well with experiments, which supports the claim that present computations provide a larger fully resolved region. Note that some jumps at inter-element boundaries are still noticeable, especially for $k = 4$. Fig. 7.20 reports the computed averaged velocity profiles in wall units, for different stations located downstream to the reattachment region, of the T3L1 test case. For $x/D \geq 9.5$ the profile approaches the turbulent law of the wall, showing some discrepancies with respect to the equilibrium boundary layer in the outer layer. For the sake of comparison, the zero pressure gradient flat plate DNS result at $\mathrm{Re}_\theta = 300$ of Spalart [131] is reported together with the numerical solution at $x/D = 12.5$, which shows almost the same Re_θ. Note that the station at $x/D = 9.5$, which results in $\mathrm{Re}_\theta \approx 270$, compares favourably with the experimental data.

It is worth noticing that, for the other cases, a reference value in terms of bubble length is not provided in the experimental database. However, the agreement of the velocity profile observed in Figures 7.17 and 7.18 suggest a favourable comparison. In particular, the reattachment length of the T3L4 case at $Tu = 5.6\%$ agrees well with that measured in [132] for the same geometry and a slightly different flow configuration, *i.e.* $Tu = 7\%$ and $Re = 3300$. Note that that the numerical value $l/D = 1.8$ evaluated in [124] appears to be significantly larger.

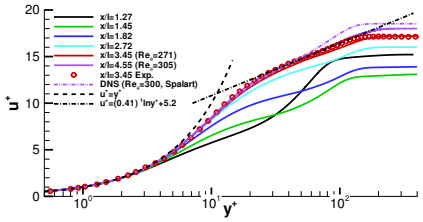


Figure 7.20: T3L1 test case, $k = 6$ solution for $Tu = 0.2\%$. Non dimensional streamwise velocity profile for different values of x/D after reattachment in comparison to the theoretical law of the wall and DNS data [131].

Performance assessment

Table 7.5 reports the computational performances of the different solution strategies obtained for a dG approximation with $k = 6$, the same polynomial degree of reference employed in previous sections. The time step size of the third-order accurate linearly-implicit Rosenbrock-type time integration scheme was 16 and 8 times larger than those used in [124] and [125], respectively. The relative defect tolerance for the linear solver reads $\text{rTol} = 10^{-5}$.

The GMRES(MB)[BJ] is considered as the reference solution strategy. Being the number of curved elements small when compared to the number of affine elements, the degrees of exactness of quadrature rules disregards the second degree geometrical representation of cells close to leading edge. Since the boundary layer is still laminar at the leading edge, this under-integration does not affect stability of the scheme.

Switching from the matrix-based (GMRES(MB)[BJ]) to the matrix-free solver (GMRES(MF)[BJ]), one can observe the same computational efficiency but 40% less memory usage. Note that the use of Additive Schwarz preconditioned GMRES (GMRES[ASM(1,ILU(0))]) decreases the overall parallel efficiency of the method. In fact, the CPU time increases by the 11% and the memory requirements raises by 60%.

Solver Prec	GMRES(MB) BJ	GMRES(MB) ASM(1,ILU(0))	GMRES(MF) BJ
CPU Ratio	1	1.11	0.95
Memory Ratio	1	1.6	0.6
ITs	115	72	115
Solver Prec	**FGMRES(MB) MG_{full}**	**FGMRES(MF) MG_{full}**	**FGMRES(MF) MG_{full} (LAG=3)**
CPU Ratio	0.50	0.47	0.31
Memory Ratio	0.65	0.15	0.15
ITs	3.0	3.0	3.31

Table 7.5: Performance comparison of the solver on the T3L1 test case. Computational time, total memory footprint non-dimensionalized with the GMRES(MB)[BJ] solver, and average number of GMRES iterations per time step, for the BJ, ASM(1,ILU(0)) and MG_{full} preconditioners (see text for settings details). Results obtained on 540 Intel Xeon CPUs of Marconi-A1@CINECA.

The p-multigrid precondioned solver specs are as follows: FGMRES(MB or MF) as outer solver, a full p-multigrid interation with $L = 3$, $k_\ell = 6, 2, 1$, GMRES(MB or MF)[EWBJ] smoother for $\ell = 0$ (8 iterations), GMRES(MB)[EWBJ] smoother for $\ell = 1$ (8 iterations) and GMRES(MB)[ASM(1,ILU(0))] smoother on the coarsest level ($\ell = 2$, 40 iterations). Note that for the finest level outer solver and smoother both matrix-based and matrix-free implementations are considered for the sake of comparison. The computational efficiency of the method improves considerably with respect to the reference (first column of Tab. 7.5): i) in a matrix-based framework a 50% decrease of the CPU time is observed and the memory requirements reduce by 35%, ii) in a matrix-free framework the CPU times gains are unaltered and the memory savings reach 85%. Such significant memory footprint reductions are mainly due the finest level strategy: only a block diagonal matrix is allocated and a small number of Krylov subspaces is employed for the GMRES algorithm. In this case the actual memory footprint has been computed through the PETSc library, and it is in line with the values that can be estimated a-priori using the model of Section 5.1.5.

As a further optimization of the matrix-free approach the possibility to lag the computation of the system matrix employed for preconditioning purposes is considered, this means that the preconditioner is "freezed" for several time steps. Clearly this strategy reduces assembly times but degrades convergence rates if the discrepancy between the matrix and the preconditioner gets too severe. In the present computation optimal performance are achieved by lagging the operators evaluation for 3 time steps. By doing so, the CPU time is further reduced to the 0.31 of the baseline, see Tab. 7.5, which corresponds to a speed-up of 3.22. As a side effect, the average number of GMRES iterations slightly increases from 3.0 to 3.31.

7.4 Boeing Rudimentary Landing Gear test case

The final validation case reported in this work deals with the preliminary result of the implicit LES of the incompressible flow around the Boeing Rudimentary Landing Gear (RLG). The purpose of the test is to demonstrate the applicability of the solution strategies proposed in this work within an industrially relevant test case.

The RLG was designed by Spalart *et al.* [133], and experimentally studied in [134], to become a benchmark for testing turbulence modelling approaches. The test case was also included within the test case suite of the ATAAC EU-funded project [135]. The flow conditions involve a Reynolds number of 10^6, based on the freestream velocity $V_\infty = 40\ m/s$ and on the wheel diameter $D = 0.406\ m$. The Mach number of the experiments was $M = 0.12$, which resemble an incompressible flow problem. It is worth mentioning the test case was aimed to assess hybrid RANS/LES approaches on complex flow configurations. For this reason, in previous computations special care is spent to control the turbulent transition location allowing a better agreement with experimental data. While the structural elements are made rectangular, and thus fixing the separation location on the edge, the boundary layers on the wheels are tripped, see [133]. The structure of the unsteady flow around the landing gear is mainly characterized by large separated and recirculating regions on the wheels and axles, as well as by the front-rear wheel interaction, which make the use of unsteady scale-resolving simulation mandatory.

The mesh for the computation has been provided by the German Aerospace Center (DLR) within the EU-funded project "Towards Industrial LES/DNS in Aeronautics – Paving the Way for Future Accurate CFD" (TILDA). The grid is delimited by four symmetry planes, one inflow, one outflow, and the landing gear wall surface. A snapshot of the grid showing the wall surface (red), the symmetry plane discretization (black) and an internal slice (blue) is reported in Figure 7.21. The mesh employed

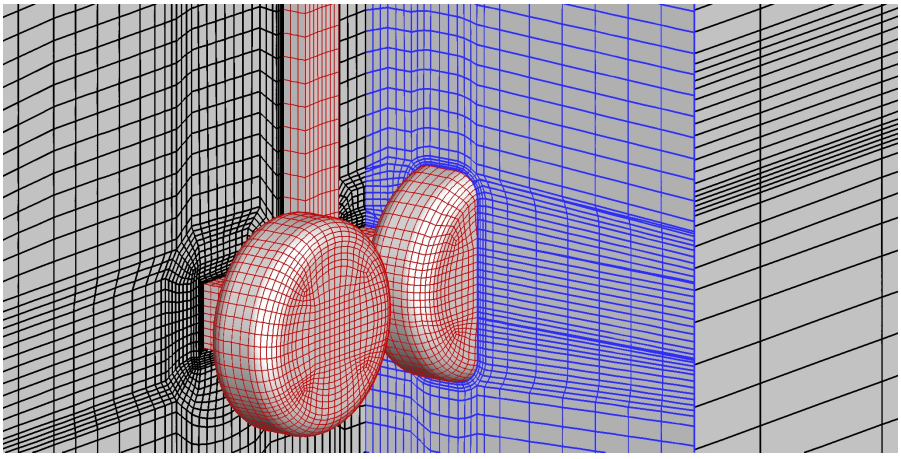


Figure 7.21: Boeing Rudimentary Landing Gear test case. Details of the multiblock structured grid provided by DLR under the ATAAC and TILDA European projects.

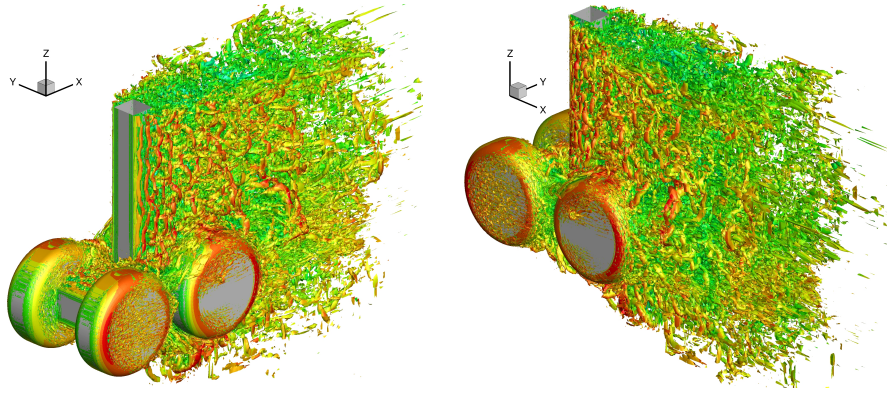


Figure 7.22: RLG test case at $Re = 10^6$. Incompressible flow solution using $k = 4$ polynomials. λ_2 iso-contour coloured by stream-wise velocity magnitude. Front view (left) and rear view (right).

takes advantage of the symmetry of the problem and it discretises only the half of the domain. The grid was made by $115 \cdot 10^3$ hexa elements with second-order geometrical representation obtained through agglomeration of a finer discretization, and it shows a severe wall refinement to accommodate a suitable wall resolution given the high Reynolds number of the case. The first cell height is $\delta y = 6.054 \cdot 10^{-5} D$, which provides an equivalent wall normal resolution of $1.851 \cdot 10^{-5}$, obtained by dividing for $(n_v)^{1/3}$. Exploiting the maximum value of the skin friction coefficient over the entire wall boundary obtained during the post-processing phase, the grid allows a maximum wall normal resolution of $y^+ = 2.8$.

The solution has been obtained by using $\mathbb{P}_4$ polynomials, providing a total of $4.025 \cdot 10^6$ degrees of freedom, while the four-stage, order-three ROSI2PW scheme was employed for time integration. Using as reference quantities the free stream velocity and the wheel diameter, the non-dimensional time step size was $\Delta t = 0.001$. To compute the average fields, the solution was advanced in time for roughly 50 convective times, some of them performed using a lower-order space discretization. The average lasted roughly $T = 23$ convective times, which may be not enough to have converged first-order statistics. However, it has been verified that the average quantities did not change sensibly from $T > 17.5$. Linear systems arising from the time integration were solved using FGMRES preconditioned with a p-multigrid strategy with similar settings to that of Table 5.8, employing an additive Schwarz preconditioned on the coarsest level of the multigrid iteration, and an element-wise block Jacobi method on the other levels to maximise the scalability of the algorithm. Despite working with an average of 19 elements per partition in such a complex test case, the iterative solver converged using an average of 3.5 iterations per stage, confirming the efficacy of this preconditioning approach.

Figure 7.22 shows the features of the flow field through the istantaneous λ_2 iso-contour plot coloured by the stream-wise velocity magnitude. The interaction between the separated flow downstream the front wheel and rear wheel is clearly visible, while the details of the harpin vortices generated on the side of the front wheels are also

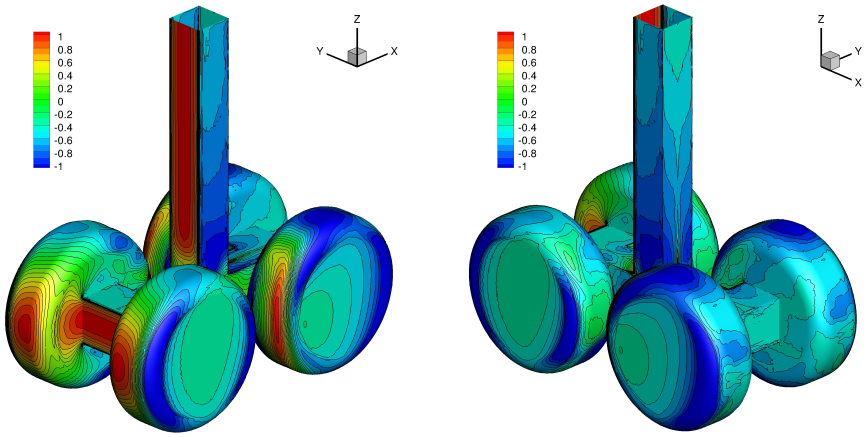


Figure 7.23: RLG test case at $Re = 10^6$. Incompressible flow solution using $k = 4$ polynomials. Mean pressure coefficient C_p contours on the wall surface. Front view (left) and rear view (right).

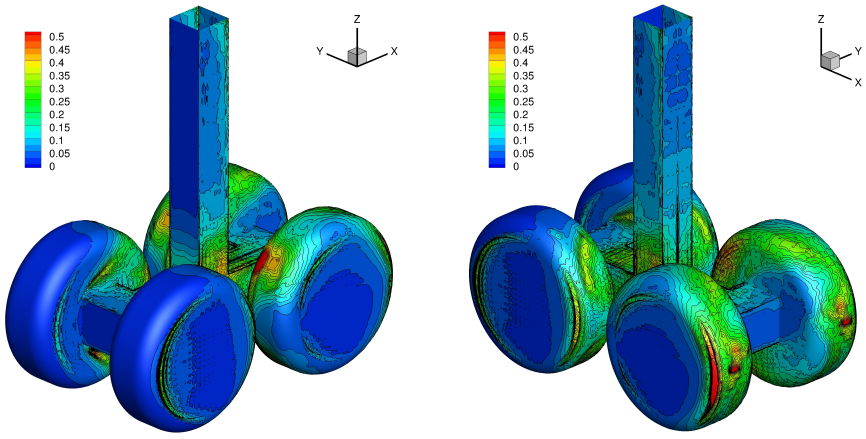


Figure 7.24: RLG test case at $Re = 10^6$. Incompressible flow solution using $k = 4$ polynomials. Pressure coefficient RMS, C_p^{RMS} contours on the wall surface. Front view (left) and rear view (right).

clearly resolved. It is worth noticing that the shape of the iso-contour suggest that the flow is mainly attached to the wheels, although a very small laminar separation similar to that of the T3L test case can be observed on the fore side of the wheel.

The averaged fields in terms of pressure coefficient C_p, the root mean square value of the pressure coefficient C_p^{RMS} on the landing gear have are reported in Figure 7.23, 7.24 and 7.25. A qualitative agreement with the surface plots reported in [133, 136], obtained through a hybrid RANS/LES approach, can be observed especially as regards the front views. On the other hand, the rear view highlight the presence of oscillations that might be symptom of an averaging period not long enough, as well as a too coarse space resolution. Those oscillations are even more evident if first order

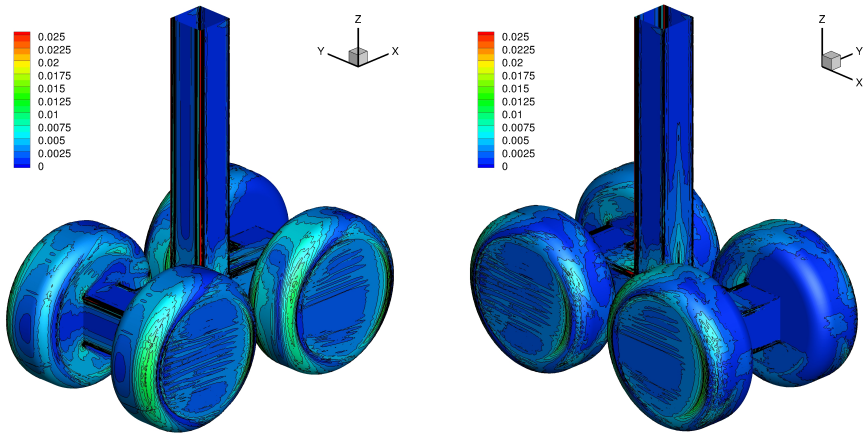


Figure 7.25: RLG test case at $Re = 10^6$. Incompressible flow solution using $k = 4$ polynomials. Mean skin friction coefficient C_f contours on the wall surface. Front view (left) and rear view (right).

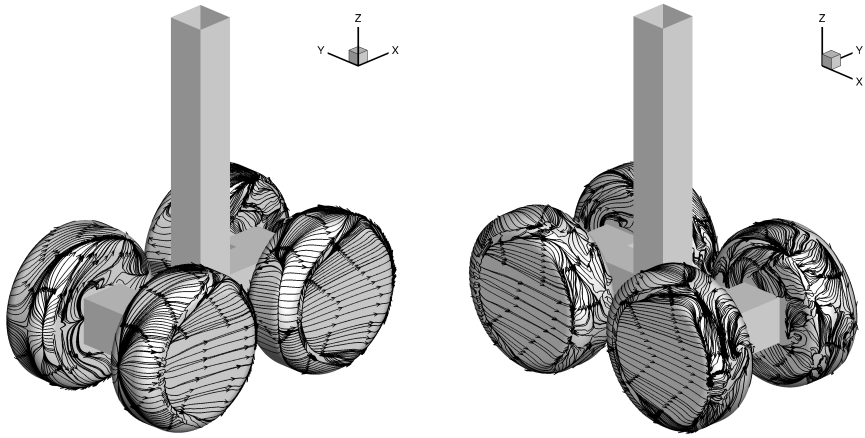
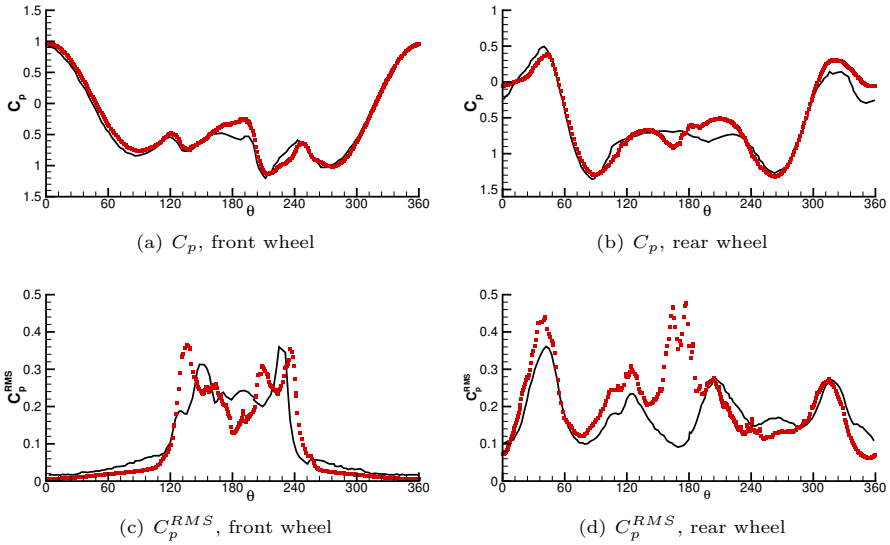


Figure 7.26: RLG test case at $Re = 10^6$. Incompressible flow solution using $k = 4$ polynomials. Average wall streamline path. Front view (left) and rear view (right).

statistics (*e.g.* C_p^{RMS}) or the skin friction coefficient (which involve state derivatives) are considered. Figure 7.25 reports the surface plot of the skin friction coefficient $C_f = 2\tau_w/\rho V_\infty^2$. As reported for the pressure coefficient fluctuation, the plot shows oscillations particularly at the inter-element connection, which is pretty typical for under-resolved simulations using discontinuous Galerkin finite element methods. However, the plots look qualitatively similar to those reported in previous numerical simulations [133, 136].

Figure 7.26 show the streamline patterns on the wheels. As described in [136], the patterns show the bifurcation line of separation and reattachment on the front and rear wheels. However, the simulation shows on both sides of the wheels a region with



(a) C_p, front wheel

(b) C_p, rear wheel

(c) C_p^{RMS}, front wheel

(d) C_p^{RMS}, rear wheel

Figure 7.27: RLG test case at $Re = 10^6$. Incompressible flow solution using $k = 4$ polynomials. Average pressure coefficient C_p and RMS C_p^{RMS} distribution on the mid-line of the fore and back wheels versus the azimuthal angle θ.

separated flow and reversed streamline patterns. Such a region seems to be different to that reported by the experiments [134] as well as previous numerical simulations. It is worth to point out that no transition tripping was employed in the current simulation, differently to what has been done with experiments and numerical simulations based on RANS modelling.

Figure 7.27 reports the average pressure coefficient C_p and its root mean square value C_p^{RMS} on the mid-line of the wheels, versus the azimuthal angle θ, in comparison with experimental data from NAL [134]. While the pressure coefficient compare favourably with experimental data, its root mean square value shows a more oscillatory behaviour, originating from low spatial resolution and possibly a too short averaging time. However, the locations of the peaks of fluctuation as well as its value is pretty well captured.

While more refined solutions of this test case are currently under investigation by increasing the order of polynomial approximation to $k = 6$ and reducing the time step size to show time convergence of the results, it is worth to remark that the present computation has been performed on roughly $6 \cdot 10^3$ cores of the Marconi A1 cluster hosted by CINECA, using a total wall clock time of 94 hours. The average wall time per convective time unit was roughly 4 hours.

Chapter 8

Conclusion and outlook

8.1 Main achievements

The work presents advances in computational efficiency of high order discontinuous Galerkin methods applied to unsteady flow simulations targeting large, three dimensional computations performed in HPC facilities. It specifically addresses two main topics: the first one deals with parallelization strategies in the context of explicit time integration, while the second one considers scalable algorithms to solve linear systems arising from the implicit time discretization of the equations.

To improve the parallel efficiency, Chapter 3 considers novel OpenMP parallelization strategies have been presented. Three different algorithms, required to avoid data race conditions of OpenMP, have been proposed and compared with each other. The numerical experiments reveal that a colouring algorithm excels for large computations, while a partition algorithm outperforms the others at low computational loads. To avoid dealing with two nested domain decompositions, the colouring algorithm proved also to be a simple yet effective choice to improve the efficiency of existing MPI solvers within a hybrid MPI/OpenMP parallelization framework. The performance of the hybrid implementation has been tested on both EE/LEE and compressible Navier–Stokes problems, on different HPC facilities, demonstrating better efficiencies than the pure MPI solver.

In the context of implicit time integration, the work mainly focuses on the introduction of scalable memory saving strategies to solve linear systems. This problem is tackled through the use of a matrix-free implementation of the GMRES linear solver, which is described in Chapter 4. Matrix-free methods replace the matrix-vector product via its first order finite difference approximation, which allows to use the Jacobian matrix only for the purpose of evaluating the preconditioner. This fact increases flexibility of the solver, since the preconditioner can be obtained cheaply by approximating the iteration matrix. The numerical experiments, performed both on a model two-dimensional case and on three-dimensional turbulent test cases, reveal a clear trend. In fact, when the numerical complexity is low, *i.e.* the problem is two-dimensional and it is discretized using a low polynomial order, the computational efficiency is in favour of the MB method which is more efficient. On the other hand, when the complexity grows, the flexibility of the MF solver can be conveniently exploited to increase the computational efficiency with respect to the MB case. This situation is particularly evident in three-dimensional cases where the lagged EWBJ-MF setting allows to reduce the CPU time up to the 80% in realistic configurations, allocating only the 7% of the memory for the time integration. See, to this end, experiments in

Chapter 7, Section 7.1 and 7.2.

For stiffer space discretizations, it has been demonstrated that the EWBJ preconditioner is not enough computationally efficient, see Chapter 5. A extension of the memory saving approach has been presented, and it is based on a p-multigrid preconditioner strategy. The algorithm relies on a matrix-free implementation of both the outer FGMRES solver as well as the finest level smoother, while matrix-based GMRES smoothers are employed on coarse levels. Coarse operators of lower polynomial degree are built using a rescaled-inherited approach which provides optimal scaling of the stabilization terms and improves p-multigrid preconditioner's efficiency and robustness. The performance of the algorithm has been evaluated on test cases of growing complexity. The work demonstrates that the p-multigrid preconditioned solver outperforms standard single-grid preconditioned iterative solvers from a CPU time viewpoint. As proof of consept, the strategy is applied to the ILES of the incompressible turbulent flow over a rounded-leading edge plate with different free-stream turbulent intensities, *i.e.* the ERCOFTAC T3L test case suite. The solver strategy is profiled and compared with state-of-the-art single-grid solvers running on large HPC facilities. If a block-diagonal preconditioner is employed on the finest level, the algorithm reduces the memory footprint of the solver of about 92% of the standard matrix-based implementation. On top the memory savings, the p-multigrid preconditioned FGMRES solver is also three times faster than the best performing single-grid solver.

Chapter 6 reports on the comparison between the implicit matrix-free strategies proposed for DG versus more recent high-order solution techniques for implicit problems such HDG. In particular, HDG is presented in both the mixed and primal form. While the former discretization is commonly adopted in the literature, the comparison with the latter consists of an element of novelty in itself. The comparison employs the same preconditioning techniques and, to this end, a numerical framework to use a p-multigrid preconditioner in the context of HDG has been proposed for the first time. The approach is based on approximate-inherited projection of the statically condensed matrix on the finest space. The results on two-dimensional test cases involving the solution of compressible Navier–Stokes equations reveal that, for considerably stiff problems involving a large number of GMRES iterations, the use of HDG allows to reduce the computational time of the simulations. However, if smaller time step sizes are employed, the costs of the static condensation and back-solve are too large to make the method convenient from the computational time point of view with respect, for example, to the best performing matrix-free p-multigrid approach with Jacobian lagging. More specifically, within HDG, the use of the proposed p-multigrid method to precondition the iterative solver is effective in reducing the number of iterations, but the costs of the static condensation and back-solve, still present, do not allow to observe benefits in CPU time.

As proof of concept, high order accurate implicit LES of four incompressible turbulent flow problems are reported, *i.e.* the Rayleigh-Benard natural convection problem up to $Ra = 10^8$, the turbulent channel flow up to $Re_\tau = 590$, the complete ERCOFTAC T3L test case suite for different levels of free stream turbulence and Reynolds, as well as the Boeing rudimentary landing gear test case of ATAAC and TILDA

EU-projects at $Re = 10^6$. Results in good agreement with experiments and previous numerical simulations show the relevance of the numerical techniques proposed to deal with test cases of industrial complexity.

8.2 Outlook and future work

Despite the improvements in computational efficiency demonstrated in this thesis, several critical points still remain to be solved. In fact, a drawback that has been highlighted within the text is the performance degradation of the matrix-free GMRES solver by increasing the geometrical order of representation of the solution, since the increased cost of the residual evaluation make it more expensive than performing a matrix-vector product. To further improve this point, the implementation of an adaptive strategy for the choice of the quadrature degree of exactness, which can be adapted in view of the actual amount of curvature of mesh faces and elements, can target the optimization of the residual evaluation and thus the overall performance of the matrix-free algorithm. In addition, the implementation of vectorization capabilities to optimize further the residuals computation on modern HPC architectures can provide large gains on modern computer architectures. The use of OpenMP to exploit hyperthreading for the residual evaluation in a matrix-free context can also be considered to improve the computational efficiency.

It is worth pointing out that two main research areas have to be considered for further improvements in computational efficiency of the proposed solver. First, the development of cheap and accurate sensors to drive run-time p-adaptation strategies, *i.e.* the variation of the polynomial degree of the solution between the elements, which allows to obtain a reduction of the simulation CPU time and memory while not spoiling the accuracy required by this class of simulations. See, for example, preliminary studies on adaptive methodologies reported in [137, 138, 139]. Second, the development numerical methods to reduce the overall the computational costs via, for example, wall-modelled LES for simulations performed at the highest Reynolds numbers.

Bibliography

[1] F. Bassi, A. Crivellini, D. Di Pietro, S. Rebay, An implicit high-order discontinuous Galerkin method for steady and unsteady incompressible flows, Computers & Fluids 36 (10) (2007) 1529–1546.

[2] A. Crivellini, V. D'Alessandro, F. Bassi, A Spalart-Allmaras turbulence model implementation in a discontinuous Galerkin solver for incompressible flows, Journal of Computational Physics 241 (2013) 388 – 415.

[3] A. Crivellini, V. D'Alessandro, F. Bassi, High-order discontinuous Galerkin solutions of three-dimensional incompressible RANS equations, Computers & Fluids 81 (2013) 122 – 133.

[4] F. Bassi, A. Crivellini, S. Rebay, M. Savini, Discontinuous Galerkin solution of the Reynolds averaged Navier-Stokes and k - ω turbulence model equations, Computers & Fluids (34) (2005) 507–540.

[5] F. Bassi, A. Crivellini, D. A. Di Pietro, S. Rebay, A high-order discontinuous Galerkin solver for 3D aerodynamic turbulent flows, in: Proceedings of the European Conference on Computational Fluid Dynamics, 2006.

[6] J.-B. Chapelier, M. De La Llave Plata, F. Renac, E. Lamballais, Evaluation of a high-order discontinuous Galerkin method for the dns of turbulent flows, Computers & Fluids 95 (2014) 210–226.

[7] C. C. Wiart, K. Hillewaert, L. Bricteux, G. Winckelmans, Implicit les of free and wall-bounded turbulent flows based on the discontinuous galerkin/symmetric interior penalty method, International Journal for Numerical Methods in Fluids 78 (6) (2015) 335–354.

[8] A. Uranga, P.-O. Persson, M. Drela, J. Peraire, Implicit large eddy simulation of transition to turbulence at low reynolds numbers using a discontinuous galerkin method, International Journal for Numerical Methods in Engineering 87 (1-5) (2011) 232–261.

[9] G. J. Gassner, A. D. Beck, On the accuracy of high-order discretizations for underresolved turbulence simulations, Theoretical and Computational Fluid Dynamics 27 (3-4) (2013) 221–237.

[10] A. D. Beck, T. Bolemann, D. Flad, H. Frank, G. J. Gassner, F. Hindenlang, C.-D. Munz, High-order discontinuous galerkin spectral element methods for transitional and turbulent flow simulations, International Journal for Numerical Methods in Fluids 76 (8) (2014) 522–548.

[11] F. Bassi, L. Botti, A. Colombo, A. Crivellini, A. Ghidoni, 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 55 (2016) 367–379.

[12] A. Abba, L. Bonaventura, M. Nini, M. Restelli, Dynamic models for large eddy simulation of compressible flows with a high order dg method, Computers & Fluids 122 (2015) 209–222.

[13] R. Moura, S. Sherwin, J. Peiró, Linear dispersion–diffusion analysis and its application to under-resolved turbulence simulations using discontinuous galerkin spectral/hp methods, Journal of Computational Physics 298 (2015) 695–710.

[14] T. Sayadi, P. Moin, Large eddy simulation of controlled transition to turbulence, Physics of Fluids 24 (11) (2012) 114103.

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

[16] F. Bassi, L. Botti, A. Colombo, A. Ghidoni, F. Massa, Linearly implicit rosenbrock-type runge–kutta schemes applied to the discontinuous galerkin solution of compressible and incompressible unsteady flows, Computers & Fluids 118 (2015) 305–320.

[17] V. John, G. Matthies, J. Rang, A comparison of time-discretization/linearization approaches for the incompressible navier–stokes equations, Computer Methods in Applied Mechanics and Engineering 195 (44) (2006) 5995–6010.

[18] X. Liu, Y. Xia, H. Luo, L. Xuan, A comparative study of rosenbrock-type and implicit runge–kutta time integration for discontinuous galerkin method for unsteady 3d compressible navier-stokes equations, Communications in Computational Physics 20 (4) (2016) 1016–1044.

[19] J. Gottlieb, C. Groth, Assessment of riemann solvers for unsteady one-dimensional inviscid flows of perfect gases, Journal of Computational Physics 78 (2) (1988) 437–458.

[20] D. Hänel, R. Schwane, G. Seider, On the accuracy of upwind schemes for the solution of the navier-stokes equations, in: 8th Computational Fluid Dynamics Conference, 1987, p. 1105.

[21] P. L. Roe, Approximate riemann solvers, parameter vectors, and difference schemes, Journal of computational Physics 135 (2) (1997) 250–258.

[22] F. Bassi, A. Crivellini, D. A. Di Pietro, S. Rebay, An artificial compressibility flux for the discontinuous Galerkin solution of the incompressible Navier-Stokes equations, Journal of Computational Physics 218 (2) (2006) 794–815.

[23] F. Bassi, S. Rebay, A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations, J. Comput. Phys. 131 (1997) 267–279.

[24] F. Brezzi, G. Manzini, D. Marini, P. Pietra, A. Russo, Discontinuous Galerkin approximations for elliptic problems, Numer. Methods Partial Differential Equations 16 (2000) 365–378.

[25] D. N. Arnold, F. Brezzi, B. Cockburn, L. D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, SIAM J. Numer. Anal. 39 (5) (2002) 1749–1779.

[26] A. Crivellini, F. Bassi, A three–dimensional parallel discontinuous Galerkin solver for acoustic propagation studies, International Journal of Aeroacoustics 2 (2) (2003) 157–173.

[27] R. Cools, An Encyclopædia of Cubature Formulas, J. Complexity 19 (2003) 445–453.

[28] C. A. Kennedy, M. H. Carpenter, R. M. Lewis, Low–storage, explicit Runge-Kutta schemes for the compressible Navier–Stokes equations, Applied Numerical Mathematics 35 (3) (2000) 177 – 219.

[29] J. Lang, J. Verwer, ROS3P—An accurate third-order Rosenbrock solver designed for parabolic problems, BIT 41 (4) (2001) 731–738.

[30] C. A. Kennedy, M. H. Carpenter, Additive runge-kutta schemes for convection-diffusion-reaction equations, Applied Numerical Mathematics 1 (44) (2003) 139–181.

[31] C. C. de Wiart, K. Hillewaert, Development and validation of a massively parallel high-order solver for DNS and LES of industrial flows, in: IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach, Springer, 2015, pp. 251–292.

[32] F. Renac, M. Llave Plata, E. Martin, J. B. Chapelier, V. Couaillier, IDIHOM: Industrialization of High-Order Methods - A Top-Down Approach: Results of a Collaborative Research Project Funded by the European Union, 2010 - 2014, Springer International Publishing, Cham, 2015, Ch. Aghora: A High-Order DG Solver for Turbulent Flow Simulations, pp. 315–335.

[33] S. R. Brus, D. Wirasaet, J. J. Westerink, C. Dawson, Performance and scalability improvements for Discontinuous Galerkin solutions to conservation laws on unstructured grids, Journal of Scientific Computing 70 (1) (2017) 210–242.

[34] R. D. Nair, H. W. Choi, H. M. Tufo, Computational aspects of a scalable high-order discontinuous Galerkin atmospheric dynamical core, Computers & Fluids 38 (2) (2009) 309 – 319.

[35] B. Reuter, V. Aizinger, Köstler, A multi-platform scaling study for an OpenMP parallelization of a discontinuous Galerkin ocean model, Computers & Fluids 117 (2015) 325 – 335.

[36] S. Dong, G. E. Karniadakis, Dual-level parallelism for high-order CFD methods, Parallel Computing 30 (1) (2004) 1 – 20.

[37] M. J. Chorley, D. W. Walker, Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters, Journal of Computational Science 1 (3) (2010) 168 – 174.

[38] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, B. Chapman, High performance computing using MPI and OpenMP on multi-core parallel systems, Parallel Computing 37 (9) (2011) 562 – 575.

[39] F. Bassi, A. Colombo, A. Crivellini, M. Franciolini, Hybrid OpenMP/MPI parallelization of a high-order Discontinuous Galerkin CFD/CAA solver, in: 7th European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS Congress 2016, National Technical University of Athens, 2016, pp. 7992–8012.

[40] A. Crivellini, M. Franciolini, On the implementation of OpenMP and Hybrid MPI/OpenMP parallelization strategies for an explicit DG solver, Advances in Parallel Computing 32 (2018) 527–536.

[41] A. Crivellini, M. Franciolini, A. Colombo, F. Bassi, Openmp parallelization strategies for a discontinuous galerkin solver, International Journal of Parallel Programming (2018) 1–36.

[42] Y. Sato, T. Hino, K. Ohashi, Parallelization of an unstructured Navier–Stokes solver using a multi-color ordering method for OpenMP, Computers & Fluids 88 (2013) 496 – 509.

[43] D. Komatitsch, D. Michéa, G. Erlebacher, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, Journal of Parallel and Distributed Computing 69 (5) (2009) 451 – 460.

[44] G. Karypis, V. Kumar, METIS, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, Technical Report Version 4.0, University of Minnesota, Department of Computer Science/Army HPC Research Center (1998).

[45] J. Hoeflinger, P. Alavilli, T. Jackson, B. Kuhn, Producing scalable performance with OpenMP: Experiments with two CFD applications, Parallel Computing 27 (4) (2001) 391 – 413, parallel computing in aerospace.

[46] A. Baggag, H. L. Atkins, D. Keyes, Parallel implementation of the discontinuous galerkin method, nASA/CR-1999-209546, ICASE Report No. 99-35 (August 1999).

[47] J. C. Hardin, J. R. Ristorcelli, C. K. W. Tam, ICASE/LaRC Workshop on Benchmark Problems in Computational Aeroacoustics (CAA), NASA conference publication, National Aeronautics and Space Administration, Langley Research Center, 1995.

[48] C. K. W. Tam, J. C. Hardin, Second Computational Aeroacoustics (CAA): Workshop on Benchmark Problems, NASA conference publication, NASA, 1997.

[49] A. Crivellini, Assessment of a sponge layer as a non–reflective boundary treatment with highly accurate gust–airfoil interaction results, International Journal of Computational Fluid Dynamics 30 (2) (2016) 176–200.

[50] A. Colombo, A. Crivellini, Assessment of a sponge layer non-reflecting boundary treatment for high-order CAA/CFD computations, Computers & Fluids 140 (2016) 478 – 499. doi:https://doi.org/10.1016/j.compfluid.2016.09.019.

[51] A. Mani, Analysis and optimization of numerical sponge layers as a nonreflective boundary treatment, Journal of Computational Physics 231 (2) (2012) 704 – 716.

[52] P. J. Morris, Scattering of Sound by a Sphere: Category 1: Problems 3 and 4, in: C. Tam, J. Hardin (Eds.), Second Computational Aeroacoustics (CAA) Workshop on Benchmark Problems, 1997, nASA CP 3352.

[53] S. P. Simonaho, T. Lähivaara, T. Huttunen, Modeling of acoustic wave propagation in time-domain using the discontinuous Galerkin method – A comparison with measurements, Applied Acoustics 73 (2) (2012) 173 – 183.

[54] 5$^{\text{th}}$ International Workshop on High–Order CFD Methods, https://how5.cenaero.be/ (2018).

[55] W. M. Van Rees, A. Leonard, D. Pullin, 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 230 (8) (2011) 2794–2805.

[56] Advanced Micro Devices, Inc., AMD Opteron 6200 series processors, Linux tuning guide, downloadable from http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/ (2012).

[57] Advanced Micro Devices, Inc., AMD Opteron 4200/6200 series processors compiler options quick reference guide, downloadable from http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/ (2012).

[58] J. Fang, A. L. Varbanescu, H. Sips, L. Zhang, Y. Che, C. Xu, "An Empirical Study of Intel Xeon Phi", ArXiv e-printsarXiv:1310.5842v2.

[59] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y. C. Liu, Knights landing: Second-generation intel xeon phi product, IEEE Micro 36 (2) (2016) 34–46.

[60] J. Waltz, J. G. Wohlbier, L. D. Risinger, T. R. Canfield, M. R. J. Charest, A. R. Long, N. R. Morgan, Performance analysis of a 3d unstructured mesh hydrodynamics code on multi-core and many-core architectures, International Journal for Numerical Methods in Fluids 77 (6) (2015) 319–333.

[61] R. Kannan, V. Harrand, M. Lee, A. Przekwas, Highly scalable computational algorithms on emerging parallel machine multicore architectures: development and implementation in CFD context, International Journal for Numerical Methods in Fluids 73 (10) (2013) 869–882.

[62] R. Kannan, V. Harrand, X. G. Tan, H. Q. Yang, A. J. Przekwas, Highly scalable computational algorithms on emerging parallel machine multicore architectures II: development and implementation in the CSD and FSI contexts, Journal of Parallel and Distributed Computing 74 (9) (2014) 2808–2817.

[63] C. Altmann, A. Beck, A. Birkefeld, G. Gassner, F. Hindenlang, C. D. Munz, M. Staudenmaier, Discontinuous Galerkin for high performance computational fluid dynamics, in: W. E. Nagel, D. H. Kröner, M. M. Resch (Eds.), High Performance Computing in Science and Engineering '12, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 225–238.

[64] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. M. Badia, E. Ayguade, J. Labarta, Productive cluster programming with OmpSs, in: European Conference on Parallel Processing, Springer, 2011, pp. 555–566.

[65] G. Matheou, P. Evripidou, Data-driven concurrency for high performance computing, ACM Transactions on Architecture and Code Optimization (TACO) 14 (4) (2017) 53.

[66] A. Crivellini, F. Bassi, An implicit matrix-free discontinuous galerkin solver for viscous and turbulent aerodynamic simulations, Computers & Fluids 50 (1) (2011) 81–93.

[67] A. Sarshar, P. Tranquilli, B. Pickering, A. McCall, C. J. Roy, A. Sandu, A numerical investigation of matrix-free implicit time-stepping methods for large cfd simulations, Computers & Fluids 159 (2017) 53–63.

[68] M. Ceze, L. Diosady, S. M. Murman, Development of a high-order space-time matrix-free adjoint solver, in: 54th AIAA Aerospace Sciences Meeting, 2016, p. 0833.

[69] P. Birken, G. Gassner, M. Haas, C. D. Munz, Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navier–Stokes equations, Journal of Computational Physics 240 (2013) 20–35.

[70] F. Renac, S. Gérald, C. Marmignon, F. Coquel, Fast time implicit–explicit discontinuous galerkin method for the compressible navier–stokes equations, Journal of Computational Physics 251 (2013) 272–291.

[71] L. Wang, D. J. Mavriplis, Implicit solution of the unsteady euler equations for high-order accurate discontinuous galerkin discretizations, Journal of Computational Physics 225 (2) (2007) 1994–2015.

[72] K. J. Fidkowski, Output-based space–time mesh optimization for unsteady flows using continuous-in-time adjoints, Journal of Computational Physics 341 (2017) 258–277.

[73] K. Shahbazi, P. F. Fischer, C. R. Ethier, A high-order discontinuous galerkin method for the unsteady incompressible navier–stokes equations, Journal of Computational Physics 222 (1) (2007) 391–407.

[74] E. Ferrer, R. Willden, A high order discontinuous galerkin finite element solver for the incompressible navier–stokes equations, Computers & Fluids 46 (1) (2011) 224–230.

[75] B. Krank, N. Fehn, W. A. Wall, 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 348 (2017) 634–659.

[76] A. Crivellini, M. Franciolini, A. Nigro, A matrix-free incompressible DG algorithm for the simulation of turbulent flows, in: Progress in Turbulence VII, Springer, 2017, pp. 153–159.

[77] M. Franciolini, A. Crivellini, A. Nigro, On the efficiency of a matrix-free linearly implicit time integration strategy for high-order discontinuous Galerkin solutions of incompressible turbulent flows, Computers & Fluids 159 (2017) 276–294.

[78] D. A. Knoll, D. E. Keyes, Jacobian-free newton–krylov methods: a survey of approaches and applications, Journal of Computational Physics 193 (2) (2004) 357–397.

[79] M. Pernice, H. Walker, Nitsol: A newton iterative solver for nonlinear systems, SIAM Journal on Scientific Computing 19 (1) (1998) 302–318.

[80] T. T. Chisholm, D. W. Zingg, A jacobian-free newton–krylov algorithm for compressible turbulent fluid flows, Journal of Computational Physics 228 (9) (2009) 3490–3507.

[81] J. Rang, L. Angermann, New Rosenbrock methods of order 3 for PDAEs of index 2, Comenius University Press, 2007.

[82] J. Rang, L. Angermann, New rosenbrock w-methods of order 3 for partial differential algebraic equations of index 1, BIT Numerical Mathematics 45 (4) (2005) 761–787.

[83] A. Crivellini, V. D'Alessandro, F. Bassi, Assessment of a high-order discontinuous Galerkin method for incompressible three–dimensional Navier–Stokes equations: Benchmark results for the flow past a sphere up to Re=500, Computers & Fluids 86 (2013) 442 – 458.

[84] J. Lang, Adaptive multilevel solution of nonlinear parabolic PDE systems: theory, algorithm, and applications, Vol. 16 of Lecture Notes in Computational Science and Engineering, Springer Science & Business Media, 2001.

[85] D. S. Blom, P. Birken, H. Bijl, F. Kessels, A. Meister, A. H. van Zuijlen, A comparison of rosenbrock and esdirk methods combined with iterative solvers for unsteady compressible flows, Advances in Computational Mathematics 42 (6) (2016) 1401–1426.

[86] B. Helenbrook, D. Mavriplis, H. Atkins, Analysis of"p"-multigrid for continuous and discontinuous finite element discretizations, in: 16th AIAA Computational Fluid Dynamics Conference, 2003, p. 3989.

[87] F. Bassi, S. Rebay, Numerical solution of the euler equations with a multiorder discontinuous finite element method, in: Computational Fluid Dynamics 2002, Springer, 2003, pp. 199–204.

[88] K. J. Fidkowski, T. A. Oliver, J. Lu, D. L. Darmofal, *p*-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, Journal of Computational Physics 207 (1) (2005) 92–113.

[89] F. Prill, M. Lukáčová-Medviďová, R. Hartmann, Smoothed aggregation multigrid for the discontinuous Galerkin method, SIAM Journal on Scientific Computing 31 (5) (2009) 3503–3528.

[90] M. Wallraff, R. Hartmann, T. Leicht, Multigrid solver algorithms for DG methods and applications to aerodynamic flows, in: IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach, Springer, 2015, pp. 153–178.

[91] P. F. Antonietti, M. Sarti, M. Verani, Multigrid algorithms for *hp*-discontinuous Galerkin discretizations of elliptic problems, SIAM Journal on Numerical Analysis 53 (1) (2015) 598–618.

[92] K. Shahbazi, D. J. Mavriplis, N. K. Burgess, Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible navier–stokes equations, Journal of Computational Physics 228 (21) (2009) 7917–7940.

[93] L. T. Diosady, D. L. Darmofal, Preconditioning methods for discontinuous galerkin solutions of the navier–stokes equations, Journal of Computational Physics 228 (11) (2009) 3917–3935.

[94] L. Botti, A. Colombo, F. Bassi, *h*-multigrid agglomeration based solution strategies for discontinuous Galerkin discretizations of incompressible flow problems, Journal of Computational Physics 347 (2017) 382–415.

[95] M. Franciolini, L. Botti, A. Colombo, A. Crivellini, p-Multigrid matrix-free discontinuous Galerkin solution strategies for the under-resolved simulation of incompressible turbulent flows, arXiv preprint arXiv:1809.00866.

[96] P. F. Antonietti, B. A. de Dios, S. C. Brenner, L. yeng Sung, Schwarz methods for a preconditioned WOPSIP method for elliptic problems, Computational Methods in Applied Mathematics Comput. Methods Appl. Math. 12 (3) (2012) 241–272. doi:10.2478/cmam-2012-0021.

[97] F. Brezzi, G. Manzini, D. Marini, P. Pietra, A. Russo, Discontinuous Galerkin approximations for elliptic problems, Numer. Methods Partial Differential Equations 16 (2000) 365–378.

[98] D. Schötzau, C. Schwab, A. Toselli, Mixed hp-DGFEM for incompressible flows, SIAM Journal on Numerical Analysis 40 (6) (2002) 2171–2194.

[99] D. A. Di Pietro, A. Ern, Mathematical Aspects of Discontinuous Galerkin Methods, Vol. 69 of Maths & Applications, Springer-Verlag, 2011.

[100] T. Warburton, J. Hesthaven, On the constants in hp-finite element trace inverse inequalities, Computer Methods in Applied Mechanics and Engineering 192 (25) (2003) 2765 – 2773.

[101] C. Schwab, *p*- and *hp*- Finite Element Methods: Theory and Applications in Solid and Fluid Mechanics, Numerical Mathematics and Scientific Computation, Clarendon Press, 1998.

[102] J. R. Meneghini, F. Saltara, C. L. R. Siqueira, J. A. Ferrari Jr, Numerical simulation of flow interference between two circular cylinders in tandem and side-by-side arrangements, Journal of Fluids and Structures 15 (2) (2001) 327–350.

[103] L. Botti, Influence of reference-to-physical frame mappings on approximation properties of discontinuous piecewise polynomial spaces, Journal of Scientific Computing 52 (3) (2012) 675–703.

[104] T. Johnson, V. Patel, Flow past a sphere up to a Reynolds number of 300, Journal of Fluid Mechanics 378 (1999) 19–70.

[105] A. G. Tomboulides, S. A. Orszag, Numerical investigation of transitional and weak turbulent flow past a sphere, Journal of Fluid Mechanics 416 (2000) 45–73.

[106] P. Ploumhans, G. Winckelmans, J. K. Salmon, A. Leonard, M. Warren, Vortex methods for direct numerical simulation of three-dimensional bluff body flows: application to the sphere at re= 300, 500, and 1000, Journal of Computational Physics 178 (2) (2002) 427–463.

[107] A. Crivellini, M. Franciolini, A. Colombo, F. Bassi, OpenMP parallelization strategies for a discontinuous Galerkin solver, International Journal of Parallel Programming 178 (2) (2018) 427–463.

[108] B. Cockburn, O. Dubois, J. Gopalakrishnan, S. Tan, Multigrid for an HDG method, IMA Journal of Numerical Analysis 34 (4) (2014) 1386–1425.

[109] N. C. Nguyen, J. Peraire, B. Cockburn, An implicit high-order hybridizable discontinuous Galerkin method for nonlinear convection–diffusion equations, Journal of Computational Physics 228 (23) (2009) 8841–8855.

[110] K. J. Fidkowski, A hybridized discontinuous Galerkin method on mapped deforming domains, Computers & Fluids 139 (2016) 80–91.

[111] R. M. Kirby, S. J. Sherwin, B. Cockburn, To cg or to hdg: a comparative study, Journal of Scientific Computing 51 (1) (2012) 183–212.

[112] S. Yakovlev, D. Moxey, R. M. Kirby, S. J. Sherwin, To cg or to hdg: a comparative study in 3d, Journal of Scientific Computing 67 (1) (2016) 192–220.

[113] M. Woopen, A. Balan, G. May, J. Schütz, A comparison of hybridized and standard dg methods for target-based hp-adaptive simulation of compressible flow, Computers & Fluids 98 (2014) 3–16.

[114] M. Kronbichler, W. A. Wall, A performance comparison of continuous and discontinuous galerkin methods with fast multigrid solvers, arXiv preprint arXiv:1611.03029.

[115] J. Schütz, V. Aizinger, A hierarchical scale separation approach for the hybridized discontinuous galerkin method, Journal of Computational and Applied Mathematics 317 (2017) 500–509.

[116] J. Dahm, Toward accurate, efficient, and robust hybridized discontinuous galerkin methods, Phd thesis, University of Michigan (2017).

[117] R. Togni, A. Cimarelli, E. De Angelis, Physical and scale-by-scale analysis of rayleigh–bénard convection, Journal of Fluid Mechanics 782 (2015) 380–404.

[118] H. Yang, Z. Zhu, Numerical simulation of turbulent rayleigh–benard convection, International communications in heat and mass transfer 33 (2) (2006) 184–190.

[119] R. Verzicco, R. Camussi, Numerical experiments on strongly turbulent thermal convection in a slender cylindrical cell, Journal of Fluid Mechanics 477 (2003) 19–49.

[120] R. D. Moser, J. Kim, N. N. Mansour, Direct numerical simulation of turbulent channel flow up to re_τ= 590, Physics of fluids 11 (4) (1999) 943–945.

[121] S. Hoyas, J. Jiménez, Reynolds number effects on the reynolds-stress budgets in turbulent channels, Physics of Fluids 20 (10) (2008) 101511.

[122] K. A. Hoffmann, S. T. Chiang, Computational fluid dynamics volume i, Engineering Education System, Wichita, Kan, USA.

[123] H. Choi, P. Moin, Effects of the computational time step on numerical solutions of turbulent flow, Journal of Computational Physics 113 (1) (1994) 1–4.

[124] M. Langari, Z. Yang, Numerical study of the primary instability in a separated boundary layer transition under elevated free-stream turbulence, Physics of Fluids 25 (7) (2013) 074106. doi:10.1063/1.4816291.

[125] Z. Yang, P. R. Voke, Large-eddy simulation of boundary-layer separation and transition at a change of surface curvature, Journal of Fluid Mechanics 439 (2001) 305–333.

[126] R. Hillier, N. Cherry, The effects of stream turbulence on separation bubbles, Journal of Wind Engineering and Industrial Aerodynamics 8 (1) (1981) 49 – 58.

[127] Y. Nakamura, S. Ozono, The effects of turbulence on a separated and reattaching flow, Journal of Fluid Mechanics 178 (1987) 477–490.

[128] Z. Yang, I. E. Abdalla, Effects of free-stream turbulence on a transitional separated-reattached flow over a flat plate with a sharp leading edge, International Journal of Heat and Fluid Flow 30 (5) (2009) 1026 – 1035, the 3rd International Conference on Heat Transfer and Fluid Flow in Microscale.

[129] E. Lamballais, J. Silvestrini, S. Laizet, Direct numerical simulation of flow separation behind a rounded leading edge: Study of curvature effects, International Journal of Heat and Fluid Flow 31 (3) (2010) 295–306.

[130] L. Cutrone, P. De Palma, G. Pascazio, M. Napolitano, Predicting transition in two-and three-dimensional separated flows, International Journal of Heat and Fluid Flow 29 (2) (2008) 504–526.

[131] P. R. Spalart, Direct simulation of a turbulent boundary layer up to $re_\theta = 1410$, Journal of Fluid Mechanics 187 (1988) 61–98.

[132] A. Palikaras, K. Yakinthos, A. Goulas, The effect of negative shear on the transitional separated flow around a semi-circular leading edge, International Journal of Heat and Fluid Flow 24 (3) (2003) 421 – 430.

[133] P. Spalart, M. Shur, M. Strelets, A. Travin, Initial rans and ddes of a rudimentary landing gear, in: Progress in Hybrid RANS-LES Modelling, Springer, 2010, pp. 101–110.

[134] L. Venkatakrishnan, N. Karthikeyan, K. Mejia, Experimental studies on a rudimentary four-wheel landing gear, AIAA journal 50 (11) (2012) 2435–2447.

[135] S.-H. Peng, M. Strelets, Test case st12, boeing rudi- mentary landing gear., in: ATAAC.

[136] Q.-L. Dong, H.-Y. Xu, Z.-Y. Ye, Numerical investigation of unsteady flow past rudimentary landing gear using ddes, les and urans, Engineering Applications of Computational Fluid Mechanics 12 (1) (2018) 689–710.

[137] M. Tugnoli, A. Abbà, L. Bonaventura, M. Restelli, A locally p-adaptive approach for large eddy simulation of compressible flows in a dg framework, Journal of Computational Physics 349 (2017) 33–58.

[138] A. Colombo, G. Manzinali, A. Ghidoni, G. Noventa, M. Franciolini, A. Crivellini, F. Bassi, A *p*-adaptive implicit discontinous Galerkin method for the under-resolved simulation of compressible turbulent flows, in: 7nd European Conference on Computational Fluid Dynamics, 2018.

[139] F. Naddei, M. de la Llave Plata, V. Couaillier, F. Coquel, A comparison of refinement indicators for p-adaptive simulations of steady and unsteady flows using discontinuous galerkin methods, Journal of Computational Physics 376 (2019) 508–533.