On the Efficiency of Implicit Discontinuous Galerkin Spectral Element Methods for the Unsteady Compressible Navier-Stokes Equations

A thesis accepted by the Faculty
of Aerospace Engineering and Geodesy of the University of Stuttgart
in partial fulfillment of the requirements for the degree of
Doctor of Engineering Sciences (Dr.-Ing.)

by

Serena Vangelatos

born in Lörrach

Main referee:	Prof. Dr. Claus-Dieter Munz
Co-referee:	Prof. Dr. Christian Rohde
Co-referee:	Prof. Dr. Jochen Schütz
Date of defence:	July 8, 2019

Institute of Aerodynamics and Gas Dynamics
University of Stuttgart

2020

For Pia-Regina, Dimitris, Elke and Franz

Preface

This thesis focuses on some of the research topics I worked on as an academic employee at the Institute of Aerodynamics and Gas Dynamics (IAG) of the University of Stuttgart.

At this point I want to thank my doctoral supervisor Prof. Dr. Claus-Dieter Munz for his support, the great working conditions in his numerical research group as well as the scientific freedom I was granted during the PhD project. Furthermore, I want to thank Prof. Dr. Christian Rohde and Prof. Dr. Jochen Schütz for being the co-referees of my thesis.

I also thank all my colleagues for the excellent working atmosphere which I enjoyed every day at the institute. Especially, I want to thank Andrea Beck for reviewing this work, and Florian Hindenlang, Thomas Bolemann, Jonas Zeifang and Philip Ortwein for the fruitful scientific discussions.

Last but not least, I am grateful for the love, the patience and the constant support of my husband Dimitris and my parents Elke and Franz. The greatest thank goes to my daughter Pia-Regina for her great patience and understanding in her first 4 years of life.

Stuttgart, December 20, 2019

Serena Vangelatos

Contents

Preface v

Table of Contents vii

Symbols and Abbreviations xi

Kurzfassung xvii

Abstract xix

1 Introduction 1

- 1.1 Implicit Discontinuous Galerkin Methods 3
- 1.2 Objectives 4
- 1.3 Outline 5

2 Numerics 7

- 2.1 Governing Equations 8
  - 2.1.1 Navier-Stokes Equations 9
- 2.2 Transformation to Reference Space 10
  - 2.2.1 Transformation of Differential Operators and Geometrical Quantities 11
  - 2.2.2 Transformation of the Governing Equations 13
- 2.3 Discontinuous Galerkin Formulation 14
- 2.4 Spatial Discretization 21
  - 2.4.1 Polynomial Approximation and Numerical Quadrature 21
  - 2.4.2 Approximation of the Solution and the Fluxes 22
  - 2.4.3 Time Derivative Integral 25
  - 2.4.4 Volume Integral 26
  - 2.4.5 Surface Integral 28
  - 2.4.6 Semi-Discrete Formulation 29
  - 2.4.7 Second Order Approximation 30
  - 2.4.8 Scaling of the DG Operator 32

2.5 Time Discretization 33
  2.5.1 Explicit Time Integration 33
  2.5.2 Implicit Time Integration 34

3 Implicit Solver 37
3.1 Diagonal Implicit Runge-Kutta Methods 39
3.2 Equation System Solver 41
  3.2.1 Inexact Newton Method 41
  3.2.2 Linear Solver 45
3.3 Preconditioning 54
3.4 Numerical Experiments 55
  3.4.1 Temporal Convergence Order 56
  3.4.2 Effect of GMRES Tolerance 57
  3.4.3 Effect of Newton Tolerance 59
  3.4.4 Effect of Krylov Subspace Dimension 61
  3.4.5 Discussion 62

4 Analytic Block-Jacobi Preconditioner for DGSEM 65
4.1 Jacobian Assembling 68
  4.1.1 Volume Integral 69
  4.1.2 Surface Integral 71
4.2 Solving the Preconditioned System 80
  4.2.1 Exact Inverse by LU Decomposition 81
  4.2.2 Symmetric Gauss-Seidel 83
  4.2.3 Incomplete LU Decomposition 85
  4.2.4 Tensor-Product 87
  4.2.5 Computational Effort of the Preconditioners 91
4.3 Numerical Tests 95
  4.3.1 Quality of the Preconditioner 95
  4.3.2 Influence of Spatial Discretization 98
  4.3.3 Preconditioner Freezing 103
  4.3.4 Influence of the Time Integration Order 104
  4.3.5 Strong Scaling of the Implicit Solver 106
4.4 Discussion 112

5 Application of Implicit DGSEM 115
5.1 Cylinder Flow 116
  5.1.1 Laminar Flow at $Re_D = 200$ 116

5.1.2 Turbulent Flow at $Re_D = 3900$ 126
5.2 Channel Flow at $Re_\tau = 590$ 133
5.3 Discussion 140

6 Conclusion and Prospects 143
6.1 Conclusion 143
6.2 Prospects 145

A Implicit Butcher Coefficients 147

B Explicit Butcher Coefficients 151

C Traveling Vortex Test Case Definitions 153

D System Architecture 155

Bibliography 157

List of Tables 171

List of Figures 175

Curriculum Vitae 181

Symbols and Abbreviations

Latin symbols

$\boldsymbol{a}^i$	i-th contravariant basis vector
$\boldsymbol{a}_j$	j-th covariant basis vector
a_{ij}, b_i, c_i	Butcher table coefficients
$\boldsymbol{A}$	Jacobian of the root function
$\mathbf{BJ}^{iElem}$	$iElem$-th block diagonal of the Jacobian
c_v, c_p	Heat capacity at constant volume/pressure
C_{iElem}	Arbitrary physical element
d	Number of spatial dimensions
D	Differential matrix
$\hat{D}$	Scaled differential matrix by quadrature weights
e	Specific total energy
E	Reference element
∂E	Boundary of reference element
$\boldsymbol{f}$	Root function of the Newton scheme
$\boldsymbol{F}$	Flux vector in physical space ($\boldsymbol{F} = \boldsymbol{F}^A + \boldsymbol{F}^D$)
$\boldsymbol{F}^A$	Advection flux vector in physical space
$\boldsymbol{F}^D$	Diffusion flux vector in physical space
$\boldsymbol{\mathcal{F}} = (\mathcal{F}^1, \mathcal{F}^2, \mathcal{F}^3)^T$	Transformed flux vector in reference space
$\hat{\mathcal{F}}_{ijk}$	Nodal coefficients of the flux $\mathcal{F}^d$
$F^* = F^*_A - F^*_D$	Riemann flux
F^*_A	Advection Riemann flux
F^*_D	Diffusion Riemann flux
G	Flux vector of the lifting equation
h	Specific enthalpy
$\boldsymbol{H}^*$	Riemann flux for the gradient equation

$\boldsymbol{I}$	Unit matrix
J	Jacobian of the element mapping
k	Specific Heat Transfer Coefficient
l	Total number of degrees of freedom
ℓ_i	i-th Lagrange interpolation polynomial
$\mathcal{L}^2$	Quadratic Lebesgue integrable function space
m	Maximum Krylov subspace dimension for restarted GMRES
Ma	Mach number
$\boldsymbol{\mathcal{N}}$	Unit normal vector in reference element
$\boldsymbol{n}$	Unit normal vector in physical element
n_{el}	Number of grid elements
n_{var}	Number of conservative variables
p	Pressure
Pr	Prandtl number
$\boldsymbol{q} = (q^1, q^2, q^3)^T$	Gradient of conservative variables ($q^d = \frac{\partial u}{\partial x^d}$)
$\hat{q}^d_{ijk}$	Nodal coefficients of the gradient q^d
r	Radius
$\boldsymbol{R}$	DGSEM operator
R_{ijk}	Element-local DGSEM operator
Re	Reynolds number
$\hat{s}$	Element surface area
t	Time
t^n, t^{n+1}	Time levels
T	Fluid temperature
u	State vector of conservative variables
$\hat{u}_{ijk}$	Nodal degrees of freedom
$\boldsymbol{u}$	Vector of nodal degrees of freedom
$\boldsymbol{u}^n$	Vector of nodal degrees of freedom at time t^n
$\boldsymbol{\mathcal{U}}$	Transformed state vector in reference space
$\boldsymbol{P}$	Preconditioner for the linear equation system
$\boldsymbol{v} = (v_1, v_2, v_3)^T$	Velocity vector
$\boldsymbol{x} = (x_1, x_2, x_3)^T$	Coordinate vector in physical space
$\boldsymbol{X}$	Mapping from reference to physical space

Greek symbols

α	Diagonal element of the ESDIRK Butcher table
δ_{ij}	Kronecker symbol
Δt	Time step
Δx	Minimum physical element size
ϵ	Specific inner energy
ϵ_N	Newton tolerance
η_{BR2}	Constant of the BR2 flux approximation
η_k	GMRES tolerance
κ	Adiabatic coefficient
λ	Heat transfer coefficient
μ	Dynamic viscosity
$\boldsymbol{\xi} = (\xi^1, \xi^2, \xi^3)^T$	Coordinate vector in reference space
ξ_i	i-the interpolation point
$\boldsymbol{\xi}_{ijk}$	Three-dimensional interpolation point
π	Mathematical constant
ψ_{ijk}	Three-dimensional basis functions
ρ	Density
$\boldsymbol{\tau}$	Shear stress tensor
ϕ	Test function
ω_i	i-th weight of the Gauss integration
Ω	Physical domain

Subscripts

$(\cdot)_\infty$	Freestream value of a quantity
$(\cdot)_L$	Left state of the Riemann solver
$(\cdot)_R$	Right state of the Riemann solver
$(\cdot)_t$	Time derivative

Superscripts

$(\cdot)_0$	Initial solution
$(\cdot)^1, (\cdot)^2, (\cdot)^3$	Components of contravariant quantity
$(\hat{\cdot})$	Nodal coefficients by interpolation
$(\cdot)^{\mathsf{T}}$	Transposed matrix or vector

Mathematical Abbreviations

$\boldsymbol{\nabla}$	Nabla operator $\boldsymbol{\nabla} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T$
$\lVert . \rVert$	Euclidean norm
$\langle . \rangle$	Jump operator
$\{ . \}$	Mean value operator

Abbreviations

ADI	Alternating-Direction-Implicit
BR1/2	First/second method of Bassi and Rebay
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy condition
CPU	Central Processing Unit
DG	Discontinuous Galerkin
DGSEM	Discontinuous Galerkin Spectral Element Method
DNS	Direct Numerical Simulation

DOF	Degree Of Freedom
EOC	Experimental Order of Convergence
ERK	Explicit Runge-Kutta
ESDIRK	Explicit first stage Singly Diagonally Implicit Runge-Kutta
FE	Finite Element
FV	Finite Volume
GMRES	Generalized Minimal Residual Method
GTS	Global Time Stepping
HOPR	High-Order Preprocessor (mesh generation software)
HLL	Harten-Lax-van Leer Riemann solver
HLRS	Höchstleistungsrechenzentrum Stuttgart
IAG	Institute of Aerodynamics and Gas dynamics
ILU	Incomplete Lower-Upper factorization
IVP	Initial Value Problem
JFNK	Jacobian-free Newton-Krylov method
LAPACK	Linear Algebra PACKage
LDG	Local Discontinuous Galerkin
LF	Lax-Friedrichs
LTS	Local Time Stepping
LES	Large Eddy Simulation
LU	Lower-Upper factorization
MPI	Message Passing Interface
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PID	Performance Index
RK	Runge-Kutta
RKDG	explicit Runge-Kutta Discontinuous Galerkin
SGS	Symmetric Gauss-Seidel

Kurzfassung

In dieser Arbeit wird die implizite Zeitintegration für die Discontinuous Galerkin Spectral Element Methode (DGSEM) untersucht im Hinblick auf ihre Leistungsfähigkeit in Effizienz und Genauigkeit im Vergleich zur expliziten Zeitintegration. Bei instationären Simulationen laminarer und turbulenter Strömungen, die durch die kompressiblen Navier-Stokes-Gleichungen beschrieben werden, ist nicht nur die räumliche, sondern auch die zeitliche Genauigkeit von großer Bedeutung. Aufgrund der durch die Stabilitätsbedingung bedingten Zeitschrittbeschränkung ist die Verwendung von expliziten Verfahren für zeitgenaue Simulationen weit verbreitet. Bei sehr steifen Problemen wird diese Bedingung schwerwiegend, was zur Verwendung impliziter Zeitintegrationsverfahren führt, die als bedingungslos stabil konstruiert werden können. Der implizite Zeitschritt wird nur durch Genauigkeitsanforderungen gesteuert. Die Lösung der entstehenden (nicht-) linearen Gleichungssysteme im DGSEM-Kontext ist jedoch nach wie vor eine Herausforderung.

Discontinuous Galerkin Verfahren hoher Ordnung haben in den letzten Jahren im Bereich der Computational Fluid Dynamics (CFD) aufgrund ihrer hohen Eignung für die Simulation von Multiskalenproblemen große Popularität erlangt. Sie zeichnen sich durch Genauigkeit hoher Ordnung und hohe Effizienz im parallelen Rechnen dank ihres kompakten Abhängigkeitsbereichs aus. Diese Kompaktheit und der element-lokale Charakter von DGSEM machen das Verfahren auch in Kombination mit impliziten Zeitintegrationsverfahren attraktiv. Die zusätzliche geringe Abhängigkeit innerhalb jedes Elements durch den Tensor-Produkt Ansatz induziert eine dünnbesetze Jacobian-Matrix des DGSEM-Operators mit geringem Speicherbedarf, die durch den linearen Löser ausgenutzt werden kann. Allerdings gibt es im Bereich impliziter Discontinuous Galerkin Verfahren noch einige offene Themen für Multiskalenprobleme: Trotz der umfangreichen Untersuchungen von linearen Lösern erfordert die Recheneffizienz in Bezug auf die CPU-Zeit und die entsprechende Genauigkeit der impliziten Löser eine intensive Betrachtung, insbesondere im Vergleich zu expliziten Zeitintegrationsver-

fahren. Darüber hinaus existieren bis heute nur wenige parallele Effizienzuntersuchungen. Wir richten unsere Aufmerksamkeit in dieser Arbeit auf diese Themen.
Wir betrachten Explicit first stage Singly Diagonally Implicit Runge-Kutta (ESDIRK) Verfahren in Kombination mit einem Jacobian-Free Newton-Krylov (JFNK) Löser, der zur Klasse der matrixfreien Lösungsansätze gehört. Im ersten Teil stellen wir eine neue Strategie zur Lösung der nichtlinearen und linearen Systeme mit adaptiven Toleranzen vor, um Über- und Unterauflösen zu vermeiden. Diese Toleranzen werden automatisch an die zeitlichen Genauigkeitsanforderungen angepasst. Diese Strategie führt nicht nur zu einer benutzerfreundlichen Handhabung, sondern auch zu einem hocheffizienten Löser. In einem zweiten Schritt führen wir den Block-Jacobi Präkonditionierer ein, der alle nebendiagonal Blöcke vernachlässigt, so dass das resultierende Linearsystem element-lokal gelöst werden kann. Der Vorteil dieses Präkonditionieres ist der geringe Speicherbedarf, die parallele Skalierbarkeit und die einfache Implementierung. Weiterhin konzentrieren wir uns auf verschiedene approximative Löser für die element-lokalen linearen Gleichungssystem, um den linearen Löser in Bezug auf die CPU-Zeit zu beschleunigen. Hier nutzen wir die schwache Besetzungsstruktur aus, die durch die Tensor-Produkt Struktur hervorgerufen wird. Wir stellen einen neuartigen Ansatz vor, indem wir die Block-Jacobi Matrix der kompressiblen Navier-Stokes-Gleichungen auf die Einträge beschränken, die sich auf die Besetzungsstruktur beziehen, die sich aus den Euler-Gleichungen ergibt, um die dünnbesetzte Matrixstruktur zu erhalten. Die Effizienz wird durch die Auswirkungen der Zeitschrittweite, des element-lokalen Lösers, der Polynomordnung, des Einfrierens des Präkonditionierers, der zeitlichen Ordnung und der Parallelisierung bewertet.
Schließlich wird die implizite DGSEM auf laminare und turbulente Testfälle angewendet, mit denen wir die Effizienz und Genauigkeit im Vergleich zu einem expliziten Zeitintegrationsverfahren demonstrieren.
Die Ergebnisse zeigen, dass implizite DGSEM mit einem speicherarmen Präkonditionierer, der die dünnbesetzte Struktur der Jacobi Matrix des DGSEM Operators ausnutzt, mit einem expliziten Runge-Kutta Verfahren in Bezug auf Rechenzeit und Genauigkeit wettbewerbsfähig sein kann. Der implizite Löser kann das explizite Verfahren bei sehr starken Zeitschrittbeschränkungen mit gleicher räumlicher Genauigkeit oder mit noch besseren Übereinstimmungen zu DNS-Ergebnissen übertreffen.

Abstract

In this work we investigate the implicit time integration for the Discontinuous Galerkin Spectral Element Method (DGSEM) with respect to its efficiency and accuracy compared to explicit time integration. Considering unsteady simulations of laminar and turbulent flows described by the compressible Navier-Stokes equations, not only the spatial but also the temporal discretization is of high importance. Due to the explicit time step restriction caused by the stability condition, explicit schemes are widely used for time accurate simulations. In the case of very stiff problems this condition becomes severe leading to the consideration of implicit time integration schemes, which can be constructed as unconditionally stable. The implicit time step is contrastingly only driven by accuracy requirements. However, solving the arising (non-) linear equation systems within the DGSEM context is still a challenging issue.

High-order Discontinuous Galerkin schemes are popular due to their high suitability for the simulation of multi-scale problems. They feature high order accuracy and high efficiency in parallel computing thanks to their compact stencil. This compactness and the element-local nature of DGSEM makes them also attractive in combination with implicit time integration schemes. The additional low dependency within each element due to the tensor-product ansatz induces a high sparsity of the Jacobian matrix of the DGSEM operator with low memory requirements, which can be exploit by the linear solver. However, there are still several open topics in the field of implicit Discontinuous Galerkin schemes for multi-scale problems: despite extensive research on linear solvers, the computational efficiency in terms of CPU time and the corresponding accuracy of the implicit solvers necessitates an intensive consideration especially in comparison with explicit time integration schemes. Furthermore, only a few parallel efficiency investigations exist up to data. In this work we direct our attention to the proposed issues.

We consider Explicit first stage Singly Diagonally Implicit Runge-Kutta (ESDIRK) schemes in combination with a Jacobian-Free Newton-Krylov (JFNK) solver, which belongs to the class of matrix-free solution approaches. In the first part, we introduce a novel strategy for solving the non-linear and linear systems with adaptive tolerances in order to avoid over- and under-solving. These tolerances are automatically adjusted to temporal accuracy requirements. This strategy leads not only to an user friendly handling but also to an highly efficient solver. In a second step, we employ the block-Jacobi preconditioner neglecting all the off-diagonals blocks, so that the resulting linear system is able to be solved element-locally. The advantages of this preconditioner are low storage requirements, parallel scalability and simple implementation. Further, we focus on various approximative solvers for the element-local linear systems in order to accelerate the linear solver in terms of CPU time. Here, we exploit the sparsity induced by the tensor-product structure. We introduce a novel approach by restricting the block-Jacobi of the compressible Navier-Stokes equations to the entries related to the matrix pattern resulting from the Euler equations in order to maintain the very sparse matrix structure. The efficiency is evaluated against multiple criteria, such as the time step size, element-local solver, polynomial order, preconditioner freezing, temporal order and parallel computing.

Finally, the proposed implicit DGSEM is applied to laminar and turbulent test cases, by which we demonstrate the efficiency and accuracy in comparison to an explicit time integration scheme.

The results highlight that implicit DGSEM with a low-storage preconditioner which exploits the sparsity of the DGSEM operator's Jacobian can be competitive with an explicit Runge-Kutta scheme in terms of computational time and accuracy. The implicit solver can outperform the explicit scheme in the case of very severe time step restrictions with the same spatial accuracy or with even better agreements to DNS results.

1 Introduction

The interest in numerical simulations of fluid flows has grown rapidly in the last decades. Due to their significant improvement in efficiency and accuracy, they have become feasible for increasing numbers of applications in the engineering field.

Physical phenomena are generally characterized by different temporal scales. For unsteady simulations of multi-scale problems, one encounters the question which range of wave lengths needs to be resolved to capture physically relevant effects while taking computational costs into account. However, if the resolution of the fastest time scales is not of practical interest, explicit schemes might be costly due to their temporal high accuracy. Considering spatially under-resolved computations, where the total error is dominated by the spatial resolution, one may think of implicit time integration schemes in order to choose higher time steps.

Generally, explicit time stepping schemes suffer from a restrictive stability constraint, termed Courant-Friedrichs-Lewy (CFL) [37] condition, which depends on the order of the temporal method as well as on the order of spatial discretization. This stability condition becomes severe for stiff problems resulting for instance from the discretized unsteady compressible Navier-Stokes equations within the method of lines approach for wall bounded and low Mach number flows. Further, explicit time integration schemes are not well suited for problems with disparate time and length scales and for steady state computations.

In contrast, implicit time marching schemes can be constructed as unconditionally stable such that the time step is only driven by temporal accuracy. However, they require an efficient solution of the resulting very large (non-linear) equation systems, while explicit schemes are easy to implement. Finding a competitive solution technique is still a challenging task for DG schemes [113] especially for the three-dimensional case. Apparently, it is not obvious which scheme is more efficient when we contrast the issues of both schemes:

- Explicit time step sizes are limited by a stability condition, which is even more restrictive for higher order spatial discretizations. Especially for thin elements, which are required for instance for the resolution of boundary layers in turbulent flows, the restriction becomes prohibitively strict resulting in a time step size much below any temporal accuracy requirements. However, they are simple to implement, feature low memory requirements and do not require the solution of an equation system.

- Implicit time steps sizes can be chosen much higher, but the cost of one implicit time step is far greater than one explicit time step. Solving efficiently (non-linear) systems of equations necessitates a more complex implementation. Additionally, the performance of the solver depends strongly on the physical properties of the flow and the chosen parameters. Also, the memory demand is much higher especially for high order spatial discretization for 3D problems.

The basis of all numerical methods consists of the spatial discretization of the governing equations. Discontinuous Galerkin (DG) schemes have become increasingly popular in recent years in the scope of high accurate solutions of conservation laws. This is attributed to the steady increase of computational resources making high resolutions possible and leads to the demand of massive parallel scalable methods. DG schemes provide high order spatial accuracy by means of high order polynomial approximations within the grid elements of arbitrary shape. The discontinuity of the solution at element interfaces are interpreted as Riemann problems, which have been extensively studied in the context of Finite Volume (FV) schemes. Adjacent elements are coupled only by the numerical flux depending on the double-valued states at the element interfaces. This discontinuous nature induces an element-local formulation and makes the scheme perfectly suitable for parallel computations. The only necessary communication required within DG schemes exists between direct neighbors.
In this work, we investigate a particular efficient variant of the DG discretization, namely the Discontinuous Galerkin Spectral Element Method (DGSEM) [79]. This method considers a tensor-product basis restricting the computational grid to quadrilateral or hexahedral elements but allowing general unstructured grids. DGSEM represents a computationally highly efficient and simple to implement method compared to other DG variants.

1.1 Implicit Discontinuous Galerkin Methods

For time dependent problems, Cockburn and Shu [35, 31, 29, 28, 30] introduced a combination of the spatial DG discretization with an explicit high order Runge-Kutta time discretization, denoted by explicit Runge-Kutta DG (RKDG) methods, for the non-linear hyperbolic conservation laws. Bassi and Rebay [9, 12] extended this approach to viscous problems as the compressible Navier-Stokes equations.

For high order DG schemes, there is still a lack of an efficient solver leading to a greater usage of explicit time integration schemes. However, several authors have concentrated on the investigation of an implicit solver for unsteady flows with a spatial DG discretization. In [114], Wang and Mavriplis compared different implicit time integration schemes including a p-multigrid solver for the two-dimensional Euler equations. They found the fourth-order accurate Explicit first stage Singly Diagonally Implicit Runge-Kutta (ESDIRK) scheme to be the more efficient in computational time for a given accuracy than the first and second-order implicit backwards differencing schemes (BDF) and the second-order Crank-Nicholson scheme. Bassi and Rebay employed in [10] the implicit backward Euler time integration scheme for the two-dimensional compressible Navier-Stokes equations. The arising linear system is solved by a preconditioned Generalized Minimum RESidual iterative method (GMRES). They compared different preconditioners as the standard Incomplete LU (ILU) factorization and a simple block-Jacobi preconditioner, which turns out to be adequate for the DG discretization of the compressible Navier-Stokes equations. Persson and Peraire studied in [100] the performance of the ILU preconditioned Newton-Krylov solver in conjunction with a BDF implicit time discretization for turbulent flows using the Reynolds averaged Navier-Stokes (RANS) equations with the one-equation Spalart–Allmaras turbulence model. In [18] Birken et al. investigated a new preconditioner ROBO-SGS for the compressible Navier-Stokes equations with regard to the parallel efficiency for high number of processors. They found a good overall performance and good strong scalability for both preconditioners, ROBO-SGS and block-Jacobi. Recently, Franciolini et al. investigated in [53] linearly implicit Rosenbrock-type Runge–Kutta schemes for the Implicit Large Eddy Simulation (ILES) of turbulent flows for the incompressible Navier-Stokes equations. They compared the efficiency of matrix-free and matrix-based implementations. In [96] Pazner and Persson developed a novel tensor-product

based preconditioner using an algebraic Kronecker-product singular value decomposition (KSVD) approximating the block-Jacobi matrix for the Euler equations. While the KSVD preconditioner increases the GMRES counts, it reduces the runtime for high polynomial degrees compared to the exact block-Jacobi.
The main objective of this thesis is the investigation of implicit time integration schemes in combination with a spatial high order DGSEM discretization. The advantage of using DG schemes for implicit time marching schemes lies in its element-local nature providing a block structured Jacobian matrix of the spatial DG operator. The coupling by only adjacent neighboring elements and the tensor-product structure brings us to a very low dependency and hence to a sparse Jacobian matrix with low memory demand compared to other methods.

1.2 Objectives

In spite of several studies about implicit time discretization for DG schemes, there are still some issues not considered. Except for the work of Wang and Mavriplis, there exists rarely any comparison between implicit and explicit time discretization in terms of computational time and accuracy. As the number of iterations is not well suited in order to judge the implicit solver, also the CPU time has to be taken into account, which is commonly neglected in literature. Since the termination criteria within the solvers influences not only the accuracy but also the CPU time, the solution quality is also a key criterion for the evaluation of implicit time integration schemes in the case of unsteady flows. Further, the question about parallel efficiency of the implicit solver on high number of processors remains often unanswered.

In this work, the investigations are directed to tackle the proposed issues. Hence, the main focus lies in the computational efficiency of implicit time integration for a DGSEM discretization for unsteady laminar and turbulent flows. DGSEM has been applied to turbulent flows with an explicit Runge-Kutta scheme by several authors [48, 65, 15, 50], particularly with the fourth order 5-stage scheme by Carpenter and Kennedy [25]. Hence, we compare the computational time required by the implicit solver against this commonly used explicit scheme. Considering the solving strategy of the algebraic systems, we have to achieve the following key requirements of a low storage, parallel scalable, fast converging, simple to implement solver so that implicit DGSEM becomes more feasible for the practical use also in the industrial context. In an attempt to unify all these properties, we introduce a novel strategy of element-local preconditioning exploiting the high sparsity of the DGSEM operator. Further, we employ adaptive parameters for the implicit solver in order to guarantee an adequate fit between efficiency an accuracy.

Within this work, we extended the open source Computational Fluid Dynamics (CFD) code FLEXI [92], which is based on DGSEM with explicit time marching schemes, by implicit time integration with solution strategies fulfilling the proposed requirements and various preconditioners for comparison. The flow solver FLEXI was developed by the Numerics Research Group of the Institute of Aero- and Gasdynamics (IAG) at the University of Stuttgart and it is still in progress involving subjects of ongoing research. Several topics have been investigated with FLEXI: Multi-phase flows [46, 47, 60, 61], DNS and Large Eddy Simulations (LES) of laminar and turbulent flows [65, 15, 6, 16], direct acoustic simulations of flow noise [48, 50, 54] and Maxwell's equations with particle in cell methods [89, 93, 36].

1.3 Outline

The structure of this work is given as follows: In Chapter 2, we start with the description of the considered compressible Navier-Stokes equations and the derivation of the numerical scheme DGSEM. Here, we focus on the detailed description of the spatial discretization including the hyperbolic and parabolic terms for the later derivation of the Jacobian matrix. The implicit solver is discussed in Chapter 3, where we introduce the specific time integration scheme ESDIRK and the solvers for the arising non-linear and linear equation systems. This chapter concludes with some basic numerical experiments for the two-dimensional Euler equations in order to evaluate the implicit solver without preconditioning. Chapter 4 is addressed to the block-Jacobi preconditioner, where we start with the computation of the analytical derivatives of the DGSEM operator. Different exact and approximative strategies are proposed for solving the element-local linear system.

Numerical tests for evaluating the efficiency of the element-local preconditioners for the compressible Navier-Stokes equations are shown with respect to different time step sizes, polynomial order, preconditiong freezing and time integration order. Further, the parallel efficiency of the implicit solver including different preconditioners is investigated. In Chapter 5, we show the capabilities of the implicit solver in comparison to the explicit scheme for two test cases, a laminar and turbulent flow around a circular cylinder and a turbulent channel flow. We conclude this work with a summary of the recent work and give a short outlook for possible further investigations within this research topic in Chapter 6.

2 Numerics

In this chapter, we concentrate on the derivation of the spatial discretization of the underlying governing equations, employing the Discontinuous Galerkin Spectral Element Method (DGSEM), which is implemented in the open source flow solver FLEXI provided in [92]. The resulting semi-discrete discontinuous Galerkin formulation yields a system of ordinary differential equations for the time dependent degrees of freedom. In this work, we focus on implicit time integration schemes investigated in the following chapters in order to discretize the semi-discrete form in time.

Discontinuous Galerkin (DG) methods have originally been proposed in [103] in 1973 for linear steady hyperbolic equations. Later, Cockburn and Shu [35, 31, 29, 28, 30] extended the method to non-linear hyperbolic conservation laws. For the compressible Navier-Stokes equations, Bassi and Rebay introduced in [9, 11] an approach by rewriting the parabolic terms into a system of first order, followed by several related formulations [34, 59, 74, 97].

DG methods represent a combination of Finite Element (FE) and Finite Volume (FV) methods. They are based on the weak formulation of the governing equations in order to allow discontinuous solutions. Subdividing the computational domain into non-overlapping elements, the DG solution is approximated by high order polynomial ansatz functions within each element, but globally the solution can posses discontinuities across grid cell interfaces. Due to the double-valued data at these interfaces, adjacent elements are coupled by Riemann solvers, which are well-known within the FV community. The element-local nature of DG methods guarantees that the mass matrix is local and hence easier to invert, compared to the classic continuous FE method using a globally coupled mass matrix. In contrast to FV methods, the accuracy of discontinuous Galerkin methods can be easily increased to arbitrary high order by adjusting the polynomial degree of the ansatz function without extending the stencil of the scheme. Consequently, these methods are promising candidates for future numerical schemes showing high parallel scalability.

In this work, we will consider a particularly efficient variant of DG schemes, namely DGSEM [79, 65]. The main issues for DGSEM are the usage of nodal basis functions i.e. Lagrange polynomials, the collocation of interpolation and integration points and the tensor-product structure, which restricts the method to hexahedral elements.

This chapter is structured as follows. As a first step, we will introduce the governing equations fitting in the type of conservations laws, which the following discretization is tailored for. As a second step, we will transform the conservation law to the reference space to obtain a general formalism for any hexahedral element. Then, we will introduce the discontinuous Galerkin formulation by rewriting the transformed equations in the weak sense. For parabolic terms of the partial differential equations, we will employ a special treatment. Due to the discontinuous solution across element faces, we need to introduce in a further step numerical flux functions. Further, we will define the polynomial representation of the solution and the fluxes within an element and the employed quadrature rule. With all the preparation in place, we can determine each integral within the discontinuous Galerkin formulation resulting in the semi-discrete form. In the last section, we will briefly discuss different time-integrators in order to obtain the fully discrete scheme.

2.1 Governing Equations

In this section, we present the governing equations we will concentrate on for deriving the discontinuous Galerkin Method. We will focus in this work on solving purely hyperbolic and hyperbolic-parabolic partial equations which belong to the class of advection-diffusion equations. In the d-dimensional domain $\Omega \subset \mathbb{R}^d$ advection-diffusion problems can be expressed in the form

$$u_t + \boldsymbol{\nabla} \cdot \boldsymbol{F}^A(u) - \boldsymbol{\nabla} \cdot \boldsymbol{F}^D(u, \boldsymbol{\nabla} u) = 0 \qquad \text{in } \Omega \times \mathbb{R}^+, \tag{2.1}$$

where the vector of conservative variables is given by $u \in \mathbb{R}^{n_{var}}$, the advection fluxes by $\boldsymbol{F}^A = (F_1^A, \cdots, F_d^A)^T \in \mathbb{R}^{d \times n_{var}}$ and the diffusion fluxes by $\boldsymbol{F}^D = (F_1^D, \cdots, F_d^D)^T \in \mathbb{R}^{d \times n_{var}}$. The divergence of a flux function

$\boldsymbol{F} = (F_1, \cdots, F_d)^T \in \mathbb{R}^{d \times n_{var}}$ is defined as

$$\boldsymbol{\nabla} \cdot \boldsymbol{F} = \sum_{i=1}^{d} \partial_{x_i} F_i \in \mathbb{R}^{n_{var}}.$$

Generally, we can express equation (2.1) as conservation law for the state vector u

$$u_t + \boldsymbol{\nabla} \cdot \boldsymbol{F}(u, \boldsymbol{\nabla} u) = 0 \qquad \text{in } \Omega \times \mathbb{R}^+, \tag{2.2}$$

where the summarized flux function $\boldsymbol{F} = \boldsymbol{F}(u, \boldsymbol{\nabla} u) \in \mathbb{R}^{d \times n_{var}}$ is given by

$$\boldsymbol{F}(u, \boldsymbol{\nabla} u) = \boldsymbol{F}^A(u) - \boldsymbol{F}^D(u, \boldsymbol{\nabla} u). \tag{2.3}$$

2.1.1 Navier-Stokes Equations

In this work, the considered fundamental conservation laws are the unsteady compressible Euler and Navier-Stokes equations which form a type of advection-diffusion equations. For the three-dimensional compressible Navier-Stokes equations, the conservative variables and the Euler fluxes are given by

$$u = \begin{pmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ \rho v_3 \\ \rho e \end{pmatrix}, \quad F_i^A(u) = \begin{pmatrix} \rho v_i \\ \rho v_i v_1 + \delta_{1i} p \\ \rho v_i v_2 + \delta_{2i} p \\ \rho v_i v_3 + \delta_{3i} p \\ \rho e v_i + p v_i \end{pmatrix} \quad i = 1, 2, 3, \tag{2.4}$$

and the diffusion fluxes are defined by

$$F_i^D(u, \boldsymbol{\nabla} u) = \begin{pmatrix} 0 \\ \tau_{1i} \\ \tau_{2i} \\ \tau_{3i} \\ \sum_j \tau_{ij} v_j + k \frac{\partial T}{\partial x_i} \end{pmatrix} \quad i = 1, 2, 3. \tag{2.5}$$

The physical quantities are listed in Table 2.1. δ_{ij} identifies the Kronecker delta and τ the viscous stress tensor defined by

$$\tau = \mu \left(\boldsymbol{\nabla} \boldsymbol{v} + (\boldsymbol{\nabla} \boldsymbol{v})^T - \frac{2}{3} (\boldsymbol{\nabla} \cdot \boldsymbol{v}) \boldsymbol{I} \right),$$

where $\boldsymbol{I}$ denotes the identity matrix. In order to close the Navier-Stokes equations, we use the equation of state of an ideal gas, which reads

$$p = \rho RT = (\kappa - 1)\rho(e - \frac{1}{2}\boldsymbol{v}^2), \quad \text{with } e = \frac{1}{2}\boldsymbol{v}^2 + c_v T.$$

The compressible Navier-Stokes equations are a non-linear hyperbolic-parabolic differential equation system. If viscous effects are neglected, the diffusion flux $\boldsymbol{F}^D$ is omitted and we obtain the purely hyperbolic Euler equations.

ρ	density
$\boldsymbol{v} = (v_1, \dots, v_d)^T$	velocity vector
e	total energy
p	static pressure
$k = \frac{\mu c_p}{Pr}$	specific heat transfer coefficient
μ	viscosity
c_p, c_v	specific heats
$\kappa = \frac{c_p}{c_v}$	adiabatic exponent
Pr	Prandtl number
T	temperature
$R = c_p - c_v$	specific gas constant

Table 2.1: Physical quantities

2.2 Transformation to Reference Space

For the derivation of numerical methods solving equation (2.2), we require a discretization of the physical domain $\Omega \subset \mathbb{R}^d$ into non-overlapping elements. It is common to transform then each physical element of the mesh into a reference space in order to obtain a general formalism for all elements. For DGSEM we are restricted in 3D to hexahedral (in 2D to quadrilateral) elements due to the tensor-product property of the method. Hence, we subdivide Ω into n_{el} non-overlapping tensor-product elements $C_{iElem} \subset \Omega$, $iElem = 1, \dots n_{el}$ and define the reference element by

$$E = [-1, 1]^d \qquad \text{with coordinates } \boldsymbol{\xi} \in \mathbb{R}^d.$$

Within this element it is possible to use standard numerical operations, such as tensor-product Gaussian quadrature rules. Particularly in the case of arbitrarily curved elements, the mapping to a linear reference element simplifies the implementation significantly. Considering without loss of generality an arbitrary physical element C_{iElem}, we employ the mapping $\boldsymbol{X}$ with

$$\begin{aligned} \boldsymbol{X}\colon E &\to C_{iElem} \\ \boldsymbol{\xi} &\mapsto \boldsymbol{X}(\boldsymbol{\xi}) = \boldsymbol{x} \end{aligned}$$

from the reference space E with coordinates $\boldsymbol{\xi} = (\xi^1, \dots, \xi^d)^T$ to the physical element C_{iElem} with physical coordinates $\boldsymbol{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$. The mapping $\boldsymbol{X}$ depends on the element C_{iElem} and has to be minimum a $\mathcal{C}^1$-diffeomorphism.

First, we study how the differential operators transform under the mapping $\boldsymbol{X}$ in order to rewrite the conservation law (2.2) in coordinates $\boldsymbol{\xi}$ of the reference element.

2.2.1 Transformation of Differential Operators and Geometrical Quantities

In order to transform the differential equations (2.2) defined on Ω into the reference element, we have to derive the transformed divergence operator which is gained by the use of the chain rule. Furthermore, we consider the integrated conservation laws over the physical space within DGSEM, so that we have to use the integration by substitution to obtain the integrals over the reference element. The following derivations are given for the three-dimensional case without loss of generality.
The gradient of an arbitrary function $g(\boldsymbol{x})$ defined in reference space is computed by means of the chain rule resulting in

$$\boldsymbol{\nabla}_{\boldsymbol{\xi}} g = \boldsymbol{J} \boldsymbol{\nabla}_{\boldsymbol{x}} g,$$

where $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\xi})$ is the Jacobian matrix of the mapping $\boldsymbol{X}(\boldsymbol{\xi})$ with

$$\boldsymbol{J}(\boldsymbol{\xi}) = \begin{pmatrix} \frac{\partial x_1}{\partial \xi^1} & \frac{\partial x_2}{\partial \xi^1} & \frac{\partial x_3}{\partial \xi^1} \\ \frac{\partial x_1}{\partial \xi^2} & \frac{\partial x_2}{\partial \xi^2} & \frac{\partial x_3}{\partial \xi^2} \\ \frac{\partial x_1}{\partial \xi^3} & \frac{\partial x_2}{\partial \xi^3} & \frac{\partial x_3}{\partial \xi^3}. \end{pmatrix}$$

Since we are interested in expressing the derivatives with respect to the physical space by derivatives with respect to the reference space $\boldsymbol{\nabla}_{\boldsymbol{x}} g = \boldsymbol{J}^{-1}\boldsymbol{\nabla}_{\boldsymbol{\xi}} g$, we need to determine the inverse $\boldsymbol{J}^{-1}$ of the Jacobian matrix. Now, we define two types of basis vectors describing the directions in physical space, following [78, 79]. First, the covariant basis vectors

$$\boldsymbol{a}_i = \frac{\partial \boldsymbol{X}}{\partial \xi^i}, \qquad i = 1, 2, 3,$$

which are tangential to the coordinate lines and represent the rows of the Jacobian matrix $\boldsymbol{J} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3)^T$. The second type of basis vectors are the contravariant basis vectors

$$\boldsymbol{a}^i = \boldsymbol{\nabla}_{\boldsymbol{x}} \boldsymbol{\xi}^i, \qquad i = 1, 2, 3,$$

which are normal to the coordinate lines and form the rows of the Jacobian inverse

$$\boldsymbol{J}^{-1} = \frac{1}{J} \begin{pmatrix} (\boldsymbol{a}_2 \times \boldsymbol{a}_3)^T \\ (\boldsymbol{a}_3 \times \boldsymbol{a}_1)^T \\ (\boldsymbol{a}_1 \times \boldsymbol{a}_2)^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{a}^1 \\ \boldsymbol{a}^2 \\ \boldsymbol{a}^3 \end{pmatrix}.$$

Here, J describes the determinant of the Jacobian matrix $\boldsymbol{J}$. Hence, the contravariant basis can be expressed by the covariant basis as

$$(J\boldsymbol{a})^i = \boldsymbol{a}_j \times \boldsymbol{a}_k, \qquad \text{with } (i, j, k) \text{ cyclic}, \tag{2.6}$$

defining the so-called metric terms. So the physical surface differential $d\boldsymbol{S}^i$ pointing in the direction of increasing ξ^i and the volume differential dV transform under the mapping $\boldsymbol{X}$ as

$$\begin{aligned} d\boldsymbol{S}^i &= (\boldsymbol{a}_j \times \boldsymbol{a}_k) d\xi^j d\xi^k = \boldsymbol{s}^i d\xi^j d\xi^k, && (i, j, k) \text{ cyclic}, \\ dV &= \boldsymbol{a}_i \cdot (\boldsymbol{a}_j \times \boldsymbol{a}_k) d\xi^i d\xi^j d\xi^k = J d\xi^i d\xi^j d\xi^k, \end{aligned}$$

expressed in covariant basis vectors.

Thus, we obtain for the non-normalized surface normal $\boldsymbol{s}^i$ and the Jacobian determinant J

$$\boldsymbol{s}^i = (\boldsymbol{a}_j \times \boldsymbol{a}_k) \tag{2.7}$$

$$J = \boldsymbol{a}_1 \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3). \tag{2.8}$$

Finally, we can write the transformed gradient of a function g and the divergence of a vector-valued function $\boldsymbol{g}$ in reference space using the basis vectors. Following Kopriva [79], the conservative form of the gradient and divergence operators are given by

$$\boldsymbol{\nabla}_{\boldsymbol{x}} g = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i g \right), \tag{2.9}$$

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{g} = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i \cdot \boldsymbol{g} \right), \tag{2.10}$$

where the contravariant basis $(J\boldsymbol{a})^i$ can be expressed by the covariant basis $\boldsymbol{a}_i$ as in (2.6).

2.2.2 Transformation of the Governing Equations

We study again the three-dimensional case for mapping the conservation law considered on a physical grid cell into the reference space. Therefor, we rewrite (2.2) restricted to an arbitrary physical element C_{iElem}

$$u_t + \boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{F}(u, \boldsymbol{\nabla}_{\boldsymbol{x}} u) = 0 \qquad \text{in } C_{iElem} \times \mathbb{R}^+. \tag{2.11}$$

The state vector can be computed in reference coordinates by composition $u(x,t) = u(\boldsymbol{X}(\xi), t)$. And as given in (2.10), the divergence of the flux function can be transformed to the reference element as

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{F} = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i \cdot \boldsymbol{F} \right). \tag{2.12}$$

Following [79], we define the contravariant flux $\boldsymbol{\mathcal{F}}$ with components including the contravariant basis vectors

$$\mathcal{F}^i = (J\boldsymbol{a})^i \cdot \boldsymbol{F} \in \mathbb{R}^n_{var}, \qquad i = 1, 2, 3, \tag{2.13}$$

in order to write (2.12) in divergence form with respect to the reference coordinates

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{F} = \frac{1}{J} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}. \tag{2.14}$$

It still remains to transform the gradient $\boldsymbol{\nabla}_{\boldsymbol{x}} u$ of the state vector, which we will discuss in Section 2.3 introducing a special treatment for the terms of second order. Hence, we can write at least for purely hyperbolic problems the conservation law in reference space as

$$J(\boldsymbol{\xi}) u_t(\boldsymbol{X}(\boldsymbol{\xi}), t) + \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}(u(\boldsymbol{X}(\boldsymbol{\xi}), t)) = 0 \qquad \text{in } E \times \mathbb{R}^+. \tag{2.15}$$

In the following investigations we omit the dependencies of the state vector for simplification.

2.3 Discontinuous Galerkin Formulation

Discontinuous Galerkin methods are based on the weak form of equation (2.15) determining the differentiability requirements on the approximate solution. Following the FE approach, we can see that the mass matrix in the discontinuous approximation is local for every grid cell and hence easily to invert, whereas in the continuous approach the mass matrix shows global dependencies. The element-local form with single coupling between von Neumann neighbors, i.e. to adjacent elements makes the discontinuous Galerkin finite element formulation very unique and attractive in terms of high parallel computing. The discrete solution space for discontinuous Galerkin methods, in which we seek a piecewise smooth numerical approximation can be expressed as

$$W := \{u \in L^2(\Omega \times \mathbb{R}^+; \mathbb{R}^{n_{var}}) \mid u(.,t)|_E \in \mathbb{P}_N(E)\}, \tag{2.16}$$

where $\mathbb{P}_N(E)$ denotes the space of polynomials defined on E with order up to N. To demonstrate the approach, we first focus on hyperbolic problems. Later in this section, we will also discuss how to treat second order terms.

We derive the general discontinuous Galerkin formulation, by the local weak formulation of the fundamental equations (2.15). Therefor, we multiply equation (2.15) by a smooth test function $\phi = \phi(\boldsymbol{\xi}) \in \mathbb{R}$ and integrate over the reference element E

$$\int_E (J u_t + \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}})\, \phi \, d\boldsymbol{\xi} = 0. \tag{2.17}$$

In order to obtain the weak discontinuous Galerkin formulation, we employ spatial integration by parts for the divergence of the flux resulting in

$$\underbrace{\int_E J u_t \phi \, d\boldsymbol{\xi}}_{\text{time integral}} - \underbrace{\int_E \boldsymbol{\mathcal{F}} \cdot \boldsymbol{\nabla}_{\boldsymbol{\xi}} \phi \, d\boldsymbol{\xi}}_{\text{volume integral}} + \underbrace{\int_{\partial E} (\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}})^* \phi \, d\boldsymbol{S}}_{\text{surface integral}} = 0, \tag{2.18}$$

where $\boldsymbol{\mathcal{N}} \in \mathbb{R}^d$ denotes the outward pointing unit normal vector of the reference element faces. Since the numerical approximation can be discontinuous across the element boundaries, the normal component of the flux $\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}}$ is not uniquely defined on the surface ∂E. Thus, we replaced it in (2.18) by an approximating numerical flux function

$$\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}} \approx (\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}})^*, \tag{2.19}$$

denoted by the superscript *.

Surface Normal The normal flux function in reference space can be expressed by the normal unit vector $\boldsymbol{n}$ in physical space using equation (2.13)

$$\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}} = \sum_{i=1}^{3} \sum_{d=1}^{3} J a_d^i F_d \mathcal{N}_i = \sum_{d=1}^{3} \left(\sum_{i=1}^{3} J a_d^i \mathcal{N}_i \right) F_d \tag{2.20}$$

$$:= \sum_{d=1}^{3} \tilde{n}_d F_d = (\boldsymbol{F} \cdot \boldsymbol{n}) |\tilde{\boldsymbol{n}}|, \tag{2.21}$$

where the non-normalized normal vector $\tilde{\boldsymbol{n}}$ in physical space is computed by means of the metric terms (2.6) evaluated at the surface $\boldsymbol{\xi}^s \in \partial E$

$$\tilde{\boldsymbol{n}}(\boldsymbol{X}(\boldsymbol{\xi}^s)) = \sum_{i=1}^{3} J \boldsymbol{a}^i(\boldsymbol{\xi}^s) \mathcal{N}_i, \quad \text{so that } \boldsymbol{n} = \frac{\tilde{\boldsymbol{n}}}{|\tilde{\boldsymbol{n}}|} := \frac{\tilde{\boldsymbol{n}}}{\hat{s}}. \tag{2.22}$$

Defining the norm of $\tilde{\boldsymbol{n}}$ as the surface element $\hat{s}$, the numerical flux function can be written in terms of quantities defined in physical space as

$$(\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}})^* = (\boldsymbol{F} \cdot \boldsymbol{n})^* \hat{s} := F^* \hat{s}. \tag{2.23}$$

Numerical Flux As mentioned above, the discontinuity across the grid cell interfaces needs to be handled in a specific way. The volume integral in the weak discontinuous Galerkin formulation (2.18) contains the smooth part, i.e the polynomial approximation within the element. In contrast, the normal flux function on the surface required by the surface integral is not uniquely defined. The same issue occurs in Finite Volume methods, which brings us to the application of the well-known Riemann solvers.

When considering hyperbolic-parabolic conservation laws, we split again as done in Section 2.1 the flux function in an advection and diffusion flux as $\boldsymbol{F} = \boldsymbol{F}^A - \boldsymbol{F}^D$ and introduce for each summand Riemann fluxes

$$F_A^* = F_A^*(u_L, u_R, \boldsymbol{n}) \tag{2.24}$$

$$F_D^* = F_D^*(u_L, u_R, \boldsymbol{\nabla}_{\boldsymbol{\xi}} u_L, \boldsymbol{\nabla}_{\boldsymbol{\xi}} u_R, \boldsymbol{n}), \tag{2.25}$$

such that $F^* = F_A^* - F_D^*$. Each Riemann flux depends only on the physical outer normal $\boldsymbol{n}$, the value from the adjacent grid element u_R and on the value from the considered element u_L on the common interface ∂E. In the case of parabolic terms, the Riemann flux depends also on the gradients of the state vector u_L and u_R. The Riemann flux F_A^* is generally given by a Riemann solver from the Finite Volume framework. Plugging the definition of the numerical flux function (2.23) into equation (2.18), the weak discontinuous Galerkin form reads as

$$\int_E J u_t \phi \, d\boldsymbol{\xi} - \int_E \boldsymbol{\mathcal{F}} \cdot \boldsymbol{\nabla}_{\boldsymbol{\xi}} \phi \, d\boldsymbol{\xi} + \int_{\partial E} (F^* \hat{s}) \phi \, d\boldsymbol{S} = 0. \tag{2.26}$$

The approximation of the diffusion flux has to be done in a different way as discussed in the next paragraph. As we can see, the only information exchange between elements is given by the numerical flux function. Hence, the choice of the approximation of the surface flux is of major importance according to the stability and consistency of the discontinuous Galerkin method. Within the used numerical framework FLEXI, several Riemann solvers are implemented, such as the trivial local Lax-Friedrich/Rusanov flux [106], some approximate Riemann solvers of Godunov-type [44] as Roe solvers [105] including an entropy fix from Harten and Hyman [57] and solvers of HLL-type [58, 112]. An extensive overview of Riemann solvers is given in [111].

Second Order Equations As mentioned in Section 2.2.2, we still have to transform the solution gradient $\boldsymbol{\nabla}_{\boldsymbol{x}} u$ in equation (2.11) to the reference space in the case of parabolic terms occurring in the Navier-Stokes equations. Second derivatives for diffusive problems have to be treated in a special way. When employing only local polynomial derivatives of the solution inside the elements, Bassi and Rebay showed in [9] that discontinuous Galerkin methods become unstable. Therefor, the information of the surrounding elements is needed to obtain a stable discontinuous approximation. The stabilization strategy of the solution gradients is called lifting. Several lifting methods are available for DG schemes. One of the earliest methods is the interior penalty scheme [1, 2] initially developed for FE methods. Two of the first DG-based methods BR1 and BR2 from Bassi and Rebay [9, 12] are very popular since they use only data from the von Neumann neighbor cells. The local discontinuous Galerkin (LDG) scheme from Cockburn and Shu [34] suffers from an extended stencil. Persson and Perraire introduced in [97] a remedy for the drawback of LDG, namely the compact DG (CDG). For a good overview we refer to Arnoldi et al. [3, 4] comparing different lifting procedures for elliptic problems.

In this work, we focus on the BR2 lifting, as we look for a method with compact stencil optimized for implicit time marching schemes. The stability procedure for second order derivatives consists of computing the gradient by means of the neighboring elements as well as of the right choice of the numerical flux function F_D^*.

For computing the gradient, we rewrite the second order problem (2.2) into a first order system

$$u_t + \boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{F}(u, \boldsymbol{q}) = 0, \tag{2.27}$$

$$\boldsymbol{q} = \boldsymbol{\nabla}_{\boldsymbol{x}} u. \tag{2.28}$$

by introducing an additional variable $\boldsymbol{q} \in \mathbb{R}^{d \times n_{var}}$ with components $q^d = \frac{\partial u}{\partial x_d} \in \mathbb{R}^{n_{var}}$. The 'new' system representing an extension of (2.2) by $d \cdot n_{var}$ additional equations. It has to be solved for the unknowns $(u, \boldsymbol{q}) \in \mathbb{R}^{n_{var}+(d \cdot n_{var})}$. Now, we want to apply the same approach as described above for deriving the discontinuous Galerkin formulation for second order equations. We transform the system into the reference space by employing the conservative derivative operators (2.9) and (2.10) in the three-dimensional case

$$u_t + \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}(u, \boldsymbol{q}) = 0, \tag{2.29}$$

$$\boldsymbol{q} = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i u \right), \tag{2.30}$$

where we used the same formulation as in (2.15) for the first equation. The second equation can be decomposed in each space direction d and then simplified by

$$q^d - \frac{1}{J} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{U}}^d = 0, \tag{2.31}$$

$$\text{with } \boldsymbol{\mathcal{U}}^d(\boldsymbol{\xi}) := ((Ja)_d^1, (Ja)_d^2, (Ja)_d^3)^T \; u(\boldsymbol{X}(\boldsymbol{\xi})) \in \mathbb{R}^{d \times n_{var}}, \tag{2.32}$$

where $\boldsymbol{\mathcal{U}}$ represents the transformed state vector in reference space similar to the flux function $\boldsymbol{\mathcal{F}}$ in (2.13). Hence, the gradient $\boldsymbol{q}$ is now given in reference coordinates, so that equation (2.29) is well-defined.

Following now the same strategy as for the purely hyperbolic equation system, we obtain the weak DG formulation for the remaining equation (2.31)

$$\int_E J q^d \phi \, d\boldsymbol{\xi} + \int_E \boldsymbol{\mathcal{U}}^d \cdot (\boldsymbol{\nabla}_{\boldsymbol{\xi}} \phi) \, d\boldsymbol{\xi} - \int_{\partial E} (\boldsymbol{\mathcal{U}}^d \cdot \boldsymbol{\mathcal{N}})^* \phi \, d\boldsymbol{S} = 0. \tag{2.33}$$

Again the flux $\boldsymbol{\mathcal{U}}^d \cdot \boldsymbol{\mathcal{N}}$ is not uniquely defined on the surface, so that we introduced a numerical flux function $(\boldsymbol{\mathcal{U}}^d \cdot \boldsymbol{\mathcal{N}})^*$. With definition (2.22) of the unit normal vector $\boldsymbol{n}$ related to the physical element evaluated at the reference surface ∂E, the numerical flux can be rewritten as

$$(\boldsymbol{\mathcal{U}}^d \cdot \boldsymbol{\mathcal{N}})^* = (u n_d)^* \hat{s} := u^* n_d \hat{s}. \tag{2.34}$$

The final weak DG formulation of the gradient equation reads

$$\int_E J q^d \phi \, d\boldsymbol{\xi} + \int_E \boldsymbol{\mathcal{U}}^d \cdot (\boldsymbol{\nabla}_{\boldsymbol{\xi}} \phi) \, d\boldsymbol{\xi} - \int_{\partial E} u^* n_d \hat{s} \phi \, d\boldsymbol{S} = 0. \tag{2.35}$$

The purpose of the different lifting methods is now to define the numerical fluxes

$$u^* = u^*(u_L, u_R), \tag{2.36}$$

$$F_D^* = F_D^*(u_L, u_R, \boldsymbol{\nabla}_{\boldsymbol{\xi}} u_L, \boldsymbol{\nabla}_{\boldsymbol{\xi}} u_R, \boldsymbol{n}). \tag{2.37}$$

For the following derivations, we introduce the arithmetic mean value operator $\{.\}$ and the jump operator $\langle . \rangle$ for a double-valued scalar $w \in \mathbb{R}$ with the left and right value $w_{L/R}$ on an interface ∂E_k as

$$\{w\} = \frac{1}{2}\left(w_L + w_R\right), \qquad \langle w \rangle = w_R - w_L.$$

For vector-valued functions, these operators has to be applied componentwise.
Bassi and Rebay introduced for the solution trace the arithmetic mean value

$$u^* = \{u\},$$

completing the discretization of the gradient $\boldsymbol{\nabla}_{\boldsymbol{\xi}} u$ within the volume of E. The numerical approximation for the diffusion flux function is in both methods, BR1 and BR2, chosen as

$$F_D^*(u_L, u_R, \boldsymbol{\nabla}_{\boldsymbol{\xi}} u_L, \boldsymbol{\nabla}_{\boldsymbol{\xi}} u_R, \boldsymbol{n}) = \{\boldsymbol{F}^D\} \cdot \boldsymbol{n}. \tag{2.38}$$

The question we have to focus on now is how to compute the surface gradients $\boldsymbol{\nabla}_{\boldsymbol{\xi}} u_L$ and $\boldsymbol{\nabla}_{\boldsymbol{\xi}} u_R$.

The BR1 scheme [9] employs for the gradients $\boldsymbol{\nabla}_{\boldsymbol{\xi}} u|_{\partial E_k}$ on the sub-surface $\partial E_k \subset \partial E$ ($\partial E = \bigcup_k \partial E_k$) needed for F_D^*

$$\boldsymbol{\nabla}_{\boldsymbol{x}} u|_{\partial E_k} = \boldsymbol{q}|_{\partial E_k},$$

simply an evaluation of $\boldsymbol{q}$ on the sub-surface. This implies that the computation of the gradient on the specific sub-surfaces ∂E_k contains also the data from the neighbors adjacent to the remaining sub-surfaces of ∂E, since the surface integral in (2.35) includes all sub-surfaces. Consequently the viscous numerical flux F_D^* depends on non-local data which employs a strongly coupled system. This effect is not desirable for implicit time stepping schemes.

In contrast, for the BR2 scheme we consider the strong form of (2.35) which is found by using backward integration by parts whereas the solution u is taken only from inside the cell, marked with index $\cdot_L$

$$\int_E \left(Jq^d - \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{U}}^d \right) \phi \, d\boldsymbol{\xi} - \int_{\partial E} (u^* - u_L) n_d \hat{s} \phi \, d\boldsymbol{S} = 0. \tag{2.39}$$

In this formulation the strong gradient $\boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{U}}^d$ is included, such that the volume integral represents the residual of the gradient equation and the surface integral specifies a penalty term which is here called lifting term. This penalty term vanishes for globally continuous solutions. As we mentioned before, attempts by using the element-local polynomial derivative fail due to instability. This lifting term depicts the stabilization of DG schemes for diffusive problems. The volume gradient q consists thus of the local derivative plus a term depending on the jump $\langle u \rangle$. Defining the numerical function

$$\boldsymbol{H}^* = \boldsymbol{H}^*(u_L, u_R, \boldsymbol{n}) := (u^* - u_L)\boldsymbol{n} = \frac{1}{2}\langle u \rangle \boldsymbol{n}, \tag{2.40}$$

the gradient on the sub-surface ∂E_k is here given by the evaluation on the interface ∂E_k of the strong gradient plus a local surface flux $[H_d^* \hat{s}]^{\partial E_k}$ over the surface ∂E_k denoted by the superscript $\cdot^{\partial E_k}$ in the weak sense

$$(\boldsymbol{\nabla}_{\boldsymbol{x}} u)_d \big|_{\partial E_k} = \frac{1}{J} \left(\boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{U}}^d + \eta_{\text{BR2}} [H_d^* \hat{s}]^{\partial E_k} \right) \bigg|_{\partial E_k}. \tag{2.41}$$

The penalty parameter η_{BR2} is a positive constant which has to be set to the number of faces in order to guarantee the stability of the diffusion operator [21]. The surface flux consists for BR2 only of the flux over the considered sub-surface compared to BR1 using the flux over all sub-surfaces. In this way the gradient on the surface depends only on the data from the two adjacent neighbors, which is the reason why we choose BR2 for the implicit DG scheme. The overhead compared to BR1 in computing the local strong gradients on the surface can be reduced by efficient implementation as considered in [63].

2.4 Spatial Discretization

The next steps after the derivation of the weak formulation (2.18) are to approximate the solution and flux by polynomials in space and replace the spatial integral by a numerical quadrature in order to derive the semi discrete DG formulation. We focus on the derivation of the DGSEM, a special variant of nodal discontinuous Galerkin schemes, using tensor-product basis functions with a nodal character. The approach of spectral collocation, using the same points for interpolation and integration, makes the scheme significantly efficient. We follow the derivation in [65].
First, we consider again purely hyperbolic problems. In Section 2.4.7 the derivation is extended to hyperbolic-parabolic equations.

2.4.1 Polynomial Approximation and Numerical Quadrature

As we derive the scheme generally on the reference element, we construct one-dimensional polynomials on the reference interval $[-1,1]$. Therefor, we define a set of $(N+1)$ interpolation nodes

$$\{\xi_i\}_{i=0}^N \subset [-1,1]. \tag{2.42}$$

on the reference interval. Within nodal discontinuous Galerkin schemes, the basis functions are defined by means of the one-dimensional Lagrange interpolation polynomials ℓ_i of degree N given by

$$\ell_i(\xi) = \prod_{\substack{k=0\\k\neq i}}^{N} \frac{\xi - \xi_k}{\xi_i - \xi_k}. \tag{2.43}$$

The Lagrange interpolation polynomials satisfy the Lagrange property

$$\ell_i(\xi_j) = \delta_{ij}, \quad i,j = 0,\dots,N. \tag{2.44}$$

Hence, a scalar function f can be approximated by the nodal approach as

$$f(\xi) \approx \sum_{i=0}^{N} f_i \ell_i(\xi), \tag{2.45}$$

where the nodal coefficients or degrees of freedom (DOF) f_i hold $f_i = f(x_i)$ due to the Lagrange property.
In the following, we evaluate integrals on discrete nodes by means of numerical quadrature rules. We use Gauss-Legendre quadratures $\{\eta_i, \omega_i\}_{i=0}^N$ consisting of points η_i in the interval $[-1, 1]$ and weights ω_i such that

$$\int_{-1}^{1} f(\eta)\, d\eta \approx \sum_{i=0}^{N} f(\eta_i)\omega_i. \tag{2.46}$$

Choosing interpolation nodes identically to quadrature nodes ($\xi_i = \eta_i$ for $i = 0, \ldots, N$) leads to very efficient collocation methods. Note that Gauss-type quadrature rules are well-suited for polynomials. Gauss-Legendre quadratures with N nodes integrate polynomials of degree up to $2N-1$ exactly.

2.4.2 Approximation of the Solution and the Fluxes

We want to approximate the quantities in the conservation law (2.15) in reference space by a polynomial tensor-product basis. We restrict the derivation again to three dimensions, but it can be similarly constructed for two dimensions. The three-dimensional basis $\{\psi_{ijk}\}_{i,j,k=0}^N$ for DGSEM consists of a tensor-product of one-dimensional Lagrange polynomials ℓ_i defined in (2.43) of degree N in each space direction, as

$$\psi_{ijk}(\boldsymbol{\xi}) = \ell_i(\xi^1)\ell_j(\xi^2)\ell_k(\xi^3), \qquad i, j, k = 0, \ldots, N, \tag{2.47}$$

with $\boldsymbol{\xi} = (\xi^1, \xi^2, \xi^3)^T \in E = [-1, 1]^3$. The three-dimensional interpolation nodes $\boldsymbol{\xi}_{ijk}$ are also chosen as the tensor-product of $N+1$ Gauss points $\{\xi_i\}_{i=0}^N$ in each direction

$$\boldsymbol{\xi}_{ijk} = (\xi^1_{ijk}, \xi^2_{ijk}, \xi^3_{ijk})^T = (\xi_i, \xi_j, \xi_k)^T.$$

In Figure 2.1, the distribution of the two-dimensional tensor-product interpolation nodes within the reference element is depicted. The number of degrees of freedom within a d-dimensional tensor-product element is thus given by

$$(N+1)^d. \tag{2.48}$$

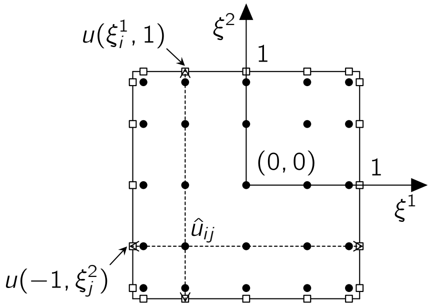


Figure 2.1: Distribution of the Gauss points • in the volume and the nodes for the boundary fluxes ▫ at the interface of the reference element in two space dimensions for a polynomial degree of $N = 4$.

The solution inside each hexahedral element is approximated in space as

$$u(\boldsymbol{X}(\boldsymbol{\xi}), t) \approx u(\boldsymbol{\xi}, t) = \sum_{i,j,k=0}^{N} \hat{u}_{ijk}(t)\psi_{ijk}(\boldsymbol{\xi}), \tag{2.49}$$

where $\hat{u}_{ijk}(t)$ are the nodal degrees of freedom representing the solution u evaluated at the node $\boldsymbol{\xi}_{ijk}$ due to the Lagrange property. The composition $u \circ \boldsymbol{X}$ is again written as u for simplification. Analogously, we approximate each component of the contravariant fluxes $\mathcal{F}^m$ as

$$\mathcal{F}^m(u) \approx \sum_{i,j,k=0}^{N} \hat{\mathcal{F}}^m_{ijk}\psi_{ijk}(\boldsymbol{\xi}), \qquad \text{with } m = 1, \dots, d \tag{2.50}$$

where $\hat{\mathcal{F}}^m_{ijk}$ is the nodal interpolation of the flux component $\mathcal{F}^m$ at the ijk-th node. It is computed from the physical flux $\boldsymbol{F}$ using equation (2.13)

$$\hat{\mathcal{F}}^m_{ijk} = \mathcal{F}^m(\hat{u}_{ijk}) = \sum_{d=1}^{3} (Ja)^m_d(\xi^1_i, \xi^2_j, \xi^3_k) F_d(\hat{u}_{ijk}). \tag{2.51}$$

Considering the surface fluxes on element interfaces, we also need to evaluate the solution at element boundaries. Since the Gauss nodes are only distributed inside of the element as illustrated in Figure 2.1, the solution has to be extrapolated on the element boundaries. Due to the tensor-product ansatz, this 'prolongation' can be computed one-dimensionally, as for the $\xi^1 = 1$ boundary, along the ξ^1-lines

$$u^{+\xi^1} = u(1, \xi^2, \xi^3, t) = \sum_{i,j,k=0}^{N} \hat{u}_{ijk}(t)\ell_i(1)\ell_j(\xi^2)\ell_k(\xi^3), \tag{2.52}$$

as depicted in Figure 2.1, where the superscript $+\xi^1$ denotes the evaluation on the $\xi^1 = 1$ face. Evaluating the solution $u^{+\xi^1}$ at the surface points $(1, \xi^2_j, \xi^3_k)$ on the face $+\xi^1$ yields then due to the Lagrange property

$$\begin{aligned} u^{+\xi^1}_{jk} &= \sum_{m,n,o=0}^{N} \hat{u}_{mno}(t)\ell_m(1)\underbrace{\ell_n(\xi^2_j)}_{\delta_{nj}}\underbrace{\ell_o(\xi^3_k)}_{\delta_{ok}} \\ &= \sum_{m=0}^{N} \hat{u}_{mjk}(t)\ell_m(1). \end{aligned} \tag{2.53}$$

These boundary states are required for the calculation of the surface integral using Riemann solvers.
Note that the analytical flux function in the case of the compressible Navier-Stokes equations is non-linear with respect to the solution, as well as the metric terms, so that we introduce not only truncation errors in the finite polynomial interpolation of the flux in (2.50), but also aliasing errors of higher order terms [24].

Now, we can insert the polynomial approximations of the solution and the contravariant flux function into the weak formulation (2.18) and approximate the integrals by the Gauss quadrature rule in order to obtain the spatial operator of the semi-discrete form of the DGSEM. Due to the collocation of the interpolation and the integration nodes the computations are significantly simplified. As in all Galerkin methods, we set the test functions as the polynomial basis functions. In the next three sections, we approximate successively each integral within the weak discontinuous formulation (2.18).

2.4.3 Time Derivative Integral

We consider the first integral of the weak discontinuous formulation (2.18) representing the time derivative integral. Inserting the polynomial approximation of the solution (2.49) and choosing the test function as the basis functions, $\phi = \psi_{ijk}$, leads to

$$\frac{\partial}{\partial t}\int_E Ju\phi \, d\boldsymbol{\xi} \approx \frac{\partial}{\partial t}\int_E J(\boldsymbol{\xi})\left(\sum_{m,n,o=0}^{N} \hat{u}_{mno}(t)\psi_{mno}(\boldsymbol{\xi})\right)\psi_{ijk}(\boldsymbol{\xi}) \, d\boldsymbol{\xi},$$

$$\text{for } i,j,k = 0,\dots N.$$

Then we split the integration over the reference element into the three coordinate directions, where we approximate each integral by the Gauss quadrature. Replacing the ansatz function by the tensor-product definition (2.47) and using the Lagrange property yields

$$= \frac{\partial}{\partial t}\iiint_{-1}^{1} J(\boldsymbol{\xi})\left(\sum_{m,n,o=0}^{N} \hat{u}_{mno}(t)\psi_{mno}(\boldsymbol{\xi})\right)\psi_{ijk}(\boldsymbol{\xi}) \, d\xi^1 d\xi^2 d\xi^3$$

$$\approx \frac{\partial}{\partial t}\sum_{p,q,r=0}^{N} J(\boldsymbol{\xi}_{pqr})\left(\sum_{m,n,o=0}^{N} \hat{u}_{mno}(t)\,\underbrace{\ell_m(\xi_p^1)}_{\delta_{mp}}\,\underbrace{\ell_n(\xi_q^2)}_{\delta_{nq}}\,\underbrace{\ell_o(\xi_r^3)}_{\delta_{or}}\right)\psi_{ijk}(\boldsymbol{\xi}_{pqr})$$

$$\omega_p\omega_q\omega_r$$

$$= \frac{\partial}{\partial t}\sum_{p,q,r=0}^{N} J(\boldsymbol{\xi}_{pqr})\hat{u}_{pqr}(t)\psi_{ijk}(\boldsymbol{\xi}_{pqr})\omega_p\omega_q\omega_r.$$

Applying again the tensor-product basis to the test function

$$= \frac{\partial}{\partial t}\sum_{p,q,r=0}^{N} J(\boldsymbol{\xi}_{pqr})\hat{u}_{pqr}(t)\,\underbrace{\ell_i(\xi_p^1)}_{\delta_{ip}}\,\underbrace{\ell_j(\xi_q^2)}_{\delta_{jq}}\,\underbrace{\ell_k(\xi_r^3)}_{\delta_{kr}}\,\omega_p\omega_q\omega_r,$$

we obtain the final approximation

$$\frac{\partial}{\partial t}\int_E Ju\phi \, d\boldsymbol{\xi} \approx J(\boldsymbol{\xi}_{ijk})\frac{\partial \hat{u}_{ijk}}{\partial t}\omega_i\omega_j\omega_k. \tag{2.54}$$

Note, the Gauss integration with $N+1$ points in each direction is exact for Jacobians J which are at most linear, hence for only linear and bilinear deformed hexahedra. For trilinear and fully curved elements an integration error is introduced, commonly termed as geometrical aliasing [87].
Defining the global mass matrix

$$M_{ijk,mno} := \int_E \psi_{ijk}(\boldsymbol{\xi})\psi_{mno}(\boldsymbol{\xi})\; d\boldsymbol{\xi},$$

we obtain due to the collocation of the interpolation and integration node and the Lagrange property

$$M_{ijk,mno} = \delta_{im}\delta_{jn}\delta_{ko}\omega_i\omega_j\omega_k$$

for $i,j,k,m,n,o = 0,\dots,N$. Thus, we can write the time derivative integral in matrix vector form

$$\frac{\partial}{\partial t}\int_E Ju\phi\; d\boldsymbol{\xi} = \frac{\partial}{\partial t}\boldsymbol{M}\boldsymbol{J}\hat{\boldsymbol{u}}$$

Since the global mass matrix $\boldsymbol{M}$ of the DGSEM is a diagonal matrix, computing the solution $\hat{\boldsymbol{u}}$ by multiplying the inverse of $\boldsymbol{M}$ requires significantly less operations than a non-collocation method consisting of a full mass matrix.

2.4.4 Volume Integral

The volume integral of the weak formulation (2.18) consists of the scalar product concerning the contravariant flux vector, which can be written as the sum of the three components

$$\int_E \boldsymbol{\mathcal{F}}\cdot\boldsymbol{\nabla}_{\boldsymbol{\xi}}\phi\; d\boldsymbol{\xi} = \sum_{d=1}^{3}\int_E \mathcal{F}^d\frac{\partial\phi}{\partial\xi^d}\; d\boldsymbol{\xi}.$$

We focus on the derivation for the first summand w.l.o.g, since the other components can be treated in the same way. Inserting the flux approximation (2.50) and setting the test function as the polynomial basis yields

$$\int_E \mathcal{F}^1\frac{\partial\phi}{\partial\xi^1}\; d\boldsymbol{\xi} \approx \iiint_{-1}^{1}\left(\sum_{m,n,o=0}^{N}\hat{\mathcal{F}}^1_{mno}\psi_{mno}(\boldsymbol{\xi})\right)\frac{\partial\psi_{ijk}(\boldsymbol{\xi})}{\partial\xi^1}\; d\boldsymbol{\xi}.$$

Approximating the integrals by Gauss quadrature using the tensor-product basis and the Lagrange property reads

$$\begin{aligned}
&\int_E \mathcal{F}^1 \frac{\partial \phi}{\partial \xi^1}\, d\boldsymbol{\xi} \\
&\approx \sum_{p,q,r=0}^{N} \left(\sum_{m,n,o=0}^{N} \hat{\mathcal{F}}^1_{mno} \underbrace{\ell_m(\xi^1_p)}_{\delta_{mp}} \underbrace{\ell_n(\xi^2_q)}_{\delta_{nq}} \underbrace{\ell_o(\xi^3_r)}_{\delta_{or}} \right) \frac{\partial \psi_{ijk}(\boldsymbol{\xi}_{pqr})}{\partial \xi^1} \omega_p \omega_q \omega_r \\
&= \sum_{p,q,r=0}^{N} \hat{\mathcal{F}}^1_{pqr} \frac{\partial \ell_i(\xi^1_p)}{\partial \xi^1} \underbrace{\ell_j(\xi^2_q)}_{\delta_{jq}} \underbrace{\ell_k(\xi^3_r)}_{\delta_{kr}} \omega_p \omega_q \omega_r \\
&= \omega_j \omega_k \sum_{p=0}^{N} \hat{\mathcal{F}}^1_{pjk} \frac{\partial \ell_i(\xi^1_p)}{\partial \xi^1} \omega_p .
\end{aligned} \tag{2.55}$$

Defining the one-dimensional differentiation matrix

$$D_{ij} = \frac{\partial \ell_j(\xi_i)}{\partial \xi}, \qquad \text{for } i, j = 0, \ldots, N, \tag{2.56}$$

we can write the last equation in (2.55) as a one-dimensional matrix vector product along the ξ^1-direction. Computing the remaining summands similarly, we obtain for the volume integral

$$\begin{aligned}
\int_E \boldsymbol{\mathcal{F}} \cdot \nabla_{\boldsymbol{\xi}} \phi \, d\boldsymbol{\xi} \approx \quad & \omega_j \omega_k \sum_{p=0}^{N} \hat{\mathcal{F}}^1_{pjk} D_{pi} \omega_p \\
& + \omega_i \omega_k \sum_{q=0}^{N} \hat{\mathcal{F}}^2_{iqk} D_{qj} \omega_q \\
& + \omega_i \omega_j \sum_{r=0}^{N} \hat{\mathcal{F}}^3_{ijr} D_{rk} \omega_r .
\end{aligned} \tag{2.57}$$

2.4.5 Surface Integral

For the surface integral in (2.18), we split the integral over the boundary of the reference element into the sum of each element face where we gather the two parallel faces evaluated at $\xi^1 = \{-1, 1\}, \xi^2 = \{-1, 1\}$ and $\xi^3 = \{-1, 1\}$. Using the definition (2.23) of the numerical flux function, we obtain

$$\begin{aligned}\int_{\partial E} (\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}})^* \phi \, d\boldsymbol{\xi}^s = & \left[\iint_{-1}^{1} (F^* \hat{s}) \phi \, d\xi^2 d\xi^3 \right]_{\xi^1=-1}^{1} \\ & + \left[\iint_{-1}^{1} (F^* \hat{s}) \phi \, d\xi^1 d\xi^3 \right]_{\xi^2=-1}^{1} \\ & + \left[\iint_{-1}^{1} (F^* \hat{s}) \phi \, d\xi^1 d\xi^2 \right]_{\xi^3=-1}^{1} .\end{aligned} \tag{2.58}$$

We exemplify the derivation only for one summand, $\xi^1 = 1$, the other summands are computed in the same way. Again, we approximate the numerical flux function by interpolation but on the Gauss nodes located on the corresponding element face as illustrated in Figure 2.1

$$[F^* \hat{s}](1, \xi^2, \xi^3) \approx \sum_{n,o=0}^{N} [F^*(u_L, u_R, \boldsymbol{n}) \hat{s}]_{no}^{+\xi^1} \ell_n(\xi^2) \ell_o(\xi^3), \tag{2.59}$$

where the evaluation on the interface $\xi^1 = 1$ is indicated by the superscript $+\xi^1$. The boundary states u_L and u_R and the physical normal vector $\boldsymbol{n}$ are related to the face $+\xi^1$ and have to be evaluated on the n, o-th Gauss point. As computed in (2.52), the solution is required on both sides of $+\xi^1$. We insert the boundary flux approximation (2.59) and the polynomial tensor-product basis for the test function into the surface integral (2.58) at $+\xi^1$ and use the Gauss quadrature as well as the Lagrange property, so that

$$\begin{aligned}& \iint_{-1}^{1} (F^* \hat{s}) \phi \, d\xi^2 d\xi^3 \bigg|_{\xi^1=1} \\ & \approx \sum_{p,q=0}^{N} \left(\sum_{n,o=0}^{N} [F^* \hat{s}]_{no}^{+\xi^1} \underbrace{\ell_n(\xi_p^2)}_{\delta_{np}} \underbrace{\ell_o(\xi_q^3)}_{\delta_{oq}} \right) \ell_i(1) \ell_j(\xi_p^2) \ell_k(\xi_q^3) \omega_p \omega_q\end{aligned}$$

$$
\begin{aligned}
&= \sum_{p,q=0}^{N} [F^* \hat{s}]_{pq}^{+\xi^1} \ell_i(1) \underbrace{\ell_j(\xi_p^2)}_{\delta_{jp}} \underbrace{\ell_k(\xi_q^3)}_{\delta_{kq}} \omega_p \omega_q \\
&= [F^* \hat{s}]_{jk}^{+\xi^1} \ell_i(1) \omega_j \omega_k .
\end{aligned}
$$

We end up with the total surface integral by applying the previous steps analogously to the remaining sides

$$
\begin{aligned}
\int_{\partial E} (\boldsymbol{\mathcal{F}} \cdot \boldsymbol{\mathcal{N}})^* \phi \, d\boldsymbol{\xi}^s \approx \quad & \left([F^* \hat{s}]_{jk}^{+\xi^1} \ell_i(1) - [F^* \hat{s}]_{jk}^{-\xi^1} \ell_i(-1) \right) \omega_j \omega_k \\
+ & \left([F^* \hat{s}]_{ik}^{+\xi^2} \ell_j(1) - [F^* \hat{s}]_{ik}^{-\xi^2} \ell_j(-1) \right) \omega_i \omega_k \\
+ & \left([F^* \hat{s}]_{ij}^{+\xi^3} \ell_k(1) - [F^* \hat{s}]_{ij}^{-\xi^3} \ell_k(-1) \right) \omega_i \omega_j .
\end{aligned} \tag{2.60}
$$

2.4.6 Semi-Discrete Formulation

In the previous sections, we have discretized all spatial integrals of the weak discontinuous Galerkin formulation (2.18), so that we can now put the rewritten time integral (2.54), the volume integral (2.57) and the surface integral (2.60) together to the semi-discrete form of the spatial operator. Defining the following one-dimensional operators, which can be precomputed as an initialization step,

$$
\hat{\ell}_i = \frac{\ell_i}{\omega_i}, \qquad \hat{D}_{ij} = -D_{ji} \frac{\omega_i}{\omega_j}, \qquad \text{for } i, j = 0, \dots, N,
$$

the time derivative of the nodal degrees of freedom within each element for DGSEM reads

$$
\begin{aligned}
\frac{\partial \hat{u}_{ijk}}{\partial t} = -\frac{1}{J_{ijk}} \Bigg[\quad & \sum_{p=0}^{N} \hat{\mathcal{F}}_{pjk}^1 \hat{D}_{ip} + \left([F^* \hat{s}]_{jk}^{+\xi^1} \hat{\ell}_i(1) - [F^* \hat{s}]_{jk}^{-\xi^1} \hat{\ell}_i(-1) \right) \\
& + \sum_{q=0}^{N} \hat{\mathcal{F}}_{iqk}^2 \hat{D}_{jq} + \left([F^* \hat{s}]_{ik}^{+\xi^2} \hat{\ell}_j(1) - [F^* \hat{s}]_{ik}^{-\xi^2} \hat{\ell}_j(-1) \right) \\
& + \sum_{r=0}^{N} \hat{\mathcal{F}}_{ijr}^3 \hat{D}_{kr} + \left([F^* \hat{s}]_{ij}^{+\xi^3} \hat{\ell}_k(1) - [F^* \hat{s}]_{ij}^{-\xi^3} \hat{\ell}_k(-1) \right) \Bigg] ,
\end{aligned} \tag{2.61}
$$

for $i, j, k = 0, \dots N$. The components $\hat{\mathcal{F}}^m$ of the contravariant fluxes are given as in (2.51) for $m = 1, 2, 3$. For computing the Riemann fluxes $F^* = F^*(u_L, u_R, \boldsymbol{n})$, we need to prolong the solution u defined on the volume Gauss nodes to the element interfaces as depicted in Figure 2.1. The evaluation on the surface nodes is again only a one-dimensional scalar product [79] as can be seen in (2.53) for the $+\xi^1$ face.

Equation (2.61) represents a semi-discrete formulation of the weak form, since the time is still continuous and only the space is discretized. Each line of the three-dimensional spatial operator corresponds to one-dimensional DG operators. The extension to higher space dimensions or the restriction to one or two dimension is thus straightforward. Thanks to the tensor-product structure on the mapped reference element we can benefit from the dimension-by-dimension splitting of the DG operator in conjunction with the collocation of the interpolation and integration, since it yields more efficiency due to less operations and less storage requirements in contrast to a full three-dimensional DG operator. Hence for DGSEM, the computation of the 3D operator is as simple as in one dimension.

As we can see in equation (2.61), the semi-discrete form of the governing equations (2.2) within DGSEM can be written as a system for all degrees of freedom of the size $l = (N+1)^d \cdot n_{var} \cdot n_{el}$ (omitting the superscript$\hat{}$ for simplicity)

$$\boldsymbol{u}_t = \boldsymbol{R}(\boldsymbol{u}, t) \qquad \text{in } [0, T], \tag{2.62}$$

where $\boldsymbol{u} \in \mathbb{R}^l$ denotes the vector of all DOFs and $\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{u}, t) \in \mathbb{R}^l$ the residual, termed in this work as the DG operator corresponding to the spatial discretization.

2.4.7 Second Order Approximation

In this Section, we derive the spatial DGSEM discretization of the solution gradient in the case of mixed hyperbolic-parabolic systems as the compressible Navier-Stokes equations. Due to the focus on implicit time integration schemes we consider the BR2 scheme [12] from Bassi and Rebay resulting in a smaller dependency as the commonly used BR1 scheme [9].
As discussed in Section 2.3, the solution derivatives $q^d = \partial_{x_d} u$ with

$d = 1, 2, 3$ in 3D are required on the volume Gauss nodes for the physical flux $\boldsymbol{F}^D$ as well as on the surface nodes for the numerical flux function F_D^*. First we derive the volume gradients, which are given for BR2 in the strong discontinuous Galerkin formulation of the gradient equation deduced in (2.39). We interpolate the function $\boldsymbol{\mathcal{U}}$ defined in (2.32) as the contravariant flux in (2.50) and the numerical flux $\boldsymbol{H}^*$ defined in (2.40) as in (2.59) for all surfaces. Analogously to the hyperbolic systems, we obtain the discretized gradient equation for the nodal coefficients $\hat{q}^d_{ijk}$ of each partial derivative

$$\begin{aligned} \hat{q}^d_{ijk} = \frac{1}{J_{ijk}} \Bigg[& \sum_{p=0}^{N} D_{ip} \hat{\mathcal{U}}^{d,1}_{pjk} + D_{jp} \hat{\mathcal{U}}^{d,2}_{ipk} + D_{kp} \hat{\mathcal{U}}^{d,3}_{ijp} \\ & + [H_d^* \hat{s}]^{+\xi^1}_{jk} \hat{\ell}_i(1) + [H_d^* \hat{s}]^{-\xi^1}_{jk} \hat{\ell}_i(-1) \\ & + [H_d^* \hat{s}]^{+\xi^2}_{ik} \hat{\ell}_j(1) + [H_d^* \hat{s}]^{-\xi^2}_{ik} \hat{\ell}_j(-1) \\ & + [H_d^* \hat{s}]^{+\xi^3}_{ij} \hat{\ell}_k(1) + [H_d^* \hat{s}]^{-\xi^3}_{ij} \hat{\ell}_k(-1) \Bigg]. \end{aligned} \tag{2.63}$$

As illustrated in (2.41), the gradient on the surface for BR2 consists of the interpolated volume contribution as well as only the local interpolated surface flux on the surface nodes, here e.g. on $+\xi^1$

$$\begin{aligned} [\hat{q}^d]^{+\xi^1}_{jk} = \sum_{i=0}^{N} \frac{1}{J_{ijk}} \Bigg[& \sum_{p=0}^{N} D_{ip} \hat{\mathcal{U}}^{d,1}_{pjk} + D_{jp} \hat{\mathcal{U}}^{d,2}_{ipk} + D_{kp} \hat{\mathcal{U}}^{d,3}_{ijp} \\ & + \eta_{\text{BR2}} [H_d^* \hat{s}]^{+\xi^1}_{jk} \hat{\ell}_i(1) \Bigg] \ell_i(1), \end{aligned} \tag{2.64}$$

where the penalty parameter η_{BR2} is chosen greater than the number of element faces for stability. The additional overhead compared to BR1 where only the stored volume gradients are interpolated to the surface, can be efficiently implemented with DGSEM as described in [63]. The first step is to compute only the volume integrals depicted in the first line of (2.63) and (2.64). In a second step, the local surface part is added for each face to the volume part for the volume gradient and the surface gradient at the same time. Whereas for the volume gradient, each local flux is summed up for each side in order to obtain the full surface integral as in (2.63). For the surface gradient, each sum of the volume integral and the local flux is stored with respect to the considered face and interpolated to the surface as in (2.64).

2.4.8 Scaling of the DG Operator

In order to examine the scaling of the DG operator, we estimate the number of operations (multiplications) for the d-dimensional Navier-Stokes equations per hexahedral grid element using collocation. In Table 2.3 the operation counts for a polynomial degree of N is illustrated. The number of degrees of freedom in DGSEM are $(N+1)^d$ on each hexahedra.

	number of operations
interpolation to surface for u and $\boldsymbol{q}$	$\mathcal{O}(N^{(d-1)}N)$
evaluate fluxes $\boldsymbol{F}(u, \boldsymbol{\nabla} u)$	$\mathcal{O}(N^d)$
transform fluxes $\boldsymbol{\mathcal{F}}$ and $\boldsymbol{\mathcal{U}}$	$\mathcal{O}(N^d)$
volume integral for u and $\boldsymbol{q}$	$\mathcal{O}(N^d N)$
surface integral for u and $\boldsymbol{q}$	$\mathcal{O}(N^d)$

Table 2.3: Scaling of the operation counts of the DGSEM operator for the polynomial degree N considering one hexahedral element.

The overall estimates present that the DG operator $\boldsymbol{R}$ scales with $\mathcal{O}(N^{d+1})$ for general dimension d. Hence the computational work per DOF is linear in N for hexahedra using DGSEM. In contrast, DG methods without tensor-product structure typically scale with $\mathcal{O}(N^d)$ per DOF [62].

2.5 Time Discretization

So far, the governing equations (2.2) are discretized in space by DGSEM resulting in equation (2.62), whereas the time is still a continuous quantity. The semi-discrete form represents an ordinary differential equation (ODE) for the time dependent polynomial coefficients. Thus, we can use any solver for initial value problems with a given initial solution in time in order to advance the solution numerically in time. We can employ different time integration schemes, as explicit, implicit or mixed explicit-implicit (IMEX) methods. The code FLEXI is basically equipped with explicit time integration methods described in the following section for the purpose of resolving highly unsteady problems very accurately. Within this work FLEXI is also extended in order to allow implicit time marching schemes, which is explained in Chapter 3.

2.5.1 Explicit Time Integration

Cockburn and Shu investigated in [31, 29, 28, 30] Runge-Kutta discontinuous Galerkin methods integrating the ODE in (2.62) in time by high order accurate explicit Runge-Kutta schemes. The most frequently used time stepping scheme in the open source code FLEXI is the explicit Runge-Kutta method of order four with five stages, termed ERK4, derived by Kennedy et al. [73]. The Butcher table can be found in Appendix B. ERK4 has optimized coefficients, especially deduced for the compressible Navier-Stokes equations. But also other explicit Runge-Kutta methods of order three and four are available in FLEXI. Explicit time integration schemes are simple to implement, computationally cheap related to each time step and are suitable for highly parallel computations [39]. However, they all suffer from severe time step restrictions depending on the spatial discretization, the physical element size and the maximum signal velocity.

Time Step Restriction The maximum stable time step for explicit schemes is given by the CFL condition by Courant, Friedrichs and Lewy [37]

$$\Delta t \leq \mathrm{CFL} \cdot \alpha_{RK}(N) \frac{\Delta x}{|\lambda^A|_{\max}(2N+1)} \qquad \text{for advection,} \tag{2.65}$$

where CFL is the CFL number, $\alpha_{RK}(N)$ a scaling factor of the Runge-Kutta method depending on the DG method and the polynomial degree N of the

spatial discretization, Δx denotes the minimum physical element size and $|\lambda^A|_{\max}$ the maximum eigenvalue of the Jacobian corresponding to the advection flux. Since the DOFs are not equidistantly distributed within a grid cell, the CFL number is additionally scaled by the factor $\frac{1}{2N+1}$ as provided in [33]. The CFL number has to be chosen less equal one in order to obtain a stable method. For the Euler equations the maximal signal velocity is given by the eigenvalues depending on the fluid velocity and the sound speed.
However, the time step restriction given in (2.65) can only be applied to hyperbolic systems or advection dominated problems for the mixed hyperbolic-parabolic Navier-Stokes equations. For the parabolic terms of the Navier-Stokes equations we have to employ the viscous time step restriction [55]

$$\Delta t \leq \text{CFL} \cdot \beta_{RK}(N) \frac{\Delta x^2}{|\lambda^D|_{\max}(2N+1)^2} \qquad \text{for diffusion.} \tag{2.66}$$

Here, β_{RK} denote the scaling factor for the diffusion part and $|\lambda^D|_{\max}$ the maximum eigenvalue of the diffusion matrix according to amount.
In order to obtain a stable method, the computational time step for the compressible Navier-Stokes equations has to be chosen as the minimum of both restrictions (2.65) and (2.66).

We can split the type of explicit time integrators into the global time stepping approach (GTS), where the time step is chosen as the minimum of all element-local time steps, and the local time stepping ansatz (LTS) [81, 32], where each cell has its own local time step. LTS schemes are more complex to implement than GTS and still suffer from the local CFL conditions. The open source code FLEXI employs only GTS due to the low implementation effort.

2.5.2 Implicit Time Integration

For fully implicit schemes the time step can be chosen arbitrarily large with no effect on the stability of the scheme, since they are unconditional stable. The time step is only restricted by physical meaningful conditions in order to resolve relevant phenomena and not by stability limits. Hence, only temporal accuracy requirements play a role, which makes implicit schemes attractive in the field of stiff problems and steady state simulations. Additionally, the time step does not depend on the grid size, which is favorable for grids with small cells capturing the geometry. However, the drawback for implicit schemes lies in solving large (non-linear) algebraic systems resulting in high computational costs per time step. Increasing the time steps causes decreasing numbers of iterations for the linear solver but higher discretization errors. Hence, implicit time-integrators are suitable for applications when the resolution of fast phenomena at small time scales, as acoustic waves, are not in the field of interest. In case of severe CFL restrictions forced by e.g. the element sizes or the problem stiffness, the high cost of the implicit method can counterbalance the cost of numerous explicit time steps even for unsteady problems. This is true when the maximum global explicit time step is much smaller compared to the physical time step. In order to resolve convective phenomena related to the explicit hyperbolic time step (2.65), IMEX schemes [5, 104] are introduced where the hyperbolic part is treated explicitly and the parabolic part implicitly. They can be efficient for diffusion-dominated problems. This kind of IMEX schemes can only outperform the explicit scheme if the time step related to the diffusion (2.66) is much smaller than the hyperbolic time step (2.65). As this is not the case for many applications using the compressible Navier-Stokes equations, we decided to focus on fully implicit time integration schemes. One frequently used IMEX strategy is the zonal IMEX method splitting the computational domain into explicit and implicit regions according to the numerical stiffness [69, 98, 77].

In the next chapter we investigate fully implicit Runge-Kutta schemes for the later application for unsteady flows.

3 Implicit Solver

Implicit time integration strategies have been already investigated by several authors [114, 14, 83, 109] for unsteady flows described by the Euler equations and the compressible and incompressible Navier-Stokes equations in combination with DG methods. As mentioned in Section 2.5, implicit schemes have the advantage of being designed as unconditionally stable in contrast to explicit time integration methods, which have a limited stability region resulting in a time step restriction. Considering unsteady problems, we still need the resolution of physical phenomena such that the time step is only driven by accuracy. However, implicit schemes require the solution of linear or non-linear equation systems inducing higher costs per time step compared to explicit schemes.
Since we use the method of lines, we obtain after the spatial DGSEM discretization an initial value problem (IVP) as described in Section 2.4.6. Gathering the element-local degrees of freedom of all grid cells, the IVP is given by

$$\begin{aligned} \frac{d}{dt}\boldsymbol{u}(t) &= \boldsymbol{R}(t, \boldsymbol{u}(t)) \qquad \text{in } [0,T], \\ \boldsymbol{u}(0) &= \boldsymbol{u}_0, \end{aligned} \tag{3.1}$$

where $\boldsymbol{u} \in \mathbb{R}^l$ denotes the state vector consisting of all nodal coefficients within the computational mesh for every conservative variable, $\boldsymbol{R} : [0,T] \times \mathbb{R}^l \to \mathbb{R}^l$ describes the spatial DG operator depending on the time and the state vector and $\boldsymbol{u}_0 \in \mathbb{R}^l$ represents a given initial state vector for $l = (N+1)^d \cdot n_{var} \cdot n_{el}$. There are several numerical methods for solving ordinary differential equation systems. The most well-known categories in which approximate solvers for IVP can be divided are the one-step and the multistep methods. Multistep methods involve several precomputed solutions in contrast to one-step methods, which need only the solution one time step before. Explicit and implicit Runge-Kutta schemes belong to the one-step methods. Basic informations on solving ordinary differential equations can be found in [115].

Solving the equation system (3.1) by an implicit scheme, we obtain in the case of the compressible Navier-Stokes equations a non-linear system, which we solve with an inexact Newton method. The arising linear system in every Newton step can be solved with any linear solver, whereat it can be distinguished between matrix-based and matrix-free methods. The resulting system matrix is here described by the Jacobian of the DGSEM operator $\boldsymbol{R}$. Matrix-based solvers store the Jacobian matrix and since it depends on the solution $\boldsymbol{u}$, it has to be recomputed in every Newton step. For these methods, matrix-vector multiplications within the used Krylov subspace linear solver can be exactly calculated. Particularly, the Jacobian can be reused to compute the preconditioner. Contrarily, matrix-free solvers do not compute the Jacobian, since it is not needed explicitly in Krylov subspace methods. The Jacobian appears only in the context of matrix-vector multiplications representing directional derivatives which can be approximated as a differential quotient. We will show in this chapter that in both approaches the number of operations for performing the matrix-vector product scale identically with respect to the polynomial order N. However, this is only valid if a compressed low-storage format is employed for the matrix-based version. For preconditioning we only assemble a reduced version of the Jacobian with less storage requirements. Hence, we call it still matrix-free. In [53] both approaches are compared for the incompressible Navier-Stokes equations. They investigated considerable improvements in CPU time as well as in memory footprint when using a matrix-free solver in the case of high order polynomials in three-dimensional test cases.
An attractive class of implicit time-integrators is given by the linearly implicit Rosenbrock-type Runge-Kutta schemes, investigated in [80, 8, 53] for the compressible Navier-Stokes equations. Here, the non-linear solution procedure in every time stage is avoided and consequently only a solver for linear systems is required. In [100] and [95] Backward Differentiation Formula (BDF) schemes are considered. Particularly, in [100] matrix-based strategies are investigated by comparing different preconditioners. A novel tensor-product based preconditioner in a matrix-free context using a Newton-Krylov solver is introduced in [95]. Space-time DG methods, where the time is treated as an additional dimension of the DG polynomial, have been investigated in [40] in relation to an Alternating Direction Implicit (ADI) preconditioner. Birken et al. [19] concentrated on Explicit first stage Singly Diagonally Implicit Runge-Kutta (ESDIRK) schemes using Jacobian-Free Newton-Krylov (JFNK) solvers with a new preconditioner based on a

modal basis. In [13] the temporal schemes ESDIRK, Rosenbrock and BDF are compared for the compressible Navier-Stokes equations in the case of full order DG schemes.

In this work, we focus on high-order ESDIRK schemes given in [71], which have never been applied in order to advance the DGSEM space discretization for the compressible Navier-Stokes equations in time. The non-linear system arising in every Runge-Kutta (RK) stage is solved by Jacobian-free Newton-Krylov solvers, which are widely used in the context of matrix-free implicit schemes [76, 17, 19]. JFNK methods consist of an inexact Newton solver for the non-linear system with an additional matrix-free Krylov-subspace solver for the linear system. Here, we employ the Generalized Minimal Residual (GMRES) method implemented as in [107], which is also applicable for non-symmetric matrices occurring for the DGSEM discretized Navier-Stokes equations. For the termination criteria of the Newton and the GMRES method we introduced adaptive tolerances in order to avoid over- and under-solving achieving the appropriate accuracy within the temporal discretization error. Investigations of the choice of preconditioners are done in the following chapter 4. We conduct some basic numerical tests without preconditioner at the end of this chapter in order to get a better understanding of the sub-methods within implicit solvers.

3.1 Diagonal Implicit Runge-Kutta Methods

For advancing the DGSEM discretized equation (3.1) in time, we employ in this work high-order ESDIRK time integration schemes, designed in [71]. ESDIRK schemes are a class of one-step multi-stage implicit Runge-Kutta methods, which can be constructed to be A- and L-stable with arbitrary temporal order of accuracy. ESDIRK methods with s stages compute the solution $\boldsymbol{u}^{n+1}$ at the time t^{n+1} of the system (3.1) by means of the previous given solution $\boldsymbol{u}^n$ at the time t^n as follows

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

$$\boldsymbol{k}_i = \boldsymbol{R}(t^n + \Delta t c_i, \boldsymbol{u}^n + \Delta t \sum_{j=1}^{i} a_{ij} \boldsymbol{k}_j) \qquad \text{for } i = 1, \dots, s, \tag{3.3}$$

whereas the Butcher coefficients $\{a_{ij}\}_{i,j=1}^{s}$, $\{b_i\}_{i=0}^{s}$, $\{c_i\}_{i=0}^{s}$ are arranged in a Butcher tableau given in Table 3.1. The meaning of the name ESDIRK is explained as follows, 'E' stands for 'explicit first stage' ($a_{11} = 0$) and 'SD' means 'singly diagonal', where the diagonal a_{ii} is given by a constant $\alpha \in \mathbb{R}$. Furthermore, if the last row of $\{a_{ij}\}_{i,j=1}^{s}$ equals the vector $\{b_i\}_{i=0}^{s}$, hence $b_j = a_{sj}$, the implicit scheme is just called stiffly accurate, if it is A-stable.

$$
\begin{array}{c|cccc}
0 & 0 & 0 & \dots & 0 \\
c_2 & a_{21} & \alpha & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & 0 \\
c_s & a_{s1} & \dots & a_{ss-1} & \alpha \\
\hline
 & a_{s1} & \dots & a_{ss-1} & \alpha
\end{array}
$$

Table 3.1: General form of an stiffly accurate ESDIRK Butcher tableau.

Defining the stage solution

$$\boldsymbol{u}_i := \boldsymbol{u}^n + \Delta t \sum_{j=1}^{i} a_{ij} \boldsymbol{k}_j \in \mathbb{R}^l,$$

we obtain $\boldsymbol{k}_i = \boldsymbol{R}(\boldsymbol{u}_i)$ and hence, we can describe stiffly accurate ESDIRK schemes as follows

$$\boldsymbol{u}_1 = \boldsymbol{u}^n$$

For stages $i = 2, \dots s$

$$\boldsymbol{u}_i = \boldsymbol{u}^n + \Delta t \sum_{j=1}^{i} a_{ij} \boldsymbol{R}(t^n + c_j \Delta t, \boldsymbol{u}_j) \tag{3.4}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}_s.$$

Depending on the operator $\boldsymbol{R}$ we have to solve $s - 1$ linear or non-linear equation systems (3.4) with $(N+1)^d \cdot n_{var} \cdot n_{el}$ unknowns of the state vector $\boldsymbol{u}_i$. The triangular form of the Butcher coefficient matrix $\{a_{ij}\}_{i,j=1}^{s}$ has the benefit, that the big equation system with $s \cdot (N+1)^d \cdot n_{var} \cdot n_{el}$ unknowns is decoupled into a sequence of s equation systems with $(N+1)^d \cdot n_{var} \cdot n_{el}$

unknowns. However, the sequence of equation systems corresponds to the application of several implicit Euler steps

$$\begin{aligned} \boldsymbol{u}_i &= \boldsymbol{u}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} \boldsymbol{R}(t^n + c_j \Delta t, \boldsymbol{u}_j) + \Delta t a_{ii} \boldsymbol{R}(t^n + c_i \Delta t, \boldsymbol{u}_i) \\ &= \boldsymbol{q}_i^n + \Delta t \alpha \boldsymbol{R}(t^n + c_i \Delta t, \boldsymbol{u}_i) \qquad i = 1, \dots s, \end{aligned} \tag{3.5}$$

since $\boldsymbol{q}_i^n$ depends only on already computed state vectors.
In this work, we compare three ESDIRK schemes, ESDIRK2-3 of order 2 with 3 stages, ESDIRK3-4 of order 3 with 4 stages, ESDIRK4-6 of order 4 with 6 stages designed by Kennedy and Carpenter in [71], which are all A- and L-stable. The corresponding Butcher tableau can be found in Appendix A.

3.2 Equation System Solver

As discussed in the previous section, for ESDIRK schemes we have to solve a sequence of implicit Euler schemes, given in (3.5). Since we consider the Navier-Stokes equations, leading to a non-linear DG operator $\boldsymbol{R} \in \mathbb{R}^l$ with $l = (N+1)^d \cdot n_{var} \cdot n_{el}$, an efficient solver for non-linear systems of the form

$$\boldsymbol{u} = \boldsymbol{q} + \alpha \Delta t \boldsymbol{R}(\boldsymbol{u}) \tag{3.6}$$

is required within every Runge-Kutta stage. Here, $\boldsymbol{u} \in \mathbb{R}^l$ indicates the vector of unknowns, $\boldsymbol{q} \in \mathbb{R}^l$ is an explicit given vector, α the diagonal element of the Butcher tableau and $\boldsymbol{R}$ describes the operator obtained by the spatial and temporal discretization.

3.2.1 Inexact Newton Method

Newton's method is a standard algorithm for finding the numerical solution of systems of non-linear equations. It solves the root problem

$$\boldsymbol{f}(\boldsymbol{u}) = 0,$$

where $\boldsymbol{f}$ is differentiable. The basic idea is to linearize the function $\boldsymbol{f}$. For a given initial starting point $\boldsymbol{u}^0$, the classical Newton's method iterates $\boldsymbol{u}^k$ as follows

$$\left.\frac{d\boldsymbol{f}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k} \Delta\boldsymbol{u}^k = -\boldsymbol{f}(\boldsymbol{u}^k)$$
$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta\boldsymbol{u}^k, \qquad k = 0, 1, \ldots$$

until a chosen termination criterion is fulfilled. In every Newton iteration k a system of linear equations has to be solved. If it is solved exactly and the initial guess is sufficiently good so that the iteration method converges, it's convergence rate is of order two. In the case of large equation systems it is reasonable to consider iterative solvers within the class of inexact Newton methods [38]

$$\left.\frac{d\boldsymbol{f}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k} \Delta\boldsymbol{u}^k = -\boldsymbol{f}(\boldsymbol{u}^k) + \boldsymbol{r}^k$$
$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta\boldsymbol{u}^k, \qquad k = 0, 1, \ldots,$$

where the linear system is solved approximately satisfying the relative residual condition limited by non-negative forcing terms η_k

$$\frac{\|\boldsymbol{r}^k\|}{\|\boldsymbol{f}(\boldsymbol{u}^k)\|} \leq \eta_k, \tag{3.7}$$

instead of computing the exact solution of the linear system by means of direct methods. The case $\eta_k = 0$ recovers Newton's method. Due to the balance of accuracy loss and cost per Newton step, the forcing sequence can be chosen to control the accuracy in order to preserve the method's quadratic convergence. In [38] it is shown, that under the weak assumption the forcing terms being less than one, inexact Newton methods are locally convergent. It is also described how this sequence influences the rate of convergence.
For solving the non-linear equation system (3.6) in every RK stage we define in this work the function $\boldsymbol{f}$ as

$$0 = \boldsymbol{f}(\boldsymbol{u}) := \boldsymbol{u} - \boldsymbol{q} - \alpha\Delta t\boldsymbol{R}(\boldsymbol{u}) \tag{3.8}$$

for the application of the inexact Newton method. As termination criterion, we use a residual based relative tolerance. Consequently, we consider the following algorithm for solving the upcoming non-linear equation system in every RK stage

$$
\begin{aligned}
&\text{DO}\quad k = 0, 1, \ldots\\
&\qquad \text{Solve iteratively}\quad \left.\frac{d\boldsymbol{f}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k} \Delta\boldsymbol{u}^k = -\boldsymbol{f}(\boldsymbol{u}^k)\\
&\qquad\qquad\qquad \text{until}\quad \left\| \left.\frac{d\boldsymbol{f}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k} \Delta\boldsymbol{u}^k + \boldsymbol{f}(\boldsymbol{u}^k)\right\| \le \eta_k \|\boldsymbol{f}(\boldsymbol{u}^k)\|\\
&\qquad \text{Set}\quad \boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta\boldsymbol{u}^k\\
&\qquad \text{Abort if}\quad \|\boldsymbol{f}(\boldsymbol{u}^{k+1})\| \le \epsilon_{\text{N}}\, \|\boldsymbol{f}(\boldsymbol{u}^0)\|
\end{aligned}
\tag{3.9}
$$

for a given initial guess $\boldsymbol{u}^0$ and $\epsilon_{\text{N}} < 1$. Note, that in this work the absolute value given by $\|.\|$ is related to the Euclidean norm. We choose the forcing terms η_k as suggested in [70]. They are based on the forcing sequence introduced by Eisenstat and Walker in [45]. Setting $\eta_k = \eta_k^A$ with

$$
\eta_k^A = \gamma \left(\frac{\|\boldsymbol{f}(\boldsymbol{u}^k)\|}{\|\boldsymbol{f}(\boldsymbol{u}^{k-1})\|} \right)^2
$$

and $\gamma \in (0, 1]$, Eisenstat and Walker proved that the local quadratic convergence of the inexact Newton iteration is maintained if η_k^A is uniformly bounded away from 1. In order to bound the sequence η_k^A and initialize it, we set

$$
\eta_k^B = \begin{cases} \eta_{\text{max}}, & k = 0\\ \min\{\eta_{\text{max}}, \eta_k^A\}, & k > 1, \end{cases}.
$$

where the parameter η_{max} describes an upper bound $0 \le \eta_k \le \eta_{\text{max}} < 1$ for each k. We set $\gamma = 0.9$ and $\eta_{\text{max}} = 0.9999$ as in [45]. A desirable property for this null sequence is, that in the first Newton iterations the linear systems should not be solved in a very accurate manner, since we do not need the best searching direction of the Newton method far away from the solution. Hence, the approach of safeguards was introduced in [45] to improve the performance of the inexact Newton algorithm in the case of volatile decreases in η_k avoiding over-solving of the linear system. We redefine the sequence η_k by

$$
\eta_k^C = \begin{cases} \eta_{\text{max}}, & k = 0\\ \min\{\eta_{\text{max}}, \eta_k^A\}, & k > 0, \gamma\eta_{k-1}^2 \le 0.1\\ \min\{\eta_{\text{max}}, \max\{\eta_k^A, \gamma\eta_{k-1}^2\}\}, & k > 0, \gamma\eta_{k-1}^2 > 0.1. \end{cases}
$$

In the final iteration it can happen, that the linear equation will be solved more accurate than actually needed for the non-linear iteration. In order to avoid this over-solving we finally set

$$\eta_k = \min\{\eta_{\max}, \max\{\eta_k^C, 0.5\tau/\|\boldsymbol{f}(\boldsymbol{u}^k)\|\}\}, \tag{3.10}$$

as suggested in [70], where in our case

$$\tau = \epsilon_{\mathrm{N}}\|\boldsymbol{f}(\boldsymbol{u}^0)\|.$$

Thereby, the norm of the current non-linear residual $\|\boldsymbol{f}(\boldsymbol{u}^k)\|$ is compared to the termination criterion of the Newton algorithm τ.

Adaptive Newton Tolerance As discussed before, we choose the termination criterion for the linear solver by certain forcing terms. They depend on the accuracy of the Newton iteration due to the influence of the Newton tolerance ϵ_{N}. Hence, we can prevent over- and under-solving of the linear system dependent on the precision of the non-linear solution. One further step is to choose the termination criterion for the Newton algorithm also automatically in order to obtain a limited set of adjustable parameters making the implicit solver more convenient. Thus, it is desirable to adapt the Newton tolerance ϵ_{N} to the solution accuracy in time. In other words, for greater CFL numbers we want to avoid over-solving of the non-linear equation systems and on the opposite for smaller CFL numbers we have to guarantee to solve it precisely enough in order to get the right time accuracy. Since an analytical solution is not always present, we take benefit of the embedded method. The idea originally comes from the choice of adaptive step sizes in order to control the error of the ODE solution. The local truncation error of a Runge-Kutta step can be estimated by means of the embedded method as

$$\alpha^n = \|\boldsymbol{u}^n - \hat{\boldsymbol{u}}^n\|_2, \tag{3.11}$$

where $\boldsymbol{u}^n$ is the numerical solution of the RK method and $\hat{\boldsymbol{u}}^n$ is the embedded solution at the time step n. The embedded method is additionally included in the RK table denoted by $\hat{b}_i$ and its solution is computed as follows

$$\hat{\boldsymbol{u}}^n = \boldsymbol{u}^{n-1} + \Delta t \sum_{i=1}^{s} \hat{b}_i \boldsymbol{k}_i.$$

The local truncation error (3.11) can be used to keep the Newton solution error below the time discretization error by defining the termination criterion $\epsilon_{\mathrm{N}}^{n+1}$ for the Newton scheme at the next time step $n+1$ as

$$\epsilon_{\mathrm{N}}^{n+1} = \min(s\,\alpha^{n}, 10^{-3}), \tag{3.12}$$

with a safety factor $s \in \mathbb{R}$. For all the following computations in 2D and 3D, we choose $s = 1/3$, which results in good agreement with the $\mathcal{L}^2$ error for very small ϵ_{N}. This is a novel approach in the presence of non-linear equation systems in order to reduce number of iterations and hence to accelerate the implicit solver. In [53] the same strategy is used to adaptively control the tolerance for the linear solver in the context of linearly implicit Rosenbrock-type Runge-Kutta schemes. A very important aspect is, this approach can be applied independently to any application test case. In Section 3.4.3, the impact of the adaptive Newton tolerance is analyzed and compared to different fixed tolerances.

3.2.2 Linear Solver

The class of solvers for linear equation systems can be split into two groups, the direct and the iterative solvers. Direct solvers are methods, which compute the exact solution, except for round-off errors, within a finite number of steps. For numerical calculations with high numbers of unknowns direct solvers are not feasible. They are often used as a preconditioner but in an incomplete way or for smaller problems. The most well known direct solver is the Gaussian elimination method, using the LU decomposition. In contrast, iterative solvers determine approximate solutions within a given tolerance. They are efficient for large systems regarding the computing time and the storage space, especially in the case of sparse matrices. However, when solving a linear equation system, which results from a discretized problem, it is not necessary to solve it exactly by a direct method, since it is also only an approximation of the solution. In this case, it is sufficient to solve the system approximately with an iterative solver in the same order as the discretization error. Hence, we will employ for the linear system arising from the discretized Navier-Stokes equations an iterative solver. A fundamental overview of linear solvers is given in [84].

As already discussed in the previous section, every Newton step k requires the solution of the linear system

$$\left.\frac{d\boldsymbol{f}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k} \Delta\boldsymbol{u}^k = -\boldsymbol{f}(\boldsymbol{u}^k), \tag{3.13}$$

where the relative termination criterion is determined by the forcing terms (3.10). For the sake of convenience, we denote the Jacobian matrix of $\boldsymbol{f}$, representing the system matrix of the linear equation system, the solution vector and the right hand side for any Newton step k with

$$\begin{aligned}\boldsymbol{A} &:= \left.\frac{d\boldsymbol{f}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k} \in \mathbb{R}^{l\times l},\\ \boldsymbol{x} &:= \Delta\boldsymbol{u}^k \in \mathbb{R}^l,\\ \boldsymbol{b} &:= -\boldsymbol{f}(\boldsymbol{u}^k) \in \mathbb{R}^l.\end{aligned}$$

In this work, we choose GMRES as iterative linear solver, introduced by Saad and Schulz [107], since it does not make any demands on the linear equations system except being regular in order to ensure the convergence and the stability of the method. GMRES belongs to the class of projection methods. Projection methods describe either orthogonal or oblique projections onto an affine m-dimensional subspace $\boldsymbol{x}_0 + K_m$ of $\mathbb{R}^l$ with an arbitrary initial guess $\boldsymbol{x}_0 \in \mathbb{R}^l$. The approximate solution $\boldsymbol{x}_m \in \boldsymbol{x}_0 + K_m$ satisfies the orthogonality condition

$$(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m) \perp L_m,$$

whereas L_m represents a m-dimensional subspace of $\mathbb{R}^l$ in the m-th iteration step. Particularly, GMRES methods are part of Krylov subspace methods with an oblique projection

$$L_m = \boldsymbol{A}K_m.$$

and the Krylov subspaces

$$K_m = K_m(\boldsymbol{A}, \boldsymbol{r}_0) = \text{span}\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{r}_0\},$$

where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$. Another common Krylov subspace method is the Conjugate Gradient (CG) method for solving large sparse symmetric, positive definite linear equation systems. Both methods, CG and GMRES, can be

interpreted as direct solvers, as they converge to the exact solution at the latest of l iterations, if rounding errors are neglected. Additionally to the class of splitting methods, projection methods denote the most important iterative techniques for solving large linear equation systems. More details concerning Krylov subspace methods can be found in [108].
The GMRES algorithm is based on Arnoldi's method calculating an orthogonal basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$, with $\boldsymbol{v}_i \in \mathbb{R}^l$, of the Krylov subspace K_m by transforming a dense matrix into Hessenberg form with a unitary transformation and using the modified Gram-Schmidt orthogonalization algorithm given in Algorithm 1.

Algorithm 1: Arnoldi-Modified Gram-Schmidt

```
Procedure ArnoldiModifiedGramSchmidt($j, (\boldsymbol{v}_i)_{i=1}^{j}), (h_{ij})_{i=1}^{j+1}, \boldsymbol{w}_j)$
    /* Computation of an orthonormal basis of the Krylov
       subspace $K_{j+1}$ by means of the given orthonormal basis
       $(\boldsymbol{v}_i)_{i=1}^{j}$ of $K_j$.                                    */
    Input: $j \in \mathbb{N}$ and $(\boldsymbol{v}_i)_{i=1}^{j} \in \mathbb{R}^l$
    Output: Hessenberg matrix $(h_{ij})_{i=1}^{j+1} \in \mathbb{R}$ and $\boldsymbol{w}_j \in \mathbb{R}^l$
    Compute $\boldsymbol{w}_j := \boldsymbol{A}\boldsymbol{v}_j$;
    for $i = 1$ to $j$ do
        $h_{i,j} := (\boldsymbol{v}_i, \boldsymbol{w}_j)$;
        $\boldsymbol{w}_j := w_j - h_{ij}\boldsymbol{v}_j$;
    end
    $h_{j+1,j} := \|\boldsymbol{w}_j\|_2$;
    /* Additional orthogonal basis vector for $K_{j+1}$ can be
       computed as $\boldsymbol{v}_{j+1} := \frac{\boldsymbol{w}_j}{h_{j+1,j}}$        */
end
```

Since the computational cost increases at least as $\mathcal{O}(m^2 \cdot l)$ for the Gram-Schmidt orthogonalization and the storage cost for an orthonormal basis increases as $\mathcal{O}(m \cdot l)$ for increasing m, we will use the restarted version of the GMRES algorithm. In this remedy, we consider a fixed small $m < l$ limiting the maximum dimension of the Krylov subspace. We denote this method as GMRES(m). If the maximum number m is reached but the residual $\|\boldsymbol{r}_m\|$ is not below a given accuracy limit the approximate solution $\boldsymbol{x}_m$ is reused

as the initial guess $\boldsymbol{x}_0$ within the next restart. The GMRES(m) algorithm of Saad and Schultz [107] is implemented in a practical way as given in [84] and is shown for our case in Algorithm 2.
As we can notice in the algorithm for GMRES(m), the only termination criterion can be specified when $\boldsymbol{w}_j$, i.e. $h_{j+1,j}$ disappears at a certain step j. In this case, it is proven that the non restarted GMRES provides the exact solution at the j-th step since the residual vector is zero for $j < l$. However, in the restarted case these conclusions are not possible, but a monotonic decreasing residuum is guaranteed. Still it is to mention that a well known difficulty can occur namely the stagnation of the residuum. To ensure the convergence of the restarted GMRES(m) $(m < l)$ it is shown that for positive definite matrices the maximum dimension m of the Krylov subspaces has to be chosen greater equal one [84].

Algorithm 2: GMRES(m)

```
Procedure RestartedGMRES$(m, -\boldsymbol{f}(\boldsymbol{u}^k), \Delta\boldsymbol{u}^k)$
    Input: $m \in \mathbb{N}$, $-\boldsymbol{f}(\boldsymbol{u}^k) \in \mathbb{R}^l$ and $\eta_k > 0$
    Output: $\Delta\boldsymbol{u}^k \in \mathbb{R}^l$
    Initialization: $\Delta\boldsymbol{u}^k = 0$, $\boldsymbol{r}_0 \coloneqq -\boldsymbol{f}(\boldsymbol{u}^k)$, $restarts \coloneqq 0$
    while $restarts \leq$ maximum of restarts do
        $\boldsymbol{v}_1 \coloneqq \boldsymbol{r}_0/\|\boldsymbol{r}_0\|_2$, $\gamma_1 \coloneqq \|\boldsymbol{r}_0\|_2$;
        for $j = 1$ to $m$ do
            $ArnoldiModifiedGramSchmidt(j, (\boldsymbol{v}_i)_{i=1}^{j}, (h_{ij})_{i=1}^{j+1}, \boldsymbol{w}_j)$;
            /* Givens-Rotations */
            for $i = 1$ to $j-1$ do
                $\begin{pmatrix} h_{ij} \\ h_{i+1,j} \end{pmatrix} \coloneqq \begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix} \begin{pmatrix} h_{ij} \\ h_{i+1,j} \end{pmatrix}$;
            end
            $\beta \coloneqq \sqrt{h_{jj}^2 + h_{j+1,j}^2}, \quad s_j \coloneqq h_{j+1,j}/\beta, \quad c_j \coloneqq h_{jj}\beta$;
            $h_{jj} \coloneqq \beta, \quad \gamma_{j+1} \coloneqq -s_j\gamma_j, \quad \gamma_j \coloneqq c_j\gamma_j$;
            if $\gamma_{j+1} < \eta_k\|\boldsymbol{f}(\boldsymbol{u}^k)\|$ or $j = m$ then
                /* converged or maximum Krylov subspace dimension
                   reached */
                for $i = j$ to $1$ do
                    $\alpha_i \coloneqq \frac{1}{h_{ii}}\left(\gamma_i - \sum_{k=i+1}^{j} h_{ik}\alpha_k\right)$;
                end
                $\Delta\boldsymbol{u}^k \coloneqq \Delta\boldsymbol{u}^k + \sum_{i=1}^{j} \alpha_i\boldsymbol{v}_i$;
                if $\gamma_{j+1} < \eta_k\|\boldsymbol{f}(\boldsymbol{u}^k)\|$ then
                    STOP;
                end
                $\boldsymbol{r}_0 \coloneqq -\boldsymbol{f}(\boldsymbol{u}^k) - \boldsymbol{A}\Delta\boldsymbol{u}^k$;
                $restarts \coloneqq restarts + 1$;
            else
                /* Extension of Krylov subspace dimension */
                $\boldsymbol{v}_{j+1} \coloneqq \boldsymbol{w}_j/h_{j+1,j}$;
            end
        end
    end
end
```

Form of the Jacobian Matrix As described previously, we have to solve in every Newton iteration the linear system (3.13) with the matrix given by the Jacobian of the non-linear function $\boldsymbol{f}$ defined in (3.8). Since, the vector $\boldsymbol{q}$ does not depend on the solution $\boldsymbol{u}$, we obtain

$$\boldsymbol{A}(\boldsymbol{u}^k) = \boldsymbol{I} - \alpha\Delta t \left.\frac{d\boldsymbol{R}(\boldsymbol{u})}{d\boldsymbol{u}}\right|_{\boldsymbol{u}^k}. \tag{3.14}$$

The Jacobian matrix $\boldsymbol{A}$ has a natural block structure related to the grid elements. It consists of n_{el} block-rows and block-columns, where each block has the dimension

$$n_{var} \cdot (N+1)^d \times n_{var} \cdot (N+1)^d. \tag{3.15}$$

We want to analyze the density of the Jacobian with respect to the polynomial degree N in order to have in mind which challenges can occur when dealing with implicit solvers for DGSEM. Density is the number of non-zero entries divided by the total number of elements of a matrix and in contrast, sparsity is related to the number of zero-valued entries. Two special characteristics of DGSEM have a great impact on the sparsity of the Jacobian $\boldsymbol{A}$: the tensor-product structure and the small dependency stencil of each element. Related to the tensor-product basis, as discussed in Section 2.3, the dependency within one element reduces to d 1D-lines. Particularly, the number of non-zero entries within each block scales linear in N for each DOF, so that the density scales as

$$\mathcal{O}\left(\frac{1}{N^{d-1}}\right) \qquad \text{for each element.} \tag{3.16}$$

Here, we can already see one of the highlights using tensor-product collocation methods. DG schemes with full-order basis functions consist in contrast of full block elements resulting in higher storage requirements and costly computations regarding matrix-vector products.

The other aspect, the small stencil, concerns the sparsity of the whole Jacobian matrix. Each element is only coupled to the adjacent elements by Riemann fluxes. Hence, the number of non-zero blocks in each block-row corresponding to a grid cell C_{iElem} is exactly the number of neighbors of C_{iElem} plus the element itself, which is $2d+1$ for hexahedral meshes. Thus, the number of non-zero entries for the Jacobian $\boldsymbol{A}$ adds up to

$$d(N+1)^{d+1} \cdot n_{Var}^2 \cdot (2d+1) \cdot n_{el}. \tag{3.17}$$

In Table 3.2 the storage requirements for one block-row of the Jacobian $\boldsymbol{A}$ assembled in double precision within the matrix-based approach are listed in the 3D case with respect to the polynomial order N. In order to obtain the whole storage demand of the Jacobian matrix, the values have to be multiplied by the number of elements n_{el}.

N	1	2	3	4	5	6	7	8	9	10
estimated memory [MB]	0.064	0.324	1.025	2.503	5.191	9.617	16.406	26.280	40.054	58.644

Table 3.2: Memory demand in double precision (8 Byte) per block row for DGSEM in 3D.

For real world problems in 3D the total number of non-zero entries of $\boldsymbol{A}$ can be of the order $\mathcal{O}(10^9)$ resulting in a memory demand within the gigabyte range.

Furthermore, we can show that the matrix $\boldsymbol{A}$ arising from the DG discretization of the Navier-Stokes equations is non-symmetric but it consists of a symmetric block structure. We can examine, that $\boldsymbol{A}$ is almost a diagonal matrix for small time steps and becomes ill-conditioned the higher the time step, resulting in greater iterations of the linear solver. Hence, it is inevitable to use preconditioning in order to accelerate the linear system for bigger time steps.

Matrix-Free Implementation of GMRES The Jacobian $\boldsymbol{A}$ has to be recalculated every Newton iteration k for a matrix-based linear solver so that the convergence of the Newton scheme is not affected. In order to circumvent this problem, we decided to consider the class of matrix-free methods for solving the linear system (3.13) with the matrix (3.14). As we can see in Algorithm 2 and in the corresponding Arnoldi Algorithm 1, in Krylov subspace methods the Jacobian $\boldsymbol{A}$ appears only in form of a matrix-vector multiplication $\boldsymbol{A}\boldsymbol{v}$ and not explicitly. The idea of the matrix-free approach is to approximate this multiplication by a difference quotient as done in [23, 75], here of first order,

$$\boldsymbol{A}(\boldsymbol{u}^k)\boldsymbol{v} \approx \frac{\boldsymbol{f}(\boldsymbol{u}^k + \epsilon\boldsymbol{v}) + \boldsymbol{f}(\boldsymbol{u}^k)}{\epsilon} = \boldsymbol{v} - \alpha\Delta t\frac{\boldsymbol{R}(\boldsymbol{u}^k + \epsilon\boldsymbol{v}) - \boldsymbol{R}(\boldsymbol{u}^k)}{\epsilon}, \quad (3.18)$$

to avoid assembling the whole Jacobian $\boldsymbol{A}$. The choice of the scalar perturbation ϵ has a strong impact on the convergence of the JFNK method. In [22] sufficient conditions for ϵ are derived to guarantee the local convergence of the Newton algorithm. We choose a very simple form as proposed in [101]

$$\epsilon = \frac{\sqrt{eps}}{\|\boldsymbol{v}\|_2},$$

where eps describes the machine accuracy. It can be seen that the right choice for ϵ can depict a tricky trade-off between round-off and truncation error. Particularly for small Mach numbers, when the variable values are not well scaled, it is almost infeasible to find a suitable ϵ. Second order finite difference approximations are not recommended due to the second additional evaluation of the DG operator. Detailed discussions in finding an appropriate ϵ can be found in [27].

With the present choice of ϵ the finite difference approximation (3.18) requires in each matrix-vector product only one additional evaluation $\boldsymbol{R}(\boldsymbol{u}^k + \epsilon\boldsymbol{v})$ of the DGSEM operator, since $\boldsymbol{R}(\boldsymbol{u}^k)$ is computed in the Newton scheme and is constant in every GMRES iteration. Hence, in the case of a matrix-free strategy, the matrix-vector multiplication scales as $\mathcal{O}(N^{d+1})$ per element, examined in Section 2.4.8 for the DGSEM operator evaluation. The operation counts for the matrix-based ansatz scale equally due to the sparsity of $\boldsymbol{A}$ given in (3.17) if and only if the Jacobian matrix is stored in a sparse format without the zero entries. Consequently a similar behavior is expected for increasing N. In [53] matrix-free and matrix-based solvers are compared, whereas they use a full-order, hierarchical polynomial basis resulting also in an identical scaling in the polynomial order of the matrix-vector product.

However, within the matrix-free ansatz the cost and memory demand associated with assembling the Jacobian is omitted, since we do not store the Jacobian matrix. Note, that we still require to build a storage-reduced approximate form of the Jacobian for the preconditioner in Section 4. But since we will consider only the block diagonals of $\boldsymbol{A}$, we reduce the system into n_{el} independent sub equation systems of size (3.15).

Linearization of the DG Operator In some cases it is recommendable to replace the finite difference quotient approximating the matrix-vector multiplication $\boldsymbol{A}\boldsymbol{v}$ in (3.18) by the DOF-wise exact derivative. Particularly, when the computation of the difference quotient is prone to round-off errors if ϵ is too small. For instance, for low Mach numbers when the values of the variables show highly different magnitudes, it is difficult to find an adequate ϵ, if not impossible.

When applying Newton's method to the equation system (3.6), we naturally linearize the problem obtaining a linear system to solve in every Newton step. According to that, we can consider the linearized flux functions receiving from the Taylor expansion around $\boldsymbol{u}_k$ in the point $\boldsymbol{u}_k + \epsilon\boldsymbol{v}$ as employed in [42]. Then the corresponding DGSEM operator $\boldsymbol{R}^{lin}$ is linear in $\boldsymbol{u}$ and the matrix-vector multiplication yields

$$\boldsymbol{A}(\boldsymbol{u}^k)\boldsymbol{v} \approx \boldsymbol{v} - \alpha\Delta t \frac{\boldsymbol{R}^{lin}(\boldsymbol{u}^k + \epsilon\boldsymbol{v}) - \boldsymbol{R}^{lin}(\boldsymbol{u}^k)}{\epsilon} = \boldsymbol{v} - \alpha\Delta t \boldsymbol{R}^{lin}(\boldsymbol{v}),$$

where the physical flux function and the Riemann fluxes within $\boldsymbol{R}^{lin}$ are substituted by its linearization

$$\boldsymbol{F}^{lin}(\boldsymbol{v}, \boldsymbol{\nabla}\boldsymbol{v}) = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}_k}\boldsymbol{v} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{\nabla}\boldsymbol{u}_k}\boldsymbol{\nabla}\boldsymbol{v}. \tag{3.19}$$

The vectors $\boldsymbol{u}_k$ and $\boldsymbol{\nabla}\boldsymbol{u}_k$ describe the Newton state and its gradient at iteration k around which we expand the series and $\boldsymbol{v}$ is the vector to multiply with. Note, that the derivatives in (3.19) are calculated DOF-wise, which means

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}_k} = \left.\frac{\partial \boldsymbol{F}(\boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}^k}.$$

Consequently, we can compute the matrix-vector product without any difficulties in finding the right perturbation parameter ϵ. Since the truncation error of the Newton algorithm is of order $\mathcal{O}(\Delta\boldsymbol{u}_k^2)$, it is obvious that the truncation error originating from the linearization of order $\mathcal{O}(\epsilon\boldsymbol{v}^2)$ does not increase the total error of the non-linear solver. However, we want to point out that this computation is more costly than evaluating the normal DGSEM operator $\boldsymbol{R}$ which is only required for the difference quotient approach, since the linearized DG operator $\boldsymbol{R}^{lin}$ requires the DOF-wise assembling of the derivatives of the flux function multiplied by the vector $\boldsymbol{v}$.

3.3 Preconditioning

The most challenging task for implicit solvers is the right choice of the preconditioner. In order to improve the convergence speed by reducing the amount of iterations from the linear solver, we introduce right preconditioning for GMRES. In contrast to left preconditioning, it maintains the residual of the linear system. We transform the original linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ into the preconditioned system

$$\boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{x}^P = \boldsymbol{b}, \tag{3.20}$$

$$\text{with} \quad \boldsymbol{P}\boldsymbol{x} = \boldsymbol{x}^P. \tag{3.21}$$

The goal is now, to choose the preconditioner $\boldsymbol{P} \in \mathbb{R}^{l \times l}$ such that the new system matrix $\boldsymbol{A}\boldsymbol{P}^{-1}$ has a smaller condition number than $\boldsymbol{A}$ itself. Hence, the standard version of GMRES is modified by solving equation (3.21) before every matrix-vector multiplication $\boldsymbol{A}\boldsymbol{x}$ in every GMRES iteration. Consequently, the choice of the preconditioner $\boldsymbol{P}$ is the key to an efficient implicit scheme. The arising trade-off can be illustrated as follows

- $\boldsymbol{P} \approx \boldsymbol{A}$. Optimal: $\boldsymbol{P} = \boldsymbol{A}$, hence $\boldsymbol{A}\boldsymbol{P}^{-1}$ has the condition number one and the solver requires only one iteration. However, this choice does not simplify the problem. But we can infer, the better the preconditioner approximates the matrix $\boldsymbol{A}$, the more it decreases the number of iterations.

- $\boldsymbol{P} \approx \boldsymbol{I}$. Optimal: $\boldsymbol{P} = \boldsymbol{I}$, hence the inversion of the preconditioner or the application of $\boldsymbol{P}^{-1}$ has the smallest computing time. However, in this case the preconditioner does not have any affect.

We can summarize, that $\boldsymbol{P}$ should be a good approximation of $\boldsymbol{A}$, but the system (3.21) should be easier solvable than the original system. Satisfying these criteria, we decided in this work to employ a block-Jacobi preconditioner, which describes the Jacobian $\boldsymbol{A}$ neglecting all off-diagonal blocks. Hence, the block row consists only of the derivative of the element by itself and not by its adjacent elements. This structure makes the block-Jacobi preconditioner very attractive in matters of parallel computing, since we have an element-wise decoupled linear equation system to solve. The derivation and effects on linear iteration reduction and computational efficiency of the block-Jacobi are described in detail in Chapter 4.

For the implementation of the right preconditioned GMRES method, the original GMRES Algorithm 2 has to be modified only at two points. Before every matrix-vector multiplication with $\boldsymbol{A}$ in line 2 of the modified Gram-Schmidt Algorithm 1, the system

$$\boldsymbol{P}\boldsymbol{u}_j = \boldsymbol{v}_j \tag{3.22}$$

has to be solved in the $j-$th GMRES iteration, since GMRES solves the preconditioned system with the system matrix $\boldsymbol{A}\boldsymbol{P}^{-1}$ in $\boldsymbol{x}^P$-variables. The matrix-vector product $\boldsymbol{A}\boldsymbol{u}_j$ is computed then with the matrix-free ansatz (3.18). The initial residual stays unaffected by the preconditioner, since

$$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{x}_0^P.$$

As described in [108], at the end of the GMRES algorithm the solution in $\boldsymbol{x}^P$-variables

$$\boldsymbol{x}_k^P = \boldsymbol{x}_0^P + \sum_{i=1}^{k} \alpha_i \boldsymbol{v}_i$$

has to replaced in line 15 of Algorithm 2 by the back-transformed solution of the unpreconditioned system in $\boldsymbol{x}$-variables as

$$\boldsymbol{x}_k = \boldsymbol{P}^{-1}\boldsymbol{x}_k^P = \boldsymbol{x}_0 + \sum_{i=1}^{k} \alpha_i \boldsymbol{P}^{-1}\boldsymbol{v}_i = \boldsymbol{x}_0 + \sum_{i=1}^{k} \alpha_i \boldsymbol{u}_i,$$

where we used the precomputed solutions $\boldsymbol{u}_i$ of equation (3.22).

3.4 Numerical Experiments

We want to conclude this chapter about implicit time integration by some fundamental numerical experiments. We compare several strategies, which we proposed in the course of this chapter by means of an isentropic vortex convection problem of Hu and Shu [67]. This test case is also summarized in Appendix C. Furthermore, we consider only the 2D Euler equations using the restarted version GMRES(20) without any preconditioner. The employed CFL numbers are related to the explicit Runge-Kutta method ERK4 of forth order listed in Appendix B. In this way, it is already obvious which time scale we consider in order to compare implicit and explicit schemes later on. Investigations including different preconditioners are shown in the next chapter 4. The following computations are conducted on the supercomputer Hazel Hen, a Cray XC40 system at HLRS on 72 cores unless otherwise noted.

3.4.1 Temporal Convergence Order

By means of the exact solution presented in Appendix C, we first examine the temporal experimental order of convergence (EOC) of the proposed ESDIRK2-3, ESDIRK3-4 and ESDIRK4-6 amplified in Appendix A. We selected these RK methods, since they incorporate the corresponding embedded scheme in order to employ the adaptive Newton tolerance.
The spatial set up is defined in Appendix C, such that the time discretization error dominates. The time steps are chosen according to the CFL numbers 4, 8, 16, 32, 64, 128, 256, related to the explicit Runge-Kutta method ERK4. Further, we fix the Newton tolerance ϵ_{N} small enough, for this set up namely $\epsilon_{\mathrm{N}} = 10^{-8}$ in order to eliminate the influence of the resolution of the Newton solver.

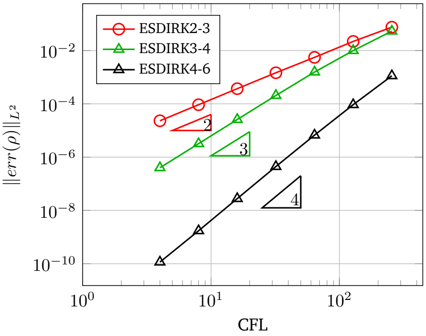


Figure 3.1: Convergence order of ESDIRK2-3, ESDIRK3-4 and ESDIRK4-6 for the isentropic inviscid traveling vortex in 2D. $\mathcal{L}^2$-error of the density is plotted for the end time $T = 1$. The Newton tolerance is fixed at $\epsilon_{\mathrm{N}} = 10^{-8}$ and no preconditioner is used.

In Figure 3.1 the $\mathcal{L}^2$-error of the density at the end time $T = 1$ is plotted over the CFL numbers. We can see that each implicit solver converges with its theoretical order for decreasing CFL numbers. Also for the smallest time steps the spatial discretization error remains smaller than the temporal.

3.4.2 Effect of GMRES Tolerance

In this section, we want to investigate the effect of the GMRES termination criteria η_k of algorithm (3.9) on the Newton residual. We compute the residuals for one RK stage using ESDIRK4-6 at CFL=10 on a single core. The spatial settings are given in Appendix C.

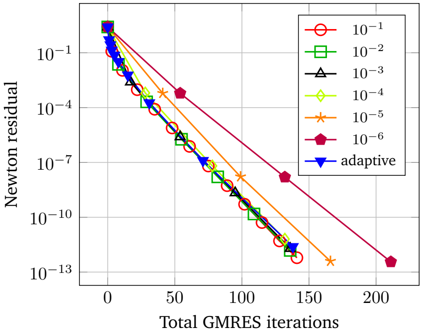


Figure 3.2: Convergence of the Newton residual for different tolerances η_k of the GMRES method at CFL=10 for one RK stage. No preconditioner is used.

In Figure 3.2, we compare the Newton residual $\|\boldsymbol{f}(\boldsymbol{u}^{k+1})\|$ for different fixed η_k and for the adaptive forcing terms given by (3.10) inspired by Eisenstat and Walker [45] in relation to the required total number of GMRES iterations. Each symbol in both figures represent one Newton iteration. The greater the fixed GMRES tolerances, the smaller are the distances of the symbols, since the required GMRES iterations decrease in order to fulfill the termination criterion (3.9) for the linear equation system. The convergence of the residual by means of the adaptive forcing terms is as good as the convergence when using the highest tolerances. Only the two lowest tolerances $\eta_k = 10^{-5}$ and $\eta_k = 10^{-6}$ induce inefficient solvers, since in every Newton step the linear equation system is over-solved indicated by the higher GMRES iterations within one Newton step with a weaker descend of the residual. In the case of a very coarse required Newton residual as 10^{-2}, we can see it is not necessary to solve the linear system very precisely. Only the GMRES tolerances $\eta_k = 10^{-1}, \eta_k = 10^{-2}$ and the adaptive can guarantee the minimum decrease in the residual. The remaining tolerances need more GMRES iterations and consequently more CPU time. The lower the required Newton residual, the more efficient are also the smaller GMRES tolerances, expect for the two lowest values, since they cause over-solving in the beginning of the Newton scheme far away from the solution. This over-solving has to be avoided, and can be complied by choosing the tolerances in the first Newton steps high, as it is done for the adaptive forcing terms.

Newton step	1	2	3	4	5
η_k	0.9999	0.9999	0.8998	0.7287	0.4779
Newton step	6	7	8	9	
η_k	0.2056	$3.03 \cdot 10^{-2}$	$8.24 \cdot 10^{-4}$	$1.02 \cdot 10^{-5}$	

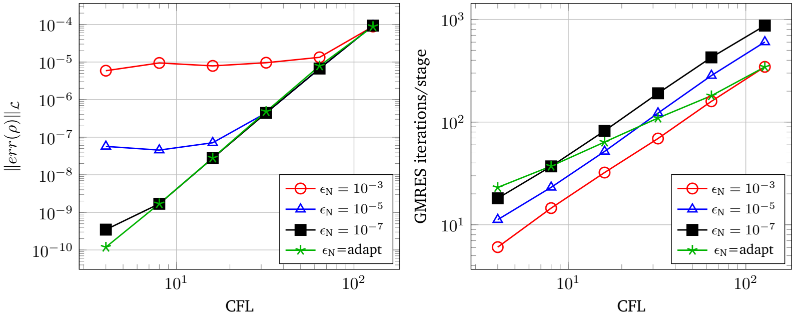
Table 3.3: Adaptive forcing terms η_k for GMRES computed as in (3.10).

The resulting values of the adaptive GMRES tolerances are listed in Table 3.3. They form a decreasing zero sequence, where the described demands in Section 3.2.1 in matters of avoiding over-solving for the first few Newton iterations are satisfied as can be seen in Figure 3.2 due to very coarse tolerances. Thus, in the beginning there are more Newton steps necessary as for the fixed GMRES iterations.

In order to have a flexible implicit solver without adjusting the GMRES tolerances for every CFL number and test case, we choose to employ the adaptive version for further computations.

3.4.3 Effect of Newton Tolerance

In contrast to the GMRES tolerance, different choices for the Newton tolerance have strong effects on the solution error. Here, we want to compare the $\mathcal{L}^2$-error of the density and the number of GMRES iterations for different fixed Newton tolerances ϵ_N and the adaptive tolerance given by (3.12). We use ESDIRK4-6, the adaptive GMRES tolerance and the restarted version GMRES(20).
In Figure 3.3 we can observe the influence of the fixed Newton tolerances $\epsilon_N = 10^{-3}, 10^{-5}$ and 10^{-7} and the adaptive tolerance on the $\mathcal{L}^2$-error and the GMRES iterations. For the highest CFL number the $\mathcal{L}^2$-error in Figure 3.3a is identical for all reported ϵ_N, since the global error is not dominated by the Newton error. In that case, it is not necessary to choose small values of ϵ_N to obtain the same error, which accelerates the implicit solver, since the number of GMRES iterations can be reduced as can be seen in Figure 3.3b. For smaller CFL numbers and high ϵ_N, the Newton scheme is under-solved and hence the error for $\epsilon_N = 10^{-3}$ and 10^{-5} does not show the expected convergence rate.



(a) $\mathcal{L}^2$- error in density over CFL

(b) GMRES iterations per time stage over CFL

Figure 3.3: Comparison of different fixed Newton tolerances ϵ_N and an adaptive strategy. The computation is run until $T = 1$ with an adaptive GMRES forcing sequence and without preconditioning.

However, the adaptive Newton tolerance determined by the corresponding embedded Runge-Kutta scheme holds the Newton solution error below the time discretization error even for small CFL numbers as depicted in 3.3a. Considering greater CFL numbers, the adaptive tolerance adopts higher values reducing the GMRES iterations significantly as illustrated in Figure 3.3b.

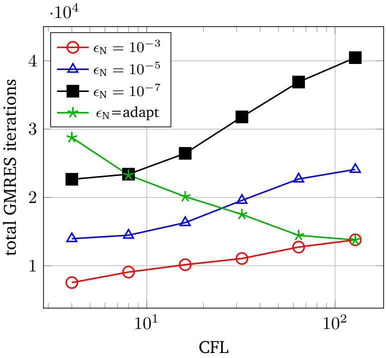
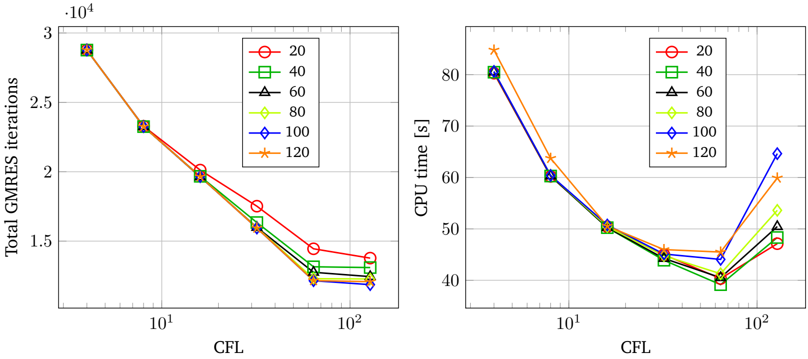


Figure 3.4: Impact of different Newton tolerances ϵ_N on the total GMRES iterations. The computation is run until $T = 1$ with an adaptive GMRES forcing sequence and without preconditioning.

The absolute efficiency gain by using an adaptive Newton tolerance can be seen in the total number of GMRES iterations in Figure 3.4. As the number of GMRES iterations increases with respect to the CFL number for each fixed Newton tolerance, the iterations decrease for the adaptive tolerance, which highlights the advantage of implicit schemes using higher time steps. Comparing with Figure 3.3b, all curves of the GMRES iterations per stage have a higher gradient than one for the fixed tolerances, which introduces increasing total computational work for greater CFL numbers. However, the slope corresponding to the curve of the adaptive Newton tolerance is significantly lower than one, so that the implicit scheme becomes for higher time steps more efficient in total.

3.4.4 Effect of Krylov Subspace Dimension

Again, we use ESDIRK4-6 in order to investigate the impact of the maximum Krylov subspace size m of the GMRES(m) method on the total number of GMRES iterations and computational cost. The computation is conducted until $T = 1$ with adaptive Newton and GMRES tolerances for different CFL numbers without preconditioning. Using a small maximum size m of Krylov subspaces results in a higher number of restarts of GMRES(m), where we can not guarantee the convergence of the GMRES solver.
In Figure 3.5 the total GMRES iterations and CPU time are plotted over the CFL number for different numbers of Krylov subspace dimensions m. In both plots there is no significant difference to observe among different numbers m for small time step sizes, since the number of GMRES iterations per linear solve is still not high enough such that restarting is required.



(a) GMRES iterations over CFL (b) CPU time over CFL

Figure 3.5: Comparison of different maximum Krylov subspace dimensions m. The GMRES iterations and CPU time denote the total amount until the computing time $T = 1$.

In Figure 3.5a we can see the trend using rather higher dimensions of the Krylov subspaces for great CFL numbers in order to reduce the number of GMRES iterations. But in contrast, Figure 3.5b illustrates the increase of the computational cost for higher m despite the lower GMRES iterations. As discussed in Section 3.2.2 the performance scales at least with $\mathcal{O}(m^2 l)$

due to the orthogonalization process. Hence, for greater m the GMRES solver becomes drastically expensive. This effect is better noticeable for higher CFL number, since the GMRES iterations are still low enough for the used CFL numbers so that GMRES already converges before reaching the maximum number m in the starting Newton steps. Hence, we hit again a trade-off in choosing lower Krylov subspace dimensions in order to the reduce computational time and the possibility of a non-converging solver. When dealing with convergence problems, one may think about introducing an adaptive dimension for the Krylov subspaces, starting in the first Newton steps with higher dimensions.

3.4.5 Discussion

In this section, a novel approach for handling the implicit solver fully adaptively was introduced. There are no adjustable settings required. The fully automatic implicit solver works appropriately for different CFL numbers ensuring the minimum number of Newton iterations.
We proposed an inexact Newton scheme as non-linear solver, where we employed an adaptive Newton tolerance avoiding the over- and under-solving problems. By means of the embedded Runge-Kutta scheme we can keep the non-linear error below the time discretization error without knowing the exact solution. In general, exact analytical solutions are not available for complex problems. This highlights the benefit of using automatic Newton tolerances independent of the investigated test case.
Further, we presented the restarted GMRES method as linear solver with a matrix-free strategy. We can underline that the approximated matrix-vector product without having the Jacobian matrix assembled scales similar to the multiplication within the matrix-based approach if the matrix is stored in a sparse format. We also introduced adaptive tolerances for the termination criterion of the linear solver, which guarantee the quadratic convergence of the Newton solver. Similar, we can avoid over-solving of the linear equation system. Setting the GMRES tolerance fixed as $\eta_k = 0.1$, we obtain a similar convergence behavior of the Newton residual as observed using the adaptive strategy.
We also investigated the usage of different dimensions of the Krylov subspaces within GMRES. The gain of reduced GMRES iterations with increased dimension can not be exploited since the construction of the orthogonal

basis scales quadratically with the dimension of the subspaces. However, when dealing with convergence problems, it is advisable to use greater dimensions in order to retrieve the convergence property of the non-restarted version of GMRES.

Summarizing the results of the presented basic tests without any preconditioner, we suggest using adaptive tolerances for the outer iterations (Newton) as well as for the inner iterations (GMRES) and to employ small dimensions for the Krylov subspaces if the convergence of GMRES does not fail.

The effect on the GMRES iterations and computational efficiency in the case of a preconditioned implicit solver is investigated in the next Chapter 4.

4 Analytic Block-Jacobi Preconditioner for DGSEM

Several preconditioners have been investigated in the last recent years for implicit Discontinuous Galerkin schemes. The natural elemental block structure of the Jacobian of the spatial DG operator implicate the usage of either a full Jacobian matrix or a reduced storage form by neglecting certain blocks for forming the preconditioner. Each block corresponds to the derivative with respect to a grid element. The diagonal blocks represent the Jacobian of the DG operator restricted to each element with respect to its own degrees of freedom.

Persson and Peraire [99, 100] studied various preconditioners as block-Jacobi (only diagonal blocks), block-Gauss-Seidel (all blocks) and block-incomplete LU (all blocks) factorization and combined them with p-multigrid schemes [49]. They investigated that the two-level scheme with a block-incomplete LU smoother requires the fewest linear iterations but without any comment on the CPU time and parallelization aspects. Birken et al. proposed in [18, 19] a new class of preconditioners, ROBO-SGS (Reduced Off diagonal Block Order-Symmetric Gauss-Seidel), exploiting the hierarchy of modal basis functions which reduces the amount of entries in the off-diagonal blocks. This mix between block-Jacobi and symmetric block-Gauss-Seidel preconditioner represents a trade-off between storage requirements and application cost. However, they found out that block-Jacobi yields similar results and illustrates for computations with a large number of processors a very viable choice with a good overall performance. In [43] the tensor-product formulation is exploited in the context of a space-time DG method, by using a diagonalized Alternating-Direction-Implicit (ADI) scheme and fast-diagonalization method (FDM) relying on the block-Jacobi preconditioner. They compared these strategies to the mass-matrix and block-Jacobi preconditioners, where the block-Jacobi showed the lowest number of GMRES iterations also for higher polynomial orders and CFL numbers. Within our presented implicit scheme, a mass-matrix precondi-

tioner is indirectly included since we divide the time derivative of the degrees of freedoms by the quadrature weights resulting in a Jacobian matrix of the form (3.14). Recently, Pazner and Persson developed in [96] a novel preconditioner based on the tensor-product using an algebraic Kronecker-product singular value decomposition (KSVD) approximating the block-Jacobi matrix. For high polynomial degrees the KSVD preconditioner reduces the runtime compared to the exact block-Jacobi while the number of GMRES iterations increases. A good overview of the basic preconditioners is given in Meister et al. [85].
Since we are interested in the computations of large scale problems on high number of processors, we focus in this work on the investigation of the block-Jacobi preconditioner. In this chapter, we present several strategies to solve the arising element-local linear equation approximately in order to increase the efficiency of the preconditioner application. The block-Jacobi method restricts the block-structured Jacobian matrix $\boldsymbol{A}$ given in (3.14) only to its diagonal block elements. Hence, each block row consists only of the derivative of the element by itself and not by its adjacent elements. This structure makes the block-Jacobi preconditioner very attractive in matters of parallel computing, since we have an element-wise decoupled linear equation system to solve. The equation system (3.22) at the j-th GMRES iteration reduces for the block-Jacobi method to

$$\begin{pmatrix} \mathbf{BJ}^1 & & & 0 \\ & \mathbf{BJ}^2 & & \\ & & \ddots & \\ 0 & & & \mathbf{BJ}^{n_{el}} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_j^1 \\ \boldsymbol{u}_j^2 \\ \vdots \\ \boldsymbol{u}_j^{n_{el}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{v}_j^1 \\ \boldsymbol{v}_j^2 \\ \vdots \\ \boldsymbol{v}_j^{n_{el}} \end{pmatrix}, \tag{4.1}$$

where

$$\mathbf{BJ}^{iElem} \in \mathbb{R}^{(n_{var}\cdot(N+1)^d)\times(n_{var}\cdot(N+1)^d)},$$

describes the $iElem$-th diagonal block of the Jacobian matrix $\boldsymbol{A}$ defined in (3.14) and

$$\boldsymbol{u}_j^{iElem}, \boldsymbol{v}_j^{iElem} \in \mathbb{R}^{n_{var}\cdot(N+1)^d}$$

represent the solution and the right-hand side vector respectively, at the j-th GMRES iterations restricted to the DOFs on the $iElem$-th grid element for $iElem = 1, \dots, n_{el}$. The $iElem$-th diagonal block corresponds to the total derivative of the DGSEM operator $\boldsymbol{R}$ restricted to the element C_{iElem} with respect to its own DOFs which reads as

$$\mathbf{BJ}^{iElem} = \boldsymbol{I} - \alpha \Delta t \frac{d\boldsymbol{R}(\boldsymbol{u}^k)}{d\boldsymbol{u}}\bigg|_{C_{iElem}}. \tag{4.2}$$

Here, $\boldsymbol{u}^k \in \mathbb{R}^l$ denotes the global solution vector at the k-th Newton step with $l = (N+1)^d \cdot n_{var} \cdot n_{el}$. Since the Jacobian $\boldsymbol{A}$ changes in every Newton step, also the block-Jacobi varies. In order to keep the computational cost low, we keep the preconditioner constant over the Newton steps and recalculate it only every time step.

In this chapter, we first derive the element-local analytical derivative $\mathbf{BJ}^{iElem}$ of the DGSEM operator for the compressible Navier-Stokes equations and examine its building costs. In order to get an impression of the complexity how to solve efficiently the preconditioned system, we look at the matrix pattern of the block diagonals $\mathbf{BJ}^{iElem}$ and investigate its density in relation to the polynomial order for the two- and three dimensional Euler- and Navier-Stokes equations. Then, we introduce different strategies to solve the element-local linear equation system approximately. For the reason of comparison, we also compute the exact inverse of $\mathbf{BJ}^{iElem}$ by the LU factorization, which ensures the minimum number of GMRES iterations. We examine for each solving strategy the building and application time in relation to the cost of one DGSEM operator call in order to obtain the analogy to the explicit time integration method. The final section is dedicated to the numerical application of the previous investigations and the comparison of the different solving strategies in matters of GMRES iterations and computational cost for the basic test case of a traveling vortex given in Appendix C. Further, we analyze the influence of the spatial discretization, the preconditioner freezing and different time integration orders on the efficiency of the implicit solver. The final strong scaling tests show the parallel efficiency of the implicit solver including the introduced preconditioners for different mesh refinement levels on very large numbers of processors.

4.1 Jacobian Assembling

In this section, we show how to compute the diagonal blocks $\mathbf{BJ}^{iElem}$ given in (4.2) for the three-dimensional viscid case supposing the conservation law in the advection-diffusion notation described in (2.1). The derivation can be reduced straightforward for the two- and one-dimensional case.
As investigated in Section 2, the discontinuous approximation of the solution yields only a weak coupling of direct adjacent elements resulting in an element-wise formulation of DGSEM given in (2.61). For further derivations of the diagonal block $\mathbf{BJ}^{iElem}$, we split the element-local DGSEM operator $R_{ijk} \in \mathbb{R}^{(N+1)^3 \cdot n_{var}}$ (omitting a superscript $iElem$) into the volume and surface integral

$$R_{ijk}(\boldsymbol{u}) = R^V_{ijk}(\boldsymbol{u}) + R^S_{ijk}(\boldsymbol{u}), \tag{4.3}$$

with

$$R^V_{ijk} = -\frac{1}{J_{ijk}} \sum_{l=0}^{N} \left[\hat{\mathcal{F}}^1_{ljk} \hat{D}_{il} + \hat{\mathcal{F}}^2_{ilk} \hat{D}_{jl} + \hat{\mathcal{F}}^3_{ijl} \hat{D}_{kl} \right] \in \mathbb{R}^{n_{var}}, \tag{4.4}$$

$$\begin{aligned} R^S_{ijk} = -\frac{1}{J_{ijk}} \Bigg[\quad & [F^* \hat{s}]^{+\xi^1}_{jk} \hat{\ell}_i(1) - [F^* \hat{s}]^{-\xi^1}_{jk} \hat{\ell}_i(-1) \\ & + [F^* \hat{s}]^{+\xi^2}_{ik} \hat{\ell}_j(1) - [F^* \hat{s}]^{-\xi^2}_{ik} \hat{\ell}_j(-1) \\ & + [F^* \hat{s}]^{+\xi^3}_{ij} \hat{\ell}_k(1) - [F^* \hat{s}]^{-\xi^3}_{ij} \hat{\ell}_k(-1) \Bigg] \in \mathbb{R}^{n_{var}}, \end{aligned} \tag{4.5}$$

for $i, j, k = 0, \dots, N$. Denoting as in Section 2.4.6 the element-local solution vector with $\hat{u}_{ijk} \in \mathbb{R}^{nVar}$ without a superscript $iElem$, all components of the $iElem-$th diagonal block reads

$$\mathbf{BJ}^{iElem}_{rijk,smno} = \delta_{rs}\delta_{im}\delta_{jn}\delta_{ko} - \alpha \Delta t \left(\frac{dR_{ijk}(\boldsymbol{u})}{d\hat{u}_{mno}} \right)_{r,s}$$

for $r, s = 1, \dots, n_{var}$ and $i, j, k, m, n, o = 0, \dots, N$ while $\frac{dR_{ijk}(\boldsymbol{u})}{d\hat{u}_{mno}} \in \mathbb{R}^{n_{var} \times n_{var}}$. Since the following derivation is identical for all elements, we omitted the superscript $iElem$ for the sake of convenience.

4.1.1 Volume Integral

Considering the volume part R^V in (4.4) of the DGSEM operator, we can see that only the contravariant fluxes $\hat{\mathcal{F}}^p$ $(p = 1, 2, 3)$ depend on the solution vector $\boldsymbol{u}$ so that the Jacobian of the volume integral yields

$$\frac{dR^V_{ijk}}{d\hat{u}_{mno}} = \frac{1}{J_{ijk}} \sum_{l=0}^{N} \left[\frac{d\hat{\mathcal{F}}^1_{ljk}}{d\hat{u}_{mno}} \hat{D}_{il} + \frac{d\hat{\mathcal{F}}^2_{ilk}}{d\hat{u}_{mno}} \hat{D}_{jl} + \frac{d\hat{\mathcal{F}}^3_{ijl}}{d\hat{u}_{mno}} \hat{D}_{kl} \right]. \tag{4.6}$$

The nodal coefficients $\hat{\mathcal{F}}^p_{ijk}$ with $p = 1, 2, 3$ defined in (2.50) are simply the transformed physical fluxes by means of the metric terms and can be rewritten for general conservation laws (2.2) as

$$\hat{\mathcal{F}}^p_{ijk} = \mathcal{F}^p(\hat{u}_{ijk}, \hat{\boldsymbol{q}}_{ijk}) = \sum_{d=1}^{3} (Ja)^p_{d,ijk} F_d(\hat{u}_{ijk}, \hat{\boldsymbol{q}}_{ijk}),$$

where $(Ja)^p_{d,ijk} = (Ja)^p_d(\xi^1_i, \xi^2_j, \xi^3_k)$ and $\hat{\boldsymbol{q}}_{ijk} \in \mathbb{R}^{d \cdot nVar}$ represents the element-local gradient vector of $\boldsymbol{u}$. For convenience, we skip the dependency of the flux in further investigations. Since the partial derivatives $\hat{q}^d_{ijk} \in \mathbb{R}^{nVar}$ also depend on the solution as can be seen in (2.63) we need to apply the chain rule

$$\begin{aligned}
\frac{d\hat{\mathcal{F}}^p_{ijk}}{d\hat{u}_{mno}} &= \sum_{d=1}^{3} (Ja)^p_{d,ijk} \left(\frac{\partial F_d}{\partial \hat{u}_{ijk}} \frac{\partial \hat{u}_{ijk}}{\partial \hat{u}_{mno}} - \sum_{r=1}^{3} \frac{\partial F^D_d}{\partial \hat{q}^r_{ijk}} \frac{\partial \hat{q}^r_{ijk}}{\partial \hat{u}_{mno}} \right) \\
&= \sum_{d=1}^{3} (Ja)^p_{d,ijk} \left(\left. \frac{\partial F_d}{\partial u} \right|_{(\hat{u}_{ijk}, \hat{\boldsymbol{q}}_{ijk})} \delta_{im} \delta_{jn} \delta_{ko} \right. \\
&\qquad \left. - \sum_{r=1}^{3} \left. \frac{\partial F^D_d}{\partial (\nabla u)_r} \right|_{(\hat{u}_{ijk}, \hat{\boldsymbol{q}}_{ijk})} \frac{\partial \hat{q}^r_{ijk}}{\partial \hat{u}_{mno}} \right),
\end{aligned} \tag{4.7}$$

where we assume the flux-splitting given in (2.3). Here, the flux derivatives can be computed analytically with the definition of the physical flux (2.4) and (2.5) for the Navier-Stokes equations. Considering purely hyperbolic problems with $\hat{\mathcal{F}}^p_{ijk} = \mathcal{F}^p(\hat{u}_{ijk})$, we obtain by inserting (4.7) into (4.6)

$$\begin{aligned}
\frac{dR^V_{ijk}}{d\hat{u}_{mno}} = -\frac{1}{J_{ijk}} \Bigg[\quad & \hat{D}_{im} \sum_{d=1}^{3} (Ja)^1_{d,mjk} \left.\frac{\partial F^A_d}{\partial u}\right|_{\hat{u}_{mjk}} \delta_{jn}\delta_{ko} \\
& + \hat{D}_{jn} \sum_{d=1}^{3} (Ja)^2_{d,ink} \left.\frac{\partial F^A_d}{\partial u}\right|_{\hat{u}_{ink}} \delta_{im}\delta_{ko} \\
& + \hat{D}_{ko} \sum_{d=1}^{3} (Ja)^3_{d,ijo} \left.\frac{\partial F^A_d}{\partial u}\right|_{\hat{u}_{ijo}} \delta_{im}\delta_{jn} \Bigg].
\end{aligned} \tag{4.8}$$

We can already recognize from this formula the high sparsity pattern of the Jacobian which we gain by the tensor-product property.

For the additional parabolic terms in (4.7), we compute the derivative of the gradient $\hat{q}^r_{ijk}$ with respect to the solution by means of the BR2 scheme, where $\hat{q}^r_{ijk}$ is given by equation (2.63). Here, the flux function $\boldsymbol{\mathcal{U}}^r$ defined in (2.32) and the numerical flux function $\boldsymbol{H}^*$ given in (2.40) depends on u as

$$\begin{aligned}
\frac{d\hat{\mathcal{U}}^{r,s}_{ijk}}{d\hat{u}_{mno}} &= (Ja)^s_{r,ijk} \frac{\partial \hat{u}_{ijk}}{\partial \hat{u}_{mno}} = (Ja)^s_{r,ijk}\delta_{im}\delta_{jn}\delta_{ko}, \\
\frac{d[H^*_r \hat{s}]^{+\xi^1}_{jk}}{d\hat{u}_{mno}} &= -\frac{1}{2}\frac{\partial \hat{u}^{+\xi^1}_{jk}}{\partial \hat{u}_{mno}} [\hat{s}n_r]^{+\xi^1}_{jk} = -\frac{1}{2} [\hat{s}n_r]^{+\xi^1}_{jk} \ell_m(1)\delta_{jn}\delta_{ko},
\end{aligned} \tag{4.9}$$

demonstrating the procedure without loss of generality only for the $+\xi^1$ side. In the last equation we used the identity

$$\hat{u}^{+\xi^1}_{jk} = \sum_{i=0}^{N} \hat{u}_{ijk}\ell_i(1). \tag{4.10}$$

Hence, the derivative of the gradient reads as

$$\begin{aligned}
&\frac{\partial \hat{q}^r_{ijk}}{\partial \hat{u}_{mno}}\\
&= \frac{1}{J_{ijk}}\Bigg[\quad \left(D_{im}(Ja)^1_{r,mjk} - \frac{1}{2}\left(L^+_{im}[\hat{s}n_r]^{+\xi^1}_{jk} + L^-_{im}[\hat{s}n_r]^{-\xi^1}_{jk}\right)\right)\delta_{jn}\delta_{ko}\\
&\qquad + \left(D_{jn}(Ja)^2_{r,ink} \quad - \frac{1}{2}\left(L^+_{jn}[\hat{s}n_r]^{+\xi^1}_{ik} + L^-_{jn}[\hat{s}n_r]^{-\xi^1}_{ik}\right)\right)\delta_{im}\delta_{ko}\\
&\qquad + \left(D_{ko}(Ja)^3_{r,ijo} \quad - \frac{1}{2}\left(L^+_{ko}[\hat{s}n_r]^{+\xi^1}_{ij} + L^-_{ko}[\hat{s}n_r]^{-\xi^1}_{ij}\right)\right)\delta_{im}\delta_{jn}\Bigg],
\end{aligned} \tag{4.11}$$

with the pre-computed quantity

$$L^{\pm}_{ij} = \hat{\ell}_i(\pm 1)\ell_j(\pm 1).$$

Plugging this into equation (4.7), we can see that the sparsity of the Jacobian related to the parabolic terms decreases compared to the hyperbolic case described by the first summand in (4.7).

4.1.2 Surface Integral

The next step is to compute the Jacobian of the surface part (4.5). Here, we consider the dependency of the numerical flux F^* exemplary at the surface $+\xi^1$ for an arbitrary element with outward pointing normal vector $\boldsymbol{n}$

$$\begin{aligned}
[F^*\hat{s}]^{+\xi^1} = [F^*\hat{s}]^{+\xi^1}(&u_L(u), u_R(u^{ne}), \boldsymbol{q}_L(u, u_L(u), u_R(u^{ne})),\\
&\boldsymbol{q}_R(u^{ne}, u_L(u), u_R(u^{ne})), \boldsymbol{n}),
\end{aligned}$$

where u describes the solution restricted to the considered element and u^{ne} to the adjacent element which have the common surface $+\xi^1$. The surface values of the solution vector and the gradient vector on $+\xi^1$ are given by u_L, u_R and $\boldsymbol{q}_L, \boldsymbol{q}_R$ respectively, where $._L$ corresponds to the considered element and $._R$ to the neighboring element. We restrict the derivation to the surface $+\xi^1$, dropping the superscript for legibility.

The Jacobian of the numerical flux is then of the form

$$\frac{d[F^*\hat{s}]_{jk}}{d\hat{u}_{mno}} = \frac{\partial([F_A^*\hat{s}]_{jk} - [F_D^*\hat{s}]_{jk})}{\partial[\hat{u}_L]_{jk}} \frac{\partial[\hat{u}_L]_{jk}}{\partial\hat{u}_{mno}} \tag{4.12}$$

$$- \sum_{d=0}^{N} \left(\frac{\partial[F_D^*\hat{s}]_{jk}}{\partial[\hat{q}_L^d]_{jk}} \frac{d[\hat{q}_L^d]_{jk}}{d\hat{u}_{mno}} + \frac{\partial[F_D^*\hat{s}]_{jk}}{\partial[\hat{q}_R^d]_{jk}} \frac{d[\hat{q}_R^d]_{jk}}{d\hat{u}_{mno}} \right), \tag{4.13}$$

where we used again the flux splitting (2.3). Beginning with the first term, the derivative of the Riemann flux F_A^* with respect to the state vector is realized as finite difference quotient

$$\frac{\partial[F_A^*\hat{s}]_{jk}}{\partial[\hat{u}_L]_{jk}} = \frac{\hat{s}_{jk}}{\epsilon_{\text{FD}}} \left(F_A^*([\hat{u}_L]_{jk} + \epsilon_{\text{FD}}, [\hat{u}_R]_{jk}, \boldsymbol{n}) - F_A^*([\hat{u}_L]_{jk}, [\hat{u}_R]_{jk}, \boldsymbol{n})\right) \tag{4.14}$$

in order to obtain a flexible implementation allowing the choice of arbitrary Riemann solvers. The perturbation parameter ϵ_{FD} is chosen constantly as $\epsilon_{\text{FD}} = 10^{-8}$, which is employed for all further computations without any numerical issue. When deriving the Riemann flux analytically we encounter problems with non-differentiable functions as for the Lax-Friedrich flux. Issues within this scope can also appear for the finite difference approach, but we have never faced difficulties in solving the linear system due to the perturbation size.

The diffusion part F_D^* of the numerical flux function is chosen in both approaches of Bassi and Rebay [9, 12] as the averaged value illustrated in (2.38), hence

$$[F_D^*]_{jk} = \frac{1}{2}\left(\boldsymbol{F}^D([\hat{u}_L]_{jk}, [\hat{\boldsymbol{q}}_L]_{jk}) + \boldsymbol{F}^D([\hat{u}_R]_{jk}, [\hat{\boldsymbol{q}}_R]_{jk})\right) \cdot \boldsymbol{n}. \tag{4.15}$$

The derivative with respect to the solution vector yields

$$\frac{\partial[F_D^*\hat{s}]_{jk}}{\partial[\hat{u}_L]_{jk}} = \frac{1}{2}\hat{s}_{jk} \left(\left.\frac{\partial\boldsymbol{F}^D}{\partial u_L}\right|_{([\hat{u}_L]_{jk}, [\hat{\boldsymbol{q}}_L]_{jk})} + \left.\frac{\partial\boldsymbol{F}^D}{\partial u_R}\right|_{([\hat{u}_R]_{jk}, [\hat{\boldsymbol{q}}_R]_{jk})} \right) \cdot \boldsymbol{n}, \tag{4.16}$$

where the derivatives of the physical diffusion flux can be computed analytically. The following derivative of the state vector interpolated on the surface $\hat{u}_L$ in equation (4.12) can be derived similar as in the volume integral from equation (4.10).

Plugging the first part (4.12) of the numerical flux derivative into the Jacobian of the DG operator restricted to the surface integral, we obtain

$$\begin{aligned}\frac{d\widetilde{R^S}_{ijk}}{d\hat{u}_{mno}} = -\frac{1}{J_{ijk}}\Bigg[& \left(\frac{\partial[F^*\hat{s}]^{+\xi^1}_{jk}}{\partial[\hat{u}_L]_{jk}} L^+_{im} - \frac{\partial[F^*\hat{s}]^{-\xi^1}_{jk}}{\partial[\hat{u}_L]_{jk}} L^-_{im}\right)\delta_{jn}\delta_{ko} \\ & + \left(\frac{\partial[F^*\hat{s}]^{+\xi^2}_{ik}}{\partial[\hat{u}_L]_{ik}} L^+_{jn} - \frac{\partial[F^*\hat{s}]^{-\xi^2}_{ik}}{\partial[\hat{u}_L]_{ik}} L^-_{jn}\right)\delta_{im}\delta_{ko} \\ & + \left(\frac{\partial[F^*\hat{s}]^{+\xi^3}_{ij}}{\partial[\hat{u}_L]_{ij}} L^+_{ko} - \frac{\partial[F^*\hat{s}]^{-\xi^3}_{ij}}{\partial[\hat{u}_L]_{ij}} L^-_{ko}\right)\delta_{im}\delta_{jn}\Bigg].\end{aligned} \tag{4.17}$$

Here, the derivative of the numerical flux function consists of the difference of the advection part (4.14) and the diffusion part (4.16). Thus, for purely hyperbolic problems the Jacobian of the surface integral $\frac{dR^S{}_{ijk}}{d\hat{u}_{mno}}$ has the same sparsity pattern as the Jacobian of the volume integral $\frac{dR^V{}_{ijk}}{d\hat{u}_{mno}}$ computed in (4.8) in the case of periodic but also Dirichlet boundary conditions.

Considering now the second part (4.13), the derivative of the numerical flux with respect to the gradient can be realized again by an analytical DOF-wise derivative of the physical flux function $\boldsymbol{F}^D$ in (4.15)

$$\frac{\partial[F^*_D\hat{s}]_{jk}}{\partial[\hat{q}^d_L]_{jk}} = \frac{1}{2}\hat{s}_{jk}\left.\frac{\partial\boldsymbol{F}^D}{\partial q^d}\right|_{[\hat{u}_L]_{jk}}\cdot\boldsymbol{n} \quad\text{and}\quad \frac{\partial[F^*_D\hat{s}]_{jk}}{\partial[\hat{q}^d_R]_{jk}} = \frac{1}{2}\hat{s}_{jk}\left.\frac{\partial\boldsymbol{F}^D}{\partial q^d}\right|_{[\hat{u}_R]_{jk}}\cdot\boldsymbol{n}.$$

Coming now to the most complex part, the total derivative of the surface gradient with respect to the volume solution vector. First, we rewrite the surface gradient of the BR2 scheme as given in (2.64)

$$\begin{aligned}[\hat{q}^d_L]^{+\xi^1}_{jk} = \sum_{i=0}^{N}\ell_i(1)\frac{1}{J_{ijk}}\Bigg[& \sum_{l=0}^{N} D_{il}\hat{\mathcal{U}}^{d,1}_{ljk} + D_{jl}\hat{\mathcal{U}}^{d,2}_{ilk} + D_{kl}\hat{\mathcal{U}}^{d,3}_{ijl} \\ & + \eta_{\mathrm{BR2}}[H^*_d\hat{s}]^{+\xi^1}_{jk}\,\hat{\ell}_i(1)\Bigg].\end{aligned}$$

Following the derivative of the volume gradient (4.11), we can reuse the formulas in (4.9), resulting in

$$\begin{aligned}\frac{d[\hat{q}_L^d]_{jk}^{+\xi^1}}{d\hat{u}_{mno}} = \sum_{i=0}^{N} \ell_i(1)\frac{1}{J_{ijk}}\Bigg[\quad & D_{im}(Ja)^1_{d,mjk}\delta_{jn}\delta_{ko} + D_{jn}(Ja)^2_{d,ink}\delta_{im}\delta_{ko} \\ & + D_{ko}(Ja)^3_{d,ijo}\delta_{im}\delta_{jn} \\ & - \left(\frac{1}{2}\eta_{\text{BR2}} L^{+}_{im}[\hat{s}n_d]^{+\xi^1}_{jk}\right)\delta_{jn}\delta_{ko}\Bigg].\end{aligned}$$

Introducing the abbreviation terms

$$\begin{aligned}[w_L^d]^{\pm\xi^1}_{jkm} &= [\hat{s}n_d]^{\pm\xi^1}_{jk}\left(\sum_{i=0}^{N}\frac{\ell_i(\pm1)L^{\pm}_{im}}{J_{ijk}}\right)\\ [w_L^d]^{\pm\xi^2}_{ikn} &= [\hat{s}n_d]^{\pm\xi^2}_{ik}\left(\sum_{j=0}^{N}\frac{\ell_j(\pm1)L^{\pm}_{jn}}{J_{ijk}}\right)\\ [w_L^d]^{\pm\xi^3}_{ijo} &= [\hat{s}n_d]^{\pm\xi^3}_{ij}\left(\sum_{k=0}^{N}\frac{\ell_k(\pm1)L^{\pm}_{ko}}{J_{ijk}}\right),\end{aligned} \tag{4.18}$$

which can be computed initially and defining the derivative of the volume integral of the gradient equation as

$$\begin{aligned}\frac{\partial\hat{q}^d_{ijk}|_{\text{volume}}}{\partial\hat{u}_{mno}} = \frac{1}{J_{ijk}}\Bigg[\quad & D_{im}(Ja)^1_{d,mjk}\delta_{jn}\delta_{ko} + D_{jn}(Ja)^2_{d,ink}\delta_{im}\delta_{ko} \\ & + D_{ko}(Ja)^3_{d,ijo}\delta_{im}\delta_{jn}\Bigg],\end{aligned} \tag{4.19}$$

the Jacobian of the surface gradients reads

$$\begin{aligned}\frac{d[\hat{q}_L^d]_{jk}^{\pm\xi^1}}{d\hat{u}_{mno}} &= \sum_{i=0}^{N}\ell_i(\pm1)\frac{\partial\hat{q}^d_{ijk}|_{\text{volume}}}{\partial\hat{u}_{mno}} - \left(\frac{1}{2}\eta_{\text{BR2}}[w_L^d]^{+\xi^1}_{jkm}\right)\delta_{jn}\delta_{ko}\\ \frac{d[\hat{q}_L^d]_{ik}^{\pm\xi^2}}{d\hat{u}_{mno}} &= \sum_{j=0}^{N}\ell_j(\pm1)\frac{\partial\hat{q}^d_{ijk}|_{\text{volume}}}{\partial u_{mno}} - \left(\frac{1}{2}\eta_{\text{BR2}}[w_L^d]^{+\xi^2}_{ikn}\right)\delta_{im}\delta_{ko}\end{aligned}$$

$$\frac{d[\hat{q}_L^d]_{ij}^{\pm\xi^3}}{d\hat{u}_{mno}} = \sum_{k=0}^{N} \ell_k(\pm 1)\frac{\partial \hat{q}_{ijk}^d|_{\text{volume}}}{\partial \hat{u}_{mno}} - \left(\frac{1}{2}\eta_{\text{BR2}}[w_L^d]_{ijo}^{+\xi^3}\right)\delta_{im}\delta_{jn}.$$

Here, the sum in each Jacobian cancels out for two summands in (4.19) due to the Kronecker delta such that the matrix structure given in (4.17) is no longer valid for the mixed hyperbolic-parabolic case. Similarly to the volume term, the derivatives related to the gradient increases again the density of the Jacobian matrix.

The remaining summand in (4.13) consists of the derivative of the gradient which belongs to the neighboring element. Since it is random which face of the adjacent element is connected to the $+\xi^1$ face of the considered element, we restrict the derivation again on $+\xi^1$ but related to the neighboring element. The surface gradient $\hat{q}_R^d$ is written in quantities of the adjacent element denoted by the superscript ^{ne} as

$$[\hat{q}_R^d]_{jk}^{+\xi^1} = \sum_{i=0}^{N} \ell_i(1)\frac{1}{J_{ijk}^{ne}}\left[\sum_{l=0}^{N} D_{il}(\hat{\mathcal{U}}^{ne})_{ljk}^{d,1} + D_{jl}(\hat{\mathcal{U}}^{ne})_{ilk}^{d,2} + D_{kl}(\hat{\mathcal{U}}^{ne})_{ijl}^{d,3} + \eta_{\text{BR2}}[H_d^{*ne}\hat{s}^{ne}]_{jk}^{+\xi^1}\hat{\ell}_i(1)\right].$$

Differentiating with respect to the solution vector $\hat{u}_{mno}$ of the considered element, we obtain by means of formula (4.9) and definition (4.18) related to the neighboring quantities $\hat{s}^{ne}, \boldsymbol{n}^{ne}, J_{ijk}^{ne}$

$$\begin{aligned}\frac{\partial[\hat{q}_R^d]_{jk}^{+\xi^1}}{\partial \hat{u}_{mno}} &= \sum_{i=0}^{N} \ell_i(1)\frac{1}{J_{ijk}^{ne}}\eta_{\text{BR2}}\frac{\partial[H_d^{*ne}\hat{s}^{ne}]_{jk}^{+\xi^1}}{\partial \hat{u}_{mno}}\hat{\ell}_i(1)\\ &= \eta_{\text{BR2}}[w_R^d]_{jkm}^{+\xi^1}\delta_{jn}\delta_{ko},\end{aligned}$$

which is similar to the derivative of the gradient q_L^d but with the opposite sign. Again, we encounter the term $[w^d]$. Thus, we can save $[w_L^d]$ and $[w_R^d]$ for every global side and send them for parallel computations to the adjacent neighbor within the initialization process. Furthermore, we need to exchange the states u_L, u_R as well as the gradients $\boldsymbol{q}_L, \boldsymbol{q}_R$ on MPI sides before we build the Jacobian. The derivatives are implemented in FLEXI exploiting the tensor-product structure, which means that we assemble only the entries of the Jacobian which are non-zero, i.e. when the product of all Kronecker deltas is one.

Verifying the Analytically Computed Jacobian In order to control the accuracy of the analytically computed Jacobian, we implemented a reference Jacobian by means of the finite difference approach

$$\left(\frac{dR_{ijk}}{d\hat{u}_{mno}}\right)_{r,s} = \frac{\left(R_{ijk}((\hat{u}_{mno})_s + \epsilon_{\text{FD}})\right)_r - \left(R_{ijk}((\hat{u}_{mno})_s)\right)_r}{\epsilon_{\text{FD}}},$$

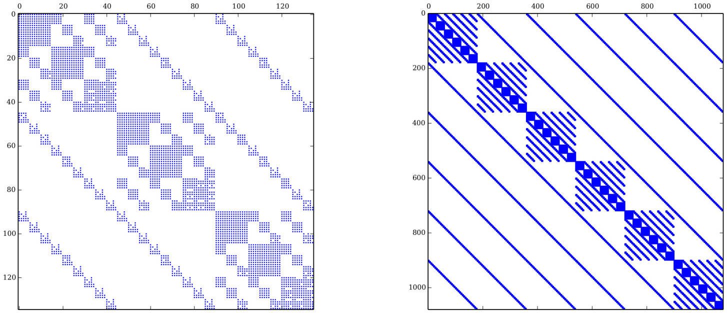
for $i,j,k,m,n,o = 0,\dots,N$ and $r,s = 1,\dots,n_{var}$, choosing $\epsilon_{\text{FD}} = 10^{-8}$. Here, $(\hat{u}_{mno})_s$ depicts the s-th entry of $\hat{u}_{mno} \in \mathbb{R}^{n_{var}}$ and respectively $(R_{ijk})_r$. This procedure requires $(N+1)^3 \cdot n_{var}$ times the displacement of the solution vector resulting in $(N+1)^3 \cdot n_{var}$ calls of the DGSEM operator R for each diagonal block or rather for each element. Thus, the finite difference approach suits only for debugging purposes considering small problem sizes. For real world problems the building time would be exceedingly high.

Matrix Pattern of Jacobian's Diagonal Blocks Due to the tensor-product ansatz, we gain a very sparse matrix structure of the diagonal block. Specific solvers can exploit this pattern by neglecting the zero-entries.

First, we want to examine for different polynomial degrees the matrix pattern and matrix density of the diagonal blocks for the purely hyperbolic case, here the three-dimensional Euler equations. The matrix density describes the rate between the number of non-zero entries and the total number of matrix entries. In Figure 4.1 the non-zero entries of one Jacobian diagonal block are marked blue for two specific polynomial degrees $N = 2$ in Figure 4.1a and $N = 5$ in Figure 4.1b. For higher N the structure of the tensor-product becomes more visible. The small dense diagonal blocks of size $(N+1) \cdot n_{var} \times (N+1) \cdot n_{var}$, in Figure 4.1b 30×30, correspond to the ξ derivative. The banded blocks within a 180×180 block are related to the η direction, where we have $(N+1)$ small diagonal η-blocks. And the $(N+1)$ off-block-diagonals as well as the $(N+1)$ 180×180 diagonal blocks result from the derivative in ζ-direction.

As already mentioned in the above derivation of the block diagonals $\mathbf{BJ}^{iElem}$, the parabolic terms within the Navier-Stokes equations extend the hyperbolic matrix pattern as illustrated in Figure 4.2.

Plotting the matrix density for the Euler- and Navier-Stokes equations in Figure 4.3, we can again see the drastic increase when adding viscous effects. In the two-dimensional case the density is shifted to much higher

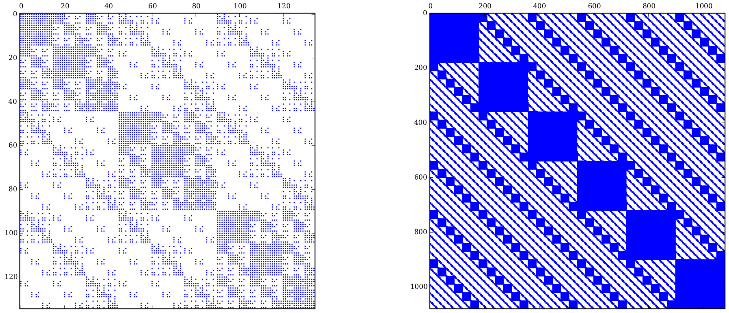


(a) N=2, matrix density: 0.23 (b) N=5, matrix density: 0.067

Figure 4.1: Matrix pattern for the non-zero entries of one diagonal block using the 3D Euler equations applied to the traveling vortex described in Appendix C.

values since the matrices are higher filled compared to 3D, due to the missing direction within the tensor-product. It is also noticeable, that for the two-dimensional equations, the slope with respect to the polynomial degree N also reduces. However, in any case the matrix density decreases with increasing N. This represents the efficiency gain for higher polynomial degrees due to the tensor-product collocation approach compared to standard DG methods requiring full matrix-vector products. Note, the worst efficiency gain is expected for the two-dimensional Navier-Stokes equations due to the very slight decrease of the density.

Building Time of the Block-Jacobi Matrix We evaluate the computational cost for assembling the block-Jacobi by means of the traveling vortex for the Navier-Stokes equations with $Re = 100$ given in Appendix C. In Figure 4.4 the relative building time is illustrated normalized by the computational time of the DGSEM operator which represents the cost of one explicit time step. Several different restrictions on the Navier-Stokes preconditioner are compared in 2D and 3D. Since we know from the sections before that the parabolic terms introduces additional entries compared to



(a) N=2, matrix density: 0.36 (b) N=5, matrix density: 0.17

Figure 4.2: Matrix pattern for the non-zero entries of one diagonal block using the 3D Navier-Stokes equations applied to the traveling vortex described in Appendix C, with $Re = 100$.

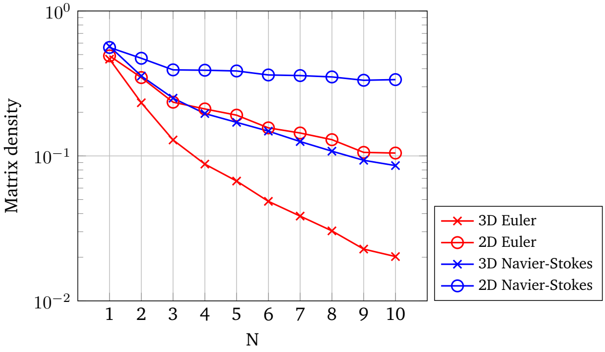


Figure 4.3: Matrix density of one diagonal block over polynomial degree N using the traveling vortex test case described in Appendix C.

an Euler preconditioner, we construct the so called Navier-Stokes NoFillIn preconditioner, which contributes to the Euler preconditioner only terms within the Euler matrix pattern as illustrated in Figure 4.1 in order to remain the sparse structure. The other preconditioner named as EulerPrecond describes the Euler preconditioner omitting the parabolic terms. For d dimensions the theoretical convergence behaves as $(N+1)^{2d}/(N+1)^{d+1}$ for the Navier-Stokes preconditioner and is plotted as dashed line in Figure 4.4. This scaling is derived by the total number of elements divided by the scaling of one DG operator examined in Section 2.4.8.

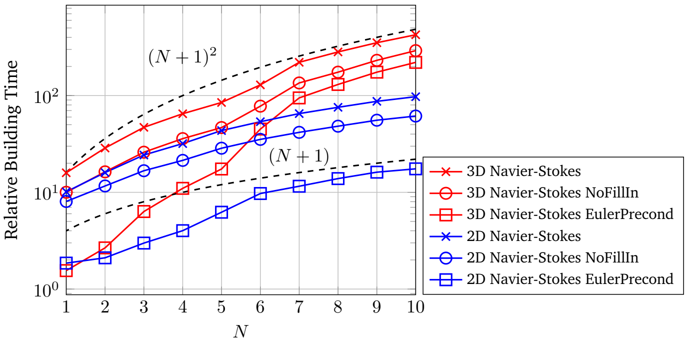


Figure 4.4: Building time of the block-Jacobi matrix in relation to the cost of one DGSEM operator call depending on the polynomial degree N.

EulerPrecond obviously requires the least building time but still its cost illustrates a significant multiple of the computational time of one explicit time step. We could extract the minimum CFL number from this figure so that we have a chance to outperform the explicit DG method neglecting the time required to solve the equation system. In Section 4.3.3, another remedy is investigated by keeping the preconditioner fixed for several time steps in order to save building time. Considering the absolute building time of the block-Jacobi in Figure 4.5, we can find a much better convergence rate as the theoretical one given by $(N+1)^{2d}$ which can be explained by the sparsity of the Jacobian. All preconditioners, the pure Navier-Stokes preconditioner, the NoFillIn version and EulerPrecond show a scaling of $(N+1)^5$ in 3D and of almost $(N+1)^3$ in 2D for higher N.
Concluding this chapter, we derived analytically the diagonal blocks of the block-Jacobian matrix of the DGSEM operator. Then, we analyzed the diagonal blocks' matrix pattern and matrix density for either the hyperbolic Eu-

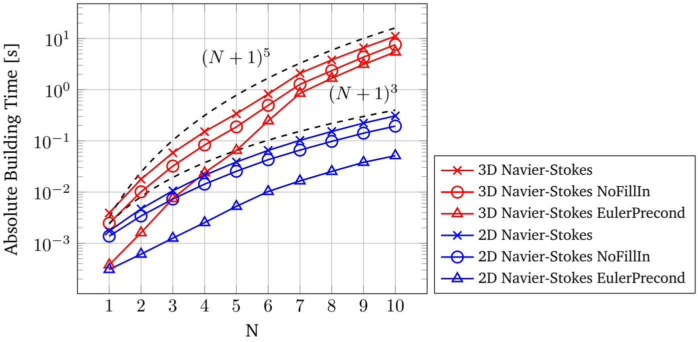


Figure 4.5: Absolute building time of the block-Jacobi matrix over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.

ler equations and the mixed hyperbolic/parabolic Navier-Stokes equations. Furthermore, we compared different restrictions of the Navier-Stokes preconditioner in relation to their building times. Obviously the Euler preconditioner depicts the lowest building time followed by the NoFillIn variant of the Navier-Stokes preconditioner. But, since they are all approximations it is still to show how they can affect the acceleration of the linear solver. In the next chapter, we will focus on solving the element-local linear equation systems related to the diagonal blocks.

4.2 Solving the Preconditioned System

As discussed in Section 3.3, the introduction of a preconditioner transforms the original linear system $\boldsymbol{Ax} = \boldsymbol{b}$ into the preconditioned system (3.22). Due to the structure of the block-Jacobi preconditioner given in (4.1), we only need to solve in each linear iteration j the linear system element-wise with the diagonal blocks as system matrices

$$\mathbf{BJ}^{iElem}\, \boldsymbol{u}_j^{iElem} = \boldsymbol{v}_j^{iElem}, \tag{4.20}$$

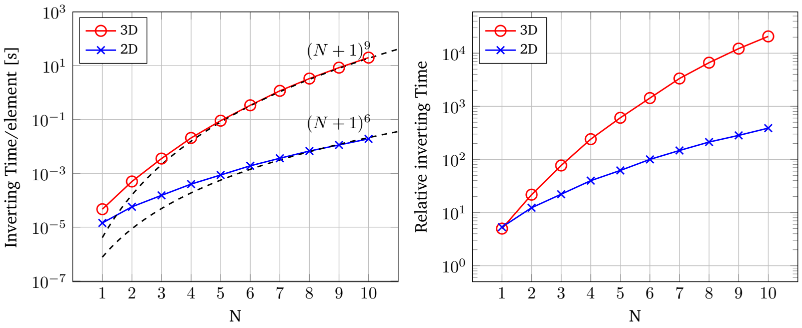
with $\boldsymbol{u}_j^{iElem}, \boldsymbol{v}_j^{iElem} \in \mathbb{R}^{(N+1)^d \cdot n_{var}}$ for $iElem = 1, \ldots, n_{el}$. The key factor for an efficient preconditioner is, finding a trade-off for the computational cost solving (4.20) within every GMRES iterations and for reducing the number of iterations.

In the following, we give an overview of the employed solving strategies including their building and application time for the two- and three-dimensional Navier-Stokes equations.

4.2.1 Exact Inverse by LU Decomposition

The simplest solution of (4.20) is the exact solution given by the lower-upper (LU) decomposition of the diagonal blocks $\mathbf{BJ}^{iElem}$. We assemble the element-local LU factorization by means of intern LAPACK routines. Further, we compute the inverse and store it since the matrix-vector multiplication is much faster than the application of the LU factorization by backward forward sweeps using LAPACK. This strategy pays off for higher numbers of GMRES iterations. Due to the exact solution, it is reasonable to assume that the minimum number of GMRES iterations is obtained when using the block-Jacobi preconditioner. However, the computation of the LU decomposition is very expensive and its inverse requires in general full storage even if $\mathbf{BJ}^{iElem}$ is a sparse matrix resulting in significantly high application costs scaling as $(N+1)^{2d}$ per element. Hence, this preconditioner loses the characteristics of the tensor-product and its sparsity property. Here, we present the LU preconditioner in terms of comparison since it solves the element-local systems exactly.

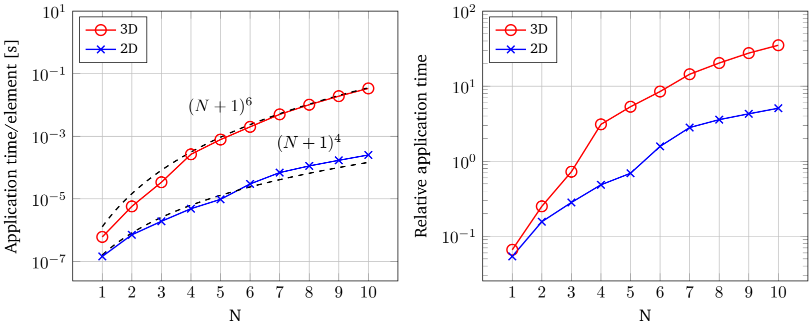
In Figure 4.6 the computational time for the inverting process of one diagonal block element using the LU decomposition for the two- and three-dimensional Navier-Stokes equations is illustrated. The left Figure 4.6a depicts the dramatical increase with respect to the polynomial degree N, since the size of the diagonal blocks scales as $(N+1)^{2d}$. However, the absolute time is in good agreement with the theoretical scaling $(N+1)^{3d}$ for higher N. Considering the right Figure 4.6b, we can see that the cost becomes prohibitive for large N in relation to the cost of one DGSEM operator call. As well as the application of the inverse which is presented in Figure 4.7 is not affordable for physical problems. The absolute application time scales as matrix-vector multiplications, i.e. as $(N+1)^{2d}$, represented in Figure 4.7a. Compared to the time of the DGSEM operator, which represents the cost for one explicit time step, we can see in Figure 4.7b that it becomes challenging for the implicit method to outperform the explicit DGSEM for high polynomial degrees.



(a) Absolute inverting time.

(b) Inverting time compared to the time of DGSEM operator.

Figure 4.6: Inverting time of the block-Jacobi matrix over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.



(a) Absolute application time.

(b) Application time compared to the time of DGSEM operator.

Figure 4.7: Application time of the exact inverse of the block-Jacobi matrix plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.

Considering the case CFL $= 10$, we would need 10 explicit steps for one implicit step so that for $N > 6$ in 3D one GMRES iteration is already more costly than the explicit scheme even without counting the time for assembling the block-Jacobi preconditioner and the inverse of it.
Note, that it is possible to obtain even less GMRES iterations when using the global Jacobi matrix, but this is not in accordance with the strategy of matrix-free methods.
Summarized, direct solvers are not suitable for higher dimensions and polynomial degrees. Hence, we need to find an approximation of each block matrix $\mathbf{BJ}^{iElem}$ exploiting its sparsity when solving the linear system in order to reduce the application cost. So, we employ not only the block-Jacobi as approximation as well as an approximating inverse of each block, resulting obviously in a larger amount of GMRES iterations.

4.2.2 Symmetric Gauss-Seidel

Lower-upper symmetric Gauss-Seidel (LU-SGS) methods were introduced by Jameson and Yoon [68] for the implicit resolution of the Euler equations. They are often used within the CFD community [26, 94] as implicit solver operating on the whole Jacobian, where each block is inverted by the LU decomposition. Luo et al. combined in [82] the GMRES iteration method with LU-SGS as preconditioner for the whole Jacobian. Here, we want to use the LU-SGS preconditioner for each block of the block-Jacobian matrix. In this case, the diagonal blocks are only scalar, so that we denote the preconditioner as Symmetric Gauss-Seidel (SGS).
Similar to the LU decomposition, the symmetric Gauss-Seidel preconditioner $\boldsymbol{P}_{SGS}^{iElem}$ is defined by

$$\mathbf{BJ}^{iElem} \approx \boldsymbol{P}_{\text{SGS}}^{iElem} := \tilde{\mathbf{L}}\tilde{\mathbf{U}},$$

but $\tilde{\mathbf{L}}$ describes a lower triangular and $\tilde{\mathbf{U}}$ a unit upper triangular with

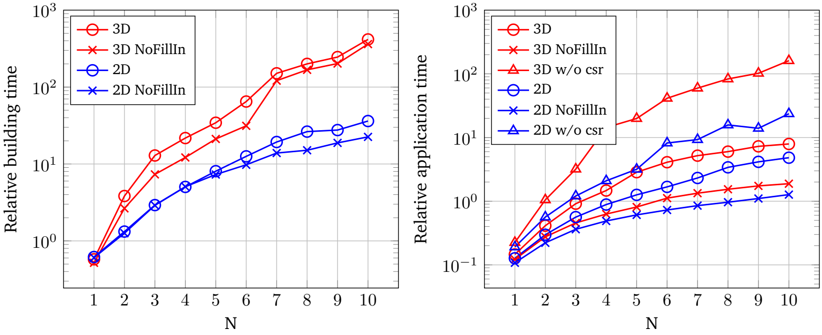
$$\tilde{\mathbf{L}} = (\mathbf{D} + \mathbf{L}), \quad \tilde{\mathbf{U}} = \mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}),$$

where $\mathbf{BJ}^{iElem}$ is split by $\mathbf{BJ}^{iElem} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ in the strictly lower triangular matrix $\mathbf{L}$, the diagonal $\mathbf{D}$ and the strictly upper triangular matrix $\mathbf{U}$. Contrastingly to the LU decomposition, we keep for the SGS preconditioner the sparsity pattern of each block $\mathbf{BJ}^{iElem}$, which is illustrated in Figure

4.2. Hence, this preconditioner does not require additional memory storage and computational effort to compute it. Now, the system (4.20) can be solved approximately by one forward sweep followed by one backward sweep given by

$$\begin{aligned} (\mathbf{D}+\mathbf{L})(\boldsymbol{u}_j^{iElem})^* &= \boldsymbol{v}_j^{iElem} && \text{forward sweep} \\ \mathbf{D}^{-1}(\mathbf{D}+\mathbf{U})\ \boldsymbol{u}_j^{iElem} &= (\boldsymbol{u}_j^{iElem})^* && \text{backward sweep.} \end{aligned}$$

SGS belongs to the class of preconditioner, where the explicit computation of its inverse is not required. We use the preconditioner implicitly by its impact through forward and backward elimination.
In order to exploit the sparsity pattern for the preconditioner application, we introduce the compressed sparse row (CSR) format storing only the non-zero entries of $\boldsymbol{P}_{SGS}^{iElem}$. The building and application costs of the SGS preconditioner in CSR format are presented in Figure 4.8 for the two- and three-dimensional Navier-Stokes equations.



(a) Relative building time.

(b) Relative application time.

Figure 4.8: Relative building and application time of the SGS preconditioner in CSR format compared to the computing time of one DGSEM operator plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.

They are plotted as relation between the absolute computing time and the cost for one DGSEM operator. Note, for each dimension d we use the corresponding d-dimensional DGSEM operator, so that the relative application time in 2D can be higher than in 3D due to the reduced sparsity. Here, the building time as illustrated in Figure 4.8a concerns only the cost of writing the block matrices into CSR format, since the SGS preconditioner itself does not need to be built as discussed above. Figure 4.8a depicts a non-negligible gradient of the building cost but with overall smaller total numbers than the exact inverting considered in the section before. Reducing the number of non-zero entries by allowing only the matrix pattern corresponding to the Jacobian of the Euler equations, we can see similar building times for both dimensions, but a significant reduction in the application cost given in Figure 4.8b. Thus, we can gain a very efficient application procedure when exploiting the sparsity by using the CSR format in contrast to the LU decomposition which loses the sparsity. However, we still have to examine the influence of the neglecting entries on the convergence speed of the linear solver since the approximation of the block-Jacobi is degraded. For reasons of comparison we also depict in Figure 4.8b the application time of the SGS preconditioner in the full format, not in CSR format, due to the benefit of no building time. But we can observe that the costs are immense also in comparison to the application of the exact LU decomposition illustrated in Figure 4.7b because of the additional multiplication of the inverted diagonal matrix in the backward sweep. For real world problems when using high CFL numbers, the cost of the preconditioner assembling carries no weight but the acceleration of the preconditioner's application is essential making the usage of a reduced storage format mandatory.

4.2.3 Incomplete LU Decomposition

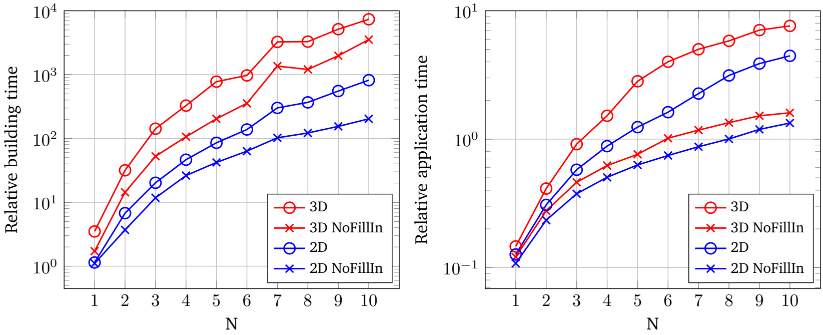
A widely used class of preconditioners is the incomplete LU (ILU) factorization. In [99, 100, 41, 19, 53] block-ILU preconditioners are employed for the whole Jacobian matrix, hence considering also the off-diagonal blocks. Here again, we want to apply the ILU factorization only to each diagonal block. As investigated before, the complete LU decomposition is very costly and requires the full storage also for sparse matrices. In contrast, the ILU decomposition exploits the sparsity pattern as the previous SGS preconditioner. Each diagonal block is decomposed as

$$\mathbf{BJ}^{iElem} = \mathbf{LU} + \mathbf{F},$$

where the ILU preconditioner $\boldsymbol{P}_{\text{ILU}}^{iElem}$ is then given by

$$\boldsymbol{P}_{\text{ILU}}^{iElem} = \mathbf{LU}.$$

The matrices $\mathbf{L}$ and $\mathbf{U}$ describe a unit lower and a upper triangular matrix respectively, with the same sparsity pattern as $\mathbf{BJ}^{iElem}$, such that $\mathbf{LU} = \mathbf{BJ}^{iElem}$ for the non-zero entries of $\mathbf{BJ}^{iElem}$. Hence, ILU preconditioners require the same storage requirements as $\mathbf{BJ}^{iElem}$ itself. In order to improve the quality of the approximation, it is possible to impose additional entries for the decomposition measured by level of fill resulting in a higher memory demand and application time. We consider no additional level of fill, denoted by ILU(0) retaining the original sparsity pattern. The application of such a preconditioner is implemented as forward and following backward elimination for solving (4.20), which is similar to the SGS preconditioner. Hence, the explicit inversion is not required.
The ILU(0) factorization is performed in-place as suggested in [108] and is then stored in the CSR format considering only the non-zero entries. We can expect that the number of GMRES iterations increases compared to a LU decomposition due to the approximation, but the computational cost of one iteration decreases as we exploit the sparsity pattern.
Figure 4.9 illustrates the building and application time for the two- and three-dimensional Navier-Stokes equations. Again, the computational times are given in relation to the required time of one DGSEM operator. The building time involves in this case the in-place ILU(0) decomposition and the restoring in CSR format, such that the left Figure 4.9a shows the overhead for computing the decomposition in comparison to Figure 4.8a for the SGS preconditioner. For the full ILU(0) preconditioner we obtain similar curves for the building time as for the exact inverse computed by the LU decomposition given in Figure 4.6b for both dimensions, but for the reduced sparsity pattern corresponding to the Euler equations denoted by NoFillIn the relative building time is significantly reduced since we compute the incomplete LU decomposition only for the non-zero entries. For the application time in Figure 4.9b the identical curves as for the SGS preconditioner presented in 4.8b are visible due to the similar forward and backward elimination procedure. Hence, ILU(0) is apparently in total more expensive as the SGS preconditioner due to its building cost, but it is to expect that $\boldsymbol{P}_{\text{ILU}}^{iElem}$ is a better approximation than $\boldsymbol{P}_{\text{SGS}}^{iElem}$ leading to less GMRES iterations for ILU(0). Further it is to mention, for higher polynomial orders N, the sparsity of $\mathbf{BJ}^{iElem}$ increases as shown in Figure 4.3,



(a) Relative building time.

(b) Relative application time.

Figure 4.9: Relative building and application time of the ILU(0) preconditioner in CSR format compared to the computing time of one DGSEM operator plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.

hence also the sparsity of the decomposed matrices **L** and **U** increases, so that the quality of the approximation **LU** decreases.

4.2.4 Tensor-Product

In this section, we want to investigate the effect of a tensor-product preconditioner, which exploits the structure of the DGSEM operator given in (2.61). One novel approach has been followed in [95] by constructing a tensor-product preconditioner by means of the Kronecker-product singular value decomposition (KSVD) in order to obtain a two-term tensor-product of Sylvester-type, which can be solved by the Bartels-Stewart algorithm [7] using a Schur decomposition. Another strategy is the usage of an alternating direction Implicit (ADI) scheme investigated in [68].
The idea of a tensor-product preconditioner is, to reduce the cost of solving one large d-dimensional linear equation system into d one-dimensional equation systems by exploiting the tensor-product structure of the spatial operator. Splitting the DGSEM operator into each dimension, $\boldsymbol{R}$ can be

written in the three-dimensional case as

$$\boldsymbol{R} = \boldsymbol{R}_{x_1} + \boldsymbol{R}_{x_2} + \boldsymbol{R}_{x_3},$$

where $\boldsymbol{R}_{x_i}$ corresponds to the i-th line of the right hand side in equation (2.61). This splitting is appropriate for hyperbolic equations. When considering additionally parabolic terms, we omit the cross-wise dependency visible in equation (4.11). The assembling of the 1D Jacobian matrices is given for the x_i-direction by

$$\begin{aligned}\mathbf{BJ}_{x_i}^{iElem} := \frac{d(\boldsymbol{R}_{x_i})_{ijk}}{d\hat{u}_{mno}} \in \mathbb{R}^{(N+1)n_{var}\times(N+1)n_{var}} \\ \text{for each } j = n = 1, \dots, N+1, \\ k = o = 1, \dots, N+1,\end{aligned}$$

resulting in N^2 1D matrices for each direction. This is done similarly for the other space dimensions, using only one line of (4.8) and (4.17) for each spatial derivative separately. Thus, we approximate the diagonal block of the Jacobian as the following-tensor-product

$$\begin{aligned}\mathbf{BJ}^{iElem} \approx & \Big((\boldsymbol{I} - \alpha\Delta t\mathbf{BJ}_{x_1}^{iElem}) \otimes \boldsymbol{I} \otimes \boldsymbol{I}\Big) \\ & + \Big(\boldsymbol{I} \otimes (\boldsymbol{I} - \alpha\Delta t\mathbf{BJ}_{x_2}^{iElem}) \otimes \boldsymbol{I}\Big) \\ & + \Big(\boldsymbol{I} \otimes \boldsymbol{I} \otimes (\boldsymbol{I} - \alpha\Delta t\mathbf{BJ}_{x_3}^{iElem})\Big),\end{aligned}$$

which has the same density pattern as the Jacobian of the Euler equations. The above approximation is exact when considering only hyperbolic eqations.

The total computational cost of building the 1D matrices is given in Figure 4.10 denoted by 3D Tensor and 2D Tensor respectively. The time is again given in relation to the cost of one DGSEM operator call. For comparison we plotted additionally in Figure 4.10 the relative time for building the whole 2D an 3D diagonal block-Jacobian and the restricted Jacobian matrices related to the density pattern of the Euler equations, denoted by 2D NoFillIn and 3D NoFillIn respectively. Obviously, the full Jacobian requires the highest computational time followed by the NoFillIn version, which neglects the same cross-dimensional entries as the tensor-product of the 1D Jacobians. The difference in time between the last two approaches is attributed to the much higher memory access for the NoFillIn version consisting of $\mathcal{O}(N^{2d})$ entries than for the one-dimensional matrices with $\mathcal{O}(dN^2)$ entries.

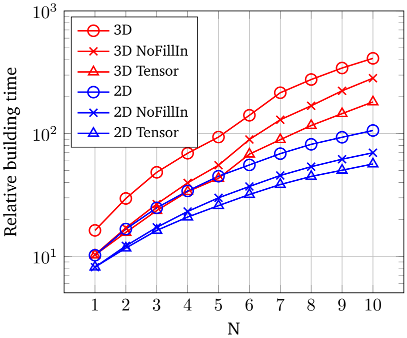
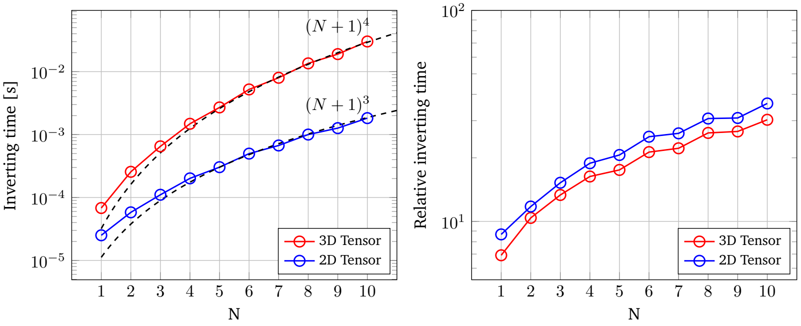


Figure 4.10: Relative building time of the d one-dimensional block-Jacobian matrices (2D/3D Tensor) compared to the relative building time of the full Jacobian (2D/3D) and NoFillIn version. Times are given in relation to one DGSEM operator call.

The linear equation system in (4.20) is then approximately solved by the sequence

$$\begin{aligned}\left((\boldsymbol{I}-\alpha\Delta t\mathbf{BJ}^{iElem}_{x_1})\otimes\boldsymbol{I}\otimes\boldsymbol{I}\right)\boldsymbol{u}^{*}&=\boldsymbol{v}^{iElem}_j\\ \left(\boldsymbol{I}\otimes(\boldsymbol{I}-\alpha\Delta t\mathbf{BJ}^{iElem}_{x_2})\otimes\boldsymbol{I}\right)\boldsymbol{u}^{**}&=\boldsymbol{v}^{iElem}_j\\ \left(\boldsymbol{I}\otimes\boldsymbol{I}\otimes(\boldsymbol{I}-\alpha\Delta t\mathbf{BJ}^{iElem}_{x_3})\right)\boldsymbol{u}^{***}&=\boldsymbol{v}^{iElem}_j\\ \boldsymbol{u}^{iElem}_j&=\boldsymbol{u}^{*}+\boldsymbol{u}^{**}+\boldsymbol{u}^{***}.\end{aligned}\tag{4.21}$$

Due to the tensor-product, only the inverse of each 1D matrix is required, so that the system (4.21) can be solved by multiplying the inverse with the appropriate entries of the right-hand side vector. Hence, the exact inverting process is reduced from computing the LU decomposition of one large matrix of the size $(N+1)^d\cdot n_{var}\times(N+1)^d\cdot n_{var}$ to the decomposition of $(N+1)^{d-1}$ small matrices of the size $(N+1)\cdot n_{var}\times(N+1)\cdot n_{var}$.
In Figure 4.11a, we can see that the absolute inverting time scales perfectly as $(N+1)^{d+1}$ for greater values of N excelling the theoretical scaling of $\mathcal{O}(N^{d-1}N^3)$. Hence, the gradient of the relative inverting time visualized in Figure 4.11b is almost the same for 2D and 3D. Among all the presented preconditioners the assembling time of the tensor-product preconditioner,



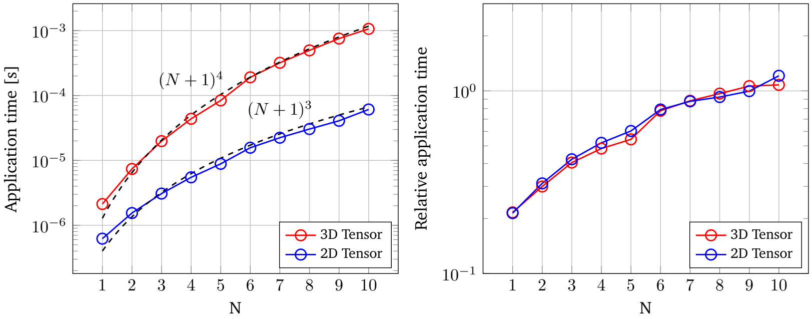
(a) Absolute inverting time.

(b) Inverting time compared to the time of DGSEM operator.

Figure 4.11: Inverting time of the tensor-product preconditioner per diagonal element plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.

which means in this case the computation of the inverse since it represents an explicit preconditioner, describes the most efficient.

Figure 4.12a shows the absolute application time following perfectly the theoretical complexity of $\mathcal{O}(N^2 N^{d-1})$. Also the relative time for the preconditioner application illustrated in Figure 4.12b depicts the same gradient in 2D and 3D. For the shown polynomial degrees N the time is even identical, which makes the tensor-product preconditioner especially for the three-dimensional case very efficient. We want to point out, that in total the application cost is reduced from $\mathcal{O}(N^{2d})$ investigated in Section 4.2.1 for the LU decomposition of a full diagonal block to $\mathcal{O}(N^{d+1})$.



(a) Absolute application time.

(b) Application time compared to the time of DGSEM operator.

Figure 4.12: Application time of the tensor-product preconditioner plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$.

4.2.5 Computational Effort of the Preconditioners

In this section, we compare the storage demand of the presented preconditioners and summarize the computational cost given in building and application time of each preconditioner as investigated in the previous section.
First, we want to concentrate on the memory space which needs to be allocated for each preconditioner. In Figure 4.13 the storage requirement in kB for one diagonal block for two- and three dimensional preconditioners is illustrated using double precision. The exact inverse achieved by the LU decomposition requires the full block memory of $(N+1)^{2d} \cdot {n_{var}}^2$ and forms in both dimension the maximum storage.
In 3D, LU requires for $N = 4$ already 3 MB per element and increases significantly to 338 MB per element at $N = 10$. This limits the implicit solver to smaller polynomial degrees in order to guarantee efficient preconditioning. The memory requirements for SGS and ILU(0) are the same since they are stored in the CSR format and benefit from the same block sparsity. The SGS/ILU(0) NoFillIn version maintains the same matrix pattern as for the Euler equations and hence reduces the storage compared to the full SGS/ILU(0) for $N = 4$ two times resulting in 0.3 MB and for $N = 10$ almost four times, resulting in 7.9 MB per element in 3D. All two-dimensional pre-

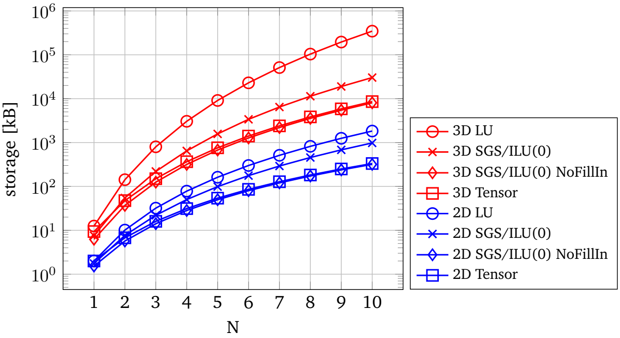
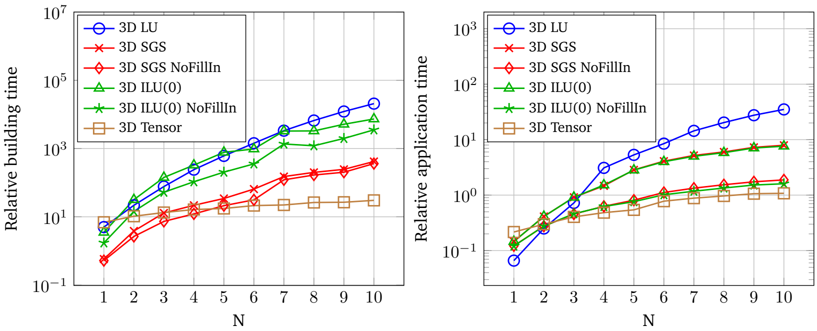


Figure 4.13: Memory demand for one diagonal block using the traveling vortex test case described in Appendix C with $Re = 100$.

conditioners require obviously much less storage, whereas the SGS/ILU(0) NoFillIn and the tensor-product preconditioners constitute again the minimum. By construction, the tensor-product preconditioner needs the same memory as the SGS/ILU(0) NoFillIn since they consist of the same entries.

Now, we consider the computational performance in terms of building and application cost as investigated already solely in Section 4.2 for both dimensions. Comparing now all presented preconditioners in 3D, the building costs for the tensor-product preconditioner (inverting time) is the lowest only for higher polynomial degrees N as can be seen in Figure 4.14a. Beside the calculation of the LU decomposition, assembling the ILU(0) preconditioner for the full Jacobian represent the highest computational costs. Maintaining the Euler density pattern by the ILU(0) NoFillIn preconditioner we can reduces the time significantly compared to the full matrix pattern. Obviously, SGS requires less building time than ILU(0), since only the storage into the CSR format is necessary, whereas ILU(0) computes additionally the LU decomposition.
In terms of application cost depicted in Figure 4.14b, ILU(0) and SGS for each version have almost identical CPU times since they exploit the same sparsity pattern. But employing the Euler density pattern, denoted by NoFillIn, reduces the application time significantly. If the number of GMRES iterations are not much affected by the reduced density pattern, it



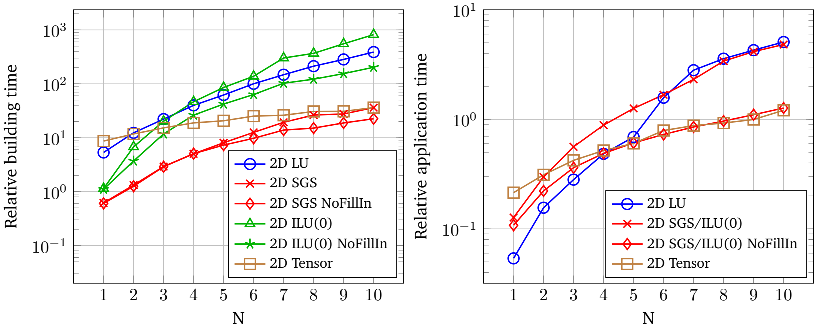
(a) Building time related to the time of DGSEM operator.

(b) Application time related to the time of DGSEM operator.

Figure 4.14: Comparison of the proposed preconditioners in terms of application and building time plotted over polynomial degree N using the traveling vortex test case described in Appendix C in 3D, $Re = 100$.

is highly recommended to use the preconditioners ILU(0) NoFillIn or SGS NoFillIn. They represent beside the tensor-product preconditioner the preconditioners with the lowest cost in application. It is clear, that the LU preconditioner represents the most expensive preconditioner. However, it includes the exact inverse of the diagonal blocks and hence it is expected to cause the fewest GMRES iterations, so that we employ it for comparison reasons.

Considering 2D, we can see a rather different behavior in Figure 4.15. In this case the building time for the SGS preconditioner is the lowest illustrated in 4.15a. ILU(0) and the LU depict the most expensive preconditioners to build. For the application time in 4.15b the lines for the SGS and ILU(0) preconditioners in each version are again identical. For smaller polynomial degrees N, SGS and ILU(0) including the complete density pattern are more expensive than applying the exact inverse which represents the lowest application cost. In contrast, for higher N the tensor-product and the SGS/ILU(0) NoFillIn are the fastest preconditioners in application. It is significant that SGS NoFillIn outperforms the tensor-product preconditioner in building and application in the two-dimensional case.



(a) Building time related to the time of DGSEM operator.

(b) Application time related to the time of DGSEM operator.

Figure 4.15: Comparison of the proposed preconditioners in terms of application and building time plotted over the polynomial degree N using the traveling vortex test case described in Appendix C in 2D. $Re = 100$.

Comparing now both dimensions, the intersection of the LU and the SGS/ILU(0) NoFillIn application curves is shifted to lower N changing from 2D to 3D since the diagonal blocks increase in size with $\mathcal{O}(N^{2d})$ which rises the cost of the LU decomposition but increases the sparsity for SGS/ILU(0). The building cost increases significantly for the LU and the SGS/ILU(0) from 2D to 3D, while the tensor-product variant remains almost at the same cost. However, it is remarkable that the application cost does not rise considerably for the SGS/ILU(0) NoFillIn preconditioner. It is worth to mention, that for both dimensions the application of SGS/ILU(0) NoFillIn are competitive with the cheap tensor-product preconditioner.

In this section we investigated the memory requirement and the performance of four different preconditioners in matters of assembling and application cost. These examinations are necessary in order to evaluate the savings when freezing the preconditioner for several time steps and the efficiency of each preconditioner per linear solve. The next step is to analyze how these preconditioners accelerate the GMRES method by reduction of the number of linear iterations.

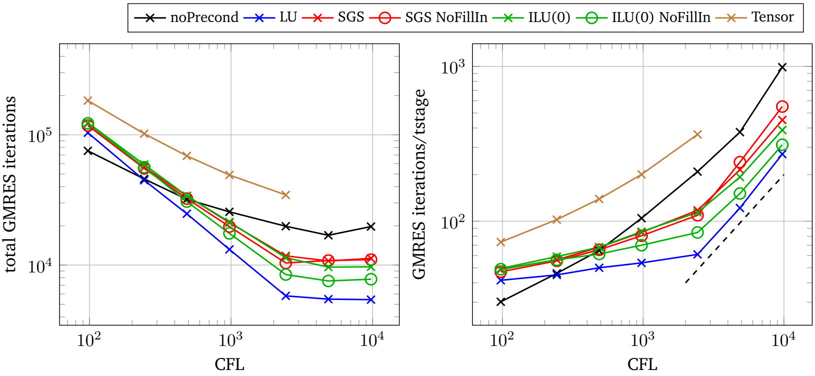
4.3 Numerical Tests

This section is dedicated to the investigation of the influence of the preconditioning strategy on several aspects. We employ for the further numerical tests the implicit time integration method of order four, ESDIRK4-6 and compare the CPU time generally against the time required by the explicit ERK4 scheme. The underlying equation system is determined by the mixed hyperbolic/parabolic Navier-Stokes equations in either two or three spatial dimensions. First, we want to evaluate in Section 4.3.1, how much the iteration number of the linear solver is reduced by the proposed preconditioners in comparison to no preconditioning. Additionally to the number of iterations, we also need to look at the required CPU time, since every preconditioner's building and application time is different. Further in Section 4.3.2, we examine the influence of varying polynomial degrees on the efficiency of the preconditioners. Since the evaluation of the preconditioning matrix can be expensive we focus in Section 4.3.3 on the impact of preconditioner freezing, keeping it constant for several time steps. By means of the preconditioning strategy it is now possible to compare time integration schemes with different orders with respect to their CPU time and $\mathcal{L}^2$ error presented in Section 4.3.4. In the final Section 4.3.5, we test the implicit solver with the presented preconditioning strategies on very high numbers of processors in order to investigate the parallel efficiency on supercomputers.

4.3.1 Quality of the Preconditioner

First, we look at different preconditioners for a fixed polynomial degree N for the two-dimensional Navier-Stokes equations with $Re = 100$. Hence, we lose the benefit of having an exact solution for the traveling vortex given in Appendix C, since it is only valid for non-viscid flows. We choose the implicit time step sizes $T/5$, $T/10$, $T/20$, $T/50$, $T/100$, $T/200$, $T/500$, the polynomial degree $N = 5$, 40×40 equidistant mesh, the local Lax-Friedrich flux as Riemann solver, adaptive Newton tolerances and $\eta_k = 0.1$ for the tolerance of the linear solver. Due to convergence difficulties of the restarted version GMRES(m) for higher CFL numbers we choose for the Krylov subspace $m = 100$. The computations are performed on 48 cores until $T = 1$. In Figure 4.16, GMRES iterations are plotted over the CFL numbers which are related to the maximum stable time step of ERK4 defined by CFL $= 1$.

All preconditioners are rebuilt every time step. Generally, the number of total GMRES iterations decreases for increasing CFL number as illustrated in Figure 4.16a until a specific CFL number when the implicit scheme can not gain any efficiency anymore due to the increasing stiffness of the linear system. In contrast, the averaged number of GMRES iterations within one time stage of ESDIRK4-6 increases due to the higher time step sizes represented in Figure 4.16b. Here we can see that LU is obviously the best approximation to the block diagonal followed by the SGS and ILU(0) preconditioner and the worst approximation is given by the tensor-product preconditioner. It is to mention, that the implicit solver including the tensor-product preconditioner does not converge for the last two time step sizes for the given setup. This is attributed to the poor approximation resulting in increasing GMRES iterations and hence in diverging of the restarted GMRES.



(a) Total GMRES iterations until $T = 1$. (b) Averaged GMRES iterations per time stage.

Figure 4.16: Comparison of the proposed preconditioners in terms of GMRES iterations plotted over CFL numbers using the traveling vortex test case described in Appendix C in 2D. $N = 5$, $Re = 100$. Dashed line represents a line with slope of one.

For lower CFL numbers, the implicit scheme without any preconditioning yields the smallest iteration numbers. This is attributed to the effect that the number of GMRES iterations in each Newton step is still too low in order to decrease it further by a non negligent approximation given by the block-Jacobi preconditioner omitting the dependency of the neighboring elements. We want to point out that the SGS and ILU(0) NoFillIn approximations which consider only a simple Euler density pattern results in lower iteration numbers than the full variant of SGS and ILU(0) except for higher CFL numbers in the SGS case. Furthermore, we can evaluate that if the slope of the curves in the log-log chart 4.16b is lower than one, an efficiency gain of the implicit scheme is expected by increasing the CFL number. Because if we double the time step size, the number of GMRES iterations should be more than halved in order to justify a bigger time step.

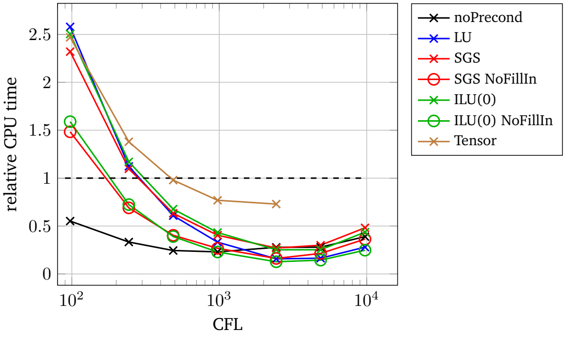


Figure 4.17: Comparison of the proposed preconditioners in terms of the implicit CPU time divided by the explicit CPU time until $T = 1$ plotted over CFL numbers using the traveling vortex test case described in Appendix C in 2D. $N = 5$, $Re = 100$.

Within the CFL range of $2 \cdot 10^3$ until 10^4, the slopes for LU, ILU(0) and SGS turning to be higher than one resulting in a stagnation or even growth of the total GMRES iterations. Hence, we can see also in Figure 4.17 no improvements in the CPU time for these specific CFL numbers. The CPU time is here given in relation to the ERK4 computing time. Within the CFL range of 10^2 until 10^4, no preconditioning is again the most efficient strategy, due to the absence of building and application work. When the GMRES iterations are significantly decreased, the preconditioned schemes turn out to be faster. Although the LU preconditioner requires less GMRES iterations than ILU(0) or SGS, the CPU time is considerable higher for smaller CFL numbers due to the expensive building costs given in Figure 4.15a. The application though is determined by the same cost for $N = 5$ depicted in Figure 4.15b. For higher CFL numbers, LU describes the fastest scheme since the times steps are greater so that the preconditioner is rarely rebuilt and the low number of GMRES iterations can counterbalance the costly application. Since the full SGS and ILU(0) preconditioners require in general more GMRES iterations and their building is obviously more expensive as their variants considering only the Euler density pattern, the computational time is much higher. Thus, we consider in the following work only the SGS/ILU(0) NoFillIn variant.

Figure 4.17 highlights for the very fine spatial resolution the acceleration of the implicit time method compared to the explicit scheme when using higher time step sizes.

4.3.2 Influence of Spatial Discretization

In the previous section, we fixed the polynomial degree in order to investigate the influence of the presented preconditioners. Now, we want to analyze how the efficiency of the preconditioners changes when we use different polynomial orders N. For this, we compute the three-dimensional Navier-Stokes equations and combine different settings of N and numbers of mesh cells such that the initial $\mathcal{L}^2$ error for the density is of the same magnitude. The computational details are reported in Table 4.1. We consider for all settings the time step sizes $T/500$, $T/200$, $T/100$, $T/50$, $T/20$, $T/10$, $T/5$. The tests are deducted for the traveling vortex test case in 3D described in Appendix C for $Re = 100$ using the local Lax-Friedrich flux until the end time $T = 1$. We employ the adaptive Newton tolerances, $\eta_k = 0.1$ as GMRES tolerance and $m = 100$ as the size of the Krylov subspaces. In the following we compare the implicit solver including LU and ILU(0) NoFillIn preconditioner with no preconditioning.

Table 4.1 highlights the gain in using high order schemes, since the number of DOFs decreases drastically from lower to higher polynomial degrees holding the same initial $\mathcal{L}^2$ error. Since the preconditioned system is solved element-locally the amount of GMRES iterations does not vary for different numbers of processors. Comparing each preconditioner in Figure 4.18, the total GMRES iterations decrease for increasing polynomial order N for every time step size. But still, the LU preconditioner represents the best

N	grid cells	initial $\|err(\rho)\|_{\mathcal{L}^2}$	DOF	#procs	explicit Δt
2	$80 \times 80 \times 80$	$1.38 \cdot 10^{-6}$	$1.38 \cdot 10^{7}$	2760	$3.93 \cdot 10^{-5}$
3	$30 \times 30 \times 30$	$1.36 \cdot 10^{-6}$	$1.73 \cdot 10^{6}$	576	$1.04 \cdot 10^{-4}$
4	$17 \times 17 \times 17$	$1.09 \cdot 10^{-6}$	$6.14 \cdot 10^{5}$	576	$1.45 \cdot 10^{-4}$
6	$7 \times 7 \times 7$	$1.85 \cdot 10^{-6}$	$1.18 \cdot 10^{5}$	72	$2.51 \cdot 10^{-4}$
8	$5 \times 5 \times 5$	$1.60 \cdot 10^{-6}$	$9.11 \cdot 10^{4}$	72	$1.97 \cdot 10^{-4}$

Table 4.1: Different spatial settings with approximately the same initial $\mathcal{L}^2$ error of density ρ.

approximation to the diagonal blocks resulting in the minimum of GMRES iterations for higher time steps followed by the ILU(0) NoFillIn preconditioner. The time step when the LU and the ILU(0) preconditioner become better than no preconditioning drops down to smaller time steps for lower polynomial degrees. Hence, we can expect a faster efficiency gain for smaller time steps by the proposed preconditioners for low order.

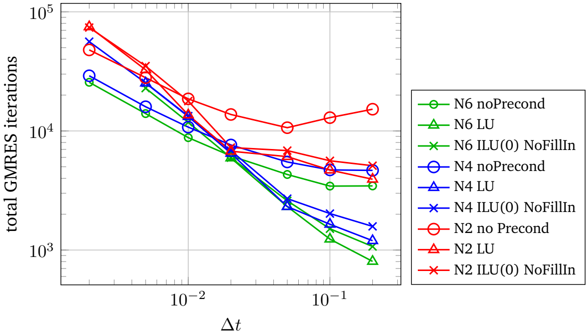
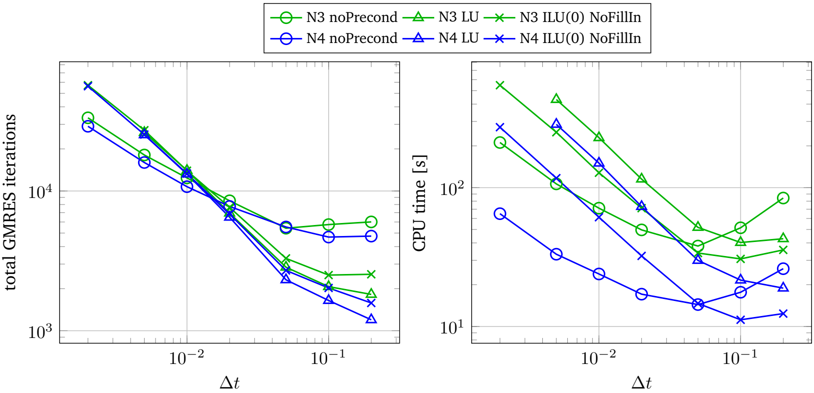


Figure 4.18: Comparison of total number of GMRES iterations plotted over the time step size Δt for different spatial settings given in Table 4.1.

In order to compare the CPU times for both cases $N = 3$ and $N = 4$ we conduct each computation on 576 cores. In Figure 4.19a we can observe that the GMRES iterations are again lower for higher N. And due to the lower number of DOFs for $N = 4$ we obtain a faster scheme for higher order depicted in Figure 4.19b. However, considering the relative CPU time in Figure 4.19c with respect to the explicit computing time, the lower order implicit scheme is faster than the corresponding explicit scheme for already smaller time steps than the implicit scheme for $N = 4$. Furthermore, the acceleration with respect to the explicit scheme is for $N = 3$ much higher than for $N = 4$ in the reported range of time steps. This can be explained by the drastic decrease of the explicit CPU time from $N = 3$ to $N = 4$ with a higher slope than for the implicit schemes.

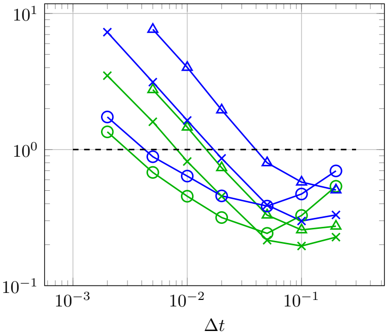
If we look at higher polynomial degrees as $N = 6$ and $N = 8$ also reported in Table 4.1, the GMRES iterations are lower for the smaller polynomial degree as illustrated in Figure 4.20a. Since the linear systems are now much more complex to solve for higher N as reported in Figure 4.14a, 4.14b and the number of DOFs are similar, the lower order scheme shows smaller total CPU times depicted in Figure 4.20b. An analog picture yields for the relative CPU time given in Figure 4.20c. For higher polynomial orders it is more difficult to outperform the explicit scheme especially for the LU decomposition, so that using ILU(0) NoFillIn or no preconditioner is the better choice for a wide range of time step sizes for this test case.

We can summarize, that it is not trivial to give general suggestions for all kind of settings since there is no unique recipe. However, we hit this difficulty in general when dealing with implicit time strategies. We can say, that we are able to outperform the explicit scheme faster and with higher ratios the smaller the polynomial degree. For higher N the LU preconditioner becomes too expensive so that only ILU(0) or SGS and no preconditioning can gain efficiency with respect to the explicit scheme for the traveling vortex test case.



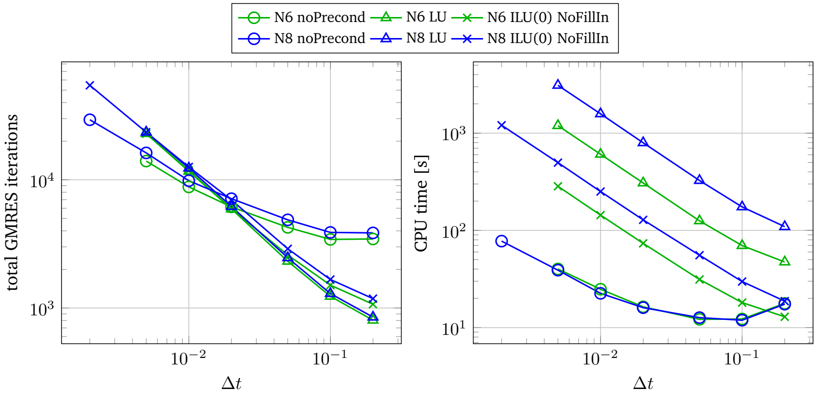
(a) Total GMRES iterations

(b) Total CPU time



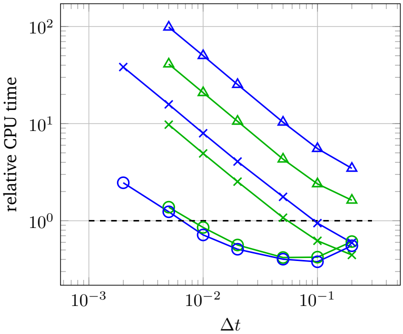
(c) Relative CPU times of the implicit and explicit scheme

Figure 4.19: Comparison of $N = 3$ and $N = 4$ as reported in Table 4.1 in terms of total GMRES iterations, total CPU time and relative CPU time plotted over the time step Δt. The computations are performed on 576 cores.



(a) Total GMRES iterations

(b) Total CPU time



(c) Relative CPU times of the implicit and explicit scheme

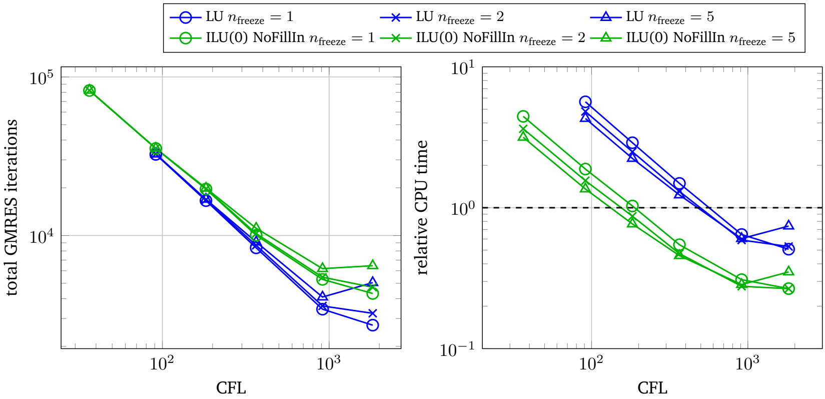
Figure 4.20: Comparison of $N = 6$ and $N = 8$ as reported in Table 4.1 in terms of total GMRES iterations, total CPU time and relative CPU time plotted over the time step Δt. The computations are performed on 72 cores.

4.3.3 Preconditioner Freezing

As investigated in the last section, for small time steps and high polynomial degrees it can be prohibitive to build the preconditioner every time step. Since in this case the Jacobian does not change quite much it is plausible to fix the preconditioner constant over several time steps, which is termed by preconditioner freezing. Actually, we already introduced the freezing strategy since we rebuild the block-Jacobi only every time step and not every Newton step. A good overview of freezing strategies is given in [110]. Also a parameter-free freezing was suggested by recalculating the preconditioner if the total computing time spent in GMRES iterations is bigger than the time to assemble the preconditioner which is also employed in [20].

For the two-dimensional case we examined no significant reduction in time, as the computational cost for evaluating the preconditioner is not expensive enough in order to compensate the increase of total GMRES iterations. Thus, for this test case the preconditioner freezing is not attractive. For three-dimensional problems freezing is of higher interest since the assembling process of the preconditioner is much more expensive comparing Figure 4.14a and 4.15a. In the following, we examine the freezing strategy for the three-dimensional Navier-Stokes equations using the ILU(0) NoFillIn preconditioner with the density pattern related to the Euler equations and the LU preconditioner since their building times are quite expensive as shown in Figure 4.14a. Further, we set $N = 5$, $m = 100$, $\eta_k = 0.1$ for the GMRES tolerances, we use the $20 \times 20 \times 20$ equidistant mesh and an adaptive Newton tolerance. The computations are conducted for the traveling vortex test case described in Appendix C with $Re = 100$ until $T = 1$ on 1440 cores.

The employed CFL numbers in Figure 4.21 correspond to the times step sizes $T/500$, $T/200$, $T/100$, $T/50$, $T/20$, $T/10$, whereas CFL $= 1$ is defined by the maximum stable time step for ERK4. For smaller CFL numbers there is no observable difference in GMRES iterations for both preconditioners illustrated in Figure 4.21a such that the CPU time in Figure 4.21b is reduced by the omitted building time of the preconditioners. Since the application of the ILU(0) NoFillIn preconditioner is one of the cheapest presented in Figure 4.15b, the reduction in CPU time for low CFL numbers is greater than for the LU preconditioner. For higher CFL numbers the freezing strategy requires more GMRES iterations such that the savings in assembling the preconditioners can hardly counterbalance the overhead of additional



(a) Total GMRES iterations.

(b) Relative CPU time compared to the time of DGSEM operator.

Figure 4.21: Comparison of different freezing numbers n_{freeze} for $N = 5$ in terms of total GMRES iterations and relative CPU time plotted over CFL numbers for the three-dimensional Navier-Stokes equations.

iterations. Covering a wide range of CFL numbers we would suggest to employ an adaptive freezing.

4.3.4 Influence of the Time Integration Order

As we have introduced some preconditioner strategies, we can now investigate which time integration order is required to reach a certain $\mathcal{L}^2$ error in the shortest CPU time. Therefor, we compute again the two-dimensional isentropic inviscid traveling vortex with the analytical solution given in detail in Appendix C as already considered in Section 3.4. We use the LU preconditioner, since we found out that the non-viscid, two-dimensional case is not complex enough in order to reduce the iterations with even weaker approximations such as the SGS or the ILU(0) preconditioners. Furthermore, we rebuild the LU preconditioner every time step to guarantee the minimum of iterations for all CFL numbers. We set $N = 5$, not too high in order to avoid the overbalance of the LU building time, divide the domain by 40×40 grid cells and set the end time to $T = 1$. The computation is conducted on 72 cores.

In Figure 4.22, the $\mathcal{L}^2$ errors of the density are illustrated in dependency of the CPU time for the ESDIRK2-3, ESDIRK3-4, ESDIRK4-6 schemes tabulated in Appendix A. The symbols on each curve are related to the CFL numbers 2, 4, 8, 16 and 32 and have to be counted from the lowest $\mathcal{L}^2$ error. This figure highlights the advantage of using high order also in time, since the more accurate solutions are reached with ESDIRK4-6 much faster than will the low order schemes. For instance an error of approximately 10^{-5} is gained by ESDIRK4-6 10 times faster than by ESDIRK3-4 and 30 times faster than by ESDIRK2-3. For higher errors the curves obviously intersect so that for very high CFL numbers it is certainly more efficient using low order time integration schemes. Since we are still interested in a fine spatial resolution, we employ in further application cases ESDIRK4-6.

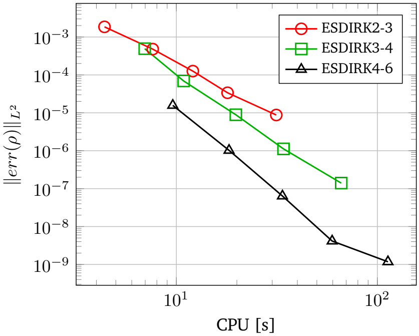


Figure 4.22: $\mathcal{L}^2$ error of the density plotted over the CPU time required until $T = 1$. Isentropic inviscid traveling vortex on a 40×40 grid at $N = 5$, with adaptive tolerance and LU preconditioner with $n_{\text{freeze}} = 1$.

4.3.5 Strong Scaling of the Implicit Solver

In the following, we want to investigate the parallel efficiency of the implicit solver with and without preconditioning and compare it against the performance of the explicit solver. Both time integration schemes are implemented within the DGSEM framework FLEXI, whereas FLEXI in combination with the explicit time marching scheme has demonstrated its ability to scale perfectly on several 10,000 cores in [6, 63]. As preconditioners, we employ the LU and the ILU(0) NoFillIn decompositions.

Computations of large scale problems with a solver of high memory capacity requires the usage of high performance computing (HPC) systems. The computational work is distributed over several processors in order to reduce the wall-clock time. Holding the total number of DOFs constant, a perfect strong scalable solver halves the wall-clock time when doubling the number of processors. For the FLEXI code, the grid elements are distributed to different processors such that the upper limit of processor numbers is given by the amount of elements when each processor owns one element. The lower limit is related to the memory size provided by each processor, since for small numbers of processors the number of elements increases and consequently also the storage per processor. Hardware configurations of the used supercomputer and the environments settings are detailed in Appendix D.

Since implicit schemes depend on the employed test case due to the different requirements for the non-linear and linear solvers, we show the parallel efficiency investigations exemplary for the three-dimensional traveling vortex described in Appendix C for the polynomial degree of $N = 4$ and the temporal ESDIRK4-6 method. In order to examine the scaling of the implicit solver we consider 10 cases with different number of elements. All mesh configurations are based on the domain $\Omega = [0, 2]^3$. The coarsest mesh consists of 6 elements in each direction and is then refined by doubling the number of elements in only one direction. Table 4.2 illustrates the different problem sizes and also the maximum number of nodes (each node consists of 24 processors), on which the elements are distributed so that each core owns only 9 elements.

The baseline configuration describes the setup against which the parallel efficiency is measured. In Table 4.3 the required nodes for the baseline computation for each case are listed. For the explicit time marching scheme the baseline for each case consists of one node (24 cores). But for the

case	1	2	3	4	5	6	7	8	9	10
#total elements: $6^3\cdot$	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
#elements in x	6	12	12	12	24	24	24	48	48	48
#elements in y	6	6	12	12	12	24	24	24	48	48
#elements in z	6	6	6	12	12	12	24	24	24	48
max #nodes	1	2	4	8	16	32	64	128	256	512

Table 4.2: Mesh configurations for the investigation of parallel efficiency. The maximum number of nodes is related to 9 elements on each core.

implicit scheme the baseline in case 10 is computed on two nodes using no preconditioning since it does not fit on one node. And especially the memory-consuming LU preconditioner restricts the baseline on two nodes in case 9 and four nodes in case 10. In contrast, the ILU(0) NoFillIn preconditioning has no issues in any case due to the low storage format and the low number of GMRES iterations requiring small Krylov subspace dimensions. The number of nodes of each baseline is doubled until the maximum number is reached.

case	1	2	3	4	5	6	7	8	9	10
Explicit	1	1	1	1	1	1	1	1	1	1
Implicit noPrecond	1	1	1	1	1	1	1	1	1	2
Implicit LU	1	1	1	1	1	1	1	1	2	4
Implicit ILU(0)	1	1	1	1	1	1	1	1	1	1

Table 4.3: Amounts of nodes for the baseline computations for the investigation of parallel efficiency.

The explicit scheme is run for 100 time steps and the implicit scheme for one time step with CFL $= 10^3$ for which the performance index (PID) is measured neglecting the time for reading or writing files and initialization. The performance index is defined by the time needed for the update of one DOF on one core

$$\mathrm{PID} = \frac{\text{wall-clock time} \cdot \#\text{cores}}{\#\text{DOFs} \cdot \#\text{time steps} \cdot \#\text{RK-stages}}.$$

We repeat the calculations for five times and calculate the median $\overline{\mathrm{PID}}$ of the performance indices in order to reduce statistical effects. The parallel efficiency is then computed by

$$\frac{\overline{\mathrm{PID}}_B}{\overline{\mathrm{PID}}_k} \cdot 100\%,$$

where $\overline{\mathrm{PID}}_B$ corresponds to the baseline median performance index and $\overline{\mathrm{PID}}_k$ to the respective on k nodes. The baseline median PID is computed on a single node expect for some cases as illustrated in Table 4.3.

In Figure 4.23, the parallel efficiency for the explicit and the implicit scheme with either no preconditioner or the LU/ILU(0) NoFillIn preconditioner is illustrated over the number of processors for all presented cases in Table 4.2. For each problem size the efficiency starts at 100% with the baseline run on the lowest amount of processors. For the explicit scheme we can see in the top left plot in Figure 4.23 superlinear scaling for almost all cases. This scaling higher than 100% can be a reason of caching effects of the processors or badly run baseline simulations. The last symbols show very high error bars since the number of DOFs per core is decreased until a minimum of 9 elements on each processor. In this case the internal work compared to the communication between the cores is quite small. Hence, for the highest number of cores in each case different jobs running on the supercomputer at the same time impact significantly the parallel performance and induce the high error bars. The parallel efficiency for the implicit schemes show similar behaviors, where no preconditioning shows the highest super scaling followed by the ILU(0) NoFillIn preconditioner but with a great loss in performance at the last two symbols (nine/eighteen elements per core) for each problem size. The LU preconditioner describes small fluctuations around the 100%. The parallel efficiency represents only relative results with respect to the baseline performance. If we want to compare between the different solvers, we need to look at the total PID, which is plotted in Figure 4.24.

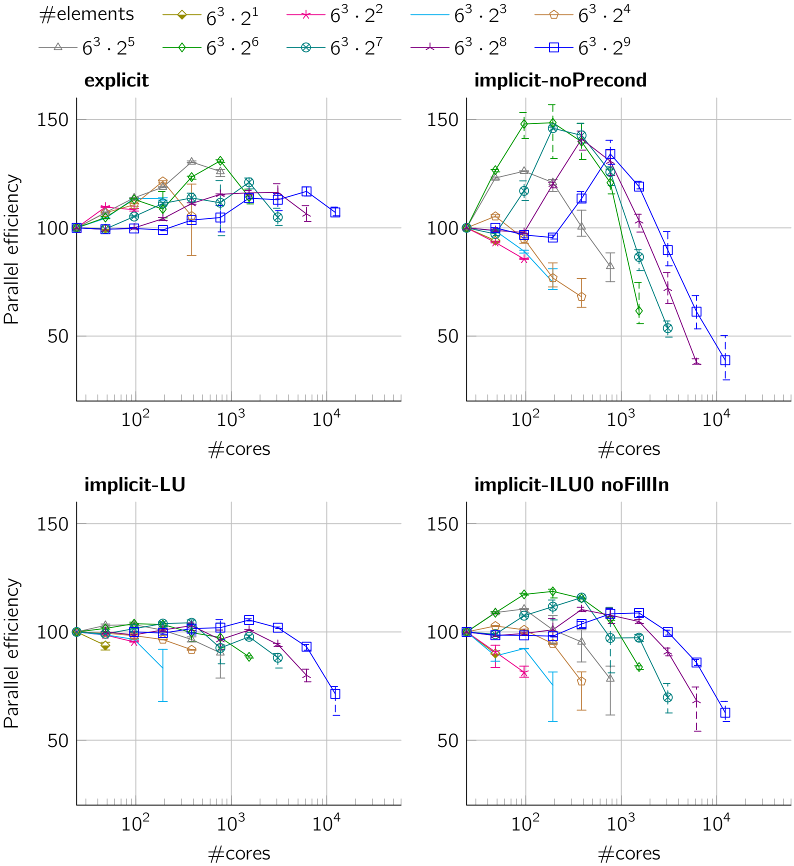


Figure 4.23: Parallel efficiency of a strong scaling of the explicit and implicit solver for different mesh sizes corresponding to each color. The baseline simulation is generally computed on a single node (24 cores) except for noPrecond and LU due to memory restrictions. The last symbol of each line is related to the case when each processor has only 9 elements. Every simulation is recomputed five times in order to obtain a statistical median. The deviations are represented by error bars.

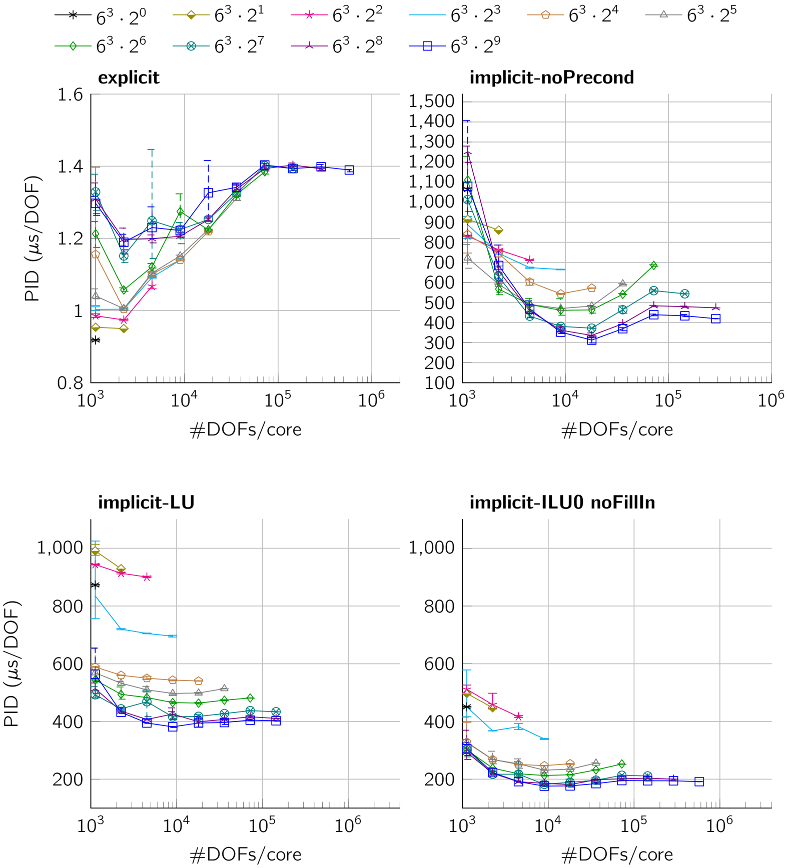


Figure 4.24: Performance index of a strong scaling of the explicit and implicit solver for different preconditioners and different mesh sizes corresponding to each color. Here, the first symbol of each line is related to the case when each processor has only 9 elements. Every simulation is recomputed five times in order to obtain a statistical median. The deviations are represented by error bars.

Here, the x-axis shows the number of DOFs per core, so that the highest number of cores is illustrated by the first symbols and the baseline point corresponds to the very right symbol in each case. Hence, the highest variability of the PID is visible for the smallest DOFs per core. In the explicit case, we can extract from the top left plot of Figure 4.24 that the PID is independent of the problem size. While each case possess a different time step size due to the CFL condition, the computational effort per DOF remains the same. For small DOFs per core, the PID can decrease which is attributed to caching effects where most of the data fits into the cache of the processor. But, due to jobs running on the supercomputer at the same time the variability of the results increases. This dependency on the shared network resources is not desirable such that 10,000 DOFs per core describes a useful guiding value achieving good performance. In contrast, the PID of the implicit scheme in Figure 4.24, which use 1,000 times greater time steps as the explicit scheme, varies for each problem size. This is due to the physical dependency of the number of GMRES iterations, whereas the PID of explicit schemes is related to the call of one DG operator which has the same cost independent of the time step size. Hence, it is not possible to compare the PID for different resolutions for each preconditioner, but the PID of each resolution case for the different implicit solvers. We choose a constant CFL number for all problem sizes in order to compare the PID among the different preconditioners. However, in each case we can see that the coarsest mesh obviously corresponds to the highest PID due to the highest time step. Since the PID is referred to the update for one time stage and the explicit PID stays around one, the implicit scheme is faster if the PID is approximately lower than 1,000. Thus, we can highlight that implicit schemes are more efficient for very fine meshes. The differences of the PID values arising for the illustrated preconditioners are not due to parallel communications since preconditioning is employed element-local, but rather due to the variations of GMRES iterations. GMRES operates on global vectors considering all grid elements, so that several communications are necessary per GMRES iteration. Remark, that the number of GMRES iterations remains constant for different numbers of cores again explained by the element-local property. It is remarkable that for every preconditioner and spatial resolution the PID is minimized around the guideline of 10,000 DOFs per core. The worst preconditioner with the highest values is given by the LU decomposition for every problem size. The ILU(0) NoFillIn preconditioner describes the most efficient acceleration in any case. Using no preconditioner results in a very high PID for very low numbers of DOFs per core, since the work on one processor is smaller than using a preconditioner and hence the communication predominates the computational effort. For the guideline of 10,000 DOFs per core no preconditioning shows for case 8, 9 and 10 a smaller PID as the LU preconditioning. But still, the implicit solver including ILU(0) NoFillIn represents the fastest scheme. Comparing the explicit and the implicit time stepping methods, the explicit scheme needs for the given setting 1,000 time steps until the implicit method makes only one. Thus, for 10,000 DOFs per core implicit time integration with ILU(0) NoFillIn outperforms the explicit scheme for any spatial resolution case and by a factor of more than 6 considering the finest grid.

4.4 Discussion

We presented in this section the analytical evaluation of the block diagonals of the block-Jacobi matrix of the DGSEM operator. Due to the tensor-product structure, we obtain especially for the 3D Euler equations a very sparse matrix structure which sparsity increases significantly for higher polynomial degrees. This is also one of the reasons why we introduce additionally a preconditioner for the Navier-Stokes equations which is restricted to the matrix pattern of the Euler equations but still considering the viscous impact for this pattern, terming it NoFillIn. However, the absolute building time of the block-Jacobi matrix shows a scaling of $(N+1)^{2d-1}$ for $d = 2, 3$, but still represents a multiple of one or two orders of magnitude of the DGSEM operator call.

Furthermore, we introduced several element-local preconditioners in order to solve the preconditioned system exactly or approximately. We compared each one in matters of GMRES iterations but also application and building/inverting time. The exact inverse obtained by the LU decomposition describes the highest reduction in GMRES iteration by definition but the most expensive preconditioner in building and application time, namely $(N+1)^{3d}$ and $(N+1)^{2d}$ respectively. The SGS preconditioner exploits the left and upper triangular matrix of block-Jacobian so that we conserve the density pattern. Thus, the building time consists only of the storage process in the CSR format and the application is executed as an efficient forward and backward sweep accounting the sparsity. The incomplete LU decomposition represents a mixture between LU and SGS since it computes an approximate LU decomposition by keeping the same density pattern as SGS. Hence the building time is still as expensive as the LU decomposition itself, but since we employ the CSR format we achieve very efficient applications identical to the SGS preconditioner. The tensor-product preconditioner which reduces the linear equation system into d one dimensional equation systems shows a promising scaling of $\mathcal{O}(N^{d+1})$ for the preconditioner application. However, the ILU(0) NoFillIn preconditioner states in 2D and 3D surprisingly similar or even lower application times as the tensor-product preconditioner. Also related to the storage requirements constitutes ILU(0) NoFillIn the minimum of the presented preconditioners. In matters of GMRES iterations, ILU(0) NoFillIn requires less iterations than SGS NoFillIn and the tensor-product preconditioner, but due to the approximative design more than the LU decomposition. We want to highlight, that the approximation quality of the tensor-product can be improved with a lot of effort, but it still holds similar application costs as the ILU(0) NoFillIn preconditioner. Summarizing all discussed aspects, ILU(0) noFIllIn describes the most efficient preconditioner in CPU time among the presented strategies.

Considering different polynomial degrees, the implicit solver gains in efficiency by using higher orders than greater numbers of grid elements similar to the explicit scheme. However, for very high polynomial orders from around $N = 6$ preconditioning becomes too expensive particularly the exact inverting approach such that in this case ILU(0) NoFillIn or even no preconditioning may be the choice.

Freezing approaches make sense for expensive preconditioners related to the building aspect as LU and ILU(0). As well, only for small time steps we can see an efficiency gain since in this case the increase of GMRES iterations due to the decreasing approximation quality can be counterbalanced by the neglecting building time.

Further, we could examine that ESDIRK schemes with high temporal orders are more efficient in CPU time compared for the same $\mathcal{L}^2$ error for the considered test case.

The strong scaling tests for several mesh size configurations showed us perfect parallel efficiency for fine meshes with limited number of cores such that each core owns minimum 10,000 DOFs. Contrarily to the explicit scheme, the PID for implicit schemes is not independent of the problem size through the increasing stiffness of the linear system for coarser meshes. Hence, it is more probable to outperform the explicit scheme for fine meshes. Furthermore, we want to highlight that for all mesh configurations the number of 10,000 DOFs per core holds the minimum PID. Also for the explicit scheme this number is a good guideline value in order to avoid high derivations when running the job several times due to higher dependencies on the shared network resources. And again the fastest implicit scheme with the minimum PID is given by means of the ILU(0) NoFillIn preconditioner.

5 Application of Implicit DGSEM

In this chapter, several well-known test cases are presented in order to evaluate the efficiency and accuracy of the proposed implicit scheme for solving unsteady flows. In all computational applications, we first examine the effect of the implicit DGSEM with different time step sizes on the solution quality compared to the results of the explicit ERK4 and the Direct Numerical Simulation (DNS) for turbulent flows. Further, we analyze the impact of different preconditioners on the number of GMRES iterations and the CPU time related to the explicit ERK4 scheme. Special configurations as different freezing numbers of the preconditioner and the choice of the GMRES termination criterion are also investigated.

We consider two applications of the circular cylinder flow with different Reynolds numbers: one laminar computation in the two-dimensional space and one turbulent computation in the three-dimensional space. Here, we also employ ESDIRK schemes of different order, evaluating the accuracy and computational cost in comparison to the ERK4 scheme. In addition, we study the parallel efficiency by evaluating the strong scaling for all implicit solvers with different preconditioners.

The other test case depicts a turbulent flow through a three-dimensional channel, representing an interesting application due to special periodicity properties. Here, we evaluate two different polynomial degrees of the spatial discretization DGSEM in combination with ESDIRK4-6.

All following computations are performed on the supercomputer Hazel Hen, a Cray XC40 system at HLRS. Detailed configurations and settings are given in D. The meshes are assembled by the grid generator HOPR [66].

5.1 Cylinder Flow

The flow around a circular cylinder is a well-established test case for the evaluation of numerical modeling in fluid dynamics. The form of the wake behind the cylinder is determined by the Reynolds number Re_D based on the cylinder diameter D: For $Re_D < 160$ we can find a laminar von Kármán vortex shedding in the wake region, $160 < Re_D < 260$ describes the transitional regime from 2D to 3D shedding phenomena so that for $Re_D > 260$ we obtain turbulent structures. Here, we want to analyze the proposed implicit solver for the different cases $Re_D = 200$ and $Re_D = 3900$. For both Reynolds numbers we employ the same physical domain and mesh setup but with different resolutions.

Computational Setup We choose a structured grid spanning a circular physical domain with radius $r_{cyl} = 100D$ in order to impose Dirichlet freestream conditions at the outer boundary. For generating the mesh we use the software HOPR detailed in [64]. The grid cells are curved by employing a polynomial basis for the mapping $\boldsymbol{X}(\boldsymbol{\xi})$ from reference to physical space with identical order as the ansatz polynomials of the unknown flow variables. In the circumferential direction, we choose a bell-shaped stretching with a grid clustering in the wake region. In radial direction we apply a stretching with a fixed factor increasing the radial extension of the cells in radial outward direction. We set at the cylinder surface isothermal wall boundaries while on the outer boundary we choose weakly enforced Dirichlet freestream conditions with the density $\rho_\infty = 1$ and the velocity vector $u_\infty = (1,0)^T$ or respectively $u_\infty = (1,0,0)^T$ for the three-dimensional turbulent flow. The freestream pressure p_∞ is adjusted to the respective Mach number. For the three-dimensional test case at $Re_D = 3900$ we choose a spanwise extension of $z = 4D$ with a uniform grid spacing and employ periodic boundary conditions at the spanwise domain faces.

5.1.1 Laminar Flow at $Re_D = 200$

For this low Reynolds number, the vortex shedding remains still laminar so that we can analyze the implicit solver in two space dimensions. The freestream Mach number is set to $Ma_\infty = 0.2$. Several authors investigated the laminar cylinder flow for implicit schemes [117, 90, 91, 52] but with much finer spatial resolution as proposed here with no improvements of the solution quality forcing the explicit time step to very small values due to mesh stiffness.

Spatial Setup For this laminar flow the grid consists of 20×50 (circumferential, radial) elements with a grid spacing at the cylinder of a radial length $l_r = 0.24D$ of the first cell from the wall and a minimum circumferential cell length $l_c = 0.08D$ in the first layer. The circular mesh with a zoomed-in view is illustrated in Figure 5.1.

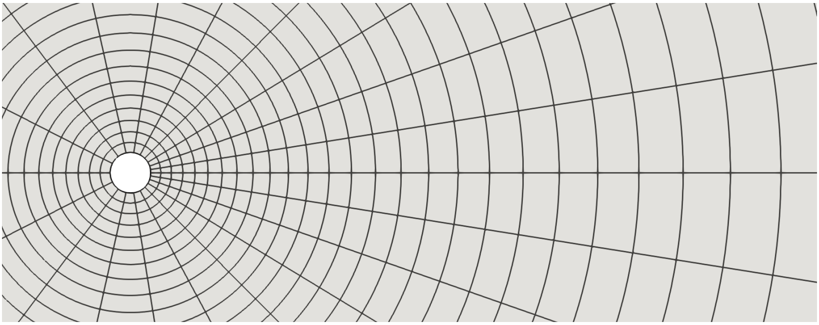


Figure 5.1: Two-dimensional curved mesh around a circular cylinder for the laminar flow at $Re_D = 200$, detailed view.

We employ the local Lax-Friedrich flux for the convective Riemann fluxes and the BR2 scheme of Bassi and Rebay [12] for the viscous flux. For the numerical DGSEM solution we choose the polynomial degree $N = 4$ as well as for the polynomial presentation of the cell boundaries, resulting in a total of $125,000$ degrees of freedom per conservative variable. Figure 5.2 shows the velocity magnitude at $100T^*$ (with the convective time unit $T^* = D/|u_\infty|$) after the vortex shedding was stabilized computed with the explicit ERK4 method detailed in Appendix B. We employ this flow field as initial condition for further investigations. The time averaging procedure is done for 100 shedding cycles (determined by the Strouhal period T_{Sr}) starting from $100T^*$.

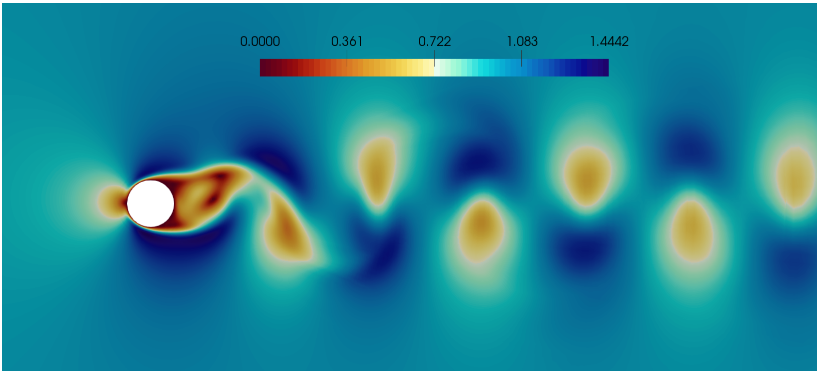


Figure 5.2: Instantaneous flow field around a circular cylinder at $Re_D = 200$ colored by the velocity magnitude.

Temporal Setup We employ for comparison the explicit ERK4 scheme and the implicit ESDIRK4-6, ESDIRK3-4, ESDIRK2-3 methods which Butcher tableaux are listed in Appendix A. In detail, we set within the implicit solver the maximum Krylov subspace dimension to 50 for the restarted GMRES, the GMRES tolerance is given by $\eta_k = 0.1$ and the relative Newton termination criterion is limited by $\epsilon_{\mathrm{N}} = 10^{-2}$ for ESDIRK4-6, ESDIRK3-4 and $\epsilon_{\mathrm{N}} = 10^{-3}$ for ESDIRK2-4. For this test case, the spatial error dominates the temporal error so that the adaptive Newton criterion forces the Newton solution to a higher resolution as actually necessary. As preconditioner, we use the LU and the ILU(0) decomposition described in 4.2.1 and 4.2.3, respectively. ILU(0) is only limited to the purely hyperbolic matrix pattern, denoted by ILU(0) NoFillIn, whose efficiency was highlighted in Section 4.3.
The explicit time step is chosen as the maximum stable one and is related to the CFL number one. For the implicit we use the time step sizes given in Table 5.1, where the maximum time step $\Delta t^* = 0.5$ corresponds to $1/10$ of the vortex shedding period.

Solution Quality First, we want to concentrate on the accuracy of the simulations with the proposed time integration schemes. In Table 5.2, we list the relevant integral quantities of the computations for either the explicit

Δt^*	$\mathbf{1.45 \cdot 10^{-3}}$	0.05	0.1	0.2	0.3	0.4	0.5
CFL	**1**	34	69	138	207	276	435

Table 5.1: Employed non-dimensionalized time step sizes $\Delta t^* = \Delta t u_\infty / D$ and corresponding CFL numbers. Bold values are related to explicit reference computation with maximum stable time step size specifying CFL $= 1$.

(denoted in bold) and the implicit schemes of different order for comparison. We consider the mean base pressure $C_{p_{Base}} = (p - p_\infty)/(0.5 \rho_\infty u_\infty^2)$, the Strouhal number Sr of the lift force, the mean drag coefficient C_D based on the streamwise force and the root mean squared lift coefficient C_L^{rms}. From Table 5.2, we can extract, that the data match the explicit results better the smaller the time step size, which is attributed to the higher numerical dissipation for higher time steps. This dissipation is significantly higher for ESDIRK3-4 and ESDIRK2-3, while the quantities for ESDIRK4-6 remain almost constant. A slight trend of increasing values can be noted for the fourth order scheme in contrast to the lower order schemes showing a continuous decrease in the proposed quantities. However, the implicit results are for all time step sizes in good agreement with the values found in literature as [102] and [86] and the literature therein for the incompressible case.
Due to the higher time step sizes up to $1/10$ of the Strouhal period time aliasing effects can occur. However, plotting the drag coefficient computed by ESDIRK4-6 over the time t^* as in Figure 5.3, we can see that the values related to all the implicit time step sizes lie exactly on the curve computed by ERK4. This is due to the fact that the highest CFL number corresponds to $1/10$ of the Strouhal period.

Solver Efficiency Here, we want to examine the influence of the LU and the ILU(0) NoFillIn preconditioners for different settings in terms of the total number of GMRES iterations, measuring the accuracy of the preconditioner, and the total CPU time. For all further investigations the end time is set to $120T^*$ which corresponds to the computation of approximately four time periods of the vortex shedding.

Δt^*	Scheme	$C_{p_{Base}}$	Sr	C_D	C_L^{rms}
$\mathbf{1.45 \cdot 10^{-3}}$	ERK4	$\mathbf{-0.9816}$	$\mathbf{0.1954}$	$\mathbf{1.3259}$	$\mathbf{0.4831}$
0.05	ESDIRK4-6	−0.9815	0.1953	1.3258	0.4830
0.1	ESDIRK4-6	−0.9815	0.1954	1.3259	0.4830
0.2	ESDIRK4-6	−0.9816	0.1955	1.3259	0.4831
0.3	ESDIRK4-6	−0.9819	0.1954	1.3260	0.4833
0.4	ESDIRK4-6	−0.9820	0.1954	1.3260	0.4836
0.5	ESDIRK4-6	−0.9822	0.1954	1.3261	0.4839
0.05	ESDIRK3-4	−0.9815	0.1955	1.3258	0.4829
0.1	ESDIRK3-4	−0.9813	0.1954	1.3257	0.4825
0.2	ESDIRK3-4	−0.9800	0.1952	1.3246	0.4804
0.3	ESDIRK3-4	−0.9738	0.1948	1.3208	0.4756
0.4	ESDIRK3-4	−0.9606	0.1938	1.3122	0.4664
0.5	ESDIRK3-4	−0.9500	0.1927	1.3050	0.4549
0.05	ESDIRK2-3	−0.9813	0.1954	1.3257	0.4830
0.1	ESDIRK2-3	−0.9806	0.1952	1.3252	0.4829
0.2	ESDIRK2-3	−0.9785	0.1944	1.3237	0.4830
0.3	ESDIRK2-3	−0.9758	0.1934	1.3217	0.4825
0.4	ESDIRK2-3	−0.9678	0.1909	1.3100	0.4783
0.5	ESDIRK2-3	−0.9670	0.1905	1.3159	0.4785

Table 5.2: Computational results for the cylinder flow at $Re_D = 200$. $\Delta t^* = \Delta t u_\infty / D$: non-dimensionalized time step size, $C_{p_{Base}}$: mean pressure coefficient at the downstream point $x = D/2, y = 0$, Sr: Strouhal number of the lift force $f_{lift} D / u_\infty$, C_D: mean drag coefficient, C_L^{rms}: RMS of the lift coefficient.

First, we compare different freezing numbers $n_{\text{freeze}} = 1, 2$ and 5 for each preconditioner as illustrated in Figure 5.4 for the ESDIRK4-6 scheme. The freezing number corresponds here to the number of time steps keeping the preconditioner constant. Generally, for the preconditioned implicit solver, the total number of GMRES iterations reduces with increasing time step

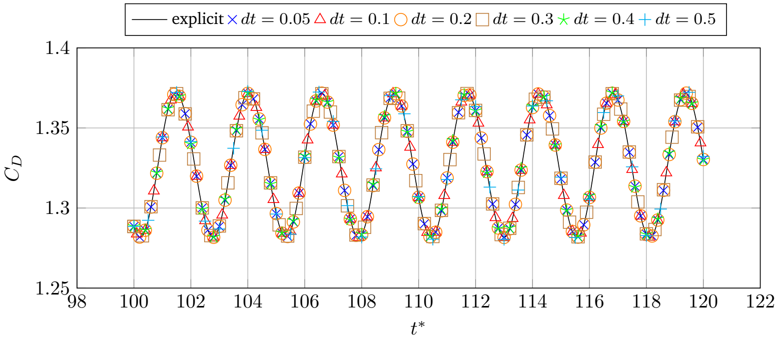


Figure 5.3: Drag coefficient C_D over non-dimensionalized time t^* for the explicit ERK4 and implicit ESDIRK4-6 schemes with given time step sizes.

size and hence also the CPU time reduces in the same way in contrast to the non-preconditioned solver. The LU preconditioner describes obviously the best approximation to the inverse of the system matrix resulting in lower GMRES iterations compared to ILU(0) NoFillIn. But, due to its significantly higher building and application costs, the implicit solver including the worse ILU(0) approximation is more efficient in matters of CPU time. Considering the freezing strategy, for higher freezing numbers the preconditioner loses its approximation quality for greater time step sizes causing an increase of GMRES iterations and hence a loss in efficiency as already investigated in Section 4.3.3. Thus, we can only gain a benefit by the freezing strategy for smaller time steps. The minimum CPU time is achieved at $\Delta t^* = 0.4$ for the ILU(0) preconditioner with $n_{\text{freeze}} = 1$ almost halving the explicit computing time.

The next Figure 5.5 compares the proposed ESDIRK schemes of different order again in terms of GMRES iterations and relative CPU time. The freezing number in this case is set to $n_{\text{freeze}} = 1$. Generally, the different preconditioners influence the efficiency for all ESDIRK schemes in the same way. Among the different temporal orders, ESDIRK2-3 scheme depicts the most efficient, followed by ESDIRK3-4 for smaller time step sizes and by ESDIRK4-6 for higher time steps. The subfigure showing the relative CPU time illustrates a zoom in, since the non-preconditioned solver turns to be

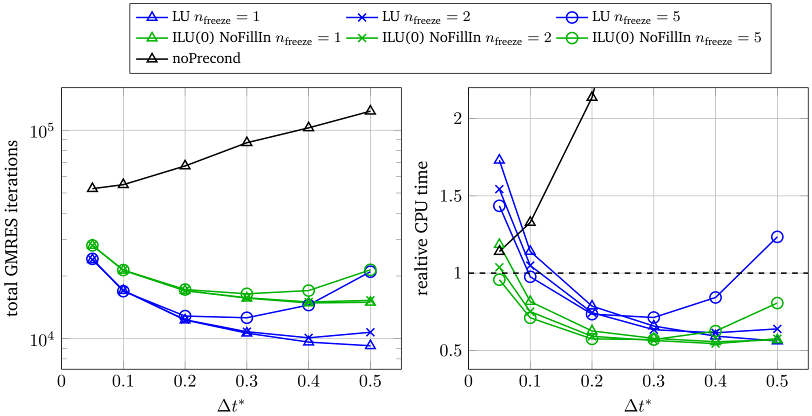


Figure 5.4: Comparison of different freezing numbers n_{freeze} for the preconditioners in terms of total GMRES iterations and relative CPU time, corresponding to the time related to ESDIRK4-6 divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size $\Delta t^* = \Delta t u_\infty / D$. The computation of the cylinder flow at $Re_D = 200$ is performed for 20 time units on 24 cores.

prohibitively expensive. However, the fastest scheme is related to ESDIRK2-3 with the LU preconditioner for the time step size $\Delta t^* = 0.5$ requiring only third of the explicit computational time.

In the previous computations we set the GMRES tolerance defined in (3.7) to $\eta_k = 0.1$. Here, we want to investigate the influence by employing the adaptive forcing terms (3.10) introduced by Eisenstat and Walker, [45]. Figure 5.6 illustrates in the case of using a preconditioner that $\eta_k = 0.1$ describes a much better choice in terms of GMRES iterations and CPU time.

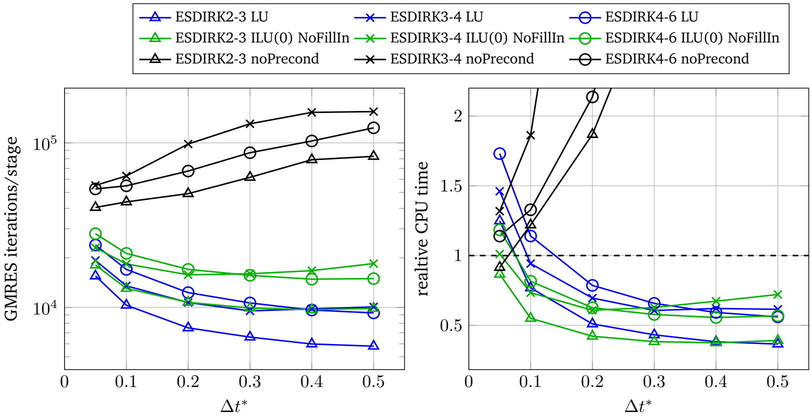


Figure 5.5: Comparison of ESDIRK schemes with different order in terms of total GMRES iterations and relative CPU time, corresponding to the time related to the specific ESDIRK scheme divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size $\Delta t^* = \Delta t u_\infty / D$. The computation of the cylinder flow at $Re_D = 200$ is performed for 20 time units on 24 cores.

For the investigation of the parallel efficiency of the different schemes we set the overall best choice $n_{\text{freeze}} = 1$ and $\eta_k = 0.1$. Looking at the fixed time step size $\Delta t^* = 0.4$ corresponding in most of the cases to the minimum CPU time, Table 5.3 shows the total number GMRES iterations and the total wall-clock time for different numbers of processors for ERK4, ESDIRK2-3 and ESDIRK4-6. Furthermore, we list the speed-up of the implicit ESDIRK scheme related to ERK4 and the strong scaling efficiency computed with respect to the baseline simulation using a single processor $\text{Time}_{\text{nProcs}=1}$. The corresponding average number of DOFs per conservative variable on one processor core for the used 1, 24 and 48 cores is 25.000, 1.041 and 520 respectively. Due to the element-local preconditioning the number of iterations remains the same for different numbers of processors. ESDIRK2-3 with either LU or ILU(0) NoFillIn preconditioning represents the most effi-

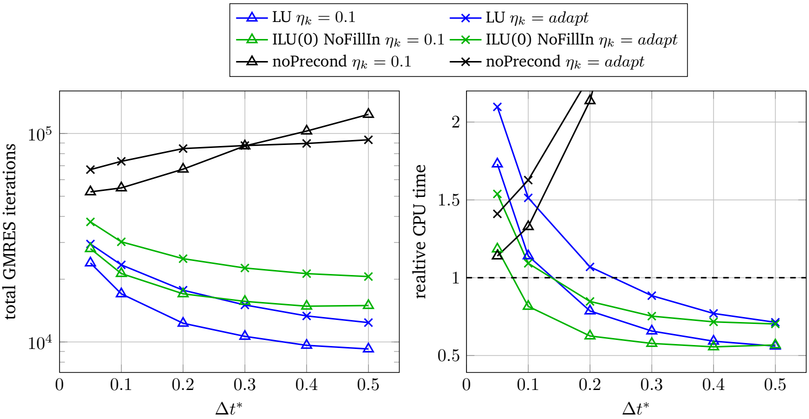


Figure 5.6: Comparison of a fixed or an adaptive GMRES tolerance η_k in terms of total GMRES iterations and relative CPU time, corresponding to the time related to ESDIRK4-6 divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size $\Delta t^* = \Delta t u_\infty / D$. The computation of the cylinder flow at $Re_D = 200$ is performed for 20 time units on 24 cores.

cient implicit solver requiring only one third of the computational time of ERK4 followed by ESDIRK4-6 which is twice faster than ERK4. Due to the different scaling factors of the given schemes, the implicit speed-up changes accordingly. The significant decrease in the scaling efficiency is attributed to the very low number of DOFs per processor also for the explicit scheme. The implicit solver including the ILU(0) or no preconditioning performs similar to the explicit scheme ERK4 in matters of strong scaling, whereas LU illustrates the worst scaling efficiency.

Summarizing, we investigated in the two-dimensional case, ESDIRK2-3 and ESDIRK4-6 with ILU(0) NoFillIn preconditioning describe the fastest implicit solvers performing almost three times faster than EKR4 and showing similar scaling efficiencies as the explicit scheme.

Method	nProcs	Iter.	Time $[s]$	Speed-up$_{\text{impl}}$	Scaling $[\%]$
ERK4	1	-	711.32	1.00	100.00
ESDIRK2-3 ILU(0) NoFillIn	1	9677	262.0	2.71	100.00
ESDIRK2-3 LU	1	5988	197.8	3.60	100.00
ESDIRK2-3 noPrecond	1	79001	2831.5	0.25	100.00
ESDIRK4-6 ILU(0) NoFillIn	1	14831	384.5	1.85	100.00
ESDIRK4-6 LU	1	9650	295.7	2.41	100.00
ESDIRK4-6 noPrecond	1	103478	3649.7	0.19	100.00
ERK4	24	-	38.31	1.00	77.36
ESDIRK2-3 ILU(0) NoFillIn	24	9677	14.49	2.64	73.96
ESDIRK2-3 LU	24	5988	14.76	2.60	55.84
ESDIRK2-3 noPrecond	24	79001	152.15	0.25	77.54
ESDIRK4-6 ILU(0) NoFillIn	24	14831	21.48	1.78	74.58
ESDIRK4-6 LU	24	9650	22.89	1.67	53.83
ESDIRK4-6 noPrecond	24	103478	194.88	0.20	78.03
ERK4	48	-	22.49	1.00	65.89
ESDIRK2-3 ILU(0) NoFillIn	48	9677	7.70	2.92	70.89
ESDIRK2-3 LU	48	5988	6.99	3.22	58.95
ESDIRK2-3 noPrecond	48	79001	103.10	0.22	57.22
ESDIRK4-6 ILU(0) NoFillIn	48	14831	10.78	2.09	74.31
ESDIRK4-6 LU	48	9650	10.38	2.17	59.35
ESDIRK4-6 noPrecond	48	103478	93.47	0.24	81.35

Table 5.3: Comparison of the computational performance of ERK4, ESDIRK2-3 and ESDIRK4-6 schemes including LU, ILU(0) NoFillIn or no preconditioner for the fixed time step $\Delta t^* = 0.4$. The computation is run for 20 time units. nProcs: number of processors, Iter: total number of GMRES iterations, Time: total wall-clock time, Speed-up$_{\text{impl}} = \frac{\text{Time}_{\text{ERK4}}}{\text{Time}_{\text{ESDIRK}}}$, Strong Scaling: $\text{Time}_{\text{nProcs}=1} / (\text{Time}_{\text{nProcs}} \cdot \text{nProcs}) \cdot 100\%$.

5.1.2 Turbulent Flow at $Re_D = 3900$

The flow around a cylinder with circular cross sections at $Re_D = 3900$ consists of a still laminar boundary layer on the upstream face up to the separation point, where it detaches forming a periodic trailing shear layer. The transition to turbulence occurs within this shear layer, which interacts with the flow in the wake region. For this turbulent flow, we focus again on a comparison of the explicit and implicit time discretization performing a spatially under-resolved Direct Numerical Simulation (uDNS). We set the free-stream Mach number to $Ma_\infty = 0.3$ for reasons of comparability to the DNS results in [16] and employ the identical computational setup, except using the half spanwise extension $z = 4D$.

Spatial Setup We discretize the three-dimensional domain for this turbulent flow again with a structured, volume-curved grid with a higher resolution of $26 \times 26 \times 18$ elements (circumferential, radial and spanwise). This results in a grid spacing in circumferential, radial and spanwise direction of $l_c^{min} = 0.035D$, $l_c^{max} = 0.228D$ in the first layer, $l_r = 0.046D$ within the first cell from the wall and $l_z = 0.222D$.

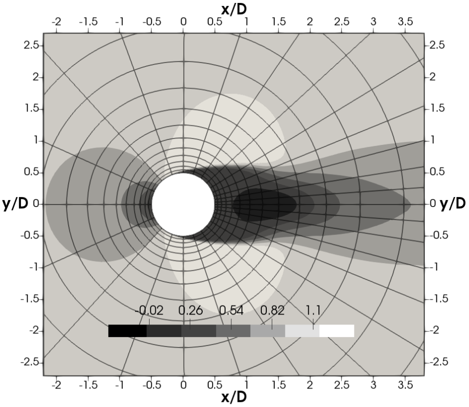


Figure 5.7: Time- and spanwise-averaged streamwise velocity at $Re_D = 3900$ and detailed grid view.

For the spatial DGSEM, the solution within each element is approximated by a polynomial tensor product ansatz of degree $N = 4$, resulting in a total number of 1.521 mio DOFs per conservative variable. Hence the factor between the DOFs of the uDNS and the DNS in [16] accounts 47. Also for the geometrical mapping we choose a polynomial representation of order four. Identical to the laminar case in Section 5.1.1, we employ the local Lax-Friedrich flux for the convective Riemann fluxes and the BR2 scheme of Bassi and Rebay [12] for the viscous flux. Figure 5.7 shows beside the employed mesh, the resulting time- and spanwise-averaged streamwise velocity distribution. In order to obtain an initial condition, we choose the explicit ERK4 scheme where the time integration errors are negligible due to the stability driven small time step. After the establishment of a stable vortex shedding at $100T^*$, with the convective time unit $T^* = D/|u_\infty|$, we start to average the results for again 100 shedding cycles. Due to the low order four of DGSEM and the resulting high numerical dissipation it is not necessary to stabilize the computations by over integration [56]. From the initial condition at $100T^*$ on, we focus again on the comparison of different implicit ESDIRK schemes and the ERK4 method. Figure 5.8 illustrates an instantaneous flow field computed with ERK4, where the vortices are visualized by the λ_2 criterion with $\lambda_2 = -5$. The iso-contours and the $x - y$ plane are colored by the velocity magnitude.

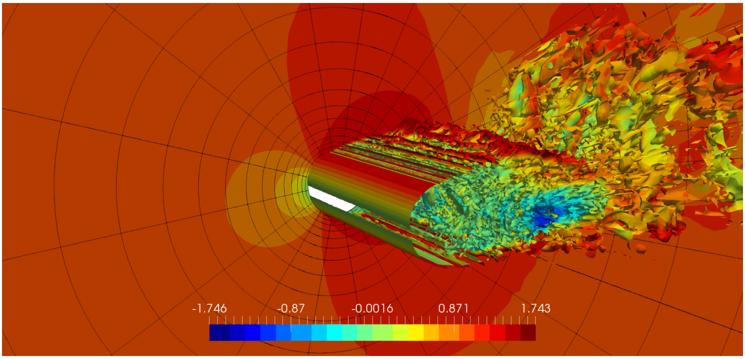


Figure 5.8: Instantaneous flow field around a circular cylinder at $Re_D = 3900$. Iso-surfaces, visualized by $\lambda_2 = -5$, and $x - y$ plane are colored by the velocity magnitude.

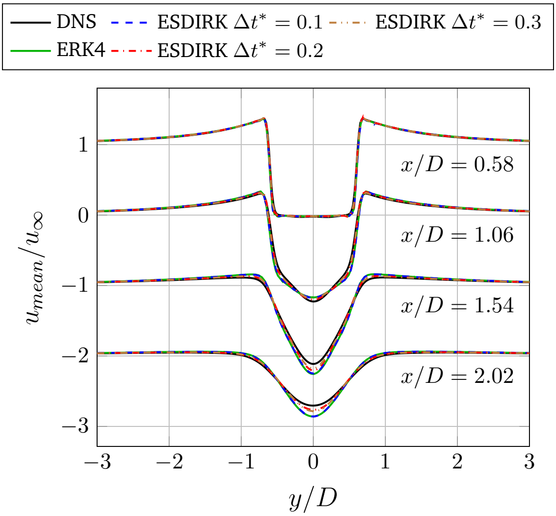
Temporal Setup Here, we compare again the ESDIRK schemes of second and fourth order, ESDIRK2-3 and ESDIRK4-6 and the explicit ERK4 scheme. The Butcher tableaux are given in Appendix A. The employed non-dimensionalized time steps $\Delta t^* = \Delta t u_\infty / D$ and the related CFL numbers are listed in Table 5.4, whereas for the ERK4 scheme we choose the maximum stable time step corresponding to CFL $= 1$. For the implicit solver, we set the maximum Krylov subspace dimension to 30 for the restarted GMRES. In contrast to the laminar case, the adaptive GMRES tolerance given in (3.10) accelerated the solver significantly more than setting $\eta_k = 0.1$, employing hence the adaptive approach. The relative Newton termination criterion is again chosen by $\epsilon_{\mathrm{N}} = 10^{-2}$ for both ESDIRK methods. Again we consider the LU, describing the exact inversion, and the ILU(0) NoFillIn preconditioners for accelerating the linear solver instead of using no preconditioner. Particularly, the preconditioners are recalculated every time step, i.e. $n_{\text{freeze}} = 1$.

Δt^*	CFL	Scheme	$C_{p_{Base}}$	Sr	C_D	C_L^{rms}	L_r/D
-	-	DNS Beck et al.	-0.979	0.204	1.082	0.246	1.160
$7.07 \cdot 10^{-4}$	1	ERK4	-0.9291	0.2065	1.033	0.183	1.347
0.1	141	ESDIRK4-6	-0.9290	0.2063	1.049	0.226	1.354
0.2	283	ESDIRK4-6	-0.9653	0.2072	1.055	0.231	1.250
0.3	424	ESDIRK4-6	-0.9734	0.2076	1.068	0.256	1.257
0.1	141	ESDIRK2-3	-0.9116	0.2066	1.020	0.171	1.420
0.2	283	ESDIRK2-3	-0.9772	0.2074	1.035	0.194	1.223
0.3	424	ESDIRK2-3	-0.8858	0.2055	0.985	0.126	1.516

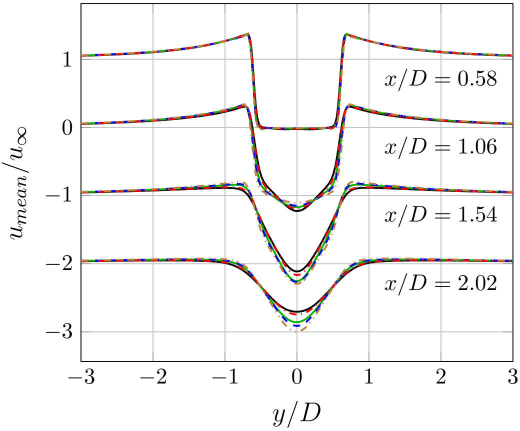
Table 5.4: Simulation results for the cylinder flow at $Re_D = 3900$. $\Delta t^* = \Delta t u_\infty / D$: non-dimensionalized time step size, $C_{p_{Base}}$: mean pressure coefficient at the downstream point $x = D/2, y = 0$, Sr: Strouhal number of the lift force $f_{lift} D / u_\infty$, C_D: mean drag coefficient, C_L^{rms}: RMS of the lift coefficient, L_r: length of separation bubble

Solution Quality Considering the accuracy of the simulations for different temporal orders and times steps, we can see in Table 5.4 the resulting integral quantities also in comparison to the DNS data given in Beck et al. [16]. In addition to the coefficients in the laminar case, we also list the normalized length of the recirculation bubble L_r/D. We can extract from Table 5.4 a better match to the explicit results for smaller time steps for both implicit schemes. For ESDIRK2-3, we can see by the base pressure $C_{p_{base}}$, the drag coefficient C_D and the separation bubble length L_r a longer, low drag recirculation zone. But for increasing time steps for each ESDIRK scheme, we achieve much better results for $C_{p_{base}}$, C_D, C_L^{rms} and L_r in comparison to the DNS data attributed to the higher numerical dissipation. Except the case $\Delta t^* = 0.3$ for ESDIRK2-3 shows non consistent behavior, which can be a reason of high loss of temporal accuracy. The overall best fit to the DNS is given by ESDIRK2-3 with the time step $\Delta t^* = 0.2$.
Figure 5.9 shows the time- and spanwise-averaged streamwise velocity profiles in the wake resulting from the implicit time discretization within DGSEM (ESDIRK4-6: Figure 5.9a, ESDIRK2-3: Figure 5.9b) in comparison to the explicit ERK4 and the DNS results. For both ESDIRK schemes we can find similar to the conclusion from Table 5.4 a perfect fit with the explicit ERK4 computation for the smaller time step $\Delta t^* = 0.1$. With increasing time step for ESDIRK4-6 the velocity profiles matches better the DNS. But in particular, the highest accuracy is achieved by ESDIRK2-3 with $\Delta t^* = 0.2$, which was already predicted by the integral sizes in Table 5.4.

Solver Efficiency Turning to the efficiency of the proposed implicit solvers, Table 5.5 shows a comparison for the different preconditioners employing the time step $\Delta t^* = 0.2$, which guarantees the best results related to the DNS data. In particular, the effects on the total number of GMRES iterations, measuring the quality of the preconditioner, the total wall-clock time, the speed up of the implicit scheme related to the ERK4 method and the strong scaling efficiency are listed for different numbers of processors. For the total number of DOFs of 1.521 mio, the proposed numbers of computing cores, 144, 288, 576 and 1152 correspond to average numbers of 10.563, 5.281, 2.641 and 1.320 DOFs per core.



(a) ESDIRK4-6



(b) ESDIRK2-3

Figure 5.9: Mean streamwise velocity at different downstream points in the wake of the cylinder flow at $Re = 3900$ for DNS [16], explicit ERK4 and implicit ESDIRK (with various time step sizes Δt^*) within DGSEM.

The scaling efficiency is computed with respect to the baseline computation of using 144 processors by

$$\frac{\text{Time}_{\text{nProcs}=144}}{(\text{Time}_{\text{nProcs}} \cdot (\text{nProcs}/144))} \cdot 100\%,$$

where $\text{Time}_{\text{nProcs}}$ denotes the wall-clock time on nProcs processors. All computations are performed until the end time of $120T^*$, which corresponds to the simulation of approximately four time periods of the vortex shedding.

The number of GMRES iterations remains for different amounts of processors constant, whereas the LU preconditioner shows for both ESDIRK schemes obviously the best approximation resulting in a minimum number of iterations. The ILU(0) NoFillIn decomposition represents also a very good approximation to the block-Jacobian as it requires only 15% more iterations than LU and it reduces them up to a factor of 6 compared to no preconditioning.

Regarding the wall-clock time, the highest reduction in iterations of the LU preconditioner does not induce the lowest computing time since the building and application time for the LU preconditioner are especially for the three-dimensional case significantly expensive as examined in the basis investigations in Section 4.2.5. The cheap application cost of the ILU(0) NoFillIn creates a very efficient implicit solver performing two times faster than the LU preconditioner and four times faster than without preconditioning. In contrast to the two-dimensional case in Section 5.1.1, where the LU and ILU(0) preconditioning showed the same computing time using the same polynomial degree, we can see here clearly the effect of higher dimensions on the cost for the LU decomposition. As well, we can recognize the increasing improvement of ILU(0) compared to LU. When we consider the speed-up of the implicit schemes with respect to the explicit ERK4 in Table 5.5, we can still see the same efficiency of the ESDIRK2-3 scheme including the ILU(0) NoFillIn preconditioner as in the two-dimensional case performing almost three times faster than the explicit scheme. ESDIRK4-6 loses efficiency compared to the two-dimensional case which can be addressed to the turbulent structures in combination with the high temporal accuracy resulting in higher iteration counts per time step.

Method	nProcs	Iter.	Time $[s]$	Speed-up$_{\text{impl}}$	Scaling $[\%]$
ERK4	144	-	1916.78	1.00	100.00
ESDIRK2-3 ILU(0) NoFillIn	144	21768	734.75	2.61	100.00
ESDIRK2-3 LU	144	19026	1671.35	1.15	100.00
ESDIRK2-3 noPrecond	144	133971	2996.42	0.64	100.00
ESDIRK4-6 ILU(0) NoFillIn	144	51323	1520.32	1.26	100.00
ESDIRK4-6 LU	144	44799	3501.70	0.55	100.00
ESDIRK4-6 noPrecond	144	282377	5938.66	0.32	100.00
ERK4	288	-	964.02	1.00	99.42
ESDIRK2-3 ILU(0) NoFillIn	288	21768	378.82	2.54	96.98
ESDIRK2-3 LU	288	19026	874.47	1.10	95.56
ESDIRK2-3 noPrecond	288	133971	1623.44	0.59	92.29
ESDIRK4-6 ILU(0) NoFillIn	288	51323	816.30	1.18	93.12
ESDIRK4-6 LU	288	44799	1837.65	0.52	95.28
ESDIRK4-6 noPrecond	288	282377	3287.36	0.29	90.33
ERK4	576	-	478.24	1.00	100.20
ESDIRK2-3 ILU(0) NoFillIn	576	21768	218.66	2.19	84.01
ESDIRK2-3 LU	576	19026	476.76	1.00	87.64
ESDIRK2-3 noPrecond	576	133971	994.85	0.48	75.30
ESDIRK4-6 ILU(0) NoFillIn	576	51323	443.72	1.08	85.66
ESDIRK4-6 LU	576	44799	1024.44	0.47	85.45
ESDIRK4-6 noPrecond	576	282377	1927.46	0.25	77.02
ERK4	1152	-	250.70	1.00	95.57
ESDIRK2 ILU(0) NoFillIn	1152	21768	139.44	1.80	65.87
ESDIRK2 LU	1152	19026	272.49	0.92	76.67
ESDIRK2 noPrecond	1152	133971	717.09	0.35	52.23
ESDIRK4 ILU(0) NoFillIn	1152	51323	264.76	0.95	71.78
ESDIRK4 LU	1152	44799	561.44	0.45	77.96
ESDIRK4 noPrecond	1152	282377	1317.41	0.19	56.35

Table 5.5: Comparison of the computational performance of ERK4, ESDIRK2-3 and ESDIRK4-6 schemes including LU, ILU(0) NoFillIn or no preconditioner for the fixed time step $\Delta t^* = 0.2$. The computation is run for 20 time units. nProcs: number of processors, Iter: Total number of GMRES iterations, Time: total wall-clock time, Speed-up$_{\text{impl}} = \frac{\text{Time}_{\text{ERK4}}}{\text{Time}_{\text{ESDIRK}}}$, strong scaling efficiency.

Table 5.5 shows also the perfect parallel scaling of the explicit and all implicit solvers for the first step of doubling the number of processors from 144 to 288 when the number of DOFs per processor is still high, i.e. more than 5000 DOFs. For further increases of the number of processors, the scaling decreases for the implicit schemes as already investigated in Section 4.3.5. When parallelizing implicit time marching schemes, we have to take into account to distribute minimum 5.000-10.000 DOFs per processor in order to have a perfect strong scaling. We can also identify that the implicit schemes with preconditioning scale better than without, which can be explained by the increasing computational load on a processor due to the additional work with respect to the preconditioner. Summarizing, applying a preconditioner increases the ratio of computation to communication and hence can improve the parallel scaling. Also the LU preconditioning shows better scaling than ILU(0) due to the significant higher computational work.

Concluding, for this three-dimensional turbulent test case ESDIRK2-3 showed for a specific high time step size the best spatial accuracy among the implicit and explicit schemes with respect to the DNS results. In comparison to the two-dimensional laminar test case, ILU(0) significantly improved the computational time compared to LU due to the increased sparsity. Overall, ESDIRK2-3 including the ILU(0) NoFillIn preconditioner performed almost three times faster than ERK4 for more than 10.000 DOFs per processor.

5.2 Channel Flow at $Re_\tau = 590$

In this Section, we want to analyze the implicit strategy for the unsteady turbulent channel flow. The channel flow is a special test case since it is periodic in stream-wise direction as well as in time. The time periodicity can be exploited for implicit time integration by freezing the preconditioner. We consider the Reynolds number $Re_\tau = 590$ related to the shear velocity $u_\tau = \sqrt{\tau_w/\rho}$, with the wall shear stress τ_w, and the channel half-width δ as $Re_\tau = \frac{u_\tau \delta}{\nu}$. The Mach number is chosen as $Ma = 0.1$, based on the channel bulk velocity u_{bulk}. We compare for this turbulent flow the LU and the ILU(0) preconditioner in terms of iteration numbers of the linear solver and CPU time for two different polynomial orders.

Spatial Setup The following computations are performed on the physical domain $[0, 2\pi] \times [-1, 1] \times [0, \pi] \ni (x, y, z)$ which is bounded by walls in y-direction and consists of periodic boundaries in x- and z-direction. Hence, the dimensions are identically set as in Moser et al. [88] in order to compare the computational results to the reference data of the DNS. The domain is decomposed into $16 \times 16 \times 16$ grid cells with a grid stretching in wall normal direction by means of a bell shape stretching with ratio 8 from smallest to largest cell and an equidistant distribution of the cells in wall parallel direction as visualized in Figure 5.10.

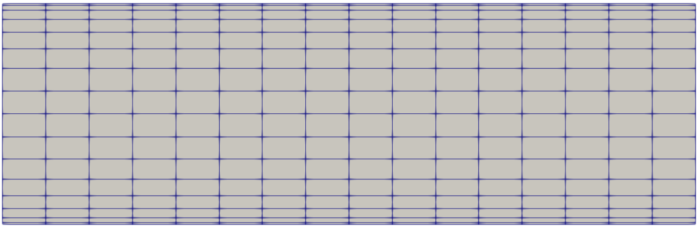


Figure 5.10: Computational mesh in the $x - y$ plane with grid stretching in wall normal direction in order to resolve the viscous sublayer for the channel flow at $Re_\tau = 590$.

The resulting computational details are listed in Table 5.7 for the polynomial order $N = 3$ and $N = 4$ for the same grid. The considered spatial resolution is chosen similarly as in [51] and coarser than the reference DNS. We employ a uDNS DGSEM approach, where mainly the Riemann solver causes numerical dissipation instead of an explicit sub-grid scale model. For the Riemann solver, we choose a Roe's approximate Riemann solver with entropy fix by Harten and Hymen [57].

First, we compute the channel with the explicit ERK4 time discretization for several flow through times in order to obtain turbulent structures. We use this solution as an initial condition for our further investigations. The averaging process for the mean velocity and fluctuation profiles is done for 100 uncorrelated instantaneous flows fields with a non-dimensional sampling period of $Tu_\tau^2/\nu = 29.5$. Figure 5.11 shows s snapshot of the turbulent channel flow for $N = 4$, where the vortices are visualized by the λ_2 criterion colored by the velocity magnitude.

N	Δx^+	Δy^+	Δz^+	Δt^+_{ex}	DOF	$\text{DOF}_\text{R}/\text{DOF}$
3	57.89	2.11	28.98	0.150	64^3	144.56
4	46.32	1.69	23.19	0.104	80^3	74.02

Table 5.7: Discretization settings for the turbulent channel flow. The non-dimensionalized minimum grid spacing is given by $\Delta(.)^+ = \Delta(.)u_\tau/\nu$ and the non-dimensionalized explicit time step size by $\Delta t^+_{\text{ex}} = \Delta t_{\text{ex}} u_\tau^2/\nu$. The relative number of DOF refers to the reference DNS degrees of freedom DOF_R from Moser [88].

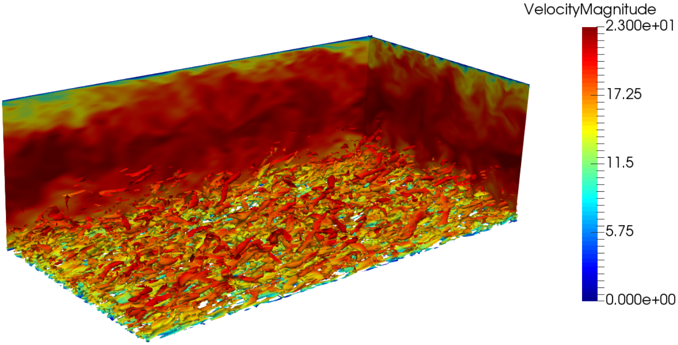


Figure 5.11: Snapshot of the uDNS with polynomial degree $N = 4$ of the channel flow at $Re_\tau = 590$. λ_2-isocontours and boundary slices are colored by the velocity magnitude.

Temporal Setup For the implicit solver, we use the ESDIRK4-6 scheme and compare again the influence of different preconditioners as LU and ILU(0) NoFillIn decomposition. The used time steps sizes with corresponding CFL numbers are given in Table 5.8 for both polynomial degrees. The choice of the maximum explicit time step for ERK4 is driven by stability reasons, defining CFL $= 1$, and is denoted in bold in Table 5.8. Employing the adaptive Newton criterion, it turns out that ϵ_{N} is equal 10^{-2} for any proposed CFL number. Further, we set the Krylov subspace dimension to maximum $m = 100$ and consider either the adaptive GMRES tolerances introduced by Eisenstat and Walker or the fixed tolerance of $\eta_k = 0.1$.

$N = 3$	Δt^*	0.295	0.413	0.59	1.18	1.77	$\mathbf{7.90 \cdot 10^{-3}}$
	CFL	37	52	75	150	224	**1**
$N = 4$	Δt^*	0.295	0.413	0.59	0.826	1.18	$\mathbf{5.45 \cdot 10^{-3}}$
	CFL	54	76	108	152	216	**1**

Table 5.8: Corresponding CFL numbers to the time step sizes Δt^*, non-dimensionalized by the bulk velocity and the half channel height. Bold values are related to the explicit ERK4 scheme.

Solution Quality First, we want to investigate the influence of different CFL numbers on the solution quality. In Figure 5.12, the mean velocity profile and mean Reynolds stresses are reported for the implicit ESDIRK4-6 scheme (for CFL=50,100,150,200) and the explicit ERK4 and in comparison to the DNS data from [88]. In the left and right column, the results are depicted for $N = 3$ and $N = 4$, respectively. We can extract from each plot the perfect agreement of the implicit and the explicit results independent of the proposed CFL numbers. The reason for this lies in the severe stability condition for the explicit scheme due to the small grid cells at the wall boundary, whereas higher time steps are feasible with no influence on the solution accuracy. Further, we can see in Figure 5.12 the significant improvement of the match to the DNS reference data from $N = 3$ to $N = 4$.

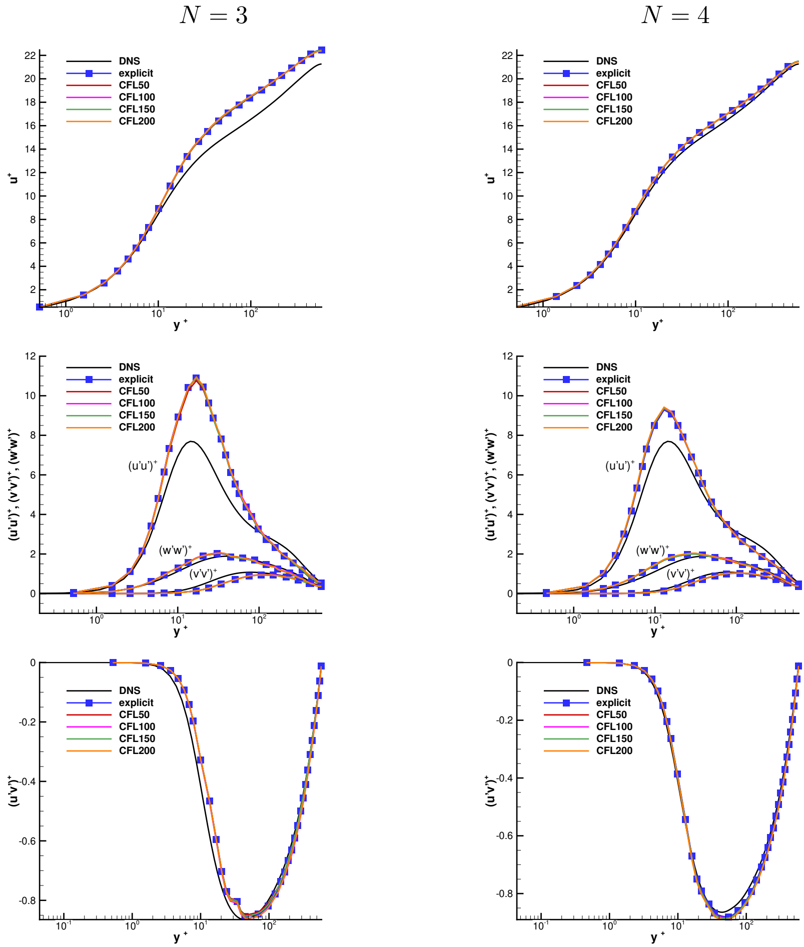


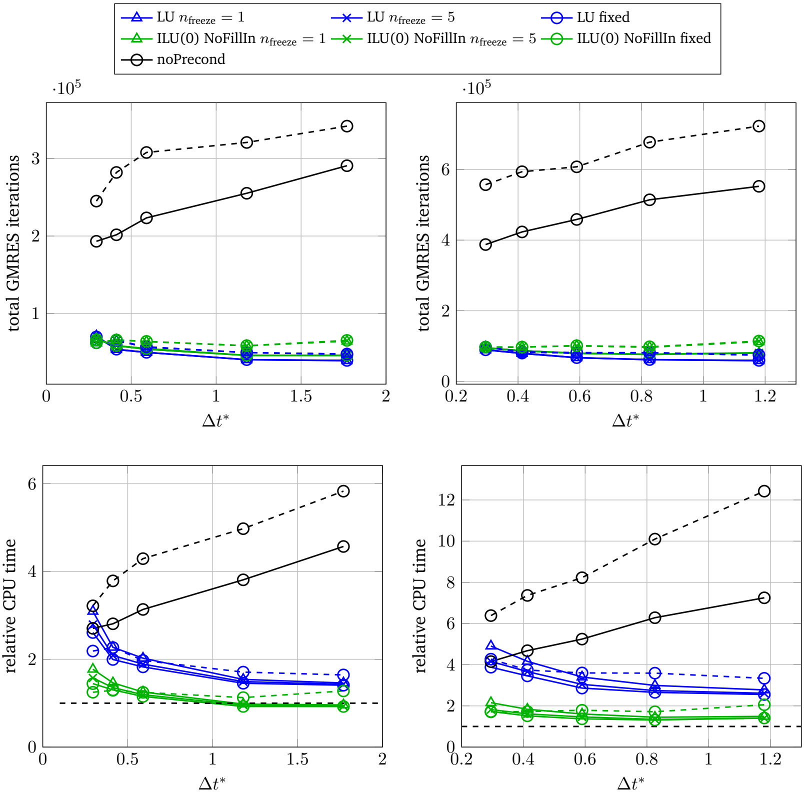
Figure 5.12: Mean velocity profile and Reynolds stresses of the turbulent channel flow at $Re_D = 590$ for the ERK4 and ESDIRK4-6 with different CFL numbers compared to DNS data from Moser et al. [88]. Left column: $N = 3$, right column: $N = 4$.

Solver Efficiency Since the channel test case is a time-periodic problem, we want to examine the influence of different freezing numbers on the number of GMRES iterations and CPU time for the different preconditioners as illustrated in Figure 5.13. We recalculate the preconditioner every time step ($n_{\text{freeze}} = 1$) as reference, freeze it for 5 time steps ($n_{\text{freeze}} = 5$) and keep it constant for the whole computation as denoted by fixed in the legend plot. The computations are performed for a simulation time of $Tu_\tau^2/\nu = 150$ corresponding approximately to 5 flows through time of half of the channel width. For $N = 3$ we use 48 cores and for $N = 4$ 240 cores, which is according to an average number of 5461 and respectively 2133 DOFs per core. The solid lines in Figure 5.13 denote the results for the adaptive GMRES tolerance and the dashed lines are related to a fixed tolerance of $\eta_k = 0.1$.

The total number of GMRES iterations for the preconditioned implicit solver decrease for both polynomial degrees with increasing time step size, whereas non preconditioning causes a high total amount and a growth in GMRES iterations. All different preconditioner freezing settings do not have any effects on the total GMRES iterations since all lines, dashed or solid, match perfectly. This is attributed to the time-periodicity of the channel test case and allows us to save unnecessary building costs of the preconditioners by keeping them fixed for the whole computation.

For this problem, we can see again that the adaptive GMRES tolerance (solid) requires less GMRES iterations than using $\eta_k = 0.1$ (dashed). In matters of iterations the LU preconditioner shows naturally the minimum. Comparing both polynomial degrees, the implicit solver with $N = 4$ needs almost double of the iterations as with $N = 3$.

The CPU time of the ESDIRK4-6 scheme is plotted in Figure 5.13 relative to the CPU time of ERK4. Here, we can extract clearly the advantage of the low application cost of the ILU(0) NoFillIn preconditioner: the implicit solver including the ILU(0) NoFillIn preconditioner almost halves the computational time of using the LU preconditioner. Fixing the preconditioner appears to be rather efficient for small time steps, and especially for the LU preconditioner due to its expensive building cost. The implicit scheme becomes faster in the case of $N = 3$ for the last two time step sizes, but not significantly as for the cylinder flow.



$N = 3$ $\qquad$ $N = 4$

Figure 5.13: Comparison of different freezing numbers n_{freeze} for the preconditioners in terms of total GMRES iterations and relative CPU time, corresponding to the time related to ESDIRK4-6 divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size. Solid lines: adaptive GMRES tolerance, dashed lines: $\eta_k = 0.1$. Left column: $N = 3$ on 48 cores, right column $N = 4$ on 240 cores.

In Table 5.9 the absolute CPU and wall-clock times are listed for ESDIRK4-6 including the ILU(0) NoFillIn preconditioner and the ERK4 scheme.

N	CFL	#procs	T^*_{CPU}(ILU(0)) $[h]$	T^*_{Wall}(ILU(0)) $[min]$	T^*_{CPU}(ex) $[h]$	T^*_{Wall}(ex) $[min]$
3	150	48	1.68	2.10	1.81	2.27
	224		1.66	2.07		
4	152	240	6.24	1.56	4.84	1.21

Table 5.9: CPU and wall-clock time of implicit scheme with ILU(0) NoFillIn preconditioning and of explicit method for the turbulent channel flow at $Re_\tau = 590$.

Summarizing, in this three-dimensional turbulent test case we could choose time steps 200 times greater than the explicit, stability driven time step without any loss in accuracy. This special test case has the benefit in computing the preconditioner once in the beginning and keeping it constant until the end of the simulations, which has no effect on the number of iterations but on the computational time by saving building costs of the preconditioner. For higher polynomial degrees we obtain greater linear systems to solve and thus, more expensive implicit solvers.

5.3 Discussion

In this Section, we applied the implicit solver with the LU and ILU(0) NoFillIn preconditioner for laminar and turbulent circular cylinder flows in two- and three space dimensions and for the turbulent channel flow. In all cases, ILU(0) NoFillIn yields the most efficient preconditioner especially for computations in 3D, where the LU preconditioner became significantly more expensive even for low polynomial orders of $N = 3$ and $N = 4$. We compared ESDIRK schemes of different orders, where obviously ESDIRK2-3, with only two implicit stages, described the fastest scheme among all proposed ESDIRK schemes and outperformed ERK4 by a factor of three. Surprisingly, the accuracy of the low order ESDIRK2-3 for higher time step sizes was still in good agreement with DNS data. Further, we could examine a perfect strong scaling of the implicit solver when using a preconditioner

due to the additional work per processor putting the communication of the processors into the background compared to no preconditioning. However, we have to remark that a distribution of minimum 5.000-10.000 DOFs per processor are necessary to maintain a good parallel scaling.
Further, we observed that for the laminar case the fixed GMRES termination criterion of $\eta_k = 0.1$ decreased the number of GMRES iterations, and hence the CPU time, even more than using the adaptive tolerances. But for the turbulent applications, we could find the opposite case.
The freezing strategy is only recommendable for smaller time step sizes when the number of time steps is high and the number of additional GMRES iterations can be balanced by the loss of building costs.
By increasing the polynomial degree, the implicit solver without preconditioning became prohibitively expensive making the usage of a preconditioner essential. LU preconditioning also increases significantly the computational cost for higher polynomial degrees, wheres ILU(0) NoFillIn remains still efficient with a slight gain of costs due to the increasing sparsity of the block-Jacobian matrix.

6 Conclusion and Prospects

6.1 Conclusion

Stiff problems lead to very severe time step restrictions for explicit time integration schemes due to the CFL condition. Implicit time integration schemes describe unconditionally stable methods enabling higher time step sizes. However, in the context of high order DG methods a strategy for efficiently solving the (non-) linear equation systems is still lacking. For a successful application to unsteady simulations of multi-scale problems, open topics as the computational efficiency in terms of CPU time and the accuracy of the implicit solver have to be addressed. This work contributes to further investigations of implicit time integration in combination with DGSEM, a particularly efficient variant of DG methods, for the two- and three-dimensional unsteady Navier-Stokes equations by improving the solver's efficiency and comparing implicit with explicit time integration concerning CPU time and accuracy.

In the first part of this work, we considered the implicit solver choosing a recently developed ansatz for defining the termination criterion of the Newton scheme. The Newton tolerances are computed adaptively with respect to the temporal accuracy. This is done by means of the embedded Runge-Kutta schemes so that the Newton error is held below the time discretization error. Hence, over- and under-solving of the equation system is prevented. Also the tolerances of the linear solver are chosen automatically in the way that over-solving is avoided with respect to the Newton error. The automatic strategy for both tolerances makes the implicit solver easier to handle and significantly more efficient as investigated for the inviscid traveling vortex test case. We also suggest to use small dimensions of Krylov subspaces within the restarted GMRES due to the expensive construction of an orthogonal basis. However, this is only possible if no convergence issues occur.

In the second part, we introduced the block-Jacobi preconditioner deduced for DGSEM including the BR2 lifting procedure. An important factor for the efficiency of a preconditioner is given by the computational cost of its application in every GMRES iteration. The best approximative preconditioner with the lowest number of iterations is not necessarily the most efficient preconditioner. Hence, we employed a specific matrix structure, named NoFillIn, for viscid flows maintaining the sparsity of the Euler Jacobian in every diagonal block. Thus, the solving cost when exploiting the matrix sparsity decreases significantly. Despite the even worse approximation to the exact block-Jacobi, we showed by means of the viscid traveling vortex test case that the required amount of iterations is not strongly affected in comparison to the full matrix. Further, we investigated several approximate solvers as SGS, ILU(0) and a tensor-product solver in comparison to the exact LU factorization for the two- and three-dimensional space. Since LU is exact, it requires the least number of GMRES iterations, but due to its enormous cost in building and application it is not affordable for high order discretizations in 3D. The tensor-product preconditioner, which reduces the multidimensional problem into 1D systems, and the ILU(0) NoFillIn preconditioner, which exploits the d-dimensional matrix sparsity by using the CSR format, showed for the application cost the same computational complexity $\mathcal{O}(N^{d+1})$ for d space dimensions. In comparison, the application of the LU factorization accounts the cost $\mathcal{O}(N^{2d})$. Also in terms of storage requirements and iteration counts ILU(0) NoFillIn describes the overall most efficient choice among the proposed approximate solvers.

Generally, the block-Jacobi preconditioner, which reduces the global linear system into several element-local systems, verifies a suitable choice for large-scale parallel simulations. Strong scaling tests point out to use minimum 10,000 DOFs per core for perfect parallel scaling.

In the last part, we evaluated the efficiency and accuracy of the implemented implicit DGSEM within the open source code FLEXI for spatially under resolved simulations of laminar and turbulent flows. We want to highlight that for under resolutions as usually used in large eddy simulations it is not necessary to employ high order temporal methods, which can be seen in the accuracy of the proposed results. ESDIRK2-3 of second order gives similar results as the explicit scheme for the laminar flow around a circular cylinder and even better results compared to the DNS of the turbulent cylinder flow. Implicit DGSEM with ESDIRK2-3 performed up to three times faster and ESDIRK4-6 up to two times faster than the standard explicit scheme. For the turbulent channel flow we investigated that for increasing order of the spatial discretization LU and no preconditioning become significantly more expensive whereas ILU(0) NoFillIn possess only a slight increase of costs. Further, we could determine that the adaptive GMRES tolerances are recommendable for the turbulent test cases.

6.2 Prospects

The proposed implicit DGSEM with element-local ILU(0) NoFillIn preconditioner including the adaptive tolerances provides an attractive solver for stiff problems, which occur also in wall-bounded flows. Especially considering computational costs with appropriate accuracy, implicit DGSEM is worthwhile for further investigations in the context of multi-scale problems.

In order to obtain a more convenient and easier to handle implementation of the implicit solver, the preconditioner freezing for smaller time step sizes could be done in an adaptive way in order to save computational cost in building the preconditioner. Another helpful feature is to determine also the maximum number of Krylov subspace dimension within GMRES adaptively. Then expensive orthogonal basis constructions and additional memory would be omitted in consideration of the convergence property of the linear solver. Hence, implicit DGSEM can be applied more generally to several test cases benefiting from the reduced number of adjustable parameters.

The block-Jacobi preconditioner consists only of diagonal block matrices limiting the acceleration of GMRES. Further investigations on extending the dependency also to neighboring elements which lay on the same processor can be of high interest.

The application of Implicit DGSEM for turbulent flows with very high Reynolds numbers with a resolved boundary layer could be developed. Tiny elements in the boundary layer lead to geometry induced stiffness and forces explicit time integration schemes to very small time step sizes, so that an implicit solver can gain much better performance. Here, the impact of under resolution on the solver's stability may require additional effort for stabilization.

A Implicit Butcher Coefficients

In this section, we list the Butcher tables of the employed second-, third- and fourth-order Explicit first stage Singly Diagonally Implicit Runge-Kutta schemes (ESDIRK). Additionally, we provide the corresponding embedded schemes holding one order lower than the original method, denoted by the superscript ˆ.

Second-Order ESDIRK

$$
\begin{array}{c|ccc}
0 & 0 & 0 & 0 \\
2\gamma & \gamma & \gamma & 0 \\
1 & 1-b_2-\gamma & b_2 & \gamma \\
\hline
b_i & 1-b_2-\gamma & b_2 & \gamma \\
\hline
\hat{b}_i & 1-\hat{b}_2-\hat{b}_3 & \hat{b}_2 & \hat{b}_3
\end{array}
$$

Table A.1: ESDIRK2-3: 2nd-order 3-stage L-stable and stiffly accurate scheme summarized in Carpenter and Kennedy [72].

The parameters are chosen as

$$
\begin{aligned}
\gamma &= 1 - \frac{\sqrt{2}}{2}, \\
b_2 &= \frac{(1-2\gamma)}{4\gamma}, \\
\hat{b}_2 &= \frac{\gamma(-2+7\gamma-5\gamma^2+4\gamma^3)}{2(2\gamma-1)}, \quad \hat{b}_3 = \frac{-2\gamma^2(1-\gamma+\gamma^2)}{2\gamma-1}.
\end{aligned}
$$

Third-Order ESDIRK

0	0	0	0	0
$\frac{1767732205903}{2027836641118}$	$\frac{1767732205903}{4055673282236}$	$\frac{1767732205903}{4055673282236}$	0	0
$\frac{3}{5}$	$\frac{2746238789719}{10658868560708}$	$-\frac{640167445237}{6845629431997}$	$\frac{1767732205903}{4055673282236}$	0
1	$\frac{1471266399579}{7840856788654}$	$-\frac{4482444167858}{7529755066697}$	$\frac{11266239266428}{11593286722821}$	$\frac{1767732205903}{4055673282236}$
b_i	$\frac{1471266399579}{7840856788654}$	$-\frac{4482444167858}{7529755066697}$	$\frac{11266239266428}{11593286722821}$	$\frac{1767732205903}{4055673282236}$
$\hat{b}_i$	$\frac{2756255671327}{12835298489170}$	$-\frac{10771552573575}{22201958757719}$	$\frac{9247589265047}{10645013368117}$	$\frac{2193209047091}{5459859503100}$

Table A.2: ESDIRK3-4: 3rd-order 4-stage L-stable and stiffly accurate scheme by Carpenter and Kennedy [71].

Fourth-Order ESDIRK

0	0	0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0
$\frac{83}{250}$	$\frac{8611}{62500}$	$-\frac{1743}{31250}$	$\frac{1}{4}$	0	0	0
$\frac{31}{50}$	$\frac{5012029}{34652500}$	$-\frac{654441}{2922500}$	$\frac{174375}{388108}$	$\frac{1}{4}$	0	0
$\frac{17}{20}$	$\frac{15267082809}{155376265600}$	$-\frac{71443401}{120774400}$	$\frac{730878875}{902184768}$	$\frac{2285395}{8070912}$	$\frac{1}{4}$	0
1	$\frac{82889}{524892}$	0	$\frac{15625}{83664}$	$\frac{69875}{102672}$	$-\frac{2260}{8211}$	$\frac{1}{4}$
b_i	$\frac{82889}{524892}$	0	$\frac{15625}{83664}$	$\frac{69875}{102672}$	$-\frac{2260}{8211}$	$\frac{1}{4}$
$\hat{b}_i$	$\frac{4586570599}{29645900160}$	0	$\frac{178811875}{945068544}$	$\frac{814220225}{1159782912}$	$-\frac{3700637}{11593932}$	$\frac{61727}{225920}$

Table A.3: ESDIRK4-6: 4th-order 6-stage L-stable and stiffly accurate scheme by Carpenter and Kennedy [71].

B Explicit Butcher Coefficients

Here, we provide the coefficients of the explicit Runge-Kutta scheme of fourth order, denoted by ERK4, which we employ for comparing the computational costs of the implicit Runge-Kutta schemes (ESDIRK). The selected ERK4 method consists of five stages and is written in low storage form of Williamson type [116] with two registers, denoted by 2N.

i	A	b	c
1	0	$\frac{1432997174477}{9575080441755}$	0
2	$\frac{567301805773}{1357537059087}$	$\frac{5161836677717}{13612068292357}$	$\frac{1432997174477}{9575080441755}$
3	$\frac{2404267990393}{2016746695238}$	$\frac{1720146321549}{2090206949498}$	$\frac{2526269341429}{6820363962896}$
4	$\frac{3550918686646}{2091501179385}$	$\frac{3134564353537}{4481467310338}$	$\frac{2006345519317}{3224310063776}$
5	$\frac{1275806237668}{842570457699}$	$\frac{2277821191437}{14882151754819}$	$\frac{2802321613138}{2924317926251}$

Table B.1: ERK4: 2N 4th-order 5-stage scheme by Carpenter and Kennedy [25].

C Traveling Vortex Test Case Definitions

For the Euler equations an exact analytical solution of the two- and three-dimensional traveling isentropic vortex problem was derived by Hu and Shu in [67]. The constant background flow with state vector $(\rho_0, \boldsymbol{v}_0^T, p_0)$ is perturbed such that

$$
\begin{aligned}
\boldsymbol{r} &= \boldsymbol{r}_{rot} \times (\boldsymbol{x} - \boldsymbol{x}_0 - \boldsymbol{v}_0 t), \\
\delta v &= \frac{v_{max}}{2\pi} exp \left(\frac{1 - \left(\frac{|\boldsymbol{r}|}{r_0} \right)^2}{2} \right), \\
\boldsymbol{v}(\boldsymbol{x}, t) &= \boldsymbol{v}_0 + \delta v \, \boldsymbol{r}, \\
\frac{T}{T_0} &= 1 - \frac{\kappa - 1}{2\kappa} \delta v^2, \\
\rho(\boldsymbol{x}, t) &= \rho_0 \left(\frac{T}{T_0} \right)^{\frac{1}{\kappa - 1}}, \\
p(\boldsymbol{x}, t) &= p_0 \left(\frac{T}{T_0} \right)^{\frac{\kappa}{\kappa - 1}},
\end{aligned}
$$

with the adiabatic coefficient $\kappa = 1.4$. For validating the implementation of the implicit solver presented in this work, we set $\rho_0 = 1$ and $\boldsymbol{v}_0 = (0.5, 0.5, 0.5)^T$. The pressure p_0 is chosen corresponding to the background Mach number $Ma = 0.5$ for the two- and three-dimensional case. The amplitude of the vortex is $v_{max} = 4$, the initial center of the vortex $\boldsymbol{x}_0 = (0.5, 0.5, 0.5)^T$ and the halfwidth of the vortex $r_0 = 0.2$. The rotational axis of the vortex is set to $\boldsymbol{r}_{rot} = (0, 0, 1)^T$. The end time of the simulation is $T = 1$ and the computational domain is given by $\Omega := [0, 2]^d$ for $d = 2$ or 3.

Solving the hyperbolic Euler equations, we employ exact Dirichlet boundary conditions. The computational mesh is determined by a Cartesian equidistant grid with 40×40 cells and the polynomial degree $N = 6$. For the implicit solver, we use the restarted version GMRES(20) as linear solver.

Considering the compressible, viscous Navier-Stokes equations the above functions are not a valid exact solution anymore. In this case, the initial vortex with above provided initial values will be transported and as well diffuse in time. Thus, we set for the computations periodic boundary conditions. The viscosity is chosen as $\mu = 0.005$ corresponding to a Reynolds number $Re = 100$ related to the unit reference length. The spatial resolution and used configurations for the linear solver are specifically announced for the particular test case.

D System Architecture

The computations in this work are conducted on the supercomputer "Hazel Hen" which is a Cray XC40 system of the High-Performance Computing Center (HLRS) in Stuttgart. The supercomputer consists of 7712 nodes with two sockets each holding an Intel Xeon CPU (E5-2680 v3) with 2.5 GHz and 12 cores. The memory per computing node is 128 GB. The code is built with the GNU Fortran (GCC) 6.0.4. compiler and the Cray MPI library 7.7.3.

Bibliography

[1] D. N. Arnold. “An Interior Penalty Finite Element Method with Discontinuous Elements”. In: *SIAM Journal on Numerical Analysis* 19.4 (1982), pp. 742–760.

[2] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. “Discontinuous Galerkin Methods for Elliptic Problems”. In: *Discontinuous Galerkin Methods. Lecture Notes in Computational Science and Engineering*. Ed. by B. Cockburn, G. Karniadakis, and C.-W. Shu. Springer, 2000, pp. 89–101.

[3] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. “Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems”. In: *SIAM J. Numer. Anal.* 39.5 (2002), pp. 1749–1779.

[4] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. “Unified analysis of discontinuous Galerkin methods for elliptic problems”. In: *SIAM Journal on Numerical Analysis* 39.5 (2002), pp. 1749–1779.

[5] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri. “Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations”. In: *Applied Numerical Mathematics* 25.2-3 (1997), pp. 151–167.

[6] M. Atak, A. Beck, T. Bolemann, D. Flad, H. Frank, and C.-D. Munz. “High fidelity scale-resolving computational fluid dynamics using the high order discontinuous Galerkin spectral element method”. In: *High Performance Computing in Science and Engineering´ 15*. Springer, 2016, pp. 511–530.

[7] R. H. Bartels and G. W. Stewart. “Solution of the matrix equation AX+ XB= C [F4]”. In: *Communications of the ACM* 15.9 (1972), pp. 820–826.

[8] F. Bassi, L. Botti, A. Colombo, A. Ghidoni, and F. Massa. “Linearly implicit Rosenbrock-type Runge–Kutta schemes applied to the Discontinuous Galerkin solution of compressible and incompressible unsteady flows”. In: *Computers & Fluids* 118 (2015), pp. 305–320.

[9] F. Bassi and S. Rebay. “A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier–Stokes Equations”. In: *Jornal of Computational Physics* 131 (1997), pp. 267–279.

[10] F. Bassi and S. Rebay. “GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations”. In: *Discontinuous Galerkin Methods*. Springer, 2000, pp. 197–208.

[11] F. Bassi and S. Rebay. “Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier–Stokes equations”. In: *International Journal for Numerical Methods in Fluids* 40 (2002), pp. 197–207.

[12] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini. “A High-Order Accurate Discontinuous Finite Element Method for Inviscid an Viscous Turbomachinery Flows”. In: *Proceedings of 2nd European Conference on Turbomachinery, Fluid and Thermodynamics*. Ed. by R. Decuypere and G. Dibelius. Technologisch Instituut, Antwerpen, Belgium, 1997, pp. 99–108.

[13] F. Bassi, A. Colombo, C. De Bartolo, N. Franchina, A. Ghidoni, and A. Nigro. “Investigation of high-order temporal schemes for the discontinuous Galerkin solution of the navier-stokes equations”. In: *Proceedings of the 11th world congress on computational mechanics, WCCM 2014, 5th European conference on computational mechanics, ECCM 2014 and 6th European conference on computational fluid dynamics, ECFD 2014*. 2014, pp. 5651–5662.

[14] F. Bassi, A. Crivellini, D. A. Di Pietro, and S. Rebay. “An implicit high-order discontinuous Galerkin method for steady and unsteady incompressible flows”. In: *Computers & Fluids* 36.10 (2007), pp. 1529–1546.

[15] A. D. Beck, T. Bolemann, D. Flad, H. Frank, G. J. Gassner, F. Hindenlang, and C.-D. Munz. “High-order Discontinuous Galerkin Spectral Element Methods for Transitional and Turbulent Flow Simu-

lations”. In: *International Journal for Numerical Methods in Fluids* 76.8 (2014), pp. 522–548.

[16] A. D. Beck, D. G. Flad, C. Tonhäuser, G. Gassner, and C.-D. Munz. “On the influence of polynomial de-aliasing on subgrid scale models”. In: *Flow, Turbulence and Combustion* 97.2 (2016), pp. 475–511.

[17] H. Bijl and M. Carpenter. “Iterative solution techniques for unsteady flow computations using higher order time integration schemes”. In: *International journal for numerical methods in fluids* 47.8-9 (2005), pp. 857–862.

[18] P. Birken, G. Gassner, M. Haas, and C. Munz. “A new class of preconditioners for discontinuous Galerkin methods for unsteady 3D Navier-Stokes equations: ROBO-SGS”. In: *Journal of Computational Physics, submitted* (2012).

[19] P. Birken, G. Gassner, M. Haas, and C.-D. Munz. “Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navier–Stokes equations”. In: *Journal of Computational Physics* 240 (2013), pp. 20–35.

[20] D. S. Blom, P. Birken, H. Bijl, F. Kessels, A. Meister, and A. H. van Zuijlen. “A comparison of Rosenbrock and ESDIRK methods combined with iterative solvers for unsteady compressible flows”. In: *Advances in Computational Mathematics* 42.6 (2016), pp. 1401–1426.

[21] S. Brdar, A. Dedner, and R. Klöfkorn. “Compact and stable Discontinuous Galerkin methods for convection-diffusion problems”. In: *SIAM Journal on Scientific Computing* 34.1 (2012), A263–A282.

[22] P. N. Brown. “A local convergence theory for combined inexact-Newton/ finite-difference projection methods”. In: *SIAM Journal on Numerical Analysis* 24.2 (1987), pp. 407–434.

[23] P. N. Brown and Y. Saad. “Hybrid Krylov methods for nonlinear systems of equations”. In: *SIAM Journal on Scientific and Statistical Computing* 11.3 (1990), pp. 450–481.

[24] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang. *Spectral Methods: Fundamentals in Single Domains*. Springer, 2006.

[25] M. Carpenter and C. Kennedy. *Fourth-Order 2N-Storage Runge-Kutta Schemes*. Tech. rep. NASA TM 109111. 1994.

[26] R. Chen and Z. Wang. “Fast, block lower-upper symmetric Gauss-Seidel scheme for arbitrary grids”. In: *AIAA journal* 38.12 (2000), pp. 2238–2245.

[27] T. T. Chisholm and D. W. Zingg. “A Jacobian-free Newton–Krylov algorithm for compressible turbulent fluid flows”. In: *Journal of Computational Physics* 228.9 (2009), pp. 3490–3507.

[28] B. Cockburn, S. Hou, and C. W. Shu. “The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: The multidimensional case”. In: *Math. Comput.* 54 (1990), pp. 545–581.

[29] B. Cockburn, S. Y. Lin, and C. Shu. “TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One dimensional systems”. In: *J. Comput. Phys.* 84 (1989), pp. 90–113.

[30] B. Cockburn and C. W. Shu. “The Runge-Kutta discontinuous Galerkin method for conservation laws V: Multidimensional systems”. In: *J. Comput. Phys.* 141 (1998), pp. 199–224.

[31] B. Cockburn and C. W. Shu. “TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: General framework”. In: *Math. Comput.* 52 (1989), pp. 411–435.

[32] B. Cockburn. “Devising discontinuous Galerkin methods for non-linear hyperbolic conservation laws”. In: *Journal of Computational and Applied Mathematics* 128.1-2 (2001), pp. 187–204.

[33] B. Cockburn and C.-W. Shu. “Runge–Kutta Discontinuous Galerkin Methods for Convection-Dominated Problems”. English. In: *Journal of Scientific Computing* 16.3 (2001), pp. 173–261.

[34] B. Cockburn and C.-W. Shu. “The local discontinuous Galerkin method for time-dependent convection-diffusion systems”. In: *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2440–2463.

[35] B. Cockburn and C.-W. Shu. “The Runge–Kutta local projection P1-discontinuous Galerkin finite element method for scalar conservation laws”. In: *RAIRO Modél. Math. Anal. Numér* 25.3 (1991), pp. 337–361.

[36] S. M. Copplestone, P. Ortwein, and C.-D. Munz. “Complex-Frequency Shifted PMLs for Maxwell’s Equations With Hyperbolic Divergence Cleaning and Their Application in Particle-in-Cell Codes”. In: *IEEE Transactions on Plasma Science* 45.1 (2017), pp. 2–14.

[37] R. Courant, K. Friedrichs, and H. Lewy. “Über die partiellen Differenzengleichungen der mathematischen Physik”. In: *Mathematische annalen* 100.1 (1928), pp. 32–74.

[38] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. “Inexact newton methods”. In: *SIAM Journal on Numerical analysis* 19.2 (1982), pp. 400–408.

[39] R. B. K. Devine and J. Flaherty. “Parallel, adaptive finite element methods for conservation laws”. In: *Appl. Numer. Math.* 14 (1994), pp. 255–283.

[40] L. Diosady and S. Murman. “Tensor-product preconditioners for higher-order space–time discontinuous Galerkin methods”. In: *Journal of Computational Physics* 330 (2017), pp. 296–318.

[41] L. T. Diosady and D. L. Darmofal. “Preconditioning methods for discontinuous Galerkin solutions of the Navier–Stokes equations”. In: *Journal of Computational Physics* 228.11 (2009), pp. 3917–3935.

[42] L. T. Diosady and S. M. Murman. “Tensor-product preconditioners for higher-order space–time discontinuous Galerkin methods”. In: *Journal of Computational Physics* 330 (2017), pp. 296–318.

[43] L. T. Diosady and S. M. Murman. “Tensor-product preconditioners for higher-order space–time discontinuous Galerkin methods”. In: *Journal of Computational Physics* 330 (2017), pp. 296–318.

[44] B. Einfeldt, C.-D. Munz, P. L. Roe, and B. Sjögreen. “On Godunov-type Methods Near Low Densities”. In: *Journal of computational physics* 92.2 (1991), pp. 273–295.

[45] S. C. Eisenstat and H. F. Walker. “Choosing the forcing terms in an inexact Newton method”. In: *SIAM Journal on Scientific Computing* 17.1 (1996), pp. 16–32.

[46] S. Fechter and C.-D. Munz. “A discontinuous Galerkin-based sharp-interface method to simulate three-dimensional compressible two-phase flow”. In: *International Journal for Numerical Methods in Fluids* 78.7 (2015), pp. 413–435.

[47] S. Fechter. “Compressible Multi-Phase Simulation at Extreme Conditions using a Discontinuous Galerkin Scheme”. Dissertation. Institute of Aerodynamics and Gasdynamics, University of Stuttgart, Germany, 2015.

[48] S. Fechter, F. Hindenlang, H. Frank, C.-D. Munz, and G. Gassner. “Discontinuous Galerkin schemes for the direct numerical simulation of fluid flow and acoustics”. In: *18th AIAA/CEAS Aeroacoustics Conference (33rd AIAA Aeroacoustics Conference)*. 2012, p. 2187.

[49] K. J. Fidkowski, T. A. Oliver, J. Lu, and D. L. Darmofal. “p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations”. In: *Journal of Computational Physics* 207.1 (2005), pp. 92–113.

[50] D. Flad, A. D. Beck, G. Gassner, and C.-D. Munz. “A discontinuous Galerkin spectral element method for the direct numerical simulation of aeroacoustics”. In: *20th AIAA/CEAS Aeroacoustics Conference*. 2014, p. 2740.

[51] D. Flad and G. Gassner. “On the use of kinetic energy preserving DG-schemes for large eddy simulation”. In: *Journal of Computational Physics* 350 (2017), pp. 782–795.

[52] M. Franciolini, L. Botti, A. Colombo, and A. Crivellini. “p-Multigrid matrix-free discontinuous Galerkin solution strategies for the under-resolved simulation of incompressible turbulent flows”. In: *ArXiv e-prints* (2018). arXiv preprint: `1809.00866`.

[53] M. Franciolini, A. Crivellini, and A. Nigro. “On the efficiency of a matrix-free linearly implicit time integration strategy for high-order Discontinuous Galerkin solutions of incompressible turbulent flows”. In: *Computers & Fluids* 159 (2017), pp. 276–294.

[54] H. M. Frank and C.-D. Munz. “Direct Aeroacoustic Simulation of Acoustic Feedback Phenomena on a Side-View Mirror”. In: *Journal of Sound and Vibration* 371 (2016), pp. 132–149.

[55] G. Gassner. “Discontinuous Galerkin methods for the unsteady compressible Navier-Stokes equations”. PhD thesis. University of Stuttgart, 2009.

[56] G. Gassner and D. Kopriva. “A Comparison of the Dispersion and Dissipation Errors of Gauss and Gauss-Lobatto Discontinuous Galerkin Spectral Element Methods”. In: *SIAM J. Scientific Computing* 33.5 (2011), pp. 2560–2579.

[57] A. Harten and J. M. Hyman. “Self Adjusting Grid Methods for One–Dimensional Hyperbolic Conservation Laws”. In: *Journal of Computational Physics* 50.2 (1983), pp. 235–269.

[58] A. Harten, P. D. Lax, and B. Van Leer. “On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws”. In: *SIAM Review* 25.1 (1983), pp. 35–61.

[59] R. Hartmann and P. Houston. “Symmetric Interior Penalty DG Methods for the Compressible Navier–Stokes Equations I: Method Formulation”. In: *International Journal of Numerical Analysis & Modeling* 3.1 (2006), pp. 1–20.

[60] F. Hempert, M. Hoffmann, U. Iben, and C.-D. Munz. “On the simulation of industrial gas dynamic applications with the discontinuous Galerkin spectral element method”. In: *Journal of Thermal Science* 25.3 (2016), pp. 250–257.

[61] F. Hempert, S. Boblest, T. Ertl, F. Sadlo, P. Offenhäuser, C. Glass, M. Hoffmann, A. Beck, C.-D. Munz, and U. Iben. “Simulation of real gas effects in supersonic methane jets using a tabulated equation of state with a discontinuous Galerkin spectral element method”. In: *Computers & Fluids* 145 (2017), pp. 167–179.

[62] J. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer, 1008.

[63] F. Hindenlang. “Mesh curving techniques for high order parallel simulations on unstructured meshes”. Ph.D. thesis. University of Stuttgart, 2014.

[64] F. Hindenlang, T. Bolemann, and C.-D. Munz. “Mesh Curving Techniques for High Order Discontinuous Galerkin Simulations”. In: *IDIHOM: Industrialization of High-Order Methods-A Top-Down Approach*. Springer, 2015, pp. 133–152.

[65] F. Hindenlang, G. Gassner, C. Altmann, A. Beck, M. Staudenmaier, and C.-D. Munz. “Explicit Discontinuous Galerkin methods for unsteady problems”. In: *Computers and Fluids* 61 (May 3, 2012), pp. 86–93.

[66] *HOPR (High Order Preprocessor), unstructured high-order mesh generator*. `https://www.hopr-project.org/index.php/Home`. 2018.

[67] C. Hu and C.-W. Shu. “Weighted Essentially Non-Oscillatory Schemes on triangular meshes”. In: *J. Comput. Phys.* 1505 (1999), pp. 97–127.

[68] A. Jameson and S. Yoon. “Lower-upper implicit schemes with multiple grids for the Euler equations”. In: *AIAA journal* 25.7 (1987), pp. 929–935.

[69] A. Kanevsky, M. H. Carpenter, D. Gottlieb, and J. S. Hesthaven. “Application of implicit–explicit high order Runge–Kutta methods to discontinuous-Galerkin schemes”. In: *Journal of Computational Physics* 225.2 (2007), pp. 1753–1781.

[70] C. Kelley. “Iterative methods for linear and nonlinear equations, SIAM, Philadelphia, 1995”. In: *MR 96d* 65002 ().

[71] C. Kennedy and M. Carpenter. “Additive Runge-Kutta schemes for convection-diffusion-reaction equations”. In: *Applied Numerical Mathematics* 44.1 (2003), pp. 139–181.

[72] C. A. Kennedy and M. H. Carpenter. “Diagonally implicit Runge-Kutta methods for ordinary differential equations”. In: *A Review. NASA Report. Langley research center. Hampton VA* 23681 (2016), p. 162.

[73] C. A. Kennedy, M. H. Carpenter, and R. M. Lewis. “Low-storage, Explicit Runge–Kutta Schemes for the Compressible Navier–Stokes Equations”. In: *Applied Numerical Mathematics* 35.3 (2000), pp. 177–219.

[74] C. Klaij, J. J. W. van der Vegt, and H. van der Ven. “Spacetime Discontinuous Galerkin Method for the Compressible Navier–Stokes Equations”. In: *Journal of Computational Physics* 217.2 (2006), pp. 589–611.

[75] D. Knoll and D. Keyes. “Jacobian-free Newton–Krylov methods: a survey of approaches and applications”. In: *Journal of Computational Physics* 193.2 (2004), pp. 357–397.

[76] D. A. Knoll and D. E. Keyes. “Jacobian-free Newton–Krylov methods: a survey of approaches and applications”. In: *Journal of Computational Physics* 193.2 (2004), pp. 357–397.

[77] M. A. Kopera and F. X. Giraldo. “Analysis of adaptive mesh refinement for IMEX discontinuous Galerkin solutions of the compressible Euler equations with application to atmospheric simulations”. In: *Journal of Computational Physics* 275 (2014), pp. 92–117.

[78] D. Kopriva. “Metric Identities and the Discontinuous Spectral Element Method on Curvilinear Meshes”. In: *Journal of Scientific Computing* 26.3 (Mar. 2006), pp. 301–327.

[79] D. A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*. 1st. Springer Publishing Company, Incorporated, 2009.

[80] X. Liu, Y. Xia, H. Luo, and 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”. In: *Communications in Computational Physics* 20.4 (2016), pp. 1016–1044.

[81] F. Lörcher, G. Gassner, and C.-D. Munz. “An explicit discontinuous Galerkin scheme with local time-stepping for general unsteady diffusion equations”. In: *Journal of Computational Physics* 227.11 (2008), pp. 5649–5670.

[82] H. Luo, J. D. Baum, and R. Löhner. “A fast, matrix-free implicit method for compressible flows on unstructured grids”. In: *Journal of Computational Physics* 146.2 (1998), pp. 664–690.

[83] H. Luo, H. Segawa, and M. R. Visbal. “An implicit discontinuous Galerkin method for the unsteady compressible Navier–Stokes equations”. In: *Computers & Fluids* 53 (2012), pp. 133–144.

[84] A. Meister. *Numerik linearer Gleichungssysteme*. Vol. 5. Springer, 2011.

[85] A. Meister and C. Vömel. “Efficient preconditioning of linear systems arising from the discretization of hyperbolic conservation laws”. In: *Advances in Computational Mathematics* 14.1 (2001), pp. 49–73.

[86] J. Meneghini, F. Saltara, C. Siqueira, and J. Ferrari Jr. “Numerical simulation of flow interference between two circular cylinders in tandem and side-by-side arrangements”. In: *Journal of fluids and structures* 15.2 (2001), pp. 327–350.

[87] G. Mengaldo, D. De Grazia, D. Moxey, P. E. Vincent, and S. J. Sherwin. “Dealiasing techniques for high-order spectral element methods on regular and irregular grids”. In: *Journal of Computational Physics* 299 (2015), pp. 56–81.

[88] R. D. Moser, J. Kim, and N. N. Mansour. “Direct numerical simulation of turbulent channel flow up to Re τ= 590”. In: *Physics of fluids* 11.4 (1999), pp. 943–945.

[89] C.-D. Munz, M. Auweter-Kurtz, S. Fasoulas, A. Mirza, P. Ortwein, M. Pfeiffer, and T. Stindl. “Coupled particle-in-cell and direct simulation Monte Carlo method for simulating reactive plasma flows”. In: *Comptes Rendus Mécanique* 342.10-11 (2014), pp. 662–670.

[90] A. Nigro, C. De Bartolo, F. Bassi, and A. Ghidoni. “High-order Discontinuous Galerkin solution of unsteady flows by using an advanced implicit method”. In: *High Order Nonlinear Numerical Schemes for Evolutionary PDEs*. Springer, 2014, pp. 135–149.

[91] A. Nigro, C. De Bartolo, F. Bassi, and A. Ghidoni. “Up to sixth-order accurate A-stable implicit schemes applied to the discontinuous Galerkin discretized Navier–Stokes equations”. In: *Journal of Computational Physics* 276 (2014), pp. 136–162.

[92] Numerics Research Group, IAG, University of Stuttgart. *Flexi Framework*. `https://github.com/flexi-framework`. 2018.

[93] P. Ortwein, T. Binder, S. Copplestone, A. Mirza, P. Nizenkov, M. Pfeiffer, T. Stindl, S. Fasoulas, and C.-D. Munz. “Parallel performance of a discontinuous Galerkin spectral element method based PIC-DSMC solver”. In: *High Performance Computing in Science and Engineering ‘14*. Springer, 2015, pp. 671–681.

[94] M. Parsani, K. Van den Abeele, and C. Lacor. “Implicit LU-SGS time integration algorithm for high-order spectral volume method with p-multigrid strategy”. In: *West-East High-Speed Flow Field Conference, Moscow, Russia*. 2007.

[95] W. Pazner and P.-O. Persson. “Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods”. In: *Journal of Computational Physics* 354 (2018), pp. 344–369.

[96] W. Pazner and P.-O. Persson. “Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods”. In: *Journal of Computational Physics* 354 (2018), pp. 344–369.

[97] J. Peraire and P. Persson. “The Compact Discontinuous Galerkin (CDG) Method for Elliptic Problems”. In: *SIAM Journal on Scientific Computing* 30.4 (2008), pp. 1806–1824.

[98] P.-O. Persson. “High-order LES simulations using implicit-explicit Runge-Kutta schemes”. In: *49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. 2011, p. 684.

[99] P.-O. Persson and J. Peraire. “An efficient low memory implicit DG algorithm for time dependent problems”. In: *44th AIAA Aerospace Sciences Meeting and Exhibit*. 2006, p. 113.

[100] P.-O. Persson and J. Peraire. “Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations”. In: *SIAM Journal on Scientific Computing* 30.6 (2008), pp. 2709–2733.

[101] N. Qin, D. K. Ludlow, and S. T. Shaw. “A matrix-free preconditioned Newton/GMRES method for unsteady Navier-Stokes solutions”. In: *International Journal for Numerical Methods in Fluids* 33.2 (2000), pp. 223–248.

[102] B. Rajani, A. Kandasamy, and S. Majumdar. “Numerical simulation of laminar flow past a circular cylinder”. In: *Applied Mathematical Modelling* 33.3 (2009), pp. 1228–1247.

[103] W. Reed and T. Hill. *Triangular Mesh Methods for the Neutron Transport Equation*. Technical Report LA-UR-73-479. Los Alamos Scientific Laboratory, 1973.

[104] F. Renac, S. Gérald, C. Marmignon, and F. Coquel. “Fast time implicit–explicit discontinuous Galerkin method for the compressible Navier–Stokes equations”. In: *Journal of Computational Physics* 251 (2013), pp. 272–291.

[105] P. Roe. “Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes”. In: *Journal of Computational Physics* 135.2 (1997), pp. 250–258.

[106] V. V. Rusanov. “The Calculation of the Interaction of Non-Stationary Shock Waves with Barriers”. In: *Zhurnal Vychislitel’noi Matematiki i Matematicheskoi Fiziki* 1.2 (1961), pp. 267–279.

[107] Y. Saad and M. H. Schultz. “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on scientific and statistical computing* 7.3 (1986), pp. 856–869.

[108] Y. Saad. *Iterative methods for sparse linear systems*. Vol. 82. siam, 2003.

[109] K. Shahbazi, P. F. Fischer, and C. R. Ethier. “A high-order discontinuous Galerkin method for the unsteady incompressible Navier–Stokes equations”. In: *Journal of Computational Physics* 222.1 (2007), pp. 391–407.

[110] R. Silva, R. Almeida, and A. Galeao. “A preconditioner freeze strategy for numerical solution of compressible flows”. In: *Communications in numerical methods in engineering* 19.3 (2003), pp. 197–203.

[111] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer Verlag, 1999.

[112] E. F. Toro, M. Spruce, and W. Speares. “Restoration of the Contact Surface in the HLL-Riemann Solver”. In: *Shock Waves* 4.1 (1994), pp. 25–34.

[113] P. E. Vincent and A. Jameson. “Facilitating the adoption of unstructured high-order methods amongst a wider community of fluid dynamicists”. In: *Mathematical Modelling of Natural Phenomena* 6.3 (2011), pp. 97–140.

[114] L. Wang and D. J. Mavriplis. “Implicit solution of the unsteady Euler equations for high-order accurate discontinuous Galerkin discretizations”. In: *Journal of Computational Physics* 225.2 (2007), pp. 1994–2015.

[115] G. Wanner and E. Hairer. *Solving ordinary differential equations II*. Springer, 1996.

[116] J. Williamson. “Low-storage Runge-Kutta Schemes”. In: *Journal of Computational Physics* 35.1 (1980), pp. 48–56.

[117] Y. Xia, X. Liu, H. Luo, and R. Nourgaliev. “A third-order implicit discontinuous Galerkin method based on a Hermite WENO reconstruction for time-accurate solution of the compressible Navier–Stokes equations”. In: *International Journal for Numerical Methods in Fluids* 79.8 (2015), pp. 416–435.

List of Tables

2.1 Physical quantities 10

2.3 Scaling of the operation counts of the DGSEM operator for the polynomial degree N considering one hexahedral element. 32

3.1 General form of an stiffly accurate ESDIRK Butcher tableau. 40

3.2 Memory demand in double precision (8 Byte) per block row for DGSEM in 3D. 51

3.3 Adaptive forcing terms η_k for GMRES computed as in (3.10). 58

4.1 Different spatial settings with approximately the same initial $\mathcal{L}^2$ error of density ρ. 99

4.2 Mesh configurations for the investigation of parallel efficiency. The maximum number of nodes is related to 9 elements on each core. 107

4.3 Amounts of nodes for the baseline computations for the investigation of parallel efficiency. 107

5.1 Employed non-dimensionalized time step sizes $\Delta t^* = \Delta t u_\infty / D$ and corresponding CFL numbers. Bold values are related to explicit reference computation with maximum stable time step size specifying CFL $= 1$. 119

5.2 Computational results for the cylinder flow at $Re_D = 200$. $\Delta t^* = \Delta t u_\infty / D$: non-dimensionalized time step size, $C_{p_{Base}}$: mean pressure coefficient at the downstream point $x = D/2, y = 0$, Sr: Strouhal number of the lift force $f_{lift} D / u_\infty$, C_D: mean drag coefficient, C_L^{rms}: RMS of the lift coefficient. . . 120

5.3 Comparison of the computational performance of ERK4, ESDIRK2-3 and ESDIRK4-6 schemes including LU, ILU(0) NoFillIn or no preconditioner for the fixed time step $\Delta t^* = 0.4$. The computation is run for 20 time units. nProcs: number of processors, Iter: total number of GMRES iterations, Time: total wall-clock time, Speed-up$_{\text{impl}} = \frac{\text{Time}_{\text{ERK4}}}{\text{Time}_{\text{ESDIRK}}}$, Strong Scaling: $\text{Time}_{\text{nProcs}=1}/(\text{Time}_{\text{nProcs}} \cdot \text{nProcs}) \cdot 100\%$. 125
5.4 Simulation results for the cylinder flow at $Re_D = 3900$. $\Delta t^* = \Delta t u_\infty / D$: non-dimensionalized time step size, $C_{p_{Base}}$: mean pressure coefficient at the downstream point $x = D/2, y = 0$, Sr: Strouhal number of the lift force $f_{lift} D / u_\infty$, C_D: mean drag coefficient, C_L^{rms}: RMS of the lift coefficient, L_r: length of separation bubble 128
5.5 Comparison of the computational performance of ERK4, ESDIRK2-3 and ESDIRK4-6 schemes including LU, ILU(0) NoFillIn or no preconditioner for the fixed time step $\Delta t^* = 0.2$. The computation is run for 20 time units. nProcs: number of processors, Iter: Total number of GMRES iterations, Time: total wall-clock time, Speed-up$_{\text{impl}} = \frac{\text{Time}_{\text{ERK4}}}{\text{Time}_{\text{ESDIRK}}}$, strong scaling efficiency. 132
5.7 Discretization settings for the turbulent channel flow. The non-dimensionalized minimum grid spacing is given by $\Delta(.)^+ = \Delta(.) u_\tau / \nu$ and the non-dimensionalized explicit time step size by $\Delta t_{\text{ex}}^+ = \Delta t_{\text{ex}} u_\tau^2 / \nu$. The relative number of DOF refers to the reference DNS degrees of freedom DOF$_{\text{R}}$ from Moser [88]. 135
5.8 Corresponding CFL numbers to the time step sizes Δt^*, non-dimensionalized by the bulk velocity and the half channel height. Bold values are related to the explicit ERK4 scheme. 136
5.9 CPU and wall-clock time of implicit scheme with ILU(0) NoFillIn preconditioning and of explicit method for the turbulent channel flow at $Re_\tau = 590$. 140

A.1 ESDIRK2-3: 2nd-order 3-stage L-stable and stiffly accurate scheme summarized in Carpenter and Kennedy [72]. 147
A.2 ESDIRK3-4: 3rd-order 4-stage L-stable and stiffly accurate scheme by Carpenter and Kennedy [71]. 148
A.3 ESDIRK4-6: 4th-order 6-stage L-stable and stiffly accurate scheme by Carpenter and Kennedy [71]. 149

B.1 ERK4: 2N 4th-order 5-stage scheme by Carpenter and Kennedy [25]. 151

List of Figures

2.1 Distribution of the Gauss points • in the volume and the nodes for the boundary fluxes ▫ at the interface of the reference element in two space dimensions for a polynomial degree of $N = 4$. 23

3.1 Convergence order of ESDIRK2-3, ESDIRK3-4 and ESDIRK4-6 for the isentropic inviscid traveling vortex in 2D. $\mathcal{L}^2$-error of the density is plotted for the end time $T = 1$. The Newton tolerance is fixed at $\epsilon_{\mathrm{N}} = 10^{-8}$ and no preconditioner is used. 56

3.2 Convergence of the Newton residual for different tolerances η_k of the GMRES method at CFL=10 for one RK stage. No preconditioner is used. 57

3.3 Comparison of different fixed Newton tolerances ϵ_{N} and an adaptive strategy. The computation is run until $T = 1$ with an adaptive GMRES forcing sequence and without preconditioning. 59

3.4 Impact of different Newton tolerances ϵ_{N} on the total GMRES iterations. The computation is run until $T = 1$ with an adaptive GMRES forcing sequence and without preconditioning. 60

3.5 Comparison of different maximum Krylov subspace dimensions m. The GMRES iterations and CPU time denote the total amount until the computing time $T = 1$. 61

4.1 Matrix pattern for the non-zero entries of one diagonal block using the 3D Euler equations applied to the traveling vortex described in Appendix C. 77

4.2 Matrix pattern for the non-zero entries of one diagonal block using the 3D Navier-Stokes equations applied to the traveling vortex described in Appendix C, with $Re = 100$. 78

4.3 Matrix density of one diagonal block over polynomial degree N using the traveling vortex test case described in Appendix C. 78
4.4 Building time of the block-Jacobi matrix in relation to the cost of one DGSEM operator call depending on the polynomial degree N. 79
4.5 Absolute building time of the block-Jacobi matrix over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 80
4.6 Inverting time of the block-Jacobi matrix over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 82
4.7 Application time of the exact inverse of the block-Jacobi matrix plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 82
4.8 Relative building and application time of the SGS preconditioner in CSR format compared to the computing time of one DGSEM operator plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 84
4.9 Relative building and application time of the ILU(0) preconditioner in CSR format compared to the computing time of one DGSEM operator plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 87
4.10 Relative building time of the d one-dimensional block-Jacobian matrices (2D/3D Tensor) compared to the relative building time of the full Jacobian (2D/3D) and NoFillIn version. Times are given in relation to one DGSEM operator call. 89
4.11 Inverting time of the tensor-product preconditioner per diagonal element plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 90
4.12 Application time of the tensor-product preconditioner plotted over polynomial degree N using the traveling vortex test case described in Appendix C. $Re = 100$. 91
4.13 Memory demand for one diagonal block using the traveling vortex test case described in Appendix C with $Re = 100$. . . 92

4.14 Comparison of the proposed preconditioners in terms of application and building time plotted over polynomial degree N using the traveling vortex test case described in Appendix C in 3D, $Re = 100$. 93

4.15 Comparison of the proposed preconditioners in terms of application and building time plotted over the polynomial degree N using the traveling vortex test case described in Appendix C in 2D. $Re = 100$. 94

4.16 Comparison of the proposed preconditioners in terms of GMRES iterations plotted over CFL numbers using the traveling vortex test case described in Appendix C in 2D. $N = 5$, $Re = 100$. Dashed line represents a line with slope of one. . 96

4.17 Comparison of the proposed preconditioners in terms of the implicit CPU time divided by the explicit CPU time until $T = 1$ plotted over CFL numbers using the traveling vortex test case described in Appendix C in 2D. $N = 5$, $Re = 100$. . . . 97

4.18 Comparison of total number of GMRES iterations plotted over the time step size Δt for different spatial settings given in Table 4.1. 99

4.19 Comparison of $N = 3$ and $N = 4$ as reported in Table 4.1 in terms of total GMRES iterations, total CPU time and relative CPU time plotted over the time step Δt. The computations are performed on 576 cores. 101

4.20 Comparison of $N = 6$ and $N = 8$ as reported in Table 4.1 in terms of total GMRES iterations, total CPU time and relative CPU time plotted over the time step Δt. The computations are performed on 72 cores. 102

4.21 Comparison of different freezing numbers n_{freeze} for $N = 5$ in terms of total GMRES iterations and relative CPU time plotted over CFL numbers for the three-dimensional Navier-Stokes equations. 104

4.22 $\mathcal{L}^2$ error of the density plotted over the CPU time required until $T = 1$. Isentropic inviscid traveling vortex on a 40×40 grid at $N = 5$, with adaptive tolerance and LU preconditioner with $n_{\text{freeze}} = 1$. 105

4.23 Parallel efficiency of a strong scaling of the explicit and implicit solver for different mesh sizes corresponding to each color. The baseline simulation is generally computed on a single node (24 cores) except for noPrecond and LU due to memory restrictions. The last symbol of each line is related to the case when each processor has only 9 elements. Every simulation is recomputed five times in order to obtain a statistical median. The deviations are represented by error bars. 109

4.24 Performance index of a strong scaling of the explicit and implicit solver for different preconditioners and different mesh sizes corresponding to each color. Here, the first symbol of each line is related to the case when each processor has only 9 elements. Every simulation is recomputed five times in order to obtain a statistical median. The deviations are represented by error bars. 110

5.1 Two-dimensional curved mesh around a circular cylinder for the laminar flow at $Re_D = 200$, detailed view. 117

5.2 Instantaneous flow field around a circular cylinder at $Re_D = 200$ colored by the velocity magnitude. 118

5.3 Drag coefficient C_D over non-dimensionalized time t^* for the explicit ERK4 and implicit ESDIRK4-6 schemes with given time step sizes. 121

5.4 Comparison of different freezing numbers n_{freeze} for the preconditioners in terms of total GMRES iterations and relative CPU time, corresponding to the time related to ESDIRK4-6 divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size $\Delta t^* = \Delta t u_\infty / D$. The computation of the cylinder flow at $Re_D = 200$ is performed for 20 time units on 24 cores. 122

5.5 Comparison of ESDIRK schemes with different order in terms of total GMRES iterations and relative CPU time, corresponding to the time related to the specific ESDIRK scheme divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size $\Delta t^* = \Delta t u_\infty / D$. The computation of the cylinder flow at $Re_D = 200$ is performed for 20 time units on 24 cores. 123

5.6 Comparison of a fixed or an adaptive GMRES tolerance η_k in terms of total GMRES iterations and relative CPU time, corresponding to the time related to ESDIRK4-6 divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size $\Delta t^* = \Delta t u_\infty / D$. The computation of the cylinder flow at $Re_D = 200$ is performed for 20 time units on 24 cores. 124

5.7 Time- and spanwise-averaged streamwise velocity at $Re_D = 3900$ and detailed grid view. 126

5.8 Instantaneous flow field around a circular cylinder at $Re_D = 3900$. Iso-surfaces, visualized by $\lambda_2 = -5$, and $x - y$ plane are colored by the velocity magnitude. 127

5.9 Mean streamwise velocity at different downstream points in the wake of the cylinder flow at $Re = 3900$ for DNS [16], explicit ERK4 and implicit ESDIRK (with various time step sizes Δt^*) within DGSEM. 130

5.10 Computational mesh in the $x-y$ plane with grid stretching in wall normal direction in order to resolve the viscous sublayer for the channel flow at $Re_\tau = 590$. 134

5.11 Snapshot of the uDNS with polynomial degree $N = 4$ of the channel flow at $Re_\tau = 590$. λ_2-isocontours and boundary slices are colored by the velocity magnitude. 135

5.12 Mean velocity profile and Reynolds stresses of the turbulent channel flow at $Re_D = 590$ for the ERK4 and ESDIRK4-6 with different CFL numbers compared to DNS data from Moser et al. [88]. Left column: $N = 3$, right column: $N = 4$. 137

5.13 Comparison of different freezing numbers n_{freeze} for the preconditioners in terms of total GMRES iterations and relative CPU time, corresponding to the time related to ESDIRK4-6 divided by the ERK4 time. Influence of LU and ILU(0) NoFillIn preconditioners and no preconditioning in dependency of the non-dimensionalized time step size. Solid lines: adaptive GMRES tolerance, dashed lines: $\eta_k = 0.1$. Left column: $N = 3$ on 48 cores, right column $N = 4$ on 240 cores. 139

Curriculum Vitae

15.03.1987	Geboren in Lörrach
1993 – 1997	Grundschule, Tannenkirch
1997 – 2006	Markgräfler Gymnasium Müllheim
2006	Allgemeine Hochschulreife
2006 – 2007	Auslandspraktikum
2007 – 2012	Studium der Mathematik mit Nebenfach Physik an der Universität Freiburg. Vertiefungsrichtungen: Numerik für Partielle Differentialgleichungen, Differentialgeometrie
2012 – 2019	Wissenschaftliche Mitarbeiterin am Institut für Aerodynamik und Gasdynamik der Universität Stuttgart
seit 2019	Softwareentwicklerin bei der Robert Bosch GmbH in Schwieberdingen

Stuttgart, den 20.12.2019 Serena Vangelatos