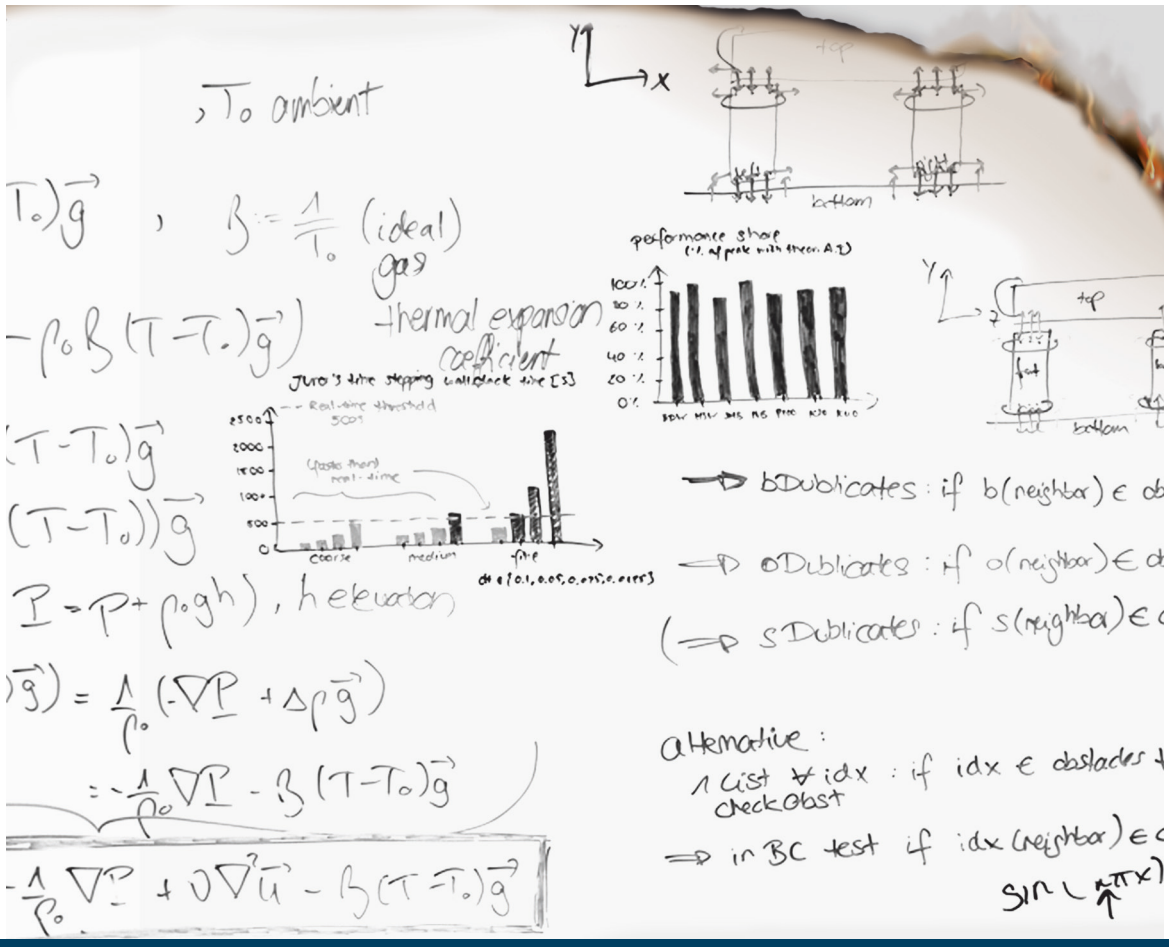


Real-Time Simulation and Prognosis of Smoke Propagation in Compartments Using a GPU

Anne Küsters

IAS Series
Band / Volume 39
ISBN 978-3-95806-379-2

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Schriften des Forschungszentrums Jülich
IAS Series Band / Volume 39

Forschungszentrum Jülich GmbH
Institute for Advanced Simulation (IAS)
Civil Safety Research (IAS-7)

Real-Time Simulation and Prognosis of Smoke Propagation in Compartments Using a GPU

Anne Küsters

Schriften des Forschungszentrums Jülich
IAS Series

Band / Volume 39

ISSN 1868-8489

ISBN 978-3-95806-379-2

Bibliografische Information der Deutschen Nationalbibliothek.
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte Bibliografische Daten
sind im Internet über http://dnb.d-nb.de abrufbar.

Herausgeber und Vertrieb:	Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag 52425 Jülich Tel.: +49 2461 61-5368 Fax: +49 2461 61-6103 zb-publikation@fz-juelich.de www.fz-juelich.de/zb
Umschlaggestaltung:	Grafische Medien, Forschungszentrum Jülich GmbH
Druck:	Grafische Medien, Forschungszentrum Jülich GmbH
Copyright:	Forschungszentrum Jülich 2018

Schriften des Forschungszentrums Jülich
IAS Series, Band / Volume 39

D 468 (Diss., Wuppertal, Univ., 2018)

ISSN 1868-8489
ISBN 978-3-95806-379-2

Persistent Identifier: urn:nbn:de:0001-2018121902

The complete volume is freely available on the Internet on the Jülicher Open Access Server (JuSER)
at www.fz-juelich.de/zb/openaccess

Acknowledgements and Dedication

This thesis summarizes my work from 2015 to 2018 at the Research Center Jülich and the National Cheng Kung University in Tainan, Taiwan.

I am grateful to all of those with whom I have had the pleasure to work with during this project. Special thanks go to my supervisors, Prof. Armin Seyfried, Dr. Lukas Arnold, and Jiri Kraus, who supported me throughout the work on this thesis. I would also like to thank my colleagues Alexander Belt, Marc Fehling, Gregor Jäger and Arne Graf for our fruitful discussions as well as My Linh Würzburger for her support. In addition, I highly appreciate the moral support of Alexander Belt and Jette Schumann, who became true friends. Together with the entire research group at the Research Center Jülich, you all made my time worthwhile.

In addition, the support of Jülich's Supercomputing Center (JSC) regarding scientific and personal training, administrative help as well as hardware support, cannot be emphasized enough. I would further like to thank Sandra Wienke from the IT Center of RWTH Aachen University for her expertise in performance analysis and granting computing time. I also gratefully acknowledge the scholarship for doctoral candidates granted by the German Academic Exchange Service (DAAD) for my stay in Taiwan. Thanks to Prof. Matthew Smith and his students for giving me a warm welcome at the National Cheng Kung University, for our discussions and for making my stay unforgettable in terms of culture, hospitality, and culinary delights.

Lastly, I dedicate this work to my husband, Dennis Küsters, who has been a constant source of support and encouragement during the challenges of life. I am truly thankful for having you in my life. This work is also dedicated to my parents, Gerhard and Annette Severt, who have always loved me unconditionally and whose good examples have taught me to work hard for the things I aspire to achieve. Danke Euch!

Abstract

The evaluation of life safety in buildings in case of fire is often based on smoke spread calculations. However, recent simulation models – in general, based on computational fluid dynamics – often require long execution times or high-performance computers to achieve simulation results in or faster than real-time.

Therefore, the objective of this study is the development of a concept for the real-time and prognosis simulation of smoke propagation in compartments using a graphics processing unit (GPU). The developed concept is summarized in an expandable open source software basis, called JuROr (*Jülich's Real-time simulation within ORPHEUS*). JuROr simulates buoyancy-driven, turbulent smoke spread based on a reduced modeling approach using finite differences and a Large Eddy Simulation turbulence model to solve the incompressible Navier-Stokes and energy equations. This reduced model is fully adapted to match the target hardware of highly parallel computer architectures. Thereby, the code is written in the object-oriented programming language C++ and the pragma-based programming model OpenACC. This model ensures to maintain a single source code, which can be executed in serial and parallel on various architectures.

Further, the study provides a proof of JuROr's concept to balance sufficient accuracy and practicality. First, the code was successfully verified using unit and (semi-) analytical tests. Then, the underlying model was validated by comparing the numerical results to the experimental results of scenarios relevant for fire protection. Thereby, verification and validation showed acceptable accuracy for JuROr's application. Lastly, the performance criteria of JuROr – being real-time and prognosis capable with comparable performance across various architectures – was successfully evaluated. Here, JuROr also showed high speedup results on a GPU and faster time-to-solution compared to the established Fire Dynamics Simulator. These results show JuROr's practicality.

Kurzfassung

Die Bewertung der Personensicherheit bei Feuer in Gebäuden basiert häufig auf Berechnungen zur Rauchausbreitung. Bisherige Simulationsmodelle – im Allgemeinen basierend auf numerischer Strömungsdynamik – erfordern jedoch lange Ausführungszeiten oder Hochleistungsrechner, um Simulationsergebnisse in und schneller als Echtzeit liefern zu können.

Daher ist das Ziel dieser Arbeit die Entwicklung eines Konzeptes für die Echtzeit- und Prognosesimulation der Rauchausbreitung in Gebäuden mit Hilfe eines Grafikprozessors (GPU). Zusammengefasst ist das entwickelte Konzept in einer erweiterbaren Open-Source-Software, genannt JuROr (*Jülich's Real-time Simulation in ORPHEUS*). JuROr simuliert die Ausbreitung von auftriebsgetriebenem, turbulentem Rauch basierend auf einem reduzierten Modellierungsansatz mit finiten Differenzen und einem Large Eddy Simulation Turbulenzmodell, um inkompressible Navier-Stokes und Energiegleichungen zu lösen. Das reduzierte Modell ist vollständig angepasst an hochparallele Computerarchitekturen. Dabei ist der Code implementiert mit C++ und OpenACC. Dies hat den Vorteil mit nur einem Quellcode verschiedenste serielle und parallele Ausführungen des Programms für unterschiedliche Architekturen erstellen zu können.

Die Studie liefert weiterhin einen Konzeptnachweis dafür, ausreichende Genauigkeit und Praktikabilität im Gleichgewicht zu halten. Zunächst wurde der Code erfolgreich mit Modul- und (semi-) analytischen Tests verifiziert. Dann wurde das zugrundeliegende Modell durch einen Vergleich der numerischen mit den experimentellen Ergebnissen für den Brandschutz relevanter Szenarien validiert. Die Verifizierung und Validierung zeigten dabei ausreichende Genauigkeit für JuROr. Zuletzt, wurden die Kriterien von JuROr – echtzeit- und prognosefähig zu sein mit vergleichbarer Leistung auf unterschiedlichsten Architekturen – erfolgreich geprüft. Zudem zeigte JuROr hohe Beschleunigungseffekte auf einer GPU und schnellere Lösungszeiten im Vergleich zum etablierten Fire Dynamics Simulator. Diese Ergebnisse zeigen JuROr's Praktikabilität.

Contents

1 Introduction **1**

1.1 Motivation 1
1.2 Challenges 3
1.3 State of the Art 4
1.3.1 Computational Resources 4
1.3.2 Integrated Tools in Fire Safety Engineering 6
1.3.3 Research Projects 8
1.4 Objectives, Limitations and Approach 12
1.5 Thesis Outline 14

2 Methodology **17**

2.1 Physical Processes of Smoke Propagation 17
2.1.1 Boundary Handling 21
2.1.2 Turbulence Modeling 23
2.1.3 Fire Modeling 29
2.2 Numerical Models of Fluid Dynamics 31
2.2.1 Numerical Grid 32
2.2.2 Discretization Method 35
2.2.3 Finite Difference Approximations 36
2.2.4 Solution Method 41
2.3 Uncertainties and Errors in CFD 60
2.3.1 Unacknowledged Errors 61
2.3.2 Acknowledged Errors 61

3 Simulation of Smoke Propagation on CPU **65**

3.1 Code Structure Using Interfaces 66
3.2 Extension to Complex Geometries in $3D$ 70
3.2.1 From $2D$ to $3D$ domains 70

3.2.2 Inner and Outer Boundary Handling 72
3.3 Intermediate Code Testing via Unit Tests 75
3.3.1 Unit Test Advection 75
3.3.2 Unit Test Diffusion 77
3.3.3 Unit Test Pressure 79

4 Verification and Validation 83
4.1 Verification of the Implementation 83
4.1.1 Analytical Verification Scenarios 84
4.1.2 Semi-Analytical Verification Scenarios 91
4.2 Validation of the Underlying Model 99
4.2.1 Fire Induced Flow Experiment in a Compartment 99
4.2.2 Open Plume Experiment Using Particle Image Velocimetry . . 105
4.3 Summary 115

5 Towards Real-Time and Prognosis Simulation Using a GPU 117
5.1 Porting to GPU Using OpenACC 118
5.1.1 Parallelization of the CPU Implementation 120
5.1.2 Optimization of the GPU Implementation 124
5.2 Performance Portability Analysis 127
5.2.1 Roofline Model 127
5.2.2 Measurement Setup 132
5.2.3 Theoretical and Measured Arithmetic Intensity 133
5.2.4 Performance Portability Results 133
5.3 Summary 136

6 Analysis of the Real-time and Prognosis Software 139
6.1 Speedup Analysis 139
6.2 Runtime Performance of Used Cases 143
6.2.1 Fire Induced Flow Experiment in a Compartment 143
6.2.2 Open Plume Experiment Using Particle Image Velocimetry . . 145
6.3 Real-Time and Prognosis Analysis 147
6.4 Summary 152

7 Closing Remarks 155
7.1 Conclusions 155

7.1.1 Societal Benefit and Value-add for Research Community 155
7.1.2 Main Contributions and Limitations 156
7.2 Outlook 159

Appendix A Compiling, Developing and Running the Software I

Appendix B Detailed Input Data of Test Cases XIX

Nomenclature XLII

Bibliography XLIII

List of Figures

1.1 Model of 'Osloer Straße' underground station in Berlin 2
1.2 Theoretical peak performance comparison of several GPUs and CPUs 5
1.3 Various work packages of this study 13

2.1 Schematic representation of scales in turbulent flows 23
2.2 Schematic comparison of DNS and LES 24
2.3 Schematic representation of spatial filtering 25
2.4 Combination of CFD and Fire Modeling 30
2.5 Examples of grid structures 33
2.6 Schematic representation of a structured regular collocated grid in $2D$ 34
2.7 Schematic comparison of variables located on a grid in $3D$ 34
2.8 Schematic representation of a spatial derivative and its approximations 37
2.9 Schematic representation of time and space discretization 39
2.10 Schematic comparison of explicit and implicit Euler methods 40
2.11 Schematic comparison of numerical and physical domains of dependency 44
2.12 Schematic representation of the Semi-Lagrangian method 48
2.13 Schematic representation of backtracing at the boundary 49
2.14 Schematic representation of matrix ordering 53
2.15 Schematic representation of pressure projection 55
2.16 Schematic representation of a V-cycle in the multigrid method 57
2.17 Schematic representation of restricting and prolonging cells in $3D$ . . 58

3.1 Class diagram of the CPU implementation using interfaces 67
3.2 Auxiliary and library classes for pre- and post-processing 68
3.3 Flow chart of an implemented fractional step solution method 69
3.4 Small-scale exemplary illustration of index lists 71
3.5 Small-scale exemplary illustration of checking duplicate indices 72
3.6 Small-scale exemplary illustration of dominant restriction in $2D$ . . . 74

3.7 Unit test for (linear) advection 76
3.8 Test for artificial diffusion 76
3.9 Artificially diffused temperature profile along the diagonal 77
3.10 Unit test for diffusion 78
3.11 Qualitative test for diffusion 79
3.12 Unit test for pressure 80

4.1 Illustrative approach to modeling, verification and validation 84
4.2 McDermott (2003) test case: horizontal velocity for 64×64 inner cells 85
4.3 McDermott (2003) test case: horizontal velocity for 512×512 inner cells 86
4.4 McDermott (2003) test case: horizontal velocity over time 87
4.5 McDermott (2003) test case: spatial convergence rate 87
4.6 Jouhaud (2010) test case: $2D$ vortex 89
4.7 Jouhaud (2010) test case: spatial convergence rate 89
4.8 Jouhaud (2010) test case: horizontal velocity profiles 90
4.9 Cavity flow test case: setup 92
4.10 Cavity flow test case: speed streamlines 93
4.11 Cavity flow test case: separation points 94
4.12 Cavity flow test case: velocity profiles 95
4.13 Flow around cube test case: setup 96
4.14 Flow around cube test case: speed streamlines 97
4.15 Flow around cube test case: reattachment point 98
4.16 Steckler et al. (1982)'s experiment N° 16: setup 100
4.17 Steckler et al. (1982)'s experiment N° 16: speed and temperature . . 102
4.18 Steckler et al. (1982)'s experiment N° 16: doorway profiles 103
4.19 Steckler et al. (1982)'s experiment N° 16: room profiles 104
4.20 Meunders (2016)'s PIV experiment: geometrical setup 106
4.21 Meunders (2016)'s PIV experiment: experimental setup 107
4.22 Meunders (2016)'s PIV experiment: speed contours, velocity profiles . 110
4.23 Meunders (2016)'s PIV experiment: temperature contours 111
4.24 Meunders (2016)'s PIV experiment: temperature profiles 112
4.25 Meunders (2016)'s PIV experiment: velocity for $T^{-}_{\mathrm{HS,\,surf,\,70\%}} = 185\,^{\circ}\mathrm{C}$ 113
4.26 Meunders (2016)'s PIV experiment: velocity as a function of Gr . . . 114

5.1 Illustrative comparison of a CPU chip and a GPU chip 117
5.2 Four stages of the APOD process 119
5.3 JuROr's runtime shares of fractional steps for various grid resolutions 122

5.4 Loop schedules when using NVIDIA architectures 123
5.5 Schematic OpenACC execution model 124
5.6 Pipelining visualized with NVIDIA's nvvp profiler using NVTX tracers 126
5.7 Roofline of an Intel Broadwell CPU for 4096×4096 inner cells 134
5.8 Roofline of an NVIDIA P100 GPU for 4096×4096 inner cells 134
5.9 Performance share of all considered architectures 135

6.1 McDermott (2003) test case: cell updates per second 141
6.2 Steckler et al. (1982)'s experiment N° 16: wallclock times 144
6.3 Meunders (2016)'s PIV experiment: corrected wallclock times 146
6.4 Real-time test case: tunnel setup with a volumetric heat source . . . 148
6.5 Real-time test case: accuracy analysis for coarse grid 148
6.6 Real-time test case: accuracy analysis for medium grid 149
6.7 Real-time test case: accuracy analysis for fine grid 150
6.8 Real-time test case: time stepping wallclock times 151

List of Tables

3.1 Five rules of the SOLID principle 66

4.1 McDermott (2003) test case: temporal convergence rate 88
4.2 Jouhaud (2010) test case: temporal convergence rate 91
4.3 Cavity flow test case: comparison of secondary vortices 94
4.4 Flow around cube test case: comparison of reattachment length . . . 98
4.5 Steckler et al. (1982)'s experiment N° 16: temperature corrections . . 100
4.6 Steckler et al. (1982)'s experiment N° 16: neutral plane location . . . 104
4.7 Steckler et al. (1982)'s experiment N° 16: thermal interface height . . 105

5.1 Used hardware architectures and compilers 128
5.2 Hardware specific floating-point performance and memory bandwidth 129
5.3 Performance counters: Flops 131
5.4 Performance counters: Bytes 131
5.5 Loop schedules for loop nests of the Jacobian stencil kernel 133
5.6 Theoretical and measured A.I. of the Jacobian stencil kernel 133
5.7 Flop/s, memory bandwidth and runtime measurement 135

6.1 McDermott (2003) test case: calculation time, CUPS and speedup . . 142
6.2 Steckler et al. (1982)'s experiment N° 16: wallclock times 144
6.3 Meunders (2016)'s PIV experiment: wallclock times 145
6.4 Meunders (2016)'s PIV experiment: corrected wallclock times 146
6.5 Real-time test case: JuROr's CUPS and real-time ratios 150
6.6 Real-time test case: FDS' wallclock times and real-time ratios 152

A.1 Minimum requirements to compile JuROr II
A.2 Various executables of JuROr IV

B.1 Advection test case: physical, numerical and solution parameters . . . XIX
B.2 Artificial diffusion test case: physical, numerical and solution parametersXX

B.3 Diffusion test case: physical, numerical and solution parameters . . . XXI
B.4 Diffusion hat test case: physical, numerical and solution parameters . XXII
B.5 Pressure test case: physical, numerical and solution parameters . . . XXIII
B.6 McDermott (2003) test case: physical and numerical parameters . . . XXIV
B.7 McDermott (2003) test case: solution method and parameters XXIV
B.8 Jouhaud (2010) test case: physical and numerical parameters XXV
B.9 Jouhaud (2010) test case: solution method and parameters XXV
B.10 Cavity flow test case: physical and numerical parameters XXVI
B.11 Cavity flow test case: solution method and parameters XXVI
B.12 Flow around cube test case: physical and numerical parameters . . . XXVII
B.13 Flow around cube test case: solution method and parameters XXVII
B.14 Steckler et al. (1982)'s experiment N° 16: physical parameters XXVIII
B.15 Steckler et al. (1982)'s experiment N° 16: physical, numerical parametersXXIX
B.16 Steckler et al. (1982)'s experiment N° 16: solution method, parameters XXX
B.17 Meunders (2016)'s PIV experiment: physical and numerical parameters XXXI
B.18 Meunders (2016)'s PIV experiment: solution method and parameters XXXII
B.19 Tunnel test case: physical and numerical parameters XXXIII
B.20 Tunnel test case: solution method and parameters XXXIV

List of Listings

2.1 Pseudocode of the V-Cycle multigrid method 58

3.1 $3D$ loops through the spatial domain 70
3.2 $1D$ row-major lists 71

5.1 OpenACC directives for $2D$ loops 122
5.2 OpenACC directives for $1D$ row-major lists 125
5.3 OpenACC's asynchronous clause 125

A.1 Cloning, building and running JuROr II
A.2 CMake options for building JuROr III
A.3 Example of compiling JuROr with a script IV
A.4 Example of PGI's compiler output V
A.5 Example of an `XML` input file: part I VIII
A.6 Example of an `XML` input file: part II IX
A.7 Example of an `XML` input file: part III X
A.8 Example of an `XML` input for a volumetric heat source XI
A.9 Example of an `XML` input file: part IV XII
A.10 Example of an `XML` input file: part V XIII
A.11 Example of an `XML` input file: part VI XIV
A.12 Example of using the script for building an `XML` file XVI
A.13 Example of using the script for starting an `XML` file XVII

Chapter 1

Introduction

"Yesterday is gone.
Tomorrow has not yet come.
We have only today.
Let us begin."

Mother Teresa

1.1 Motivation

Time is precious. Consequently, there exists a high desire for real-time and prognosis software in almost every field of application – be it in computer gaming, weather prediction, industrial applications in automotive, traffic or health management as well as civil safety for crowd-management or in cases of smoke and fire hazards.

In civil engineering, the consideration and evaluation of life safety in buildings in case of fire usually occur during planning, restoration or extension of buildings and building structures and are often based on smoke spread calculations. Thereby, complex buildings and building structures, like underground metro stations or modern architectures with new building materials, need individual evaluation, since existing descriptive requirements ensuring life safety (such as the German Model Building Code enabling escape, rescue and effective fire fighting measures) mostly cannot be consistently applied for such buildings. In these cases, real-time and prognosis simulations of smoke propagation in compartments could not only support engineers in the planning phase of a building but might also assist decision-makers responsible for direct emergency response. However, recent simulation models – in general, based on

computational fluid dynamics (CFD) – require long execution times.

Further, developments in fire safety such as legal requirements, building constructions, and technical specifications continuously advance. Additionally, modern computing architectures in High Performance Computing (HPC) and numerical methods, drive simulations utilized by designers and architectures to be state of the art with a fast time-to-solution. These developments are acknowledged and reconciled in the ORPHEUS (2018) project funded from 2015 to 2018 by the German Federal Ministry of Education and Research (BMBF) Program on 'Research for Civil Security - Protection and Rescue in complex Disaster Situations' (funding code 13N13266). The main objective was to optimize concepts for smoke control and evacuation based on experiments and simulations – especially for existing, complex infrastructures such as the underground station 'Osloer Straße' in Berlin, which was used as a target for real-scale experiments (cf. Fig. 1.1). The main objective was achieved in three steps. First, experiments of large-scale fires were conducted to assess the status-quo and to serve as a basis for concepts and the validation of simulations. Secondly, innovative, active and customized smoke extraction systems were conceptualized, and thirdly, communication strategies and state of the art CFD simulations capturing the status-quo were designed to support fire fighters and operators.

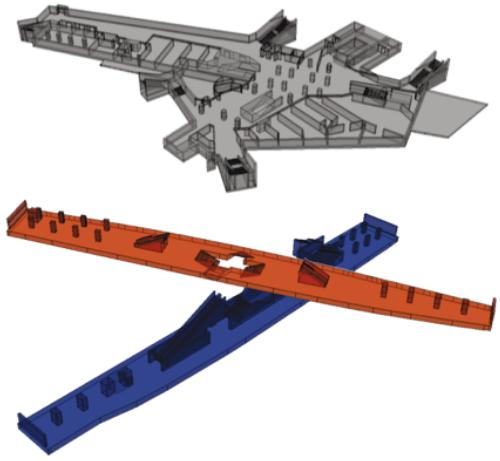


Figure 1.1: Model of the 'Osloer Straße' underground station in Berlin: the concourse 35 m above sea level (in gray) and the platforms U9 31 m above sea level (in red) and the U8 27 m above sea level (in blue), each roughly 110 m long and 20 m wide, ranging over three floors connected by staircases

Incidents such as the Jungangno Station fire in Daegu, South Korea in 2003 or, more recently, the fire at Grenfell Tower, in North Kensington, London, England in 2017[1], show the importance of a short reaction time and rescue measures of fire fighters and operators. Thereby, real-time or even faster than real-time executions of CFD models could capture the high-risk situation as status-quo and support effective fire fighting measures by a scenario-based adoption of fire fighting tactics. Additionally, (faster than) real-time simulations would allow for further application fields, where the predicted data may be used to steer technical systems like smoke extractions systems for danger prevention or dynamic escape routing for the benefit of self- and external rescue. Here, the decision-makers, such as the operational leadership of a fire brigade or police responsible for direct emergency response measures, and the fire fighters at site, would require practicable resources. This resource should be financially feasible and space-saving for the usage at site, while being able to calculate highly parallel computations in or faster than real-time as well as visualize the results.

These aspects motivate the study at hand to take up the challenge developing a concept for the real-time and prognosis simulation of smoke propagation in compartments, where the propagation of smoke is complex and hard to predict. Thereby, the developed concept is summarized in an expandable open-source software basis, called JuROr (*Jülich's Real-time simulation within ORPHEUS*). The focus of this study lies in providing a proof of JuROr's concept while handling the challenge of balancing practicality with sufficient accuracy.

1.2 Challenges

The balance of practicality and accuracy is the major challenge in this study.

Practicality ensures the real-time capability of JuROr and is regarded in terms of computational cost, performance portability and also acquisition costs of the computational architecture as well as its mobility. For the benefit of runtime, the modeling approach needs to be reduced in order to fully adapt the CFD model to highly parallel computer architectures. This design concept includes finite differences and highly parallelizable numerical models to match the target hardware. The choice of a suitable programming model, thereby, ensures performance portability, meaning to obtain a

[1]After the fire in the apartment of the Grenfell Tower was under control, the outside forces noticed that the fire was spreading across the facade. This spread was, so far, not considered with its speed. The flammable building's exterior cladding was an unknown uncertainty factor. The lack of knowledge about this possible course caused that the approached tactic, to stay inside the apartments, was not changed during the rescue. In total 72 people died.

comparable performance of the parallel computation across various architectures.

At the same time, sufficient accuracy is essential. In this study, sufficient accuracy is always regarded from the perspective of JuROr's purpose – creating a real-time system for smoke spread prediction to support fire fighters in cases of emergencies in complex infrastructures – where decisions are not based on simulation results with a resolution in terms of millimeters but rather multiple centimeters. To accurately predict smoke movement in buildings of any design, field modeling such as CFD is increasingly used in fire safety engineering. Field modeling allows for complex geometries, where traditional zone models are inaccurate. Nevertheless, complex geometries pose a challenge in computational fluid dynamics. Hence, the underlying computational grid (besides the temporal resolution) needs to be resolved with care ensuring sufficient accuracy for the application. Additionally, the accuracy needs to be continuously tested by verification and validation. To further increase the accuracy in future developments, a coupling layer for data assimilation from sensors could be added to the base code.

In order to find the appropriate balance between practicality and accuracy, several (research) projects have already been conducted using various computational resources.

1.3 State of the Art

1.3.1 Computational Resources

The increasing challenges and demands for computational power are driven by complex scientific simulations, such as CFD simulations, and lead to a performance growth of HPC systems (cf. Top500 (2017)). HPC thereby includes parallel computations over multiple compute elements, such as central processing units (CPUs) or graphics processing units (GPUs), with a fast network connecting the compute elements. Therefore, suitable programming models or interfaces such as Message Passing Interface (MPI) or Open Multi-Processing (OpenMP) are used on CPUs and, for instance, CUDA or OpenACC (for open accelerators) respectively on GPUs. To translate the code into an executable application, an appropriate compiler (e.g., from Intel, GNU, Cray or PGI) is needed. The application can then be executed on CPUs or accelerators such as GPUs.

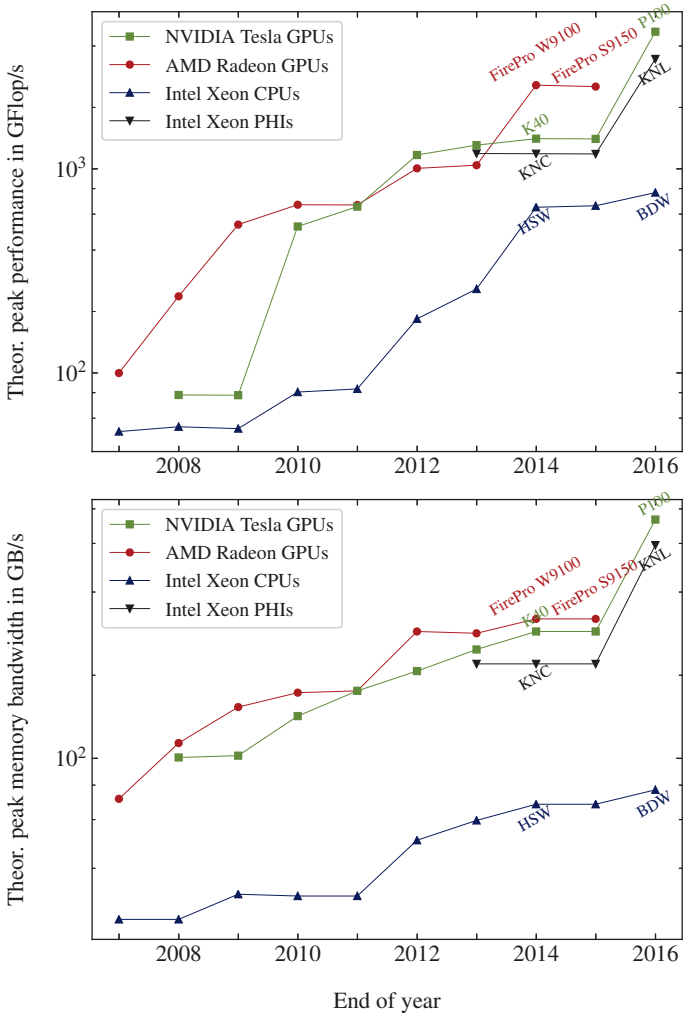


Figure 1.2: Theoretical peak performance comparison (top: floating-point operations, bottom: memory bandwidth) of NVIDIA and AMD GPUs as well as Intel CPUs and Phis over time, based on Rupp (2016)

The theoretical performance of such architectures in terms of floating-point operations per second (in Flop/s) and memory bandwidth (in GB/s) steadily increased over the last decade (cf. Rupp (2016)). Figure 1.2 shows the historical development of the theoretical peak performance of NVIDIA Tesla and AMD Radeon GPUs compared with Intel Xeon CPUs and Phis (top: in Flop/s, bottom: in GB/s). The Flop/s

development demonstrates a performance gap between GPUs and CPUs from 2010 to 2014. From 2014 on, the decision between CPU and GPU is however driven by the application's purpose, acquisition costs, ease of implementation, maintenance and mobility. Besides that GPUs are ideal for floating-point intensive applications, also memory-bound applications can nowadays be run on GPUs as the theoretical peak memory bandwidth at the bottom of Figure 1.2 shows. Here, a significant improvement of the GPU bandwidth performance can be observed in 2016.

On-demand resources such as clouds can overcome the necessity of the resource being mobile and space-saving. For instance, the recently developed tool, SparkCloud (cf. Garg et al. (2018)), simulates bush fires on a cloud on a first-come-first-serve basis, whereby the execution time of the simulation (in parallel on CPUs) is estimated and the request is completed within a given deadline. Nevertheless, cloud computing still reveals insurmountable challenges for High Performance Computing combined with the emergency character of the application at hand.

Thus for this study, the resource of a GPU is used for parallel computing, since GPUs are highly performant, affordable and mobile as well as space-saving. In detail, an NVIDIA Pascal P100 GPU (cf. Harris (2016)) was acquired with funding from the ORPHEUS (2018) project to develop, test and benchmark the real-time and prognosis capable CFD software JuROr predicting the smoke propagation in compartments.

1.3.2 Integrated Tools in Fire Safety Engineering

Since the desire for real-time simulation is high, there already exist tools for modeling fire and smoke and utilizing highly parallel computer architectures. Thereby, commercial and open source tools can be differentiated, which are already fully integrated into the fire safety community.

Commercial tools often are intended for general purpose computational fluid dynamics that can also be used for fire modeling applications. Examples thereof are ANSYS' CFX (cf. ANSYS (2013)) and FLUENT (cf. ANSYS (2015)) software, which are based on the finite volume method and include turbulence modeling, heat transfer, radiation and combustion modeling (and many more features). These products are parallelized for CPUs and (solely) NVIDIA GPUs (cf. NVIDIA Corporation (2018a)). Since explicitly NVIDIA products are utilized, CFX and FLUENT do not allow for the flexibility in computational architectures and therefore, do not assure the performance portability the study aims for. The same holds for CD-adapco's finite volume solver STAR-CCM+ (cf. Siemens PLM software (2017)), CHAM's PHOENICS software of

finite-volume type (cf. Ludwig (2011)), the finite element analysis software Autodesk CFD (cf. Autodesk Inc. (2017); Schnipke (1986)) and Solidworks' Flow Simulation applying the finite volume method (cf. Sobachkin and Dumnov (2013)). Regarding fire modelling, the FLACS (FLame ACcelerator Simulator)-Fire from Gexcon solving the compressible conservation equations for mass, momentum, enthalpy and mixture fraction using the finite volume method (cf. Gant and Hoyes (2010); Gexcon (2015)) and AVL-Fire, a thermo-fluid simulation software with implicit time and finite volume discretization (cf. AVL LIST GmbH (2014); RCPE (2018)) also lack the flexibility in using various computational architectures. All of these general purpose CFD tools are parallelized for CPUs, whereby GPUs are used for visualization instead of general purpose computation.

Specific for fire modeling is SOFIE (Simulation of Fires in Enclosures) developed by Cranfield University with the support of the Fire Research Station in England. It includes a finite volume procedure to solve the governing Navier-Stokes equations in a non-orthogonal coordinate system (cf. Melaaen (1990)). Further, SMARTFIRE, developed by the Fire Safety Engineering Group of the University of Greenwich (cf. Ewer et al. (2008, 2013)), is a finite volume CFD code which runs in parallel on CPUs using MPI and is additionally coupled to the free two-zone fire model CFAST of NIST to save computational time (cf. Ewer et al. (2010)). Again, these fire specific tools do not allow for the architectural flexibility and mobility the study intends. Being transparent regarding applied models and implemented methods as well as being of no cost, open source software is often used.

Open source examples are the widely applied Fire Dynamics Simulator (FDS) developed by NIST, which is based on the numerical model of finite differences with turbulence modeling. Conversely to JuROr, it uses methods yielding stability conditions on the time stepping size. Further, FDS runs in parallel on CPUs with the help of domain decomposition and MPI as well as OpenMP but does not run on GPU. Nonetheless, FDS' verification and validation cases (cf. McGrattan et al. (2017a,c)) and their results are (partly) used as a reference to verify, validate and compare JuROr's results with. Another example is FireFOAM, a package of the open source CFD software OpenFOAM (cf. The OpenFOAM Foundation (2017); Husted et al. (2017)). FireFOAM is a finite volume solver for unstructured grids with turbulence modeling running in parallel on CPUs with the interface OpenMP. Also based on finite volumes is Code_Saturne (cf. EDF Group (2017); Anzt et al. (2017)), which runs only partly on GPUs (status of 2017), but full GPU support is intended using libraries.

NIOSH's mine fire simulation program, MFIRE, has also been used in tunnel fire modeling (cf. NIOSH (2016); Zhou et al. (2016)) and describes a $1D$ empirical model based on observations and experiments in contrary to more accurate CFD models based on physical laws.

None of the above mentioned commercial and open source tools are eligible for achieving the study's objectives due to the lack of performance portability across various architectures and missing mobility.

1.3.3 Research Projects

Also, several research projects are concerned with the issue of real-time simulation of fire or smoke propagation, accuracy improvement by sensor coupling and performance portability.

Fire and smoke propagation modeling in real-time is also handled in civil safety research. For instance, Glimberg (2009) studied the real-time simulation of CFD models describing smoke propagation. In the joined work of Glimberg et al. (2009), GPUs were employed to solve the governing equations in simplified geometries using a fractional step method. This approach resulted in a solution within less than a minute of runtime for ten seconds of simulation time. For comparison, the simulation took more than one hour on a CPU. Hejn and Rosenkvist (2009) continued the work by employing multiple GPUs. Nevertheless, the applied programming model CUDA exclusively runs on NVIDIA GPUs and therefore does not allow for the architectural flexibility this study is interested in. The same holds for the second-order finite volume code by Cohen and Molemaker (2009) with an eight-time speedup of the CUDA-accelerated code versus the Fortran code on an eight-core CPU. Limiting the architectural flexibility is also the approach of Liu et al. (2004), who implemented the code (including obstacles and complex boundary conditions) with assembly language which is specific to a particular computer architecture. Other projects also show interest in producing real-time predictions, like those investigated in the FireGrid project by Han et al. (2010). However, the utilized fire simulation model in FireGrid is a zone model, which splits the domain of interest into very few zones (cf. Koo (2010)). Properties like temperature or smoke density are computed via a set of coupled ordinary differential equations and thus only allow for very crude approximations in contrary to CFD models. Further, zone modeling limits the applications to simple geometries.

A different approach to saving computational time is adaptive mesh refinement. The CFD software JuFIRE, developed by Drzycimski and Arnold (2015) and refined by Fehling et al. (2017) within the ORPHEUS (2018) project, simulates the smoke propagation based on the finite element method and assures fast calculation by adaptive mesh refinement in parallel on CPUs using MPI. Thereby, the adaptive mesh refinement allows for a flexible determination of the local grid resolution using refinement criteria and therefore fewer grid cells need to be updated compared to a fixed resolution resulting in speeding up the calculation. At the same time of writing this thesis, JuFIRE was still under development and needed to be validated.

Although achieving flexibility in computing architectures by providing a linear algebra framework on GPUs using OpenCL (, yet not in one source code), Krüger (2006) lacks the integration of Navier-Stokes based smoke, fire and fluid effects in his software. The same holds for the CFD-based Zonal Flow Solver ZFS developed by the Institute of Aerodynamics at RWTH Aachen (cf. Lintermann and Schröder (2018)) featuring (besides others) a Lattice Boltzmann and finite volume solver with MPI and OpenMP parallelization as well as GPU portation via OpenACC (cf. Kraus and Schlottke (2014)). Nonetheless, the coupling to buoyancy forces due to natural convection is again missing.

In conclusion, the existing research projects can only build a methodological basis for the development of JuROr, since none of them provides architectural flexibility, mobility or the accuracy the study's purpose requests.

Performance Portability across various computational architectures is the main driver for implementing JuROr with OpenACC. Although the architecture-specific assembler optimization by OpenACC compilers eases the maintenance of a single code base (e.g., over using OpenCL, Pennycook et al. (2013)), the performance portability of OpenACC implementations as analyzed in Küsters et al. (2017) have been scarcely studied so far. While OpenACC performance comparisons across different architectures have been targeted in research, they have mostly been conducted by analyzing absolute numbers such as the application's runtime, floating point operations per second or speedup over CPU runs. For example, Lopez et al. (2016) show memory bandwidth or speedup numbers for a Jacobi and n-body kernel for different OpenACC implementations (PGI, Cray) on NVIDIA Kepler GPUs. They failed to use OpenACC on multicore CPUs. Sabne et al. (2014) evaluated the performance by showing speedup numbers based on OpenARC's OpenACC implementation on NVIDIA GPUs, AMD GPUs and Intel Xeon Phi coprocessors using 12 kernels. The

hydrodynamic mini-app CloverLeaf by Herdman et al. (2014) has been tested on NVIDIA GPUs, Intel Xeon Phi Coprocessors, one AMD APP and different CPUs using the vendor OpenACC implementations from CAPS, PGI and Cray. For real-world codes, Nicolini et al. (2016) present runtimes of an aeroacoustic simulation software package using PGI's OpenACC implementation for NVIDIA Kepler GPUs and Intel CPUs. Calore et al. (2016) investigate a lattice Boltzmann application also using PGI's OpenACC implementation on NVIDIA GPUs, AMD GPU, and Intel CPU. However, performance portability investigations should not only consider absolute numbers but need to account for the hardware's and application's characteristics. For that, studies of Sabne et al. (2014); Herdman et al. (2014); Calore et al. (2016) compare their gained OpenACC performance to hand-tuned low-level implementations written in CUDA or OpenCL, or to libraries like MKL or CUBLAS. As a percentage of peak performance, Lopez et al. (2016) present their DAXPY and DGEMV kernels. For two non-trivial kernels, Calore et al. (2016) show an OpenACC efficiency of 54 % to 70 % of peak across different architectures for memory-bound code, while compute-bound code achieves 14 % to 24 % efficiency.Modeling OpenACC performance using a roofline model has only been conducted by Wang et al. (2013), who base their model on STREAM and Flop/s measurements and apply CAPS' OpenACC implementation to NVIDIA GPUs and Intel Xeon Phi coprocessors. However, they only examine basic kernels from the EPCC OpenACC Benchmark Suite. There, they get up to 82 % of sustained performance compared to peak performance of specific benchmarks, whereas more benchmarks only achieve about 10 % performance on average. In contrast, the performance portability analysis in this study does not only focus on absolute performance but especially applies the roofline model to the real-world code JuROr.

In summary, JuROr distinguishes itself from the named projects, tools and approaches by predicting the smoke propagation in compartments, deploying computational fluid dynamics while focusing on the real-time simulation, being performance portable on a wide range of computational architectures and measuring its performance relative to the hardware's characteristics via a roofline model.

Data assimilation is recently applied in the field of fire modeling, where data is coupled to an underlying fire model to improve accuracy. For instance, FirstCast3.0 by Hamins et al. (2014, 2015) is based on creating, storing, exchanging, analyzing, and integrating information from a wide range of databases and sensor networks to assign every building the New York City Fire Department inspects with a fire

risk score. Their integration of computer-based fire modeling with sensor technology uses existing approaches such as inverse zone modeling of enclosure fire dynamics by Cowlard et al. (2010); Jahn et al. (2009); Jahn (2010); Jahn et al. (2012); Jahn (2017) and ensemble-based data assimilation to predict wildfire spread of Rochoux et al. (2013a,b, 2014, 2015).

Daniel and Rein (2016) implemented the FireNavigator forecasting the spread of building fires. Using the techniques of a cellular automata building fire model (instead of CFD), they employed sensor data assimilation, inverse modeling, and genetic algorithm techniques. With this approach, the governing parameters of a fire, such as the flame spread rate, the smoke ceiling jet velocity and the outbreak location and time, can be indirectly uncovered and then used to produce real-time as well as forecast maps of the flame spread and smoke propagation. Therewith, the Fire Navigator achieves positive lead times of several minutes meaning the predictions are actual forecasts (without the usage of GPUs). Nonetheless, cellular automata simulations simplify the problem and do not produce results as accurate as CFD models.

Beata et al. (2018) provide a computing framework for the integration of real-time fire data with computation and visualization. Components of the real-time fire monitoring system include a sensor simulator, the main program coordinating data to sub-models, and a real-time sub-model for event-detection evaluating the progression of fire events (in the post-ignition state of a building). The event-detection is deterministically modeled from sensor data using thresholds based on literature and experimental data as opposed to simulating or forecasting the fire or to applying empirical methods. If a threshold has been reached, a warning level is assigned for each hazard. Visualization of an aggregated threat level (in color coding) is then performed using the Building Information Model (BIM) as a post-processing feature.

Data coupling is also applied in other fields, such as crowd-management. For instance, the aim of the HERMES project (cf. Kemloh et al. (2013)) was to improve the life safety in large buildings and at big events by applying an evacuation assistant. This assistant uses available data about the distribution of people and the availability of escape routes. Based on these data, a parallel computer program will then predict the movement of all people present, thereby providing immediate simulation results for crowd management. For future developments of JuROr, a potential approach to add a coupling layer would be based on these existing approaches for data assimilation from sensors.

In line with the discussed research findings, motivation and challenges, the study's

objectives and the strategic approach for reaching these are defined below.

1.4 Objectives, Limitations and Approach

The main objective of this thesis is to design and implement a CFD code base, called JuROr, simulating turbulent smoke spread thermodynamically driven by buoyancy in compartments making use of a GPU to be real-time and prognosis capable. A proof of concept thereof shall be given, first, by successfully verifying the code using unit, analytical and semi-analytical tests, secondly, by validating the underlying physical model with scenarios relevant for fire protection, and lastly, by showing its real-time and prognosis capability. Further, JuROr's speedup using a GPU should be benchmarked against its serial and multicore execution on CPUs as well as against the performance of the commonly used fire modeling software FDS. Motivated by the use of JuROr as a decision tool to support fire fighters with a prognosis of smoke propagation, JuROr must be performance portable on CPU and GPU as well as contained in a single, open source code.

Various aspects could not be integrated and are left to future development. First and foremost, a comprehensive test suite for the verification and validation of JuROr to evaluate and quantify its accuracy cannot be build up in the shortage of time. Further, the sensitivity and uncertainty of applied parameters cannot be assessed in detail. For the same reason, JuROr's real-time and prognosis capability cannot be thoroughly tested. Therefore, the suggested test suite could be reused for a comprehensive evaluation of JuROr's performance regarding its real-time and prognosis capability. Additionally, the mentioned coupling layer in order to assimilate data from sensors using JuROr as prediction software lies not within the scope of this study. Since JuROr is intended to serve as a fresh code base, the scope of the application is limited. To widen its scope, static data such as the geometry with openings and obstacles or initial conditions should be automatically imported in the future. Finally, in order to serve as a support tool, a graphical user interface must be added in future allowing for dynamically steering the simulation and simultaneously visualizing its results whilst the simulation is running.

Based on the specified objectives, challenges and limitations discussed throughout this chapter, the following work packages (cf. Fig. 1.3) are pursued:

- WP 1: Specification of physical processes relevant for smoke propagation, the mathematical modeling and numerical discretization thereof matched to the target hardware of a GPU

- WP 2: CPU implementation of the solver and visualization of results for simple $2D$ geometries using the object-oriented programming language C++ and providing a (one source) code structure, which can be further developed by easily adding or interchanging numerical methods and models
- WP 3: Extension to more complex $3D$ rooms or buildings with inner boundaries and appropriate boundary conditions
- WP 4: Verification of the code using $2D$ unit, analytical and semi-analytical tests (throughout the implementation) as well as validation of the underlying model with $3D$ scenarios relevant for fire protection to show sufficient accuracy of the software
- WP 5: Porting of the solver to GPU using the pragma-based programming model OpenACC and analysis thereof regarding its performance portability across various architectures in $2D$
- WP 6: Analysis of the software's performance regarding speedup of the GPU version compared to its serial and multicore performance on CPUs (in $2D$) as well as against FDS' runtime performance and assessment of JuROr's real-time and prognosis capability based on a scenario relevant for fire protection (in $3D$)

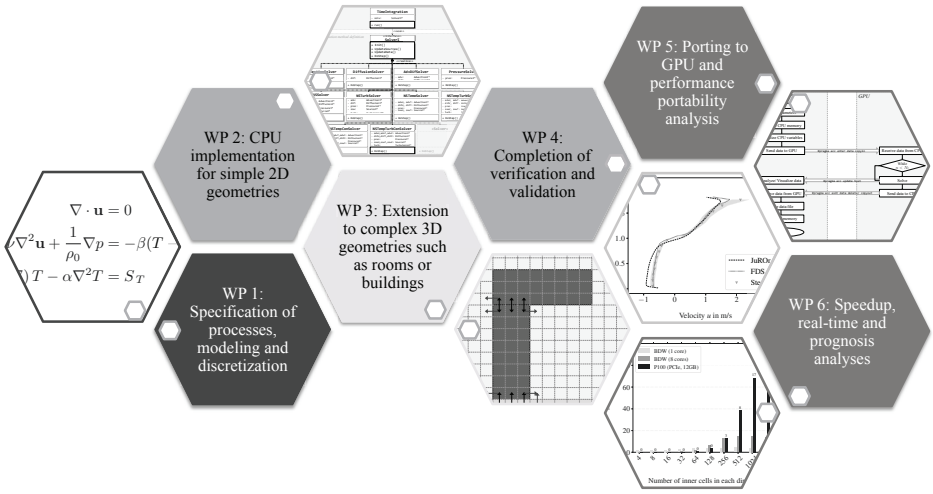


Figure 1.3: Various work packages based on the objectives, challenges and limitations of this study

Basically following the outlined approach, the thesis is structured as follows.

1.5 Thesis Outline

The thesis is divided into seven chapters. Chapter 1 already addressed the motivation, challenges, and limitations of this study and reviewed the current state of the art regarding integrated tools and research projects in fire modeling, performance portability analysis, and data assimilation. On this basis, the study's objectives and the approach to achieving those were outlined.

Chapter 2 sets up the methodological frame of the resulting concept of JuROr. First, the processes relevant for smoke propagation are physically and mathematically modeled. In the next instance, the numerical discretization of these models is chosen to be fully adapted to highly parallel computer architectures with the deliberate choice of a reduced modeling approach.

Based on this modeling approach, the software design concept for CPUs as well as its implementation is described in Chapter 3. The design concept integrates interfaces to ensure an abstract, single purpose and expandable code base. Further, the extension from $2D$ to $3D$ domains including obstacles as well as inner and outer boundary handling are presented. Constructing the obstacles, the adaption to highly parallel computer architectures is already kept in mind. During the implementation using the approach of fractional steps for the numerical solver, unit testing is applied.

The intermediate and continuous verification of the code as well as the validation of the underlying model are summarized in Chapter 4. Here, the code is verified using (semi-) analytical test cases and the model is validated by comparing JuROr's numerical results with experimental outcomes as well as the results of the fire simulation software FDS based on two scenarios relevant for fire protection.

Porting of the successfully verified and validated CPU code to a GPU-accelerated code is addressed in Chapter 5. The transfer includes a step-by-step parallelization of the CPU code using the pragma-based programming model OpenACC to maintain a single source code and the optimization of the arising GPU version. The performance portability across various computational architectures using a roofline model is thereafter analyzed based on the $2D$ GPU implementation of JuROr.

In Chapter 6 further analyses of the GPU-ported software are conducted. First, JuROr's speedup is evaluated by comparing the cell updates per second achieved when deploying a GPU against a serial and parallel (multicore) CPU execution of a verification test case. Then, the validation test cases are revisited to assess JuROr's runtime performance while maintaining sufficient accuracy. Additionally, its runtime is compared with FDS' time-to-solution. Lastly, JuROr's real-time and prognosis

capability is demonstrated on another scenario relevant for fire protection.

To conclude this thesis, Chapter 7 provides summarizing remarks on the contributions and limitations of this study and proposes improvements and developments for future research.

Now to begin, the methodology, on which the CFD simulation of smoke propagation is based, is presented in detail in the following chapter.

Chapter 2

Methodology

In order to simulate smoke propagation, first, the physical processes thereof are distinguished leading to governing mathematical equations. Then, the solution of these governing equations is numerically approximated by methods suitable for parallelization on GPUs in order to achieve real-time and prognosis capability for JuROr. Within this approach, from defining physical processes over deriving mathematical equations to solving them via numerical approximation, uncertainties and errors of different sorts can arise and are therefore introduced.

2.1 Physical Processes of Smoke Propagation

In physics, the transport of a fluid can be derived from the conservation laws of mass, momentum, and energy:

i Mass is neither created nor destroyed.

ii The rate of change of momentum of a body is directly proportional to the force applied to it.

iii Energy is neither created nor destroyed.

Therefore, let $\mathbf{x} = (x, y, z)^\top$ be a point in a region filled with fluid, and let $\mathbf{u}(\mathbf{x}, t)$ be the velocity of an element of fluid moving through $\mathbf{x}$ at time t. Assume that the fluid has a well-defined mass density $\rho(\mathbf{x}, t)$ for each time t and space $\mathbf{x}$.

i *Continuity equation*

The conservation of mass can then be more precisely described as follows: The rate of increase of mass in a region equals the rate at which the mass is passing the region's boundary, i.e.,

$$\int_W \partial_t \rho + \nabla \cdot (\rho \mathbf{u})\, dV = 0 \tag{2.1}$$

$$\iff \partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0\,, \quad \text{for any region } W. \tag{2.2}$$

ii *Momentum conservation*

By Newton's second law, mass × acceleration = force, it holds that

$$\rho(\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u}) = -\nabla p + \nabla \cdot (\sigma + \tau) + \rho \mathbf{g}\,, \tag{2.3}$$

where the forces consist of gravitational volumetric body forces $\mathbf{f}_B = \rho\mathbf{g}$ as well as surface forces $\mathbf{f}_{\text{surf}}$ (per area). The surface forces are due to fluid pressure, p, and viscous stresses resolved into normal stresses, $\sigma = -\frac{2}{3}\mu(\nabla \cdot \mathbf{u})\mathbf{I}$, and (parallel) shear stresses, $\tau = \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^\top)$ for a Newtonian fluid with viscosity μ.

iii *Energy conservation*

The temperature field $T(\mathbf{x}, t)$ inside the fluid is obtained by applying the First Law of Thermodynamics,

$$\begin{pmatrix}\text{rate of increase}\\ \text{of energy}\\ \text{inside a region}\end{pmatrix} = \begin{pmatrix}\text{net rate of energy flow}\\ \text{into the region by}\\ \text{bulk fluid motion}\end{pmatrix} + \begin{pmatrix}\text{flow of heat through}\\ \text{the surface by}\\ \text{conduction}\end{pmatrix} + \begin{pmatrix}\text{rate of work done by}\\ \text{body \& surface forces}\end{pmatrix} + \begin{pmatrix}\text{energy generated}\\ \text{inside the region}\end{pmatrix},$$

yielding

$$\rho c_p(\partial_t T + (\mathbf{u} \cdot \nabla)T) = \nabla(k\nabla T) + (\partial_t p + (\mathbf{u} \cdot \nabla)p) + \mathbf{\Phi}\,. \tag{2.4}$$

Here, c_p describes the specific heat capacity at constant pressure, k denotes thermal conductivity and $\mathbf{\Phi} = 2\mu((\partial_x u)^2 + (\partial_y v)^2 + (\partial_z w)^2) + \mu((\partial_x v + \partial_y u)^2 + (\partial_y w + \partial_z v)^2 + (\partial_z u + \partial_x w)^2) - 2/3\mu(\nabla \cdot \mathbf{u})^2$ is called viscous dissipation.

Detailed mathematical derivations and further insights can be found in Chorin and Marsden (1979); Ferziger and Peric (2002); McDonough (2007) for fluid flow in general and in Yeoh and Yuen (2009); Quintiere (2006); Drysdale (1999); Hurley et al. (2016) within the specific context of fire-related flow.

Simplifying Assumptions: In order to solve the above set of coupled nonlinear partial differential equations (2.2) - (2.4), several simplifying assumptions are applied:

1. *Low velocity*: The flow velocity is assumed to be low i.e., $Ma := u/c < 1/3$, whereby the dimensionless Mach number relates the local flow velocity u to the local speed of sound c. This assumption yields several approximations for incompressible flow:

2. *Incompressible flow*: Fluid mass density $\rho(\mathbf{x}, t) = \rho_0$ is assumed to be constant – resulting in incompressibility $\nabla \cdot \mathbf{u} = 0$, and, therefore, $\sigma = 0$ and $\tau = \mu\nabla^2\mathbf{u}$ – except in the buoyancy force $\mathbf{f}_B = \rho(T)\mathbf{g}$ (called *Boussinesq approximation*). For simplicity reasons, the reference density ρ_0 is set to unity.

3. *Small changes of density*: For the buoyancy force it is assumed that the changes of density, $\Delta\rho = \rho - \rho_0$, as a function of temperature are small, $|\Delta\rho| \ll \rho_0$, and are linearly dependent on temperature, $\Delta\rho = -\rho_0\beta\Delta T$, with thermal expansion coefficient β of an ideal gas with $pM = \rho RT$, where $R \approx 8.134\,\mathrm{J/(mol\,K)}$ denotes the universal gas constant and M the molar mass of the gas.[1]

4. *Constant properties*: The specific heat capacity c_p, thermal conductivity k and the viscosity μ are constant.

5. *Isobaric condition*: The pressure in the thermodynamic process is assumed to be constant ($\Delta p = 0$).

6. *Zero viscous dissipation:* Since the viscous dissipation is of second-order in terms of the Mach number, the contribution of viscous dissipation can be ignored in the energy equation, i.e., $\mathbf{\Phi} = 0$ (cf. Hurley et al. (2016)).

[1] The assumption of an ideal gas also allows to write the energy equation in terms of temperature derived from the thermodynamic definition of specific enthalpy $h := e + p/\rho$ with specific internal energy e and $dh = c_p dT$ for an infinitesimal process with $d(\cdot)$ denoting infinitesimal change.

These assumptions lead to the following simplified equations of conservation

$$\nabla \cdot \mathbf{u} = 0 \tag{2.5}$$

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho_0} \nabla p = \frac{1}{\rho_0} \mathbf{f}_B(T) \tag{2.6}$$

$$\partial_t T + (\mathbf{u} \cdot \nabla)\,T - \alpha \nabla^2 T \qquad = S_T\,, \tag{2.7}$$

where $\nu = \frac{\mu}{\rho_0}$ describes the kinematic viscosity, $\alpha = \frac{k}{\rho_0 c_p}$ the thermal diffusivity and S_T denotes an external energy source, which is further defined in Section 2.1.3 for fire specific flow.

Now, in order to solve the governing equations the buoyancy force $\mathbf{f}_B = \rho(T)\mathbf{g}$ needs to be further detailed. Assuming that the changes of density as a function of temperature are small (i.e., $|\Delta\rho| = |\rho - \rho_0| \ll \rho_0$), the force density $\mathbf{f}_B(T)$ can be written as

$$\frac{1}{\rho_0}\mathbf{f}_B(T) = \frac{1}{\rho_0}\rho(T)\mathbf{g} = \frac{1}{\rho_0}(\rho_0 + \Delta\rho)\mathbf{g}\,.$$

With a linear dependence on temperature, i.e., $\Delta\rho = -\rho_0 \beta \Delta T$, it holds that

$$\frac{1}{\rho_0}\mathbf{f}_B(T) = (1 - \beta(T - T_0))\mathbf{g}\,, \tag{2.8}$$

where $\beta = \frac{1}{T_0}$ denotes the thermal expansion coefficient of an ideal gas with ambient temperature T_0. To avoid potential round-off errors in the calculation of the forces in the momentum equations (2.6), a pressure shift is introduced, meaning p is replaced by $p = P - \rho_0 g h$ (dynamic and hydrostatic pressure) with elevation h (by Bernoulli's principle). This shift results in

$$\begin{aligned} &\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho_0} \nabla p = \frac{1}{\rho_0} \mathbf{f}_B(T) \\ \iff\ &\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho_0} \nabla P = -\beta(T - T_0)\mathbf{g}\,, \end{aligned}$$

using (2.8). Often the dynamic pressure P is again renamed into p.

In summary, smoke propagation and heat transfer can be mathematically described with the incompressible Navier-Stokes and energy equations

$$\nabla \cdot \mathbf{u} = 0 \tag{2.9}$$

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\, \mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho_0} \nabla p = -\beta (T - T_0) \mathbf{g} \tag{2.10}$$

$$\partial_t T + (\mathbf{u} \cdot \nabla)\, T - \alpha \nabla^2 T \qquad = S_T \,, \tag{2.11}$$

for a gas with velocity $\mathbf{u}$, pressure p and temperature T.

In (2.10) and (2.11), the *diffusion* terms $\nu \nabla^2 \mathbf{u}$ and $\alpha \nabla^2 T$ describe a process that drives a balancing of differences in the flow properties (velocity and temperature, respectively). The *advection* terms, $(\mathbf{u} \cdot \nabla)\mathbf{u}$ and $(\mathbf{u} \cdot \nabla)T$, describe the transport of the properties due to the flow. The advection of velocity is further called *convection* which is the fluid's intrinsic movement.

In case of various species, their concentration can be modeled as a passive scalar C by

$$\partial_t C + (\mathbf{u} \cdot \nabla)\, C - D \nabla^2 C = S_C \,, \tag{2.12}$$

where D is the mass diffusivity and S_C denotes an external source, which is further defined in Section 2.1.3 for fire specific flow. Equation (2.12) states that the rate of change of a scalar property equals the net rate of the scalar property added plus the rate of creation or destruction caused by an external source.

2.1.1 Boundary Handling

In order to solve Equations (2.9) - (2.12) in a distinct space (and time) and therefore close the system mathematically, appropriate initial conditions (at time $t = 0$) and boundary conditions need to be specified. In general, the value of the variable at the boundary (i.e., *Dirichlet boundary condition*), its gradient normal to the boundary (i.e., *Neumann boundary condition*) or a linear combination of the two conditions is given. Examples of the Dirichlet boundary condition for velocities are the *no-slip* and *inflow* condition, whereas the *outflow* condition is an example of the Neumann boundary condition for velocities. Based on Yeoh and Yuen (2009), the boundary conditions used here are defined as follows:

Definition 2.1.1 (No-slip boundary condition)**.** The solid surface has zero relative velocity between the surface and the fluid right at the surface. Assuming the surface

is stationary, all the velocity components can be set to be zero, i.e.,

$$u = v = w = 0 \quad \text{at the solid wall}\,. \tag{2.13}$$

Definition 2.1.2 (Inflow boundary condition)**.** At inflow boundaries, at least one velocity component needs to be prescribed, i.e.,

$$\mathbf{u} = \mathbf{U} \quad \text{at the inflow boundary}\,, \tag{2.14}$$

where $\mathbf{U}$ can be specified as a vector of constant values with at least one component unequal to zero.

Definition 2.1.3 (Zero-gradient boundary condition)**.** If the shear forces along the surface are taken to be zero, the outflow condition is provided by

$$\partial_n u = \partial_n v = \partial_n w = 0 \quad \text{at the outflow boundary}\,, \tag{2.15}$$

where n is the direction normal to the surface.

Definition 2.1.4 ((Zero-gradient) pressure boundary condition)**.** When setting the pressure boundary condition, the derivative of the pressure in normal direction to the open boundary is set, for instance,

$$\partial_n p = 0\,. \tag{2.16}$$

Here, caution must be given combining boundary conditions of velocity (at inflow/ outflow) and pressure.

Definition 2.1.5 (Wall temperature boundary condition)**.** For the temperature at a wall surface, the Dirichlet boundary condition can be used. If the material at the wall surface has temperature T_w, then the temperature fluid layer directly in contact with the wall should also be T_w, i.e.,

$$T = T_w \quad \text{at the wall}\,. \tag{2.17}$$

Definition 2.1.6 (Adiabatic surface boundary condition)**.** If a perfectly insulated surface needs to be assured (thus, no flux shall flow through the wall), the normal derivative for temperature is set to zero

$$\partial_n T = 0 \quad \text{at the surface}\,. \tag{2.18}$$

Prescribing a non-zero value means prescribing a certain constant heat flux c in K/m through the surface, i.e.,

$$\partial_n T = c \quad \text{at the surface}\,. \tag{2.19}$$

For the work at hand, there exists no explicit boundary layer model for the heat transfer from solids into the fluid (or from the fluid into the solid) other than setting the wall temperature.

Definition 2.1.7 (Periodic boundary condition)**.** If the geometry and the expected flow pattern are of a periodically repeating nature, so-called *periodic* boundary conditions are used. Thus, the transport property of one of the surfaces, ϕ_1, is taken to be equal to the transport property ϕ_2 of the second surface depending on which two surfaces experience periodicity, i.e.,

$$\phi_1 = \phi_2\,. \tag{2.20}$$

2.1.2 Turbulence Modeling

Turbulence is induced by shear flows as introduced in Section 2.1 and was first characterized by Reynolds (1883). Due to the existence of random fluctuations in the fluid (called *turbulence*) additional complexities arise in modeling turbulent flow since these fluctuations occur in different spatial scales from being large to very small (cf. Fig. 2.1). In order to predict turbulent flows there exist various approaches. Two of them are the *large eddy simulation* (LES) and the *direct numerical simulation* (DNS).

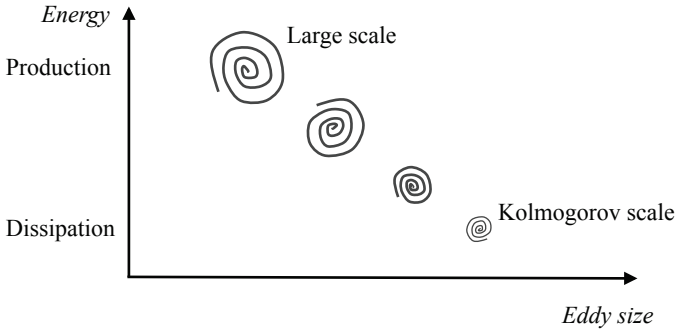
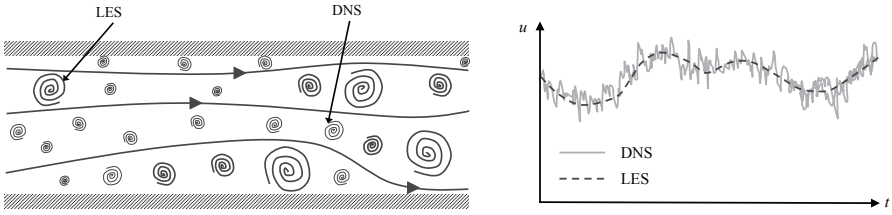


Figure 2.1: Schematic representation of scales in turbulent flows

Direct numerical simulation: In DNS the Navier-Stokes equations are solved without modeling turbulence. The scale of the largest eddy can be measured by the *large scale* L. Further, the simulation must also capture the smallest scales, on which the viscosity dominates, in order to assure that all of the significant structures of the turbulent flow are captured. Thus the numerical grid spacing must be no larger than the so-called *Kolmogorov scale*, $\eta \approx (\nu^3 L/U^3)^{1/4}$, with characteristic length L and characteristic velocity U. Since for highly turbulent flows the Kolmogorov scale is very small (in the micrometer range) and the scales are wide apart, these requirements on the resolution would make most real-world simulations not practical, being too expensive to calculate. What DNS is beneficial for are its detailed results which may be regarded as the equivalent of experimental data and thus can be used for validation where no experimental data exist.



(a) Schematic representation of turbulent motion

(b) Time dependence of a velocity component at a point in the flow

Figure 2.2: Schematic comparison of DNS and LES (based on Ferziger and Peric (2002))

Large eddy simulation: In contrast to DNS, in LES the large eddies are dissolved whereas the small ones are modeled (cf. Fig. 2.2 and Fig. 2.3). Thus, LES is less costly than DNS and preferred in highly turbulent flows or flows in complex geometries. LES is based on the assumption that any arbitrary quantity ϕ can be separated into a mean value $\overline{\phi}$ and a fluctuation ϕ', so that

$$\phi = \overline{\phi} + \phi' \quad \text{with} \quad \overline{\phi}(\mathbf{x}) = \int G(\mathbf{x}, \mathbf{x}')\phi(\mathbf{x}')dx' \,, \tag{2.21}$$

where spatial filtering is applied by a convolution kernel G (cf. Fig. 2.3). This approach leads to a scale separation into resolvable large-scale modes $\overline{\phi}$ of low frequency and unresolved (modeled) subgrid-scale (SGS) modes ϕ' of high frequency.

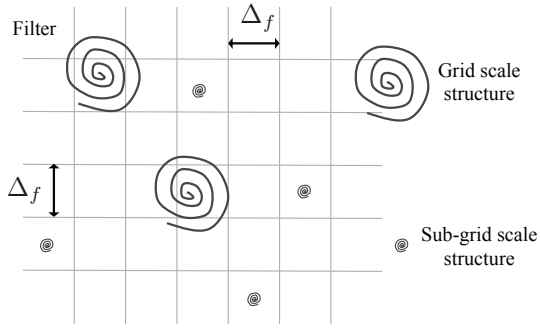


Figure 2.3: Schematic representation of spatial filtering

The filter function G depends on the filter width Δ_f and satisfies the normalization constraint

$$\int G(\mathbf{x}, \mathbf{x}')dx' = 1\,. \tag{2.22}$$

If the filter width is larger than the spatial resolution of the domain, the filtering approach is called *explicit* and if both scales are equal, the spatial resolution itself does an *implicit filtering*. Now, filtering the Navier-Stokes equations (2.9) - (2.12) leads to the LES equations

$$\nabla \cdot \overline{\mathbf{u}} = 0 \tag{2.23}$$

$$\partial_t \overline{\mathbf{u}} + (\overline{\mathbf{u}} \cdot \nabla)\, \overline{\mathbf{u}} - \nu \nabla^2 \overline{\mathbf{u}} + \frac{1}{\rho_0} \nabla \overline{p} + \frac{1}{\rho_0} \nabla \cdot \tau_u^{SGS} = \frac{1}{\rho_0} \mathbf{f}_B(\overline{T}) \tag{2.24}$$

$$\partial_t \overline{T} + (\overline{\mathbf{u}} \cdot \nabla)\, \overline{T} - \alpha \nabla^2 \overline{T} + \frac{1}{\rho_0 c_p} \nabla \cdot \tau_T^{SGS} = S_T \tag{2.25}$$

$$\partial_t \overline{C} + (\overline{\mathbf{u}} \cdot \nabla)\, \overline{C} - D \nabla^2 \overline{C} + \nabla \cdot \tau_C^{SGS} = S_C\,, \tag{2.26}$$

where $\tau_u^{SGS} = \rho_0 \overline{\mathbf{u}\mathbf{u}} - \rho_0 \overline{\mathbf{u}}\, \overline{\mathbf{u}}$ is called the *residual stress tensor* and similarly defined are $\tau_T^{SGS} = \rho_0 c_p \overline{\mathbf{u}T} - \rho_0 c_p \overline{\mathbf{u}}\overline{T}$ as well as $\tau_C^{SGS} = \overline{\mathbf{u}C} - \overline{\mathbf{u}}\overline{C}$. The main objective of the subgrid-scale models is now to approximate the residual stress tensors such that they represent the scale interaction. Therefore, Smagorinsky (1963) suggested that the Boussinesq hypothesis (cf. Boussinesq (1877)) can be applied. It states that the effects of the unresolved subgrid-scales are linearly proportional to the mean large-scale strain rates by a turbulent viscosity μ_T:

$$\tau_u^{SGS} - \frac{1}{3} \mathrm{Tr}(\tau_u^{SGS}) \mathbf{I} = -2\mu_T \overline{\mathbf{S}}\,, \tag{2.27}$$

where $\overline{S}_{i,j} = \frac{1}{2}(\partial_{x_j}\overline{u}_i + \partial_{x_i}\overline{u}_j)$ for $i,j = 1,2,3$ and $i \neq j$ denotes the filtered stress tensor (also called *rate-of-strain tensor*) for the resolved scale. If μ_T were to be known and the bars (for the mean quantities) are further omitted, the LES equations

$$\nabla \cdot \mathbf{u} = 0 \tag{2.28}$$

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - (\nu + \nu_T)\nabla^2 \mathbf{u} + \frac{1}{\rho_0}\nabla p = -\beta(T - T_0)\mathbf{g} \tag{2.29}$$

$$\partial_t T + (\mathbf{u} \cdot \nabla)\,T - (\alpha + \alpha_T)\nabla^2 T = S_T \tag{2.30}$$

$$\partial_t C + (\mathbf{u} \cdot \nabla)\,C - (D + D_T)\nabla^2 C = S_C \tag{2.31}$$

could be solved, where $\nu_T = \frac{\mu_T}{\rho_0}$, α_T, D_T are the turbulent kinematic viscosity, turbulent thermal and turbulent mass diffusivities, respectively, whereby α_T and D_T depend linearly on ν_T.

Thus, as a final step in the SGS model, the turbulent viscosity, μ_T, needs to be determined. Therefore, the Constant Smagorinsky-Lilly model of Smagorinsky (1963) assumes that

$$\mu_T = \rho_0 C_S^2 \Delta_f^2 \left\|\overline{\mathbf{S}}\right\| \quad \text{with} \quad \left\|\overline{\mathbf{S}}\right\| = \sqrt{2\sum_{i=1}^{3}\sum_{j=1}^{3}\overline{S}_{i,j}\overline{S}_{i,j}}\,, \tag{2.32}$$

where the Smagorinsky constant C_S is an empirical constant commonly used as $C_S \in \{0.1, 0.2\}$ (cf. Sagaut (2006)). Besides the Constant Smagorinsky-Lilly model a Dynamic Smagorinsky model (cf. Germano et al. (1991); Moin et al. (1991)), the Deardorff model (cf. Deardorff (1972)) and Vreman model (cf. Vreman (2004)) exist, which are not applied for the problem at hand due to the simplicity of the Constant Smagorinsky-Lilly model. Another approach similar to LES is RANS (Reynolds-Averaged-Navier-Stokes), but it is less accurate than LES since RANS only models the large eddies (instead of fully capturing them as in LES) and most of the turbulent energy is contained in the large eddies, which are responsible for a large part of the momentum transfer and turbulent mixing. Further, modeling all scales in the (single model) RANS approach is more difficult than modeling the subgrid-scale motions in LES.

Now, with $\nu_T = \frac{\mu_T}{\rho_0}$ to be known, the turbulent thermal and mass diffusivity can be determined using dimensionless numbers:

$$\alpha_T = \frac{\nu_T}{Pr_T} \quad \text{and} \quad D_T = \frac{\nu_T}{Sc_T}\,. \tag{2.33}$$

Dimensionless Numbers are introduced to describe the characteristics of the fluid.

Definition 2.1.8 (Prandtl number)**.** The *Prandtl number*, which is named after Ludwig Prandtl (cf. Prandtl (1949)), represents the ratio of diffusion of momentum, ν, to diffusion of heat, α, in a fluid:

$$Pr = \frac{\nu}{\alpha} = \frac{\mu/\rho}{k/\rho c_p} = \frac{\mu c_p}{k}\,. \tag{2.34}$$

Thus, the turbulent Prandtl number reads $Pr_T = \frac{\nu_T}{\alpha_T}$. For air at room temperature $Pr = 0.71$ holds, while most gases have similar values. Although the turbulent Prandtl number varies in space and the proper choice is not known, it is often assumed to be constant and varies in literature within $Pr_T \in [0.1, 1.0]$ according to Sagaut (2006).

Definition 2.1.9 (Schmidt number)**.** Analogously, the *Schmidt number* describes the ratio of kinematic viscosity, ν, and mass diffusivity, D, and is named after Ernst Heinrich Schmidt (1892 - 1975)

$$Sc = \frac{\nu}{D}\,. \tag{2.35}$$

Similarly, the turbulent version, $Sc_T = \frac{\nu_T}{D_T}$, describes the rates of turbulent transport of momentum and the turbulent transport of mass. Since $Sc/Pr \approx 1$ is frequently adapted in many investigations (cf. Yeoh and Yuen (2009)), the turbulent Schmidt number is taken to also be in the range of $Sc_T \in [0.1, 1.0]$.

Definition 2.1.10 (Reynolds number)**.** The ratio between inertial and friction force is described by the *Reynolds number*, Re, named after Osbourne Reynolds (in Reynolds (1895)),

$$Re = \frac{\text{inertial force}}{\text{friction force}} = \frac{\rho U L}{\mu} = \frac{U L}{\nu}\,, \tag{2.36}$$

where U and L represent the characteristic velocity and length scale, respectively, typical for the flow or setup at hand. Examples for the characteristic velocity are the average or maximum velocity, and exemplary characteristic lengths are the radius or (hydraulic) diameter of a pipe the fluid is flowing through or the plume diameter (cf. White (1991)).

The Reynolds number, Re, thereby measures the effect of viscosity and therefore is used to distinguish laminar from turbulent flows by comparing it to an appropriate

transition criterion. The Reynolds number is also used to identify the similarity of flows in the design of experiments. Similarity of two flows with the same geometry occurs if they hold the same Reynolds number.

Definition 2.1.11 (Grashof number)**.** Since the choice of the characteristic velocity is not always obvious, the *Grashof number* can be taken into account in buoyancy-driven flows to determine the transition of a laminar to a turbulent flow. The Grashof number, Gr, describes the ratio between buoyancy and viscous forces

$$Gr = \frac{\text{buoyancy force}}{\text{viscous force}} = \frac{g\beta\Delta T L^3}{\nu^2}\,, \tag{2.37}$$

where $\Delta T = T_{\text{surf}} - T_0$ is the difference between the surface and ambient temperatures. The relation of the Grashof to the squared Reynolds number can be used to determine if buoyancy effects are important ($Gr \gg Re^2$).

Definition 2.1.12 (Rayleigh number)**.** Also associated with buoyancy-driven flows is the *Rayleigh number* relating the heat transfer in the form of (natural) convection to the heat transfer in the form of conduction near a vertical wall:

$$Ra = \frac{g\beta}{\nu\alpha}(T_{\text{surf}} - T_0)L^3\,, \tag{2.38}$$

where g is the gravitational acceleration, ν describes the kinematic viscosity, and α and β denote the thermal diffusivity and expansion, respectively. The temperatures, T_{surf} and T_0, are the surface temperature of the vertical wall and ambient temperature (far from the wall), respectively. The characteristic length is denoted by L. Hence, the Rayleigh number is related to the Grashof and Prandtl numbers by $Ra = GrPr$.

Definition 2.1.13 (Nusselt number)**.** For heat transfer at a boundary or surface within the fluid, the *Nusselt number* relates the (natural or forced) convective heat transfer to the conductive heat transfer by

$$Nu = \frac{\text{convective heat transfer}}{\text{conductive heat transfer}} = \frac{h}{k/L} = \frac{hL}{k}\,, \tag{2.39}$$

with heat transfer coefficient h of the flow, the fluid's thermal conductivity k and characteristic length L. Empirically, the Nusselt number for natural convection functionally correlates with the Rayleigh and Prandtl numbers by $Nu = f(Ra, Pr)$.

2.1.3 Fire Modeling

So far, the flow of a turbulent fluid and its heat transfer is modeled using the Navier-Stokes and energy equations (2.28) - (2.30). With the passive scalar equation (2.31) the concentration of species can also be described. In the scope of this work is neither fire modeling including the processes of combustion, radiation or chemical reactions such as pyrolysis nor external influences such as pedestrians (cf. Fig. 2.4).

Fire and evacuation modeling is computationally expensive and would, therefore, negatively impact the goal of simulating pure smoke propagation in real-time or faster than real time. Further, fire and evacuation modeling is not crucial for the fire fighters to quickly adjust their tactics based on a fast and rough check of the smoke propagation motivating this work in the first place.

What needs to be accounted for in the validation of the software through experiments including fire is the reduction of the heat source in the energy equation to only the convective heat transfer fraction. Thus, the radiative fraction $\chi_{\text{rad}} \in [0.02, 0.5]$ from the flame (dependent amongst others on the fire diameter and fuel burned, cf. Drysdale (1999); Koseki (1989); McCaffrey and Cox (1982); Hamins et al. (1996)) needs to be subtracted from the total heat source. The (volumetric) heat source is then modeled as

$$S_T = (1 - \chi_{\text{rad}}) S_T^{\text{total}} \,, \tag{2.40}$$

where

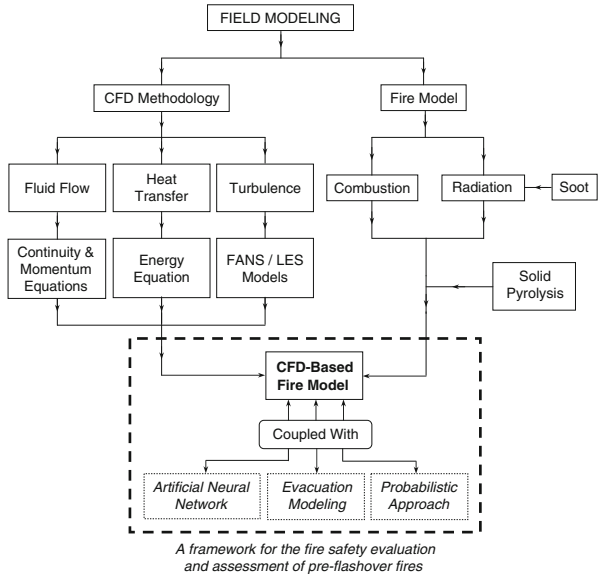
$$S_T^{\text{total}} = \frac{\dot{Q}}{\rho c_p V} \cdot f^{\text{vol}}(\mathbf{x}) \cdot f^{\text{ramp}}(t) \,, \tag{2.41}$$

with total heat release rate (HRR), $\dot{Q}$, discrete volume $V = \int_V f^{\text{vol}} \, dV$ with volume function f^{vol} (e.g., of Gaussian form) such that $\int_V \rho c_p S_T^{\text{total}} \, dV = \dot{Q} \cdot f^{\text{ramp}}$ and time ramp-up function f^{ramp} (e.g., $f^{\text{ramp}}(t) = \tanh(t/\tau)$). This approach is error-prone due to simplifications and assumptions such as the empirical estimation of radiative friction and constant density.

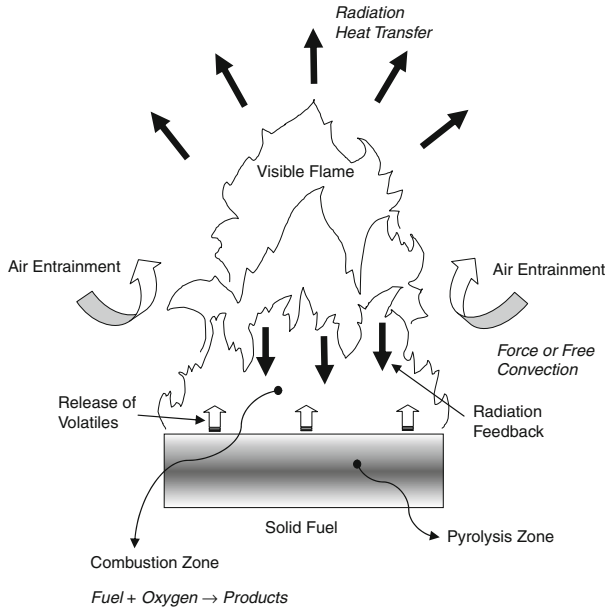
The concentration source of soot, S_C in $\text{kg}/(\text{m}^3\,\text{s})$, is modeled similarly by

$$S_C = Y_s \cdot \frac{\dot{Q}}{H_c V} \cdot f^{\text{vol}}(\mathbf{x}) \cdot f^{\text{ramp}}(t) \,, \tag{2.42}$$

where Y_s is the soot yield and H_c is the heat of combustion.



(a) CFD-based fire modeling



(b) Physical processes of a fire

Figure 2.4: Combination of CFD and Fire Modeling (cf. Yeoh and Yuen (2009))

Clarified which governing equations best describe the physical problem at hand – the smoke propagation – they need to be solved for the unknown velocity $\mathbf{u}$, pressure p and temperature T (and concentration). There only exist a few known analytical solutions and experimental testing is costly, difficult and could be dangerous. As an alternative, computational fluid dynamics is a way to obtain approximated solutions to the governing equations through numerics and computational power. Therefore, numerical approximations are applied to simplify the non-linear structure of spatial and temporal derivatives. With numerics, the so-called *discretized equations* in space and time can then be solved using modern calculators such as CPUs and GPUs. There exist various methods to numerically approximate the governing equations. In Section 2.2 selected solution models are introduced and methods suitable for the calculation using a GPU are described.

2.2 Numerical Models of Fluid Dynamics

In general, numerical models (besides analytical models, cf. Hurley et al. (2016)) can be divided into two types: *zone modeling* and *field modeling*. Zone models partition the computational domain into few subdomains with zone-global homogeneous properties (cf. Klote et al. (2012); Hurley et al. (2016)). Although zone models are commonly used in fire modeling they are too vague for the purpose of this work. Further, they can only represent simple geometries. Thus, the focus of the present work lies on field modeling (cf. e.g., Peiró and Sherwin (2005); Ferziger and Peric (2002); LeVeque (2007); Yeoh and Yuen (2009); Hurley et al. (2016)).

Field modeling is based on computational fluid dynamics solving the governing equations of mass, momentum, energy and passive scalars – derived in Section 2.1 – for each element in a computational domain with the help of computer architectures. In order to do so, the following steps need to be applied

1. The computational domain needs to be divided into smaller units (called *space discretization* resulting in a *numerical grid*) depending on an appropriate coordinate system for the flow at hand.

2. Based on the nature of the grid, a suitable *discretization method* has to be chosen, i.e., a method of approximating the governing differential equations (2.28) - (2.31) by a system of algebraic equations for the unknown variables at a set of discrete locations in space and time.

3. Based on the discretization method and numerical grid, *finite approximations* need to be selected. The choice influences, besides others, the ease of implementation, the accuracy of the solution (cf. Section 2.3) as well as the execution speed or efficiency of the code (cf. Section 6.1). Thus, a compromise has to be made, especially targeting parallel computing for real-time simulation, since approximations designed for traditional serial machines may not run efficiently on parallel computers.

4. In order to solve the non-linear algebraic equations of time-dependent variables (like those at hand), a *solution method* needs to be defined consisting of an iterative and a time marching solution method. In turn, convergence criteria for the iterative method(s) need to be set steering the accuracy and efficiency.

2.2.1 Numerical Grid

Depending on the flow at hand and the physical domain, (fixed or moving) coordinate systems can be chosen, for instance cylindrical, spherical, curvilinear orthogonal or non-orthogonal systems. Since the target geometry for the simulation of smoke propagation is set to be of cubic shape, the coordinate system remains fixed and Cartesian. Based on the Cartesian coordinate system a discrete representation of the geometric domain – the numerical grid – needs to be defined in order to calculate the unknown variables at discrete locations. One distinction in the choice of the numerical grid is its structure.

Unstructured grid: In case of very complex geometries, such as curved surfaces, an unstructured grid is beneficial for its flexibility to fit an arbitrary domain. Unstructured or irregular grids are thereby constructed by simple shapes, such as triangles or tetrahedra, in an irregular pattern. Thus, they do not consist of a regular array of cells that can be grouped into rows, columns, and layers, but the elements or volumes may have any shape (e.g., cf. top part of Fig. 2.5) yielding that the algebraic system of equations does not have an ordered structure. This irregularity, in turn, affects the memory access pattern and thus the efficiency of the solution method on a computer architecture, since node locations and neighbor connections need to be explicitly specified. Therefore, a structured grid is used if neighbor connectivity and uniquely defined node positions are crucial for the solution method as it is the case for the study at hand (cf. bottom part of Fig. 2.5).

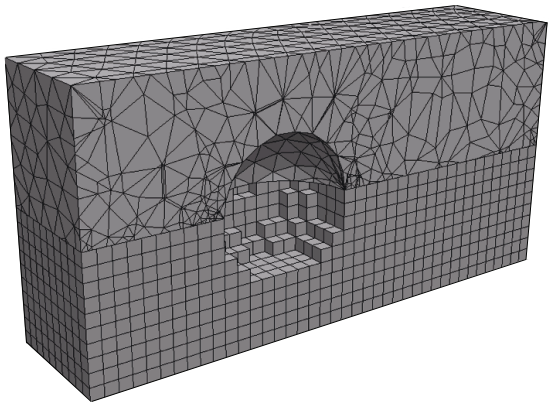


Figure 2.5: Examples of a structured grid (below) and an unstructured grid (top) in $3D$, adjusted from Brieda (2015)

Structured grid: In structured grids, the position of any grid point within the domain is uniquely identified by a tuple, (i, j), in $2D$ and a 3-tuple, (i, j, k), in $3D$ for $i = 0, \ldots, N_x - 1,\ j = 0, \ldots, N_y - 1$ and $k = 0, \ldots, N_z - 1$ (with N_x, N_y, N_z being the number of cells in the respective direction). Thereby, each point has four direct neighbors in $2D$ being $(i \pm 1, j \pm 1)$, and six in $3D$, $(i \pm 1, j \pm 1, k \pm 1)$. Thus, the memory access pattern for computer architectures is simple (e.g., in a row-wise (lexicographical) manner with global index $ix = i + N_x \cdot j + N_x \cdot N_y \cdot k$). Due to the simplicity a (Cartesian) structured grid with orthogonal mesh lines is used for the problem at hand (cf. Fig. 2.6) with cell distances, $\Delta x \neq \Delta y \neq \Delta z$, being different in each direction (called a *regular* grid) instead of *uniform*, i.e., $\Delta x = \Delta y = \Delta z$. This simplicity also reveals the main disadvantages of structured grids: they can only be applied for geometrically simple (rectangular) domains with the possibility to lose accuracy if the distances are chosen to be too wide and a waste of computational resources if no flow needs to be calculated in regions with fine grid spacing.

Once a grid structure is chosen, it is important to decide where the unknown variables, $\mathbf{u}$, p and T (and C), are located on the chosen grid. They can either all be set to the cell center (called *collocated grid*, cf. Fig. 2.7a) or only the scalars are located in the cell centers and vectors are defined at the cell faces (called *staggered grid*, cf. Fig. 2.7b). The variable location has an effect on the ease of implementation (pro collocated) and application of boundary conditions (pro staggered). For the ease of implementation, a collocated grid is used for the problem at hand. Therewith, the boundary condition for pressure needs to be explicitly set at all boundaries.

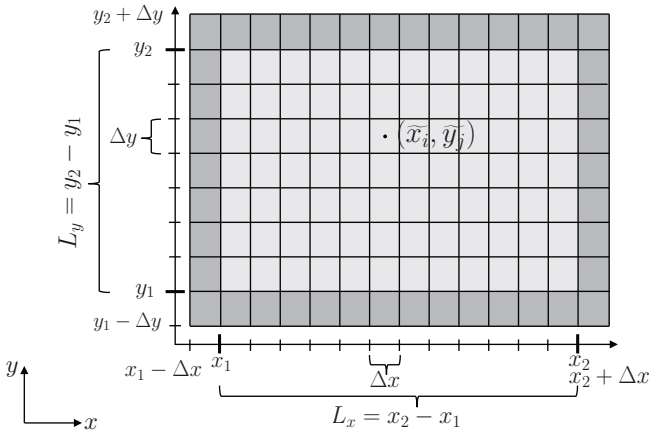
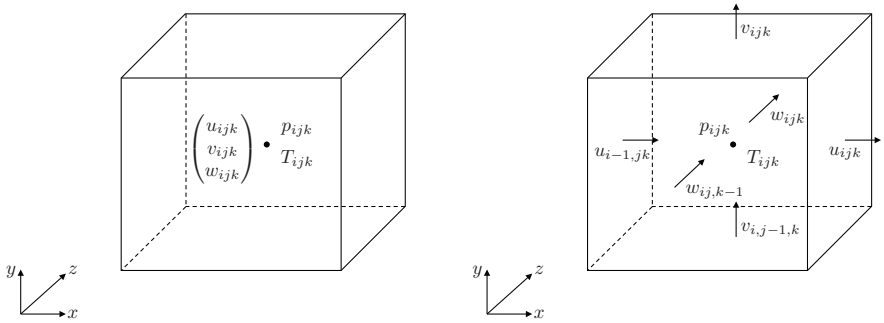


Figure 2.6: Schematic representation of a structured regular collocated grid in $2D$

Now, using a collocated grid so-called *ghost cells* (cf. dark gray cells in Fig. 2.6) need to be added to the domain to prescribe the boundary conditions at the physical domain borders $x_1, x_2, y_1, y_2, z_1, z_2$. The width of the computational domain can then be defined as $L_x = x_2 - x_1$ with height $L_y = y_2 - y_1$ and depth $L_z = z_2 - z_1$. Further, the cell centers are defined by $\tilde{x}_i = x_1 + (i - 0.5)\Delta x$, where $\Delta x = \frac{L_x}{N_x - 2} = x_{i+1} - x_i$ with N_x being the number of cells in x-direction including the ghost cells. Cell centers $\tilde{y}_j, \tilde{z}_k$ are defined analogously. The mesh lines themselves lie on $x_i = x_1 + (i-1)\Delta x$, $y_j = y_1 + (j-1)\Delta y$, and $z_k = z_1 + (k-1)\Delta z$. Therewith, an unknown variable, ϕ, is approximated by $\phi \approx \phi_i = \phi(\tilde{x}_i)$.



(a) Collocated grid with all variables at cell center

(b) Staggered grid with scalars at cell center and vectors at cell faces

Figure 2.7: Schematic comparison of variables located on a grid in $3D$

2.2.2 Discretization Method

At these discrete locations in space (and time), the unsteady differential equations now need to be approximated by a system of algebraic equations for the unknown variables. Therefore, various approaches exist depending on the numerical grid and problem at hand. The most common are *finite difference* (FDM), *finite volume* (FVM) and *finite element* (FVM) methods. Relatively new simulation techniques for complex fluid systems in CFD are the *Lattice Boltzmann* (LBM) and *cellular automata* (CA) method.

Lattice Boltzmann method: Instead of solving the Navier-Stokes equations, LBM solves the discrete Boltzmann equation simulating the flow with collision models. Thereby, the fluid consists of fictitious particles performing consecutive streaming and collision processes on a discrete mesh in terms of the probabilities of the particles' presence. Although LBM is designed to run efficiently on parallel architectures and can handle complex geometries and boundaries, a consistent thermodynamic scheme is absent in LBM. For this reason, LBM is not applicable for the incompressible equations at hand (or is at least associated with great effort to couple heat transfer into LBM).

Cellular automata: The CA approach follows the same idea as LBM (replaced by the explicit interaction between the particles instead of probabilities) with the additional advantage to be numerically stable. Due to the same drawback, CA is not applicable here.

Finite volume method: Applying FVM, the computational domain is divided into a finite number of control volumes and the integral formulation of the governing Navier-Stokes equations is then approximated. Surface integrals are evaluated as fluxes through the control volume. Thus, the FVM is conservative by construction, since the flux entering the volume equals the flux leaving the adjacent volume. Further, FVM allows for unstructured meshes, therefore it is suitable for complex geometries and local mesh refinement. However, a disadvantage of FVMs is that higher- (more than second-) order methods are hard to implement. More importantly, regarding the aim of real-time simulation, FVM is evaluated slowly on computer architectures since the unstructured mesh yields irregular memory accesses.

Finite element method: The same advantage of complex geometries with the resulting disadvantage of a less efficient solution holds the finite element method. The domain is divided into finite elements that are unstructured. Then, basis functions, such as a piecewise polynomial basis, are chosen as weight functions for the governing equations in integral form, which are approximated to form a set of non-linear algebraic equations. These basis functions allow for a higher-order solution method but make the method more complex.

Finite difference method: With the finite difference method, the pros and contras of the FEM or FVM are reversed. Although using FDM only allows simple geometries divided into a structured grid, many efficient solutions methods exist making the most of the restriction to simple geometries regarding ease of implementation and parallelization. Thus, again keeping in mind the goal of a real-time and faster than real-time prognosis simulation, FDM is the choice of discretization method for the problem at hand. Starting with the governing equations in differential form, each partial derivative is approximated (e.g., by Taylor series expansion or polynomial fitting) at each grid point of the structured mesh. Thus, on structured grids, FDM is very simple and effective with the possibility to obtain higher-order schemes very easily. An important drawback of FDM is that the conservation is not enforced. Thus, special care has to be taken for the governing equations to remain conservative since numerical errors have to be taken into account.

2.2.3 Finite Difference Approximations

For applying FDM on the governing equations (2.28) - (2.31), the derivatives of first and second-order need to be approximated at the grid points. All approximations are shown for derivatives in x-direction but also hold for y, z-directions.

Definition 2.2.1 (First derivative in space)**.** Let ϕ be a continuous differentiable function of x. Then the first spatial derivative of ϕ at point x_i is defined as

$$\left(\frac{\partial \phi}{\partial x}\right)_i := \lim_{\Delta x \to \infty} \frac{\phi(x_i + \Delta x) - \phi(x_i)}{\Delta x} \,. \tag{2.43}$$

As a short notation $\partial_x \phi\Big|_i$ is used and describes the slope of a tangent to the curve $\phi(x)$ at point x_i.

This slope can now be approximated by the slope of a line passing through two adjacent points on the curve. Depending on the chosen two points, the *forward*, *backward* or *central difference* is defined (cf. Fig. 2.8).

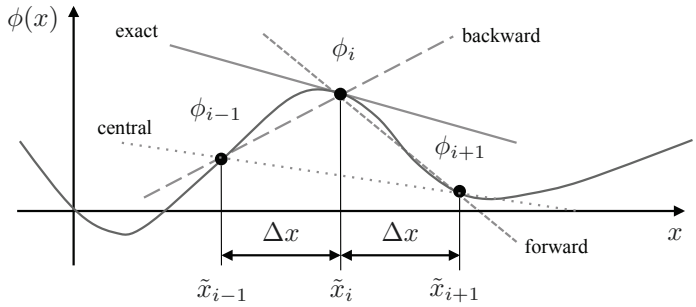


Figure 2.8: Schematic representation of a spatial derivative and its approximations, based on Ferziger and Peric (2002)

Definition 2.2.2 (Forward, backward and central difference)**.** The first derivative of a continuous differentiable function ϕ can be approximated by

$$\left(\frac{\partial\phi}{\partial x}\right)_i = \frac{\phi_{i+1}-\phi_i}{\Delta x} + \mathcal{O}(\Delta x) \qquad \approx \frac{\phi_{i+1}-\phi_i}{\Delta x} \qquad \text{forward difference (FD)}\,, \quad (2.44)$$

$$\left(\frac{\partial\phi}{\partial x}\right)_i = \frac{\phi_i-\phi_{i-1}}{\Delta x} + \mathcal{O}(\Delta x) \qquad \approx \frac{\phi_i-\phi_{i-1}}{\Delta x} \qquad \text{backward difference (BD)}\,, \quad (2.45)$$

$$\left(\frac{\partial\phi}{\partial x}\right)_i = \frac{\phi_{i+1}-\phi_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2) \qquad \approx \frac{\phi_{i+1}-\phi_{i-1}}{2\Delta x} \qquad \text{central difference (CD)} \quad (2.46)$$

using the Taylor series expansion of ϕ in the vicinity of x_i with uniform grid spacing

$$\begin{aligned}\phi(x) = \phi(x_i) + (x-x_i)\left(\frac{\partial\phi}{\partial x}\right)_i + \frac{(x-x_i)^2}{2!}\left(\frac{\partial^2\phi}{\partial x^2}\right)_i \\ + \frac{(x-x_i)^3}{3!}\left(\frac{\partial^3\phi}{\partial x^3}\right)_i + \cdots + \frac{(x-x_i)^n}{n!}\left(\frac{\partial^n\phi}{\partial x^n}\right)_i + \epsilon\,,\end{aligned} \quad (2.47)$$

which is aborted at a given order $n+1$ and evaluated at x_{i+1} for FD, at x_{i-1} for BD. CD is then obtained by taking the average of FD and BD.

Definition 2.2.3 (Truncation error, accuracy, rate of convergence)**.** The remainder in (2.47), ϵ, is called *truncation error* of order $\mathcal{O}((x-x_i)^{n+1})$. It measures the *accuracy* of the approximation and determines the rate at which the error decreases

when refining the grid. The speed in which the error goes to zero as $x \to x_i$ is called *rate of convergence*.

Definition 2.2.4 (Method of p-th order)**.** A method is referred to as a *method of p-th order* if the truncation error is of order $\mathcal{O}((x - x_i)^p)$, i.e., $\epsilon(\Delta x) \approx c\Delta x^p$ evaluated on discrete points with uniform grid spacing (cf. LeVeque (2005)). The higher the order of the scheme, the more accurate is its solution.

FD and BD are schemes of first-order $\mathcal{O}(\Delta x)$ and CD is a method of second-order $\mathcal{O}(\Delta x^2)$. It holds that the smaller the grid spacing, Δx, the higher the quality of the approximation.

For the estimation of the second derivative at a point in space the approximation for the first derivative can be used twice (either FD then BD or first BD then FD or even centered with half of the step size) since $\partial_x^2 \phi := \partial_x(\partial_x \phi)$ or the Taylor series expansion is applied again.

Definition 2.2.5 (Central difference for the second derivative)**.** The second derivative of a continuous differentiable function ϕ can be approximated by the central scheme

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i \approx \frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{\Delta x^2}\,, \tag{2.48}$$

which is second-order accurate for a uniform grid with spacing Δx.

Higher-order approximations for the second derivative can be obtained by including more data points (cf. Ferziger and Peric (2002); LeVeque (2005)).

The time derivatives in the governing equations (2.28) - (2.31) also need to be approximated. Thus, a time integration scheme is needed. Since they are also first-order derivatives, the schemes for the spatial first-order derivative can be applied here. Therefore, the time needs to be divided into discrete time steps $t^{(n)}$ for $n = 0, \ldots, N_t - 1$ with time spacing Δt (cf. Fig. 2.9).

Time integration schemes can roughly be divided into *two-level* and *multilevel methods*. Two-level methods only use evaluation at two time dates, whereas multilevel methods use more than two. The most simple and computationally inexpensive (two-level) schemes are the *explicit* and *implicit Euler method* besides other multilevel schemes such as Runge-Kutta methods (RK) and higher-order backward differentiation formulas (BDF, cf. Peiró and Sherwin (2005); Ferziger and Peric (2002); LeVeque (2005, 2007)).

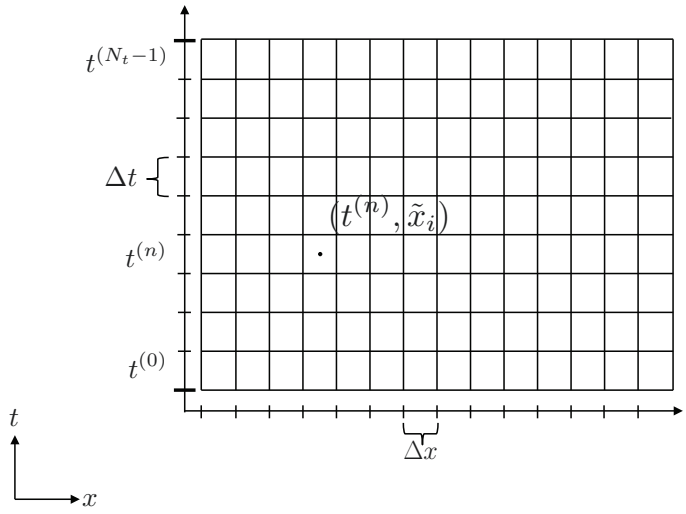


Figure 2.9: Schematic representation of time and space discretization

Definition 2.2.6. (Forward and backward Euler scheme) Let ϕ be a continuous differentiable function depending on time and space complying $\partial_t \phi = f(\phi)$. Then the temporal derivative can be approximated by

$$\frac{\phi^{(n+1)} - \phi^{(n)}}{\Delta t} = f(\phi)\,, \tag{2.49}$$

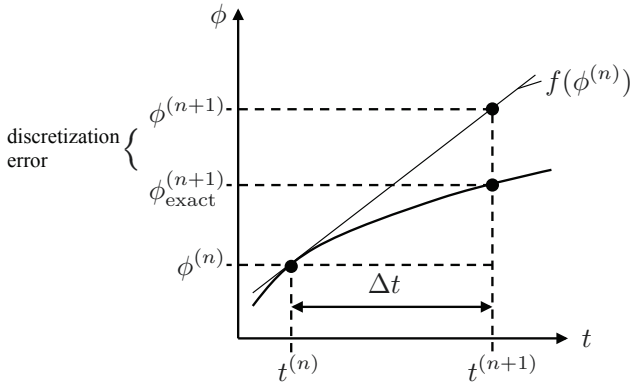
where $\phi^{(n)} = \phi(\mathbf{x}, t^{(n)})$ and $t^{(n)} = n \cdot \Delta t$. Depending on the point in time the right hand side of (2.49) is evaluated, the Euler methods are defined as

$$\frac{\phi^{(n+1)} - \phi^{(n)}}{\Delta t} = f(\phi^{(n)}) \quad \Longleftrightarrow \quad \phi^{(n+1)} = \phi^{(n)} + \Delta t \cdot f(\phi^{(n)}) \qquad \text{Forward Euler}\,, \tag{2.50}$$

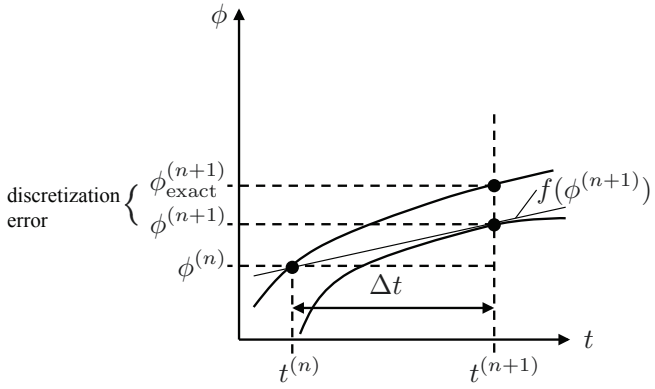
$$\frac{\phi^{(n+1)} - \phi^{(n)}}{\Delta t} = f(\phi^{(n+1)}) \quad \Longleftrightarrow \quad \phi^{(n+1)} = \phi^{(n)} + \Delta t \cdot f(\phi^{(n+1)}) \qquad \text{Backward Euler}\,. \tag{2.51}$$

Since the Forward Euler (FE) scheme can directly be evaluated using the previously calculated values $f(\phi^{(n)})$ it describes an *explicit* method (cf. Fig. 2.10a). Evaluating $f(\phi)$ on the current time step $n+1$ (Backward Euler, BE) results in an *implicit* method (cf. Fig. 2.10b). Both methods are of first-order $\mathcal{O}(\Delta t)$. Evaluating $f(\phi)$ at midpoint, $t^{(n+1/2)}$ leads to the *midpoint rule* and interpolating between $f(\phi^{(n)})$ and

$f(\phi^{(n+1)})$ yields the *trapezoid rule* (cf. Peiró and Sherwin (2005); Ferziger and Peric (2002); LeVeque (2005, 2007)).



(a) Schematic representation of forward (explicit) Euler method



(b) Schematic representation of backward (implicit) Euler method

Figure 2.10: Schematic comparison of explicit and implicit Euler methods, based on Laurien and Oertel (2013)

Although only of first-order, the Euler scheme is preferred for the problem at hand, since multilevel schemes such as RK are computationally more expensive as they involve matrix operations and are not trivially parallelized due to their data-dependent, thus variable task length (cf. Murray (2011)). Two-level schemes are, however, easy to program, computationally inexpensive regarding memory and need little computational time per step. A drawback of explicit schemes is their instability

in cases of large time steps. Nevertheless, the explicit Euler scheme is less expensive than an implicit scheme (being unconditionally stable). Stability is explained in detail when applying the solution method yet to be chosen.

2.2.4 Solution Method

The finite difference approximations now provide a system of (non-linear and time-dependent) algebraic equations for the Navier-Stokes equations. Thus, the discretized equations must be solved in space and time. Therefore, again various methods exist including the MAC (Marker And Cell) method for staggered grids (cf. Harlow and Welch (1965)), Semi-Implicit Methods for Pressure-Linked Equations (SIMPLE, cf. Patankar and Spalding (1972)) and developments thereof such as SIMPLER – revised (cf. Patankar (1981)), SIMPLEC – consistent (Van Doormaal and Raithby (1984)) and PISO (Pressure-Implicit with Splitting of Operators, cf. Issa (1986)), as well as projection methods introduced by Chorin (1968), further developed by Kim and Moin (1985) and Choi and Moin (1994). The MAC and ADI (Alternating Direction Implicit, cf. Peaceman and Rachford Jr. (1955)) methods are similar to fractional step methods but computationally costly. Whereas SIMPLE type methods have been successfully applied to steady flows, projection methods are dominantly used for unsteady flows. For these reasons, a fractional step method with projection is applied for the problem at hand resulting into four (sequential) updates at each time step (such as in Stam (1999); Harris (2004); Crane et al. (2007); Glimberg et al. (2009)) for the momentum's equations (2.29)

$$\partial_t \mathbf{u}^{(1)} = -(\mathbf{u}^{(n)} \cdot \nabla)\mathbf{u}^{(n)} \qquad \text{convection } \mathbb{C} \tag{2.52}$$

$$\partial_t \mathbf{u}^{(2)} = \nu_{\text{eff}} \nabla^2 \mathbf{u}^{(1)} \qquad \text{diffusion } \mathbb{D} \tag{2.53}$$

$$\partial_t \mathbf{u}^{(3)} = -\beta(T - T_0)\mathbf{g} \qquad \text{force } \mathbb{F} \tag{2.54}$$

$$\partial_t \mathbf{u}^{(n+1)} = -\frac{1}{\rho_0}\nabla p \qquad \text{pressure } \mathbb{P}\,, \tag{2.55}$$

where $(\cdot)^{(f)}$ describes a variable at fractional step f, the input $(\cdot)^{(n)}$ results from the last discrete time step $t^{(n)}$ and the solution solved for is denoted by $(\cdot)^{(n+1)}$. Further, $\nu_{\text{eff}} = \nu + \nu_T$ denotes the effective viscosity including the turbulent viscosity. The operators comply with

$$\partial_t \mathbf{u} = \mathbb{P} \circ \mathbb{F} \circ \mathbb{D} \circ \mathbb{C}(\mathbf{u}) =: \mathbb{L}(\mathbf{u})\,, \tag{2.56}$$

and thus,

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \Delta t(\mathbb{P} \circ \mathbb{F} \circ \mathbb{D} \circ \mathbb{C})(\mathbf{u}^{(n)})\,, \tag{2.57}$$

when explicit time stepping is applied.

The incompressibility constraint (2.28), $\nabla \cdot \mathbf{u}^{(n+1)} = 0$, is enforced by a Helmholtz-Hodge decomposition after the last fractional step. Since the force is dependent on the temperature, the energy equation (2.30) (and, if applicable, the concentration equation (2.31)) needs to be solved in every time step as well. This step can be done using the same methods as applied in solving the momentum equation without the need to implement any additional solvers. As an alternative, the *Scharfetter-Gummel method* can be used (cf. Scharfetter and Gummel (1969)) to solve passive scalar equations by exponential fitting. However, exponential fitting adds an implementation effort and is therefore not applied here.

Each of the fractional steps can now be handled independently, whereby the boundary conditions need to be applied after each fractional step. Therefore, a solution method for each step must be chosen. Here, again many variations depending on the problem and aim at hand are possible. Though, before these schemes are defined, consistency, numerical stability, convergence, and conservation need to be explained, since they constitute many of the crucial characteristics of a method and impact the methods' efficiency.

Definition 2.2.7 (Consistency)**.** Consistency is a condition on the numerical scheme. A numerical scheme is considered *consistent* if the discretization becomes exact (i.e., tends to the differential equation) as the steps in time and space tend to zero. Thus, the truncation error ϵ of order $\mathcal{O}(\Delta t^q, \Delta x^p)$ vanishes, i.e.,

$$\epsilon \to 0 \quad \text{for} \quad \Delta t, \Delta x \to 0\,. \tag{2.58}$$

Definition 2.2.8 (Stability)**.** Stability is a condition on the numerical solution. A numerical solution is considered *stable* if all errors (including round-off errors, errors in boundary or initial conditions) remain bounded when the iteration process progresses, i.e., for finite values of Δt and Δx the error has to remain bounded when the number of time steps n tends to infinity. Let $\epsilon_i^{n,\,\mathrm{num}} = \phi_i^{(n)} - \phi_i^{n,\,\mathrm{num}}$ be the difference between the computed solution, $\phi_i^{(n)}$ (including round-off errors, errors in boundary or initial conditions), and the exact solution, $\phi_i^{n,\,\mathrm{num}}$, of the discretized equation, then the stability condition reads

$$\lim_{n\to\infty} |\epsilon_i^{n,\,\mathrm{num}}| \leq c \quad \text{at fixed } \Delta t \text{ and any point } x_i\,. \tag{2.59}$$

Stability does not ensure that the error will not become unacceptably large at intermediate time steps $t^{(n)}$ while remaining bounded. The Von Neumann stability analysis (cf. Crank and Nicolson (1947); Charney et al. (1950)) can be used to further obtain a condition for stability by expanding the computed solution in a finite Fourier series. All numerical solutions of implicit schemes are unconditionally stable, while explicit schemes are at best conditionally stable.

Example (Convective and diffusive stability constraints)**.**

a) Let the scalar linear advection equation $\partial_t u + c\partial_x u = 0$ with $c > 0$ be discretized by BD in space and FE in time. Then the Von Neumann stability analysis leads to the (necessary) *CFL condition* named after Richard Courant, Kurt Friedrichs und Hans Lewy in 1928 (cf. Courant et al. (1928))

$$0 < CFL = c\frac{\Delta t}{\Delta x} \leq 1\,. \tag{2.60}$$

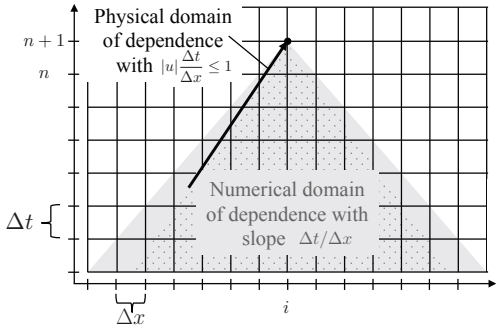
For the non-linear advection equation $\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} = 0$ in $3D$ the convective constraint reads

$$0 < CFL = \|\mathbf{u}\|\frac{\Delta t}{\Delta x_i} \leq 1\,, \tag{2.61}$$

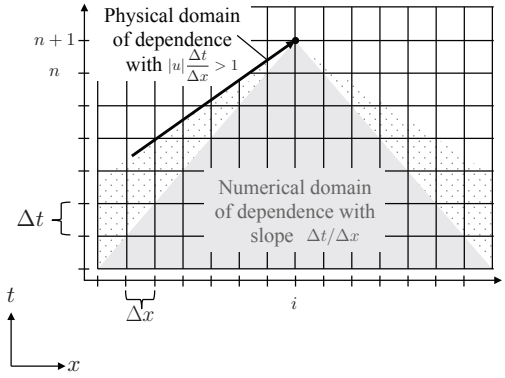
where

$$\frac{\|\mathbf{u}\|}{\Delta x_i} = \begin{cases} \frac{\|\mathbf{u}\|_\infty}{\Delta x_i} & := \max\left(\frac{|u|}{\Delta x}, \frac{|v|}{\Delta y}, \frac{|w|}{\Delta z}\right) \\ \frac{\|\mathbf{u}\|_1}{\Delta x_i} & := \frac{|u|}{\Delta x} + \frac{|v|}{\Delta y} + \frac{|w|}{\Delta z} \\ \frac{\|\mathbf{u}\|_2}{\Delta x_i} & := \sqrt{\frac{u^2}{\Delta x^2} + \frac{v^2}{\Delta y^2} + \frac{w^2}{\Delta z^2}}\,. \end{cases} \tag{2.62}$$

Physically, the CFL condition states that a fluid element should not traverse more than one cell within a time step. Numerically, this condition can be expressed with the domains of dependency (cf. Fig. 2.11). The numerical solution is stable if the physical lies within the numerical domain of dependency (cf. Fig. 2.11a), otherwise the numerical solution is unstable (cf. Fig. 2.11b).



(a) Physical domain lies inside numerical domain of dependence (stable)



(b) Physical domain lies outside numerical domain of dependence (unstable)

Figure 2.11: Schematic comparison of numerical and physical domains of dependency yielding stability (top) or instability (bottom)

b) Let the diffusion equation $\partial_t u - \nu \partial_x^2 u = 0$ be discretized by CD in space and FE in time. Then the Von Neumann stability analysis leads to the diffusive constraint for numerical stability of the scheme

$$\nu \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \quad \text{with } \nu > 0\,. \tag{2.63}$$

For the diffusion equation $\partial_t \mathbf{u} - \nu \nabla^2 \mathbf{u} = \mathbf{0}$ in $3D$ the diffusive constraint reads (cf. Peiró and Sherwin (2005))

$$\nu \Delta t \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2} \right) \leq \frac{1}{2} \quad \text{with } \nu > 0\,. \tag{2.64}$$

Definition 2.2.9 (Convergence)**.** Convergence is a condition on the numerical solution. The numerical solution is considered *convergent* if it tends to the exact solution of the mathematical model when refining the mesh, thus when the steps in time and space tend to zero. Let ${\epsilon_i}^{\text{n, ana}} = \phi_i^{(n)} - {\phi_i}^{\text{n, ana}}$ be the difference between the computed solution, $\phi_i^{(n)}$ (including round-off errors, errors in boundary or initial conditions), and the exact solution, ${\phi_i}^{\text{n, ana}} = \phi^{\text{ana}}(x_i, t^{(n)})$, of the analytical equation representing the mathematical model, then the convergence condition reads

$$\lim_{\Delta x, \Delta t \to 0} |{\epsilon_i}^{\text{n, ana}}| = 0\,. \tag{2.65}$$

In practice, convergence can be investigated by comparing the results obtained by successively refining the grid. Therefore, let $\epsilon(\Delta x) = c\Delta x^p$ be the truncation error of the discretization scheme. Refining the grid by reducing the grid size to $\Delta x/2$ results in $\epsilon(\Delta x/2) = c(\Delta x/2)^p$, i.e, the convergence rate is

$$p = \log_2 \left(\frac{\epsilon(\Delta x)}{\epsilon(\Delta x/2)} \right)\,. \tag{2.66}$$

To also assess the temporal error in a time integration scheme alone, the spatial error needs to be filtered out since they will typically dominate. Therefore, a method based on *Richardson extrapolation* (cf. Moin (2001)) is applied. Let ϕ be any quantity (e.g., the calculated numerical solution of a variable or the error to an analytical solution), then expanding ϕ in a Taylor series with different time increments at a constant ratio $r = \Delta t_{\text{fine}}/\Delta t_{\text{coarse}} < 1$ yields

$$\phi_1 = \phi_0 + c\Delta t^p + \epsilon(\Delta t^{p+1}) \tag{2.67}$$

$$\phi_2 = \phi_0 + c(r\Delta t)^p + \epsilon(r^{p+1}\Delta t^{p+1}) \tag{2.68}$$

$$\phi_3 = \phi_0 + c(r^2\Delta t)^p + \epsilon(r^{2p+2}\Delta t^{p+1})\,, \tag{2.69}$$

where ϕ_0 is the continuum value at zero spacing and c is a constant. The temporal convergence order is then given by

$$p = \frac{\ln\left(\frac{\phi_3 - \phi_2}{\phi_2 - \phi_1}\right)}{\ln r}\,. \tag{2.70}$$

This method is especially useful if no analytical solution to the mathematical model exists since ϕ can be any quantity related to the numerical scheme.

Definition 2.2.10 (Discretization error)**.** The discretization error is defined as the difference between the exact solution of the governing equations and the exact solution of the discrete approximation. It can only be approximated (due to the truncation error). A way to approximate the discretization error is to use the p-th order of convergence

$$\epsilon_2 \approx \frac{\phi_2 - \phi_1}{r^p - 1}\,, \tag{2.71}$$

where r is the constant grid refinement ratio, $r = \Delta x_{\text{fine}}/\Delta x_{\text{coarse}} < 1$, and therefore, ϕ_1 a quantity on the coarser grid and ϕ_2 on the finer grid.

Definition 2.2.11 (Conservation)**.** Physical principles such as mass, momentum and energy conservation, should apply at a discrete level. A finite difference scheme is considered *conservative* if it can be written in the form

$$\frac{\phi_i^{(n+1)} - \phi_i^{(n)}}{\Delta t} + \frac{F_{i+1/2} - F_{i-1/2}}{\Delta x} = S_i\,, \tag{2.72}$$

where F is called *flux*.

Definition 2.2.12 (Numerical dissipation and dispersion)**.** Let $\phi_i^{n,\,\text{num}}$ be the exact solution to the numerical scheme for $\partial_t\phi + \mathbb{L}(\phi) = 0$. Then $\phi_i^{n,\,\text{num}}$ satisfies the modified differential equation

$$\partial_t\phi + \mathbb{L}(\phi) = \sum_{p=1}^{\infty} c_{2p}\partial_x^{2p}\phi + \sum_{p=1}^{\infty} c_{2p+1}\partial_x^{2p+1}\phi\,, \tag{2.73}$$

where the even-order derivatives cause *numerical dissipation* (also called artificial or numerical diffusion) resulting in additional damping (amplitude errors) of the numerical solution $\phi_i^{(n)}$, whereas the odd-order derivatives cause *numerical dispersion* leading to jumps (phase errors) in the numerical solution $\phi_i^{(n)}$.

Depending on these characteristics suitable numerical schemes for the convection, diffusion and pressure terms are now chosen. Thereby $(\cdot)_{ijk}^{(f)}$ denotes a discretized variable in space at grid cell center (i, j, k) at fractional step f, where the input $(\cdot)_{ijk}^{(n)}$ results from the last discrete time step $t^{(n)}$ and the solution solved for is denoted as $(\cdot)_{ijk}^{(n+1)}$.

Convection: First the convection equation (2.52), i.e.,

$$\partial_t \mathbf{u}^{(1)} = -(\mathbf{u}^{(n)} \cdot \nabla)\mathbf{u}^{(n)} \tag{2.74}$$

is numerically solved. Explicit time stepping yields

$$\frac{\mathbf{u}^{(1)} - \mathbf{u}^{(n)}}{\Delta t} = -(\mathbf{u}^{(n)} \cdot \nabla)\mathbf{u}^{(n)}\,, \tag{2.75}$$

which is limited by a maximum time step size to avoid numerical instability. This restriction was overcome by Stam (1999) introducing the so-called *Semi-Lagrangian advection* (SL) method, which is equivalent to an implicit time integration and is based on the *method of characteristics*. Thereby, consider the convection equation

$$\partial_t \phi(\mathbf{x}, t) + \mathbf{u}(\mathbf{x}) \cdot \nabla \phi(\mathbf{x}, t) = 0 \tag{2.76}$$

with steady vector field $\mathbf{u}$ and unknown scalar field $\phi = \phi(\mathbf{x}, t)$ with $\phi(\mathbf{x}, 0) = \phi_0(\mathbf{x})$. Let $\mathbf{p}(\mathbf{x}_0, t)$ denote the characteristics of vector field $\mathbf{u}$ flowing through the point $\mathbf{x}_0$ at time $t = 0$, i.e.,

$$\frac{d}{dt}\mathbf{p}(\mathbf{x}_0, t) = \mathbf{u}(\mathbf{p}(\mathbf{x}_0, t)) \tag{2.77}$$

with $\mathbf{p}(\mathbf{x}_0, 0) = \mathbf{x}_0$ and *material derivative* $\frac{d}{dt} \coloneqq \partial_t + \mathbf{u} \cdot \nabla$ (cf. Chorin and Marsden (1979)). Now, let $\hat{\phi}(\mathbf{x}_0, t) = \phi(\mathbf{p}(\mathbf{x}_0, t), t)$ be the value of the scalar field along the characteristic passing through point $\mathbf{x}_0$ at time $t = 0$. Then using the chain rule the variation of $\hat{\phi}$ over time can be computed with the convection equation (2.76) as

$$\frac{d}{dt}\hat{\phi} = \partial_t \hat{\phi} + \mathbf{u} \cdot \nabla \hat{\phi} = \partial_t \phi + \mathbf{u} \cdot \nabla \phi = 0 \tag{2.78}$$

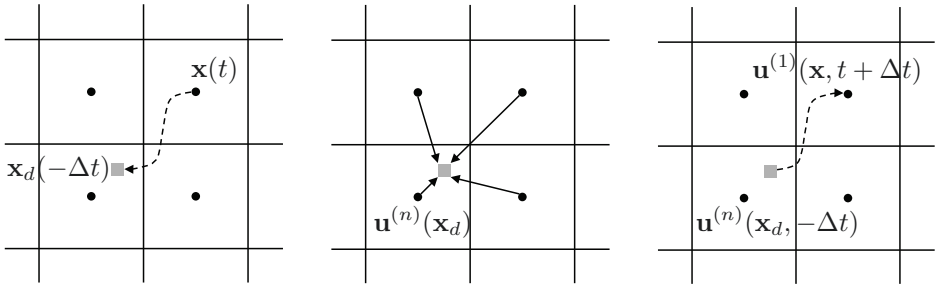
showing that the value of the scalar $\hat{\phi}$ does not vary along the streamlines, i.e., $\hat{\phi}(\mathbf{x}_0, t) = \hat{\phi}(\mathbf{x}_0, 0) = \phi_0(\mathbf{x}_0)$. Thus, the initial field ϕ_0 and the characteristics $\mathbf{p}$ entirely define the solution of the convection problem by first tracing the location $\mathbf{x}$ back in time along the characteristic to get to point $\mathbf{x}_0$ and then evaluating the initial field at that point:

$$\phi(\mathbf{p}(\mathbf{x}_0, t), t) = \phi_0(\mathbf{x}_0)\,. \tag{2.79}$$

Translating this approach for the convection equation (2.52) to obtain velocity $\mathbf{u}^{(1)}$ at

point $\mathbf{x}$ at time $t + \Delta t$, the point $\mathbf{x}$ is traced back through velocity field $\mathbf{u}^{(n)}$ over the time step Δt obtaining a path $\mathbf{p}(\mathbf{x}, -\Delta t) = \mathbf{x} - \Delta t \mathbf{u}^{(n)} =: \mathbf{x}_d(-\Delta t)$ (cf. Fig. 2.12a). The new velocity $\mathbf{u}^{(1)}$ is then set to the velocity that the element had at its previous location Δt ago (cf. Fig. 2.12c)

$$\mathbf{u}^{(1)}(\mathbf{x}, t + \Delta t) = \mathbf{u}^{(n)}(\mathbf{p}(\mathbf{x}, -\Delta t)) = \mathbf{u}^{(n)}(\mathbf{x}_d) = \mathbf{u}^{(n)}(\mathbf{x} - \Delta t \mathbf{u}^{(n)}) \,. \tag{2.80}$$



(a) Trace back from point $\mathbf{x}(t)$ with current velocity $\mathbf{u}^{(n)}$ to its previous location $\mathbf{x}_d(-\Delta t)$ Δt ago

(b) Interpolate with current velocities of grid around departure point $\mathbf{x}_d$ to obtain $\mathbf{u}^{(n)}(\mathbf{x}_d)$

(c) Assign the interpolated $\mathbf{u}^{(n)}(\mathbf{x}_d)$ to the unknown velocity $\mathbf{u}^{(1)}$ to get its state at time $t + \Delta t$

Figure 2.12: Schematic representation of the Semi-Lagrangian method, based on Verma et al. (2014)

Since the departure point $\mathbf{x}_d$ will normally not be a grid point, the value at the departure point must be calculated by (trilinear) interpolation from surrounding points (cf. Fig. 2.12b). Thus, find indices (i, j, k) such that $\mathbf{x}_{i,j,k} \leq \mathbf{x}_d \leq \mathbf{x}_{i+1,j+1,k+1}$ or $\mathbf{x}_{i-1,j-1,k-1} \leq \mathbf{x}_d \leq \mathbf{x}_{i,j,k}$ depending on the direction of $u^{(n)}, v^{(n)}, w^{(n)}$, respectively, and use the velocity at these grid cells to interpolate. In $1D$ this interpolation reads

$$u_i^{(1)} = u^{(n)}_{i - \frac{\Delta t}{\Delta x} u^{(n)}} = u^{(n)}_{i-c-\omega} \approx (1 - \omega) u^{(n)}_{i-c} + \omega u^{(n)}_{i-c-1} \tag{2.81}$$

with $c = \left\lceil \frac{\Delta t}{\Delta x} u^{(n)} \right\rceil$ and $\omega = \frac{\Delta t}{\Delta x} u^{(n)} - c$. The interpolation with non-negative coefficients smaller than unity in each direction ensures that the path falls within the numerical domain of dependence since $\max_{ijk}(\mathbf{u}^{(1)}) \leq \max_{ijk}(\mathbf{u}^{(n)})$ at all times (cf. Bonaventura (2004)). Therefore, the SL scheme is unconditionally numerically stable by definition leading to a computationally efficient scheme. Nevertheless, using trilinear interpolation the SL scheme is only of first-order. Thus, time stepping and grid resolution

still need to be carefully chosen to obtain enough accuracy. Also at the boundary caution has to be taken. If the backtracing reaches beyond the boundary, the indices used for interpolation are set to the nearest boundary indices (cf. Fig. 2.13). Another drawback to keep in mind is that the SL scheme introduces numerical dissipation (artificial viscosity) since backtracing with the forward Euler scheme introduces numerical truncation errors (cf. Huang et al. (2015)).

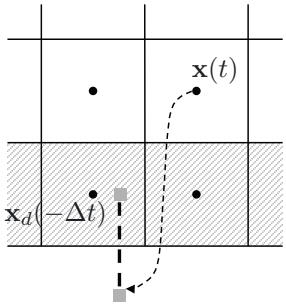


Figure 2.13: Schematic representation of backtracing at the boundary, based on Jin et al. (2012)

Further, the SL scheme is not fully conservative (cf. Lentine et al. (2011)). Thus, enhancements of the SL were made such as the unconditionally stable, second-order accurate MacCormack scheme (cf. Selle et al. (2008)) or the fully conservative scheme of Lentine et al. (2011), both with higher computational costs. For the problem at hand, the (classic) SL scheme is integrated since its simplicity for parallel implementation and its implicit nature compensate for the numerical dissipation.

Diffusion: Now for solving the diffusion equation (2.53), i.e.,

$$\partial_t \mathbf{u}^{(2)} = \nu_{\text{eff}} \nabla^2 \mathbf{u}^{(1)} \tag{2.82}$$

an implicit scheme is directly used to support the idea of an unconditionally stable solution method as it is for the convection equation. Thus with BE in time the right-hand side of the diffusion equation (2.53) is evaluated at the next time step, yielding

$$\frac{\mathbf{u}^{(2)} - \mathbf{u}^{(1)}}{\Delta t} = \nu_{\text{eff}} \nabla^2 \mathbf{u}^{(2)} \,. \tag{2.83}$$

Now, reordering gives a linear system of equations $\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$

$$(\mathbf{I} - \Delta t \nu_{\text{eff}} \nabla^2)\mathbf{u}^{(2)} = \mathbf{u}^{(1)} \tag{2.84}$$

with matrix $\mathbf{A} = \mathbf{I} - \Delta t \nu_{\text{eff}} \nabla^2 \in \mathbb{R}^{N \times N}$, unknown $\boldsymbol{\phi} = \mathbf{u}^{(2)} \in \mathbb{R}^N$ and right-hand side $\mathbf{b} = \mathbf{u}^{(1)} \in \mathbb{R}^N$ known from the previous fractional step. After discretizing the second derivatives in space with central differences, any matrix solver for a linear and diagonally dominant system of equations can be applied here.

The size of the matrix, $N \coloneqq (N_x - 2)(N_y - 2)(N_z - 2)$, quickly rules out direct solvers such as LU decomposition or Gaussian elimination since the inverse or decomposition are often again fully occupied resulting in high computational costs. Further, direct methods solve the system more accurately than necessary since the discretization error is usually larger than the arithmetic accuracy (round-off error ϵ) of a computer architecture. Therefore, it suffices to search for a method with an accuracy smaller than the accuracy of the discretization scheme (i.e., for first-order FDM it holds that $\Delta x \geq \sqrt{\epsilon}$ and $\Delta x \geq \sqrt[4]{\epsilon}$ for the CD of the second derivative).

Thus, iterative methods are preferred for the problem at hand. A major advantage of iterative solvers is the significantly lower memory usage than for direct solvers for the same problem size; this holds since the (non-changing) coefficient matrix can be stored in a memory efficient way (e.g., in compressed sparse row format). As a drawback of iterative solvers, the errors resulting from the finite nature of iterative processes (being stopped after a certain tolerance is reached) need to be taken into account. Thus, for the problem at hand an iterative method should be chosen whose single iteration is cheap (i.e., low number of read and write accesses) as well as data independent (to be parallelized) and whose number of iterations to obtain convergence is small.

In general, an iteration method is described by its iteration specification (cf. for instance Ferziger and Peric (2002)). Thereby, an initial guess of the solution is systematically improved by iterating until it converges to the exact solution.

Definition 2.2.13 (Iteration method)**.** Let $\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$ be a linear system of N equations. Then, for each row $i = 0, \ldots, N-1$ it holds that

$$\sum_{j=0}^{N-1} \mathbf{A}_{ij}\boldsymbol{\phi}_j = \mathbf{b}_i \tag{2.85}$$

$$\iff \sum_{j=0}^{i-1} \mathbf{A}_{ij}\boldsymbol{\phi}_j + \mathbf{A}_{ii}\boldsymbol{\phi}_i + \sum_{j=i+1}^{N-1} \mathbf{A}_{ij}\boldsymbol{\phi}_j = \mathbf{b}_i \,. \tag{2.86}$$

In the case of $\mathbf{A}_{ii} \neq 0$, this relation yields

$$\boldsymbol{\phi}_i = \frac{1}{\mathbf{A}_{ii}} \left(\mathbf{b}_i - \sum_{j=0}^{i-1} \mathbf{A}_{ij} \boldsymbol{\phi}_j - \sum_{j=i+1}^{N-1} \mathbf{A}_{ij} \boldsymbol{\phi}_j \right) . \tag{2.87}$$

Iterating this approximation l times with initial guess $\boldsymbol{\phi}^{(0)}$ defines the iteration specification for $\boldsymbol{\phi}^{(l+1)}$ based on the previous iteration approximation $\boldsymbol{\phi}^{(l)}$. In matrix notation this method can also be written as

$$\mathbf{M}\boldsymbol{\phi}^{(l+1)} = \mathbf{N}\boldsymbol{\phi}^{(l)} + \mathbf{b} , \tag{2.88}$$

where $\mathbf{A} = \mathbf{M} - \mathbf{N}$ since at convergence $\boldsymbol{\phi}^{(l+1)} = \boldsymbol{\phi}^{(l)} = \boldsymbol{\phi}$ holds.

Definition 2.2.14 (Residual, Iteration error)**.** Let $\boldsymbol{\phi}^{(l)}$ be an approximate solution of an iteration method for a linear system, $\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$, after l iterations. Since $\boldsymbol{\phi}^{(l)}$ does not satisfy the linear system exactly there exists a non-zero *residual* $\mathbf{r}^{(l)}$ satisfying

$$\mathbf{r}^{(l)} = \mathbf{b} - \mathbf{A}\boldsymbol{\phi}^{(l)} . \tag{2.89}$$

The difference between the exact solution, $\boldsymbol{\phi}$, and the approximate solution, $\boldsymbol{\phi}^{(l)}$, is called *iteration error* $\epsilon^{(l)} = \boldsymbol{\phi} - \boldsymbol{\phi}^{(l)}$ and satisfies

$$\mathbf{A}\epsilon^{(l)} = \mathbf{r}^{(l)} . \tag{2.90}$$

Definition 2.2.15 (Iteration convergence, stopping criterion)**.** Consider an iteration method $\mathbf{M}\boldsymbol{\phi}^{(l+1)} = \mathbf{N}\boldsymbol{\phi}^{(l)} + \mathbf{b}$. At convergence, i.e., $\boldsymbol{\phi}^{(l+1)} = \boldsymbol{\phi}^{(l)} = \boldsymbol{\phi}$, it holds that $\mathbf{M}\boldsymbol{\phi} = \mathbf{N}\boldsymbol{\phi} + \mathbf{b}$, i.e., $\mathbf{M}\epsilon^{(l+1)} = \epsilon^{(l)}$ or $\epsilon^{(l+1)} = \mathbf{M}^{-1}\mathbf{N}\epsilon^{(l)}$. Thus, an iterative method *converges* if

$$\lim_{l \to \infty} \epsilon^{(l)} = 0 . \tag{2.91}$$

To be able to estimate the iteration error is therefore crucial for the decision when to stop iterating. The most common procedure to estimate a stopping criterion is to take the (normed) difference between two successive iterates. The iteration is then stopped when

$$\|\boldsymbol{\phi}^{(l+1)} - \boldsymbol{\phi}^{(l)}\| < \delta_{\text{tol}} \tag{2.92}$$

for a pre-defined tolerance value δ_{tol}. This criterion can be misleading if the difference

is small (e.g., for small values of $\boldsymbol{\phi}^{(l)}$) but the error, $\epsilon^{(l)} = \boldsymbol{\phi} - \boldsymbol{\phi}^{(l)}$, is not. Hence, the reduction of the residual, $\mathbf{r}^{(l)} = \mathbf{b} - \mathbf{A}\boldsymbol{\phi}^{(l)}$, is often used as it is accompanied by the reduction of the iteration error by $\mathbf{A}\epsilon^{(l)} = \mathbf{r}^{(l)}$. Thus, the iteration is stopped (after a maximal number of iterations or) after

$$\|\mathbf{r}^{(l)}\| < \delta_{\text{tol}}\,. \tag{2.93}$$

The simplest iteration method is the *Jacobi method* named after Carl Gustav Jacob Jacobi (cf. Jacobi (1845)) which only takes the previous iterates into account

$$\boldsymbol{\phi}_i^{(l+1)} = \frac{1}{\mathbf{A}_{ii}} \left(\mathbf{b}_i - \sum_{j=0}^{i-1} \mathbf{A}_{ij} \boldsymbol{\phi}_j^{(l)} - \sum_{j=i+1}^{N-1} \mathbf{A}_{ij} \boldsymbol{\phi}_j^{(l)} \right) , \tag{2.94}$$

for rows $i = 0, \ldots, N-1$ and $\mathbf{A}_{ii} \neq 0$. Adding weights, $\omega \in (0, 2)$, to the iteration method such that the new iterate also consists of parts of the previous iterate, i.e.,

$$\boldsymbol{\phi}_i^{(l+1)} = (1-\omega)\boldsymbol{\phi}_i^{(l)} + \frac{\omega}{\mathbf{A}_{ii}} \left(\mathbf{b}_i - \sum_{j=0}^{i-1} \mathbf{A}_{ij} \boldsymbol{\phi}_j^{(l)} - \sum_{j=i+1}^{N-1} \mathbf{A}_{ij} \boldsymbol{\phi}_j^{(l)} \right) . \tag{2.95}$$

can accelerate the convergence. This approach is called a *weighted Jacobi method* (cf. Saad (2003)) or *over- or under relaxation* dependent on the magnitude of the weights, $\omega > 1$ or $\omega < 1$, respectively.

For the $3D$ diffusion equation (2.53), in particular, with matrix $\mathbf{A} = \mathbf{I} - \Delta t \nu_{\text{eff}} \nabla^2$, unknown $\boldsymbol{\phi} = \mathbf{u}^{(2)}$ and right-hand side $\mathbf{b} = \mathbf{u}^{(1)}$ the (weighted) Jacobi method in one component reads

$$\begin{aligned} \left(u_{ijk}^{(2)}\right)^{(l+1)} = (1-\omega)\left(u_{ijk}^{(2)}\right)^{(l)} + \omega\beta \Bigg[& u_{ijk}^{(1)} + \alpha_x \left(\left(u_{i+1,j,k}^{(2)}\right)^{(l)} + \left(u_{i-1,j,k}^{(2)}\right)^{(l)} \right) \\ & + \alpha_y \left(\left(u_{i,j+1,k}^{(2)}\right)^{(l)} + \left(u_{i,j-1,k}^{(2)}\right)^{(l)} \right) \\ & + \alpha_z \left(\left(u_{i,j,k+1}^{(2)}\right)^{(l)} + \left(u_{i,j,k-1}^{(2)}\right)^{(l)} \right) \Bigg] , \end{aligned} \tag{2.96}$$

where the coefficients are defined as $\alpha = (\alpha_x, \alpha_y, \alpha_z)^\top := (\frac{\nu_{\text{eff}}\Delta t}{\Delta x^2}, \frac{\nu_{\text{eff}}\Delta t}{\Delta y^2}, \frac{\nu_{\text{eff}}\Delta t}{\Delta z^2})^\top$ and $\beta := 1/(1 + 2(\alpha_x + \alpha_y + \alpha_z))$ and the result of the convection step serves as initial guess $(u_{ijk}^{(1)})^{(0)}$. The velocities $v_{ijk}^{(2)}$ and $w_{ijk}^{(2)}$ are approximated analogously.

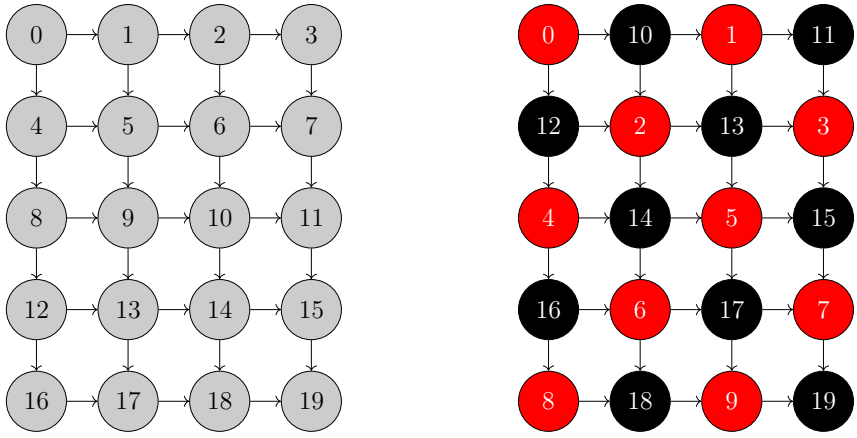
Since the calculation of the new iterate is data independent (only dependent on already calculated iterates), the (weighted) Jacobi method is perfectly suitable for

parallelization on modern architectures. However, the previous iterate must be retained until after the new iterate is constructed. Thus, both iterates need to be stored in different parts of the memory. The situation is different in the case of the *Gauss-Seidel method* (a *one-step method*, cf. Jeffreys and Jeffreys (1956)) with iteration specification taking the new iterate into account

$$\boldsymbol{\phi}_i^{(l+1)} = (1-\omega)\boldsymbol{\phi}_i^{(l)} + \frac{\omega}{\mathbf{A}_{ii}}\left(\mathbf{b}_i - \sum_{j=0}^{i-1}\mathbf{A}_{ij}\boldsymbol{\phi}_j^{(l+1)} - \sum_{j=i+1}^{N-1}\mathbf{A}_{ij}\boldsymbol{\phi}_j^{(l)}\right) \tag{2.97}$$

for rows $i = 0, \ldots, N-1$ and $\mathbf{A}_{ii} \neq 0$. The weighted version is also called *successive over-relaxation* (SOR). Due to its data dependency, the Gauss-Seidel method cannot directly be parallelized.

To parallelize the Gauss-Seidel method so-called coloring is needed resulting in the *Colored Gauss-Seidel method* (CGS, cf. Bertsekas and Tsitsiklis (1989)). Using two colors, red and black, the *Red-Black Gauss-Seidel method* restructures the row-major ordered linear equation system (cf. Fig. 2.14a) such that there exist no more data dependencies within one iteration step. Thereby, the matrix components are decomposed into two disjunct subsets, red for $0, \ldots, r-1$, and black for $r, \ldots, r+b+1$ with $r+b = n \cdot m$ for a matrix of size $n \times m$, such that red and black indices alternate (cf. Fig. 2.14b) without changing the solution.



(a) Row-major ordering (b) Red-black ordering

Figure 2.14: Schematic representation of matrix ordering (cf. Würzburger (2016))

The iteration specification for regular grids then reads

$$\left(\boldsymbol{\phi}_i^{(l+1)}\right)_r = \frac{1}{\mathbf{A}_{ii}} \left((\mathbf{b}_i)_r - \sum_{j \in N(i)} \mathbf{A}_{ij} \left(\boldsymbol{\phi}_j^{(l)}\right)_b \right) , \qquad i = 0, \dots r-1 \tag{2.98}$$

$$\left(\boldsymbol{\phi}_i^{(l+1)}\right)_b = \frac{1}{\mathbf{A}_{i+r,i+r}} \left((\mathbf{b}_i)_b - \sum_{j \in N(i)} \mathbf{A}_{i+r,j} \left(\boldsymbol{\phi}_j^{(l+1)}\right)_r \right) , \qquad i = 0, \dots b-1 , \tag{2.99}$$

where $N(i)$ denotes the respective neighbor indices. Therewith, first, the red iterates are calculated in parallel using the previous black iterates and, then, the new red iterates are used to calculate the new black iterates in parallel. Thus, fewer colors mean more parallelism but slower convergence. Although, CGS can converge faster than the Jacobi method (depending on the nature of the matrix $\mathbf{A}$ with non-zero diagonal elements, cf. Ferziger and Peric (2002)) it needs twice as many parallel steps resulting in a similar runtime (cf. Smart and White (1988); Tritsiklis (1989) and Würzburger (2016) for the implementation at hand).

In summary, the ease of implementation and parallelization of the simple Jacobi method compensate the fact that the convergence rate is slower than the rate of more sophisticated methods such as Krylov subspace methods (e.g., Conjugate Gradient method (CG, cf. Hestenes and Stiefel (1952)), BiConjugate Gradient method with stabilization (BiCGStab, cf. van der Vorst (1992)), Generalized Minimal Residual method (GMRES, Saad and Schultz (1986))) or Colored Gauss-Seidel methods which need to be adapted to be parallelized (e.g., due to data dependencies).

Sources: In the third step of the fractional scheme the external sources need to be added, $\partial_t \mathbf{u}^{(3)} = \mathbf{f}_B$. For the momentum equation (2.29) the buoyancy force, $\mathbf{f}_B = \mathbf{f}_B(T) = -\beta(T - T_0)\mathbf{g}$, depends on the temperature, which is obtained by solving the energy equation (2.30). Therefore, the explicit Euler scheme is used for fast calculation

$$\mathbf{u}^{(3)} = \mathbf{u}^{(2)} + \Delta t \mathbf{f}_B(T^{(n)}) \tag{2.100}$$

$$= \mathbf{u}^{(2)} - \Delta t \beta (T^{(n)} - T_0)\mathbf{g} . \tag{2.101}$$

Since the buoyancy force, $\mathbf{f}_B$, is evaluated at the previous time step, the energy equation (2.30) can be solved independently, contrarily to an implicit time stepping approach where the temperature approximation is needed simultaneously. Again, time stepping and grid resolution still need to be carefully chosen to obtain enough accuracy.

Pressure and Incompressibility: The last fractional step consists of two single steps, namely solving the pressure Poisson equation and correcting the velocity to be incompressible. Again using the explicit Euler scheme for (2.55), $\partial_t \mathbf{u}^{(n+1)} = -1/\rho_0 \nabla p$, yields

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(3)} - \frac{\Delta t}{\rho_0} \nabla p \,. \tag{2.102}$$

Applying the divergence operator on (2.102) gives

$$0 \overset{!}{=} \nabla \cdot \mathbf{u}^{(n+1)} = \nabla \cdot \mathbf{u}^{(3)} - \frac{\Delta t}{\rho_0} \nabla^2 p \tag{2.103}$$

$$\iff \nabla^2 p \overset{!}{=} \frac{\rho_0}{\Delta t} \nabla \cdot \mathbf{u}^{(3)} \tag{2.104}$$

with the incompressibility constraint (2.28), $\nabla \cdot \mathbf{u}^{(n+1)} = 0$. Since the fractions $\rho_0/\Delta t$ and $\Delta t/\rho_0$ cancel each other out in (2.102) and (2.104), it suffices to solve

$$\nabla^2 p = \nabla \cdot \mathbf{u}^{(3)} \tag{2.105}$$

for pressure p with known velocity $\mathbf{u}^{(3)}$ to get

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(3)} - \nabla p \tag{2.106}$$

with an orthogonal pressure projection of a (possibly) divergent velocity field $\mathbf{u}^{(3)}$ to a divergence-free field $\mathbf{u}^{(n+1)}$ (based on the Helmholtz-Hodge decomposition introduced by Chorin (1967, 1968), cf. Fig. 2.15).

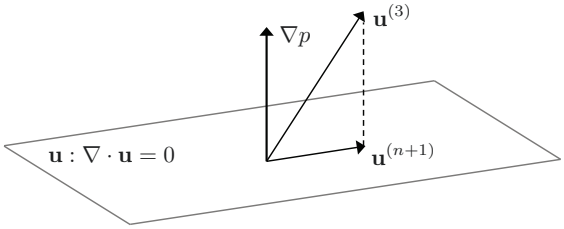


Figure 2.15: Schematic representation of pressure projection

Ideally, the pressure Poisson equation (2.106) should be solved exactly in order to satisfy the incompressibility constraint. Therefore, a more accurate scheme is applied

here which utilizes the Jacobi method whose computational stencil for pressure reads

$$p_{ijk}^{(l+1)} = (1-\omega)p_{ijk}^{(l)} + \omega\beta\Bigg[-\nabla\cdot\mathbf{u}^{(3)} + \alpha_x\left(p_{i+1,j,k}^{(l)} + p_{i-1,j,k}^{(l)}\right) + \alpha_y\left(p_{i,j+1,k}^{(l)} + p_{i,j-1,k}^{(l)}\right) + \alpha_z\left(p_{i,j,k+1}^{(l)} + p_{i,j,k-1}^{(l)}\right)\Bigg] \tag{2.107}$$

with $\omega < 1$, $\beta = 1/(2(\alpha_x + \alpha_y + \alpha_z))$ and $\alpha = (\alpha_x, \alpha_y, \alpha_z)^\top = (\frac{1}{\Delta x^2}, \frac{1}{\Delta y^2}, \frac{1}{\Delta z^2})^\top$. The spatial derivatives $\nabla\cdot\mathbf{u}^{(3)}$ and ∇p are hereby approximated with central differences. Other methods such as the colored Gauss-Seidel method could also be applied.

The iteration error $\epsilon^{(l)}$ of those iteration schemes can be expressed as (a mixture of) smooth and oscillating functions of the spatial coordinates. By applying an iteration method, the rapidly varying components of the iteration error can be removed and the error becomes a smooth function. If the error is smooth (such as with the weighted Jacobi method), the approximation to the iteration error (called *smoothing* or *relaxation*) can be calculated on a coarser grid (called *restriction*) with less computational effort and faster convergence. Then, the error approximation is interpolated (called *prolongation*) from the coarse to the fine grid to correct the fine grid solution by this error. This procedure is repeated until a coarse grid is reached, where the computational cost of a direct solution of the iteration error is negligible. Hereby, a collocated grid (as it is used for the problem at hand) is beneficial due to the ease of transfer of information between the various grids.

This approach describes the main idea of the *multigrid method* (MG) (cf. Fedorenko (1962, 1964); Bakhvalov (1966); Brandt (1973, 1977); Hackbusch (1977)): accelerate the convergence of a basic iteration method (such as Jacobi or Gauss-Seidel) by solving a computationally cheaper problem on a coarse grid and therewith correcting the fine grid solution globally. There are various ways to apply the multigrid method, for instance only two grids are used (called *two-level multigrid*) or more, starting with the finest grid, restricting to the coarsest then prolonging back to the finest grid (called *V-cycle*, cf. Fig. 2.16). Restricting to the coarsest grid, prolonging back only one level, restricting again once and prolonging two levels to restrict again to the coarsest grid and then repeating restriction and prolongation for one level before prolonging to the finest grid is called a *W-cycle*. Laying between the V-cycle and W-cycle is the *F-cycle*, whereby the restriction starts to the coarsest grid and then after having reached each level for the first time in the prolonging process a restriction

to the coarsest grid is performed. Those cycles can also be repeated several times one after another.

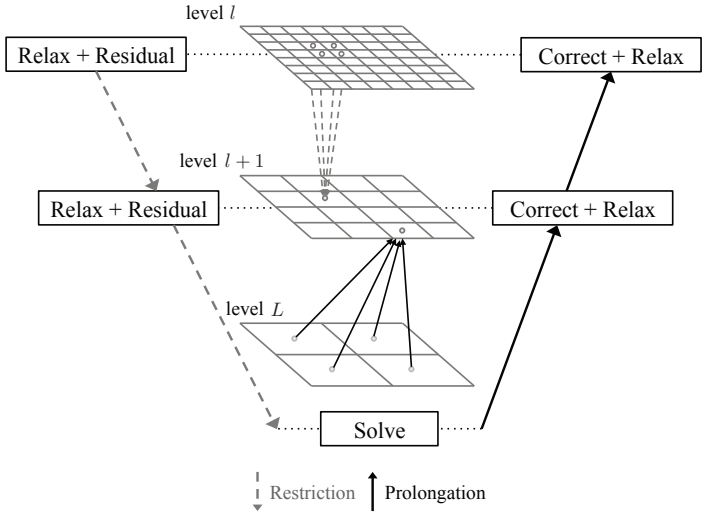


Figure 2.16: Schematic representation of a V-cycle in the multigrid method

Since Glimberg et al. (2009) showed that two V-cycles suffice to get faster convergence for the problem at hand than the Jacobi method itself, the V-cycle multigrid method is also used here whereat more level restrictions yield better convergence. Thereby, it is crucial to solve the pressure Poisson equation as accurate as possible in order to ensure incompressibility of the velocity field. Therefore, a pre-conditioning of the initial guess is applied in the first time step. Here, as many V-cycles in the multigrid method and iterations in the Jacobi method are applied as a residual tolerance, δ_{tol}, in the Jacobi iteration and a tolerance for the preceding pressure approximation (or a set maximum) is reached.

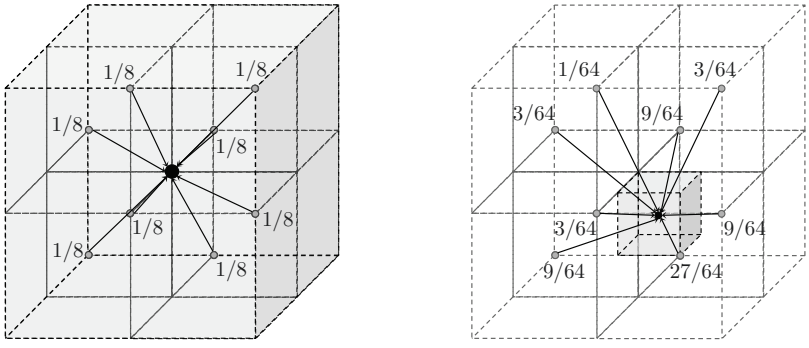
Definition 2.2.16 (V-Cycle Multigrid Method)**.** Let $\mathbf{A}_h\boldsymbol{\phi}_h = \mathbf{b}_h$ be the linear system of equations to be solved at the finest grid with resolution h. Further, let $\mathbf{R}_h^{2h}$ be a restriction operator taking a vector field from a fine grid with resolution h to a coarser grid with double the cell size in each direction, i.e., $\boldsymbol{\phi}_{2h} = \mathbf{R}_h^{2h}\boldsymbol{\phi}_h$.

Analogously, let $\mathbf{P}_{2h}^h$ be a prolongation operator taking a vector field from a coarse grid to a finer grid with half the cell size in each direction, i.e., $\boldsymbol{\phi}_h = \mathbf{P}_{2h}^h\boldsymbol{\phi}_{2h}$. Assuming that there are $l+1$ grids, $l \geq 0$, with finest grid $l = 0$ and coarsest grid L, where the number of cells is decreased by the power of two, the *V-Cycle multigrid method* is illustrated in Listing 2.1.

Listing 2.1: Pseudocode of the V-Cycle multigrid method

```
- Relax A_h φ_h = b_h with initial guess φ_h^(0).
- Compute the residual r_h = b_h − A_h φ_h and restrict to
  r_2h = R_h^2h r_h.
- Relax the residual equation A_2h ε_2h = r_2h with initial
  guess ε_2h^(0) = 0.
                .                   .
                .                   .
                .                   .
  - Solve the residual equation A_Lh ε_Lh = r_Lh with
    initial guess ε_Lh^(0) = 0.
                .                   .
                .                   .
                .                   .
- Prolongate the error ε_h = P_2h^h φ_2h.
- Correct the approximation φ_h = φ_h + ε_h.
- Relax A_h φ_h = b_h with the updated approximation φ_h as
    new initial guess.
```

In order to send the information between the grids, the restriction and prolongation operators need to be defined (cf. Fig. 2.17). Again, there exist multiple ways (neither correct nor incorrect) to define them (e.g., injection or full/ half weighting for restriction, (bi-/ tri-) linear interpolation for prolongation).



(a) Restriction (average)

(b) Prolongation (trilinear interpolation)

Figure 2.17: Schematic representation of restricting and prolonging cells in $3D$

Definition 2.2.17 (Restriction operator)**.** The applied restriction operator $\mathbf{R}_h^{2h}$ takes the average of all neighboring cells of the finer grid while the domain boundaries are fixed (cf. Fig. 2.17a), i.e.,

$$\mathbf{R}_h^{2h} = \frac{1}{2}\begin{bmatrix}1 & 1\end{bmatrix} \qquad \text{in } 1D \tag{2.108}$$

$$\mathbf{R}_h^{2h} = \frac{1}{4}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \qquad \text{in } 2D \tag{2.109}$$

$$\mathbf{R}_h^{2h} = \frac{1}{8}\left[\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \quad \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\right] \qquad \text{in } 3D \tag{2.110}$$

resulting for three dimensions in

$$\begin{aligned}\left(\boldsymbol{\phi}_{ijk}\right)_{2h} = \frac{1}{8}\Big(&\left(\boldsymbol{\phi}_{2i-1,2j-1,2k-1}\right)_h + \left(\boldsymbol{\phi}_{2i,2j-1,2k-1}\right)_h + \left(\boldsymbol{\phi}_{2i-1,2j,2k-1}\right)_h + \left(\boldsymbol{\phi}_{2i,2j,2k-1}\right)_h \\ &+ \left(\boldsymbol{\phi}_{2i-1,2j-1,2k}\right)_h + \left(\boldsymbol{\phi}_{2i,2j-1,2k}\right)_h + \left(\boldsymbol{\phi}_{2i-1,2j,2k}\right)_h + \left(\boldsymbol{\phi}_{2i,2j,2k}\right)_h\Big).\end{aligned} \tag{2.111}$$

Definition 2.2.18 (Prolongation operator)**.** The applied prolongation operator $\mathbf{P}_{2h}^h$ (cf. Fig. 2.17b) uses linear interpolation with weight $\omega = \frac{1}{4}$ in $1D$, bilinear interpolation $2D$ and trilinear interpolation in $3D$ (reusing the $1D$ weight in each direction). This interpolation results for three dimensions in

$$\begin{aligned}\left(\boldsymbol{\phi}_{2i,2j,2k}\right)_h = \frac{1}{64}\Big(&27\left(\boldsymbol{\phi}_{ijk}\right)_{2h} + 9\left(\boldsymbol{\phi}_{i+1,j,k}\right)_{2h} + 9\left(\boldsymbol{\phi}_{i,j+1,k}\right)_{2h} + 9\left(\boldsymbol{\phi}_{i,j,k+1}\right)_{2h} \\ &+ 3\left(\boldsymbol{\phi}_{i+1,j+1,k}\right)_{2h} + 3\left(\boldsymbol{\phi}_{i+1,j,k+1}\right)_{2h} + 3\left(\boldsymbol{\phi}_{i,j+1,k+1}\right)_{2h} \\ &+ \left(\boldsymbol{\phi}_{i+1,j+1,k+1}\right)_{2h}\Big).\end{aligned} \tag{2.112}$$

Since the restriction and prolongation operations divide the grid by two for each level in each direction (without considering boundary cells), the grid at zero level needs to comply with $N_x = 2^o + 2$, $N_y = 2^p + 2$ and $N_z = 2^q + 2$ for $o, p, q \in \mathbb{N}^0$. Then the maximal number of levels is $l = \max(o, p, q)$ for the last level only consisting of one cell or $l = \max(o-1, p-1, q-1)$ for remaining two cells in each direction (given that the exponents (minus one) smaller than l are set to one when there is no more restriction possible in the respective direction).

For the pressure Poisson equation (2.105) it holds that $\mathbf{A} = \nabla^2$, $\phi = p$ and $\mathbf{b} = \nabla \cdot \mathbf{u}^{(3)}$ in Definition (2.2.16) of the V-cycle. Since the matrix corresponds to the

Laplacian operator, it never changes. Thus, the matrix is never actually constructed and the weighted Jacobi method with $\omega = 2/3$ can be reused for relaxation which is stable for the Poisson equation.

Further, boundary conditions need to be applied in each step of the multigrid method. At the finest level pressure boundary conditions are set, whereas at coarser grids boundary conditions for the errors need to be set. Depending on the nature of the pressure boundary condition (Dirichlet or Neumann), the error boundary condition is set to match with values $\epsilon = 0$ or $\partial_n \epsilon = 0$. In case of inner boundary conditions for obstacles, special care has to be taken to prevent the obstacles to vanish. See Section 3.2.2 for details.

As already mentioned, the solution method of fractional steps with its methods to approximate the single steps can now be reused for the energy equation (2.30) and concentration equation (2.31) with the same advantages (stability, ease of implementation, memory usage, parallelism) and disadvantages (low-order of accuracy, convergence, numerical dissipation). Although the solution method is unconditionally stable (cf. Stam (1999)), time stepping and grid resolution still need to be carefully chosen to obtain enough accuracy due to the applied low-order schemes which result in a general first-order solution method (in time and space).

The possibly occurring uncertainties and errors which determine the accuracy of the solution method are explained in the next section in more detail.

2.3 Uncertainties and Errors in CFD

The results of computational fluid dynamic simulations may differ from their exact values since uncertainties and errors occur. In the following section, the terms uncertainty and error are characterized using the AIAA (1998) Standards and error terms are partitioned further.

Definition 2.3.1 (Uncertainty and error)**.** In the AIAA (1998) Standards, *uncertainty* is defined as "a potential deficiency in any phase or activity of the modeling process that is due to the lack of knowledge", whereas *error* is "a recognizable deficiency in any phase or activity of modeling and simulation that is not due to lack of knowledge".

Therewith, uncertainty, on the one hand, is only a potential source of imbalance if physical processes and parameters are not well understood. Uncertainty (and sensitivity) analyses can be applied in order to better determine the uncertainty in

the modeling phase. Therefore, a number of simulations with a variety of models can be run to determine how modeling affects the results. This uncertainty quantification, however, would extend the scope of this work.

Errors, on the other hand, are a known source of imbalance identifiable upon examination and are classified further into acknowledged and unacknowledged errors (cf. Ferziger and Peric (2002)).

2.3.1 Unacknowledged Errors

Unacknowledged errors include computer programming errors or usage errors.

Computer Programming Errors: These mistakes made while programming the code are the responsibility of the programmer(s) and can be discovered by systematically verifying (parts of) the code (by unit tests). *Verification* is thereby defined as the "process of determining that the implementation of a calculation method accurately represents the developer's conceptual description of the calculation method and the solution of the calculation method" (cf. ASTM (2005)).

Usage Errors: When applying the code, the user chooses the models (if applicable), defines the grid and time resolution, selects the boundary and initial conditions and sets the (physical) input parameter used in a simulation (to a defined extent) establishing the accuracy of the simulation. Here, and in the post-processing process user or usage errors can occur which can (to a certain level) be controlled through training and a detailed user guide documentation.

2.3.2 Acknowledged Errors

Acknowledged errors are further divided into physical, numerical and computer errors.

Computer Errors: Round-off errors occur when computers store floating point numbers at a certain accuracy (e.g., with $16, 32$, or 64 bits). These errors are not considered significant compared to other errors such as numerical errors.

Numerical Errors: Numerical errors can be further characterized by discretization and iteration errors. The *iteration* error is defined as the difference between the exact and the iterative solution of the discretized equations (cf. Def. (2.2.14)). This error occurs since the iteration process is stopped after a defined convergence criterion

based on the sum of residuals (cf. Def. (2.2.15)). The tolerance thereby determines the accuracy in case the error tends towards zero since the norm of residuals is tightly connected to the error.

Discretization errors are referred to as the difference between the exact solution of the governing equations and the exact solution of the discrete approximation. Due to the truncation error (cf. Def. (2.2.3)), the discretization error can only be measured by comparing discrete approximations on refined grids. Thereby, the quality of an approximation is described by its order p (cf. Def. (2.2.4)) which relates the truncation error to the grid spacing to the power of p (here, first-order due to first-order time discretization and linear interpolation in the advection scheme). It indicates, how the error changes when changing the grid spacing, hence it is not directly measuring the magnitude of the error. The order of a method can be determined for example with Richardson extrapolation (cf. Def. (2.2.9)) and can be related to the discretization error through Definition (2.2.10).

Also, local and global errors need to be distinguished. *Local errors* occur at each cell and are transported (through advection and diffusion) throughout the grid causing numerical dissipation and dispersion (cf. Def. (2.2.12)). *Global errors* refer to errors over the entire domain (in space and time). Errors can thereby be measured by different norms, for instance by the root mean square (RMS), where the squared difference of the numerical to the analytical solution at the center of the domain, $\mathbf{x}_c$ with corresponding cell index (i, j, k), is summed over all time steps, then weighted by the number of time steps and, finally, the square root is taken:

$$\epsilon_{RMS} \coloneqq \sqrt{\frac{1}{N_t} \sum_{n=1}^{N_t} \left[\phi_{ijk}^{(n)} - \phi(\mathbf{x}_c, t^{(n)})\right]^2} \tag{2.113}$$

or the summed and squared difference is taken at fixed time step n over the whole domain

$$\epsilon_n \coloneqq \sqrt{\frac{1}{N} \sum_{i=1}^{N_x-1} \sum_{j=1}^{N_y-1} \sum_{k=1}^{N_z-1} \left[\phi_{ijk}^{(n)} - \phi(\mathbf{x}_{ijk}, t^{(n)})\right]^2} \tag{2.114}$$

with $N \coloneqq (N_x - 2)(N_y - 2)(N_z - 2)$ being the total amount of inner cells.

At $t^{(N_t)} = t_{\text{end}}$, $\epsilon_{N_t} =: \epsilon_{\text{abs}}$ describes the *absolute error* and thereof, the *relative*

error is defined by

$$\epsilon_{\text{rel}} := \frac{\epsilon_{\text{abs}}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N_x-1}\sum_{j=1}^{N_y-1}\sum_{k=1}^{N_z-1}\phi(\mathbf{x}_{ijk}, t_{\text{end}})^2}}\,. \tag{2.115}$$

Physical Modeling Errors: The *modeling error* is defined as the difference between the real flow and the exact solution of the mathematical model. This error cannot be quantified since the real flow is not known and the exact solution might not be defined. However, the error does occur due to assumptions or deliberate simplifications in the continuum model, as for the result of a more efficient computation. For the problem at hand, the following assumptions and simplifications depict an extract:

- Newton's laws of conservation and the First Law of Thermodynamics,
- Constant, but uncertain properties,
- Boussinesq approximation in the buoyancy force,
- Bernoulli's principle and its assumptions (i.e., incompressibility and negligible friction due to viscous forces),
- Setting of initial and boundary conditions as well as omitting boundary layer equations,
- Simplified turbulence model (Constant Smagorinsky-Lilly with uncertain parameters),
- No models for combustion, radiation or pyrolysis.

Under the assumption of Cartesian grids, details in complex geometries are further neglected for the ease of representing the geometry.

A way of estimating physical modeling errors is the concept of validation. Thereby, *validation* is defined as "the process of determining the degree to which a calculation method is an accurate representation of the real world from the perspective of the intended use of the calculation method" (cf. ASTM (2005)). Therefore, a comparison of the solution data of the calculation method with experimental data is performed. Here, it is crucial to choose suitable experiments which reflect the physical model well within the limitations of the calculation method. If this mapping is not possible, the differences and made assumptions to approximate the numerical setup to the experimental setup need to be discussed. Accompanied by validating the underlying models with experiments additional experimental errors occur.

Experimental Errors: In experimental studies, two types of errors occur – systematic and random errors. *Systematic errors* affect the accuracy of the measurement (meaning how close a measured value is to the true or accepted value). Common sources of systematic errors are incorrect calibration of measuring instruments, their maintenance, or faulty readings of instruments by the user. *Random errors*, in turn, affect the precision of a measurement, i.e., how closely repeated measurements agree with each other. Common sources of random errors are fluctuating readings during measurements or extrapolation or estimation of quantities. Experimental accuracy is reported either as significant figures (i.e., the least significant digit which can be measured using the instrument), as percentage error (i.e., fractional difference between the measured and true or accepted value or repeated measured values in case of precision) or mean and standard deviation (for repeated measurements).

The experimental errors affect the set values of all physical parameters in the simulation and therefore influence the accuracy of the simulation as well as all of the above explained errors. Since there are ways to identify some of the errors (and possibly reduce them), unit testing, verification, and validation are applied throughout the development process. Before doing so in Chapter 4, the introduced solution method needs to be implemented on modern architectures.

Chapter 3

Simulation of Smoke Propagation on CPU

The design concept of the solution method derived in Section 2.2 is fully adapted to highly parallel computer architectures and accelerators like GPUs by choosing the numerical schemes to match the target hardware with a deliberate decision for a reduced modeling approach. Thus, the next step in the development cycle is to translate the numerical solution method into a computer code, called JuROr (*Jülich's Real-Time simulation within ORPHEUS*). Since JuROr shall run on various modern architectures (open to the end-user to choose from CPU or GPU) with comparable performance, the implementation uses the pragma-oriented OpenACC programming model (cf. Chapter 5) applied to a C++ code base. This model has several advantages: ease of implementation and maintenance of only one source code, performance portability on various (multicore) CPU and GPU architectures making the application independent on the hardware (cf. Section 5.2), and speedup towards real-time (cf. Chapter 6). To start, a $2D$ code is first implemented in C++ for simple geometries and then expanded to a $3D$ code including inner boundaries (cf. Section 3.2.1). The code design is intended to be simply structured using interfaces and, therefore, easy to maintain and develop by using the free and open source distributed version control system git (cf. Hamano et al. (2005)). To verify new methods, unit tests are set up with analytical solution scenarios which need to be applied whenever a new method or model is implemented. After CPU implementation, the code is then verified and the model is validated in Chapter 4 before porting JuROr to GPU in Chapter 5.

This chapter is designed to resemble the code development process using a CPU.

3.1 Code Structure Using Interfaces

JuROr is developed as an open source software basis using an object-oriented design complying with the SOLID principle of class design. Thereby, SOLID follows the set of rules outlined in Table 3.1 (cf. Hunt and Thomas (1999); Martin (2003) and Liskov (1987)).

Table 3.1: Five rules of the SOLID principle

	Principle	Description
S	Single Responsibility	Use independent components with a single purpose.
O	Open-Closed	Entities are open for extension, closed for modification.
L	Liskov Substitution	Subtypes must be substitutable for their base types.
I	Interface Segregation	Interfaces have a single, well defined purpose.
D	Dependency Inversion	Depend on abstractions, not on concretions.

These rules ensure that the software's complexity is reduced by managing interdependencies, and that it is adaptable to frequent changes, hence maintainable, flexible and reusable for multiple developers. In JuROr, these rules are met using the behavioral strategy pattern. This pattern captures the abstraction in an interface, burying implementation details in derived classes realizing the interface (open-closed, no repetition, single responsibility), but being open for additional add-ons. The classes obey behavioral inheritance (Liskov substitution) and the interfaces have only a single purpose (interface segregation). Further, macros and singletons are used fulfilling the dependency inversion principle. Singletons, thereby, ensure that a class only has one instance and provide a global point of access to it.

In total, JuROr consists of approximately 22 000 lines of code in 35 classes with four miscellaneous classes or headers. Based on the derived numerical model, these classes can be structured into four levels:

1. A time integration scheme starting the solver, measuring clock time, launching analyses and visualization,
2. A solution scheme with an interface for various solvers defining the problem and fractional steps to be solved,
3. Numerical methods for single fractional steps abstracted by suitable interfaces,
4. Auxiliaries and libraries for pre- and post-processing (reading, writing, analyses, and visualization), macros, and data structures.

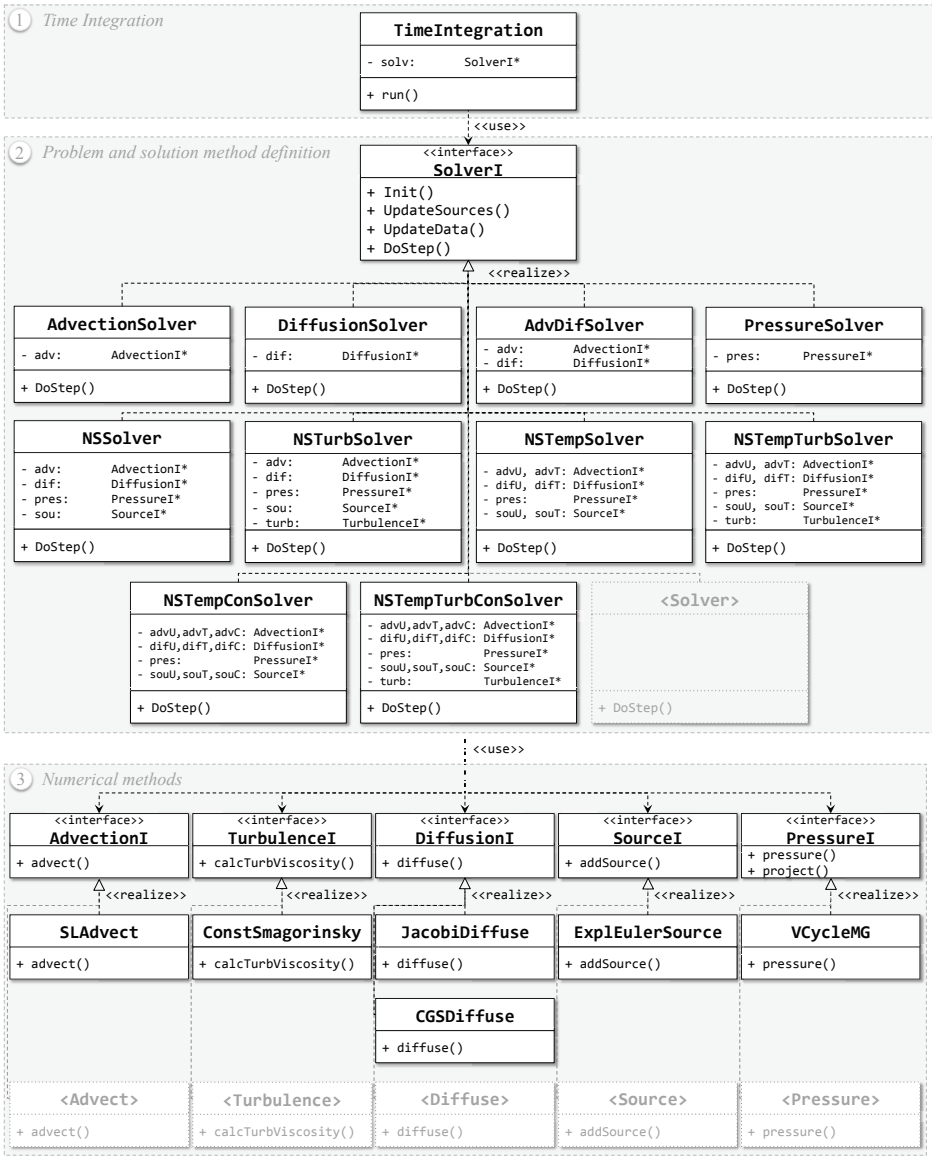


Figure 3.1: Class diagram of the CPU implementation using interfaces

Figure 3.1 illustrates these four levels and therefore the structure of the implementation in a class diagram. After parsing the user input data provided by an `XML` (Extensible Markup Language) file, the problem is defined with initial and boundary

conditions, and physical as well as numerical parameters are set.

Based on this input, the variables are parsed and initialized and the class named `TimeIntegration` starts the time integration scheme (1) with its function `run()`. Here, boundary conditions are first set, and then the solution scheme (fractional steps) is started by calling the virtual function `DoStep()` of the interface class `SolverI`. After each time step, the temperature dependent variables (e.g., buoyancy force) and all temporary variables are updated to the newly calculated variables. This update needs to be done since the variables at time step n serve as input data to calculate the new variables at time step $n+1$. To measure the wall-clock time (i.e., the time the calculation actually needs) time stamps (for start and end) are set as well. Further, analyses (such as RMS error calculations or difference to an analytical solution if applicable), visualization and saving data output (cf. Fig. 3.2) is handled in the `TimeIntegration` class as well and can be shut off for speedup calculations (cf. Chapter 6 and Appendix A.1).

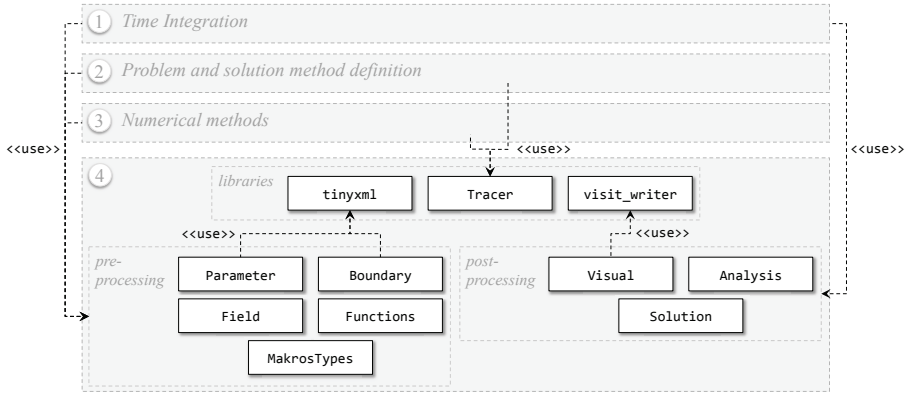


Figure 3.2: Auxiliary and library classes for pre- and post-processing

The fractional steps of the solution scheme (2) are defined via the interface `SolverI`, more precisely in the virtual function `DoStep()`. By employing an interface, the solution method can be realized as flexible as possible allowing for different problems such as an `AdvectionDiffusionSolver` with pure diffusion and advection (without external sources) or more complex problems such as a turbulent Navier-Stokes problem with thermally driven buoyancy force, as the `NSTempTurbSolver` whose implementation is exemplarily shown in Figure 3.3 as a flow chart. With the structural advantage of interfaces, new solutions methods can easily be added.

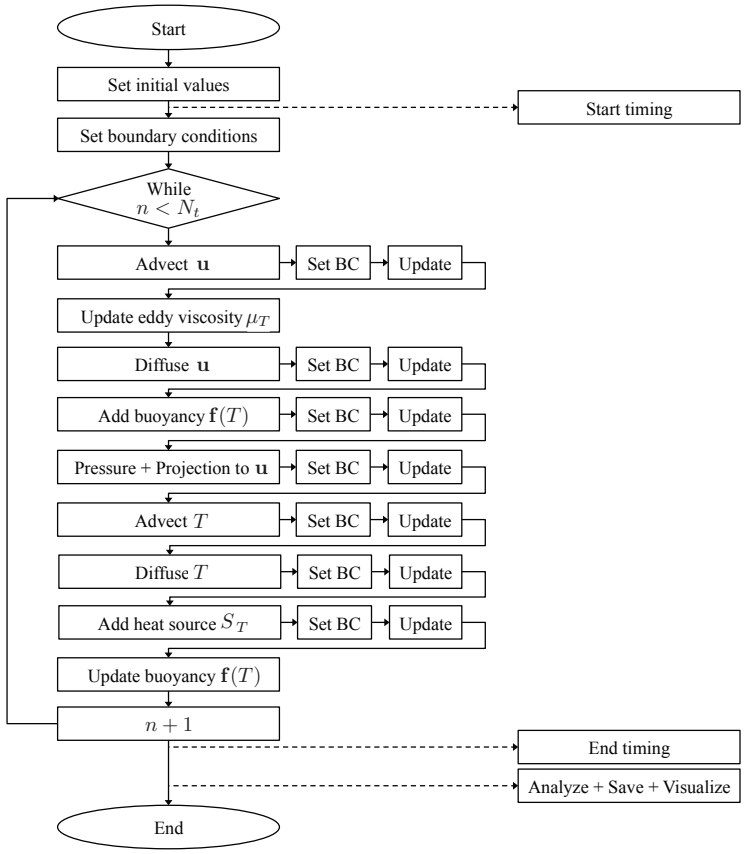


Figure 3.3: Flow chart of an implemented fractional step solution method

The way the single fractional steps are numerically solved is defined by interfaces for each step in the numerical methods (3) (e.g., advection, diffusion, calculating pressure or turbulent viscosity, adding sources) where a pool of methods is already realized; again being open for further add-ons (cf. Fig. 3.1). Additionally, auxiliary classes (4) are implemented for pre- and post-processing (cf. Fig. 3.2), e.g., for applying boundary conditions (in `Boundary`), performing analyses (in `Analysis`, `Solution`), reading input, setting values or writing output (in `Parameters`, `Functions`, `Visual`) and defining certain structures (in `Field`, `GlobalMacrosTypes`), which in turn use supporting libraries to read data from an `XML` file via `tinyxml` and to write data to `vtk` (visualization toolkit)-format for visualization via the `visit_writer` library.

In order to apply the code to real-world scenarios, it is extended for $3D$ geometries including inner obstacles.

3.2 Extension to Complex Geometries in $3D$

Including the third dimension as well as allowing inner boundaries elevates the CFD code, JuROr, to model complex geometries, such as rooms and buildings, as well as flow around obstacles. Therefore, a change of concept was performed regarding the way of iterating through the computational domain.

3.2.1 From $2D$ to $3D$ domains

In $2D$, the spatial grid is built up by two nested, but independent loops along each direction complying a row-major approach. In order to also model inner boundaries, a different approach other than simply adding a third inner loop to include a third dimension (as in Listing 3.1) is needed. This different approach is again justified by efficiency and flexibility since a query (if a cell is an inner boundary) inside a loop or using multiple loops with different start and end points is not efficient – neither on CPU nor on GPU. Further, most compilers use the most inner loop (here the z-dimension) as kernel dimension for parallelization slowing down the performance when using a coarse grid in z-direction.

Listing 3.1: $3D$ loops through the spatial domain

```
1 // including domain boundaries
2 for (size_t i = 0; i < Nx; ++i){
3   for (size_t j = 0; j < Ny; ++j){
4     for (size_t k = 0; k < Nz; ++k){
5     ...
6     }
7   }
8 }
```

The new concept is realized by replacing the three nested, independent loops by lists of indices (cf. Fig. 3.4 and Listing 3.2) representing either the domain interior (very light gray) with `iList`, the domain boundary (light gray) with `bList`, inner boundaries (so-called *obstacles* in dark gray) with `oList` or patches on the domain boundary (so-called *surfaces* in medium gray) with `sList`. For each type of boundary, there also exist lists for each direction (front and back in z-direction; top and bottom in y-direction; left and right in x-direction) to set the relevant boundary conditions more easily (cf. Section 3.2.2). In C++ the lists are implemented using `std::vector` because

setup, sorting and accessing vectors are most efficient using this type of container. Since the indices do not change during the calculation, i.e., obstacles are static, they can be initialized directly after parsing the `XML` input file containing all information about the problem, numerics and geometry (cf. Appendix A.3 for an example `XML` file).

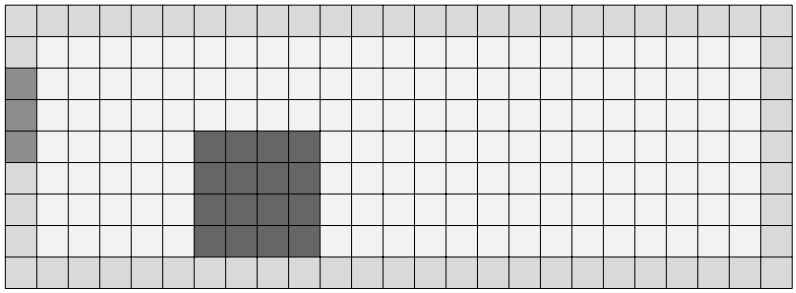


Figure 3.4: Small-scale exemplary illustration of index lists (very light gray: interior, light gray: boundary, dark gray: obstacle, medium gray: surface)

Listing 3.2: $1D$ row-major lists

```
// going through inner indices (iList)
for (auto idx: iList){
  ...
}
// going through domain boundary indices (bList)
for (auto idx: bList){
  ...
}
// going through obstacle indices (oList)
for (auto idx: oList){
  ...
}
// going through surface indices (sList)
for (auto idx: sList){
  ...
}
```

Thereby, all obstacles need to be defined as a rectangle spanned by its two opposing corners (front-bottom-left and back-top-right). Adding multiple obstacles (e.g., for a

combination of obstacles) and surfaces is possible in order to build a more complex setup. This approach results in index lists for every single obstacle or surface. To assure that the combination of adjacent obstacles is waterproof, i.e., there is no gap where fluid can escape, and to avoid that the boundary conditions for each direction do not interfere with each other, all duplicate indices are marked in order to ensure that the respective boundary condition is only applied on cells with no obstacle neighbor (cf. Fig. 3.5, where black arrows mark duplicates, and gray arrows show duplicate-free cells).

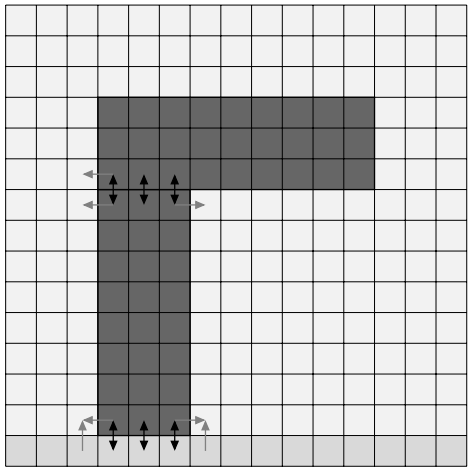


Figure 3.5: Small-scale exemplary illustration of checking duplicate indices (gray arrow: not marked, black arrow: marked as duplicate)

In order to still model 2D cases such as the unit tests in the upcoming section, the number of (inner) cells in the z-direction is set to one such that the total number of cells equals three including one ghost cell on each side, i.e., $N_z = 3$.

3.2.2 Inner and Outer Boundary Handling

Now being able to also model inner boundaries with the help of index lists, the boundary handling within the numerical methods such as the multigrid method as well as setting the boundary conditions need to be further refined. As for the domain boundaries (cf. Section 2.1.1) there also exist Dirichlet and Neumann boundary conditions for the inner boundaries. For the domain boundaries the discrete value of the

boundary (ϕ_{out}) depends on the discrete value of the adjacent inner value (ϕ_{in}), i.e.,

$$\phi_{\text{out}} = -\phi_{\text{in}} + 2c \quad \text{for a Dirichlet condition on the outer boundary} \tag{3.1}$$

$$\phi_{\text{out}} = \phi_{\text{in}} + c\Delta s \quad \text{for a Neumann condition on the outer boundary,} \tag{3.2}$$

where c is the prescribed constant value at the boundary or the gradient, respectively, and $\Delta s \in \{\Delta x, \Delta y, \Delta z\}$ is the grid spacing in the direction of the regarded boundary (i.e., $\Delta s = \Delta x$ for left and right domain boundary, $\Delta s = \Delta y$ for top and bottom domain boundary, and $\Delta s = \Delta z$ for front and back domain boundary). Since a collocated grid is used, the physical boundary lies between two cells. Therefore, the Dirichlet condition represents setting the average between the two adjacent cells to the desired value, c, at the boundary. The Neumann condition represents a spatial derivative in direction of the normal vector, thus the BD or FD schemes are used in discrete space depending on the considered boundary. Since the boundary condition is applied sequentially to all the boundary directions, the corner and edge cells are therewith set to the average based on all adjacent cells in the inner domain.

For inner boundaries, the discrete Dirichlet condition remains unchanged, whereby the inner adjacent cell is part of the inner domain, not the inside of the obstacle. The Neumann condition is changed since the inner adjacent cell lies in the opposite direction as before regarding the domain boundary, i.e., changing the direction of the normal

$$\phi_{\text{out}} = -\phi_{\text{in}} + 2c \quad \text{for a Dirichlet condition on the inner boundary} \tag{3.3}$$

$$\phi_{\text{out}} = \phi_{\text{in}} - c\Delta s \quad \text{for a Neumann condition on the inner boundary.} \tag{3.4}$$

The boundaries of the surfaces are treated exactly as the outer boundaries since the former are a subset of the latter.

As addressed in the previous section, applying boundary conditions on obstacles, surfaces or domain boundaries being in direct contact with other obstacles needs special treatment. To assure that a boundary condition is only applied on cells with no obstacle as input cell, ϕ_{in}, all boundaries need to be checked for adjacent boundary cells. Therefore, for each type of boundary, each direction (front, back, top, bottom, left, right), and each instance (in case there are multiple obstacles or surfaces), it is checked if the respective input cell, ϕ_{in}, is already marked as inner or outer boundary (cf. Fig. 3.5 with black arrows indicating conflicting boundary cells). If it is the case the respective boundary cell is marked and the boundary condition (for the direction

at hand) is not applied to this boundary cell. This way it is avoided that multiple boundary conditions (for the different directions) are applied to the same boundary cell.

Applying boundary conditions on the restricted levels of the multigrid method also needs special care when introducing obstacles. If inner boundaries are fully ignored at the restricted levels, the fluid penetrates the obstacle and causes incorrect movement downstream the obstacle (cf. Glimberg et al. (2009)). If inner boundaries are restricted naively with injection (being an obstacle cell at the next level only if the restricted cell was an obstacle at the previous level), thin obstacles might vanish, again causing the fluid to penetrate obstacles. To avoid this penetration, a dominant restriction (and prolongation) is employed (cf. Fig. 3.6): if any of the eight neighboring cells forming the restricted cell is marked as an obstacle (or surface) the restricted cell is also marked as obstacle (or surface, respectively) and is hence included in the appropriate index list of that level and type of boundary. Since dominant restriction does not prevent the obstacles become one cell thick, either the resolution needs to be increased, or the obstacle needs to be thick enough from the beginning (which is most often the case in fire-related problems). Preventing the obstacle cells to fully occupy the computational domain (also at restricted levels) should additionally be avoided. Here, the check for restricted outer and inner domain boundary cells lying adjacent to each other is also necessary at each level of the multigrid scheme. This check is done in the same manner as described at the finest level.

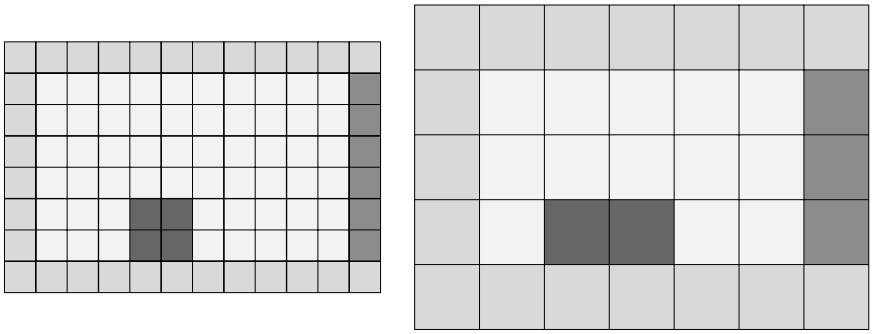


Figure 3.6: Small-scale exemplary illustration of dominant restriction in 2D (left: fine grid before restriction, right: coarse grid after restriction)

In order to test the implementation at each step, unit tests for most of the fractional steps have been integrated (cf. Appendix A for compiling, developing and running the code).

3.3 Intermediate Code Testing via Unit Tests

During code development it is essential to test newly implemented parts of the code. Therefore, unit tests are created to isolate each part of the solver and determine whether they are fully operational (cf. Kolawa and Huizinga (2007)).

3.3.1 Unit Test Advection

Applying (linear) advection to a fluid with velocity $\mathbf{u}$ and bulk velocity $\mathbf{c} \neq \mathbf{0}$, i.e.,

$$\partial_t \mathbf{u} + (\mathbf{c} \cdot \nabla)\mathbf{u} = 0\,, \tag{3.5}$$

results in the transport of fluid in the direction of the bulk velocity. Solving this convection equation numerically with the Semi-Lagrangian scheme results in additional artificial diffusion (dissipation) as explained in Section 2.2. Thus, the numerical solution to the linear advection equation (3.5) experiences additional diffusive behavior. Here, the test case solves the advection equation with initial fluid velocity distribution

$$\mathbf{u}_{ijk}^{(0)} := A \exp\left(-\frac{1}{2\sigma^2}\left[\left(\frac{x_i - x_0}{c_x} - t\right)^2 + \left(\frac{y_j - y_0}{c_y} - t\right)^2 + \left(\frac{z_k - z_0}{c_z} - t\right)^2\right]\right)_{|t=0}\,, \tag{3.6}$$

where $A = 1$ describes the amplitude of the Gaussian curve, the center of the curve is denoted by $(x_0, y_0, z_0) = (1.025, 1.025, 0.5)\,\mathrm{m}$ within the computational domain $[0, 2]^2 \times [0, 1]\,\mathrm{m}^3$, $(c_x, c_y, c_z) = (0.50, 0.50, 0.25)\,\mathrm{m/s}$ describes the bulk velocity and $\sigma \approx 0.03\,\mathrm{m}$ determines the width of the curve. The boundary in each direction adheres to the no-slip boundary condition $\mathbf{u} = \mathbf{0}\,\mathrm{m/s}$ and space as well as time are discretized with resolutions $(\Delta x, \Delta y, \Delta z) = (0.05, 0.05, 1.00)\,\mathrm{m}$ and $\Delta t = 0.001\,\mathrm{s}$, respectively (for more details see Tab. B.1 in Appendix B).

Figure 3.7 shows the fluid velocity in x-direction at $t = 0\,\mathrm{s}$ (on the left) and at $t = 1\,\mathrm{s}$ (on the right) in every grid cell (on the top) and as a profile along the diagonal (on the bottom). The fluid travels in the direction of the bulk velocity (upper right corner) but also gets diffused through artificial diffusion. This diffusion is characterized by a decreased amplitude and an increased width as indicated in the right column of Figure 3.7 by the color range and ordinate which are reduced to 1 % of the original. Therefore, the convection unit test demonstrates the expected behavior.

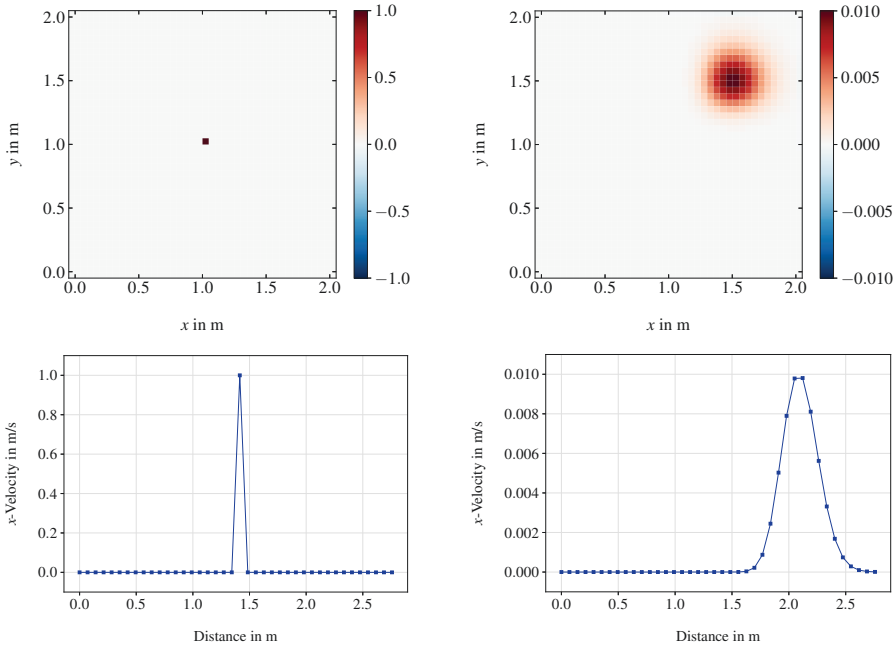
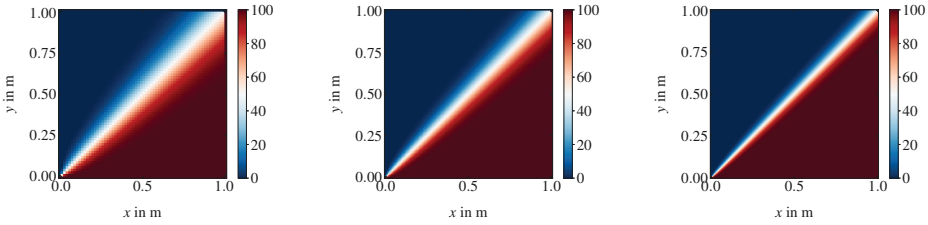


Figure 3.7: Unit test for (linear) advection showing the x-velocity (top: pseudocolor plot, bottom: curve along diagonal, left: at initial stage, right: at $t = 1\,\mathrm{s}$ with different color and ordinate range (!))

The spatial (absolute) error after $t_{\mathrm{end}} = 2\,\mathrm{s}$ over the whole computational domain is of order $\epsilon \approx 2.85 \times 10^{-4}$. Hereby, the resulting artificial diffusion can be reduced by increasing the grid resolution as seen in Figure 3.8, where a passive scalar (here temperature) is transported for $t_{\mathrm{end}} = 10\,\mathrm{s}$ with advection due to cooled walls on the left and top ($T = 0\,^{\circ}\mathrm{C}$) and heated walls ($T = 100\,^{\circ}\mathrm{C}$) on the right and bottom. The bulk velocity is set to $\mathbf{c} = (2, 2, 0)\,\mathrm{m/s}$.



(a) Coarse grid (b) Medium grid (c) Fine grid

Figure 3.8: Test for artificial diffusion

Here, different grid resolutions from a coarse grid with $(\Delta x, \Delta y) \approx (1.6, 1.6)\,\mathrm{cm}$, over a medium resolution with $(\Delta x, \Delta y) \approx (0.8, 0.8)\,\mathrm{cm}$ to a fine grid with a small cell spacing of roughly $(\Delta x, \Delta y) \approx (0.4, 0.4)\,\mathrm{cm}$, are tested, whereas the grid size of 1.0 m in z-direction and time stepping remain unchanged with $\Delta t = 0.001\,\mathrm{s}$ (cf. Tab. B.2 in Appendix B). The numerical solution approximates a sharp distribution along the diagonal of the computational domain of $[0, 1]^3\,\mathrm{m}^3$ the better, the higher the grid resolution is set (cf. Fig. 3.9). The mixing of temperatures along the diagonal represents the artificial diffusion, which has a greater influence on the numerical solution the coarser the grid is.

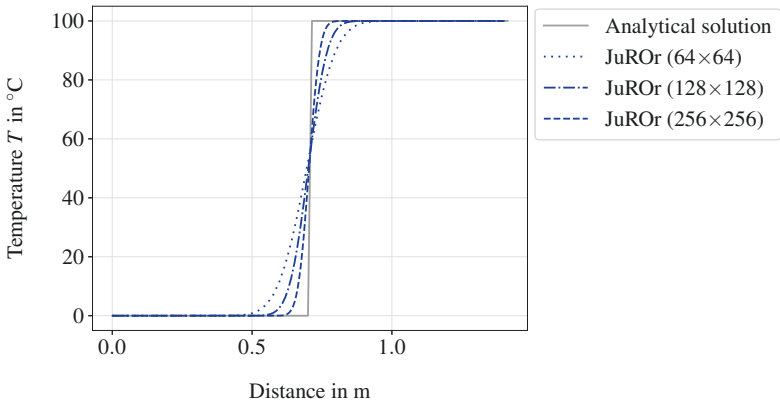


Figure 3.9: Artificially diffused temperature profile along the diagonal

3.3.2 Unit Test Diffusion

a) A manufactured solution for the diffusion equation, $\partial_t \mathbf{u} = \nu \nabla^2 \mathbf{u}$, is a multiplication of the exponential function and sinuses

$$\mathbf{u}^{\mathrm{ana}}(\mathbf{x}, t) := A \exp(-3l^2 \nu t) \sin(lx) \sin(ly) \sin(lz)\,. \tag{3.7}$$

Solving the diffusion equation numerically for $t_{\mathrm{end}} = 1\,\mathrm{s}$ with time step size $\Delta t = 0.0125\,\mathrm{s}$ on a domain of size $[0, 2]^2 \times [0, 1]\,\mathrm{m}^3$ with grid cell sizes of 0.05 m for Δx and Δy, and $\Delta z = 1.00\,\mathrm{m}$, kinematic viscosity $\nu = 1 \times 10^{-3}\,\mathrm{m}^2/\mathrm{s}$, amplitude $A = 1$, and wave number $l = 5/2\pi$ (so that the number of extremal points per direction equals five), shows the expected diffusive behavior (i.e., smoothing in regions of steep gradients, cf. Fig. 3.10).

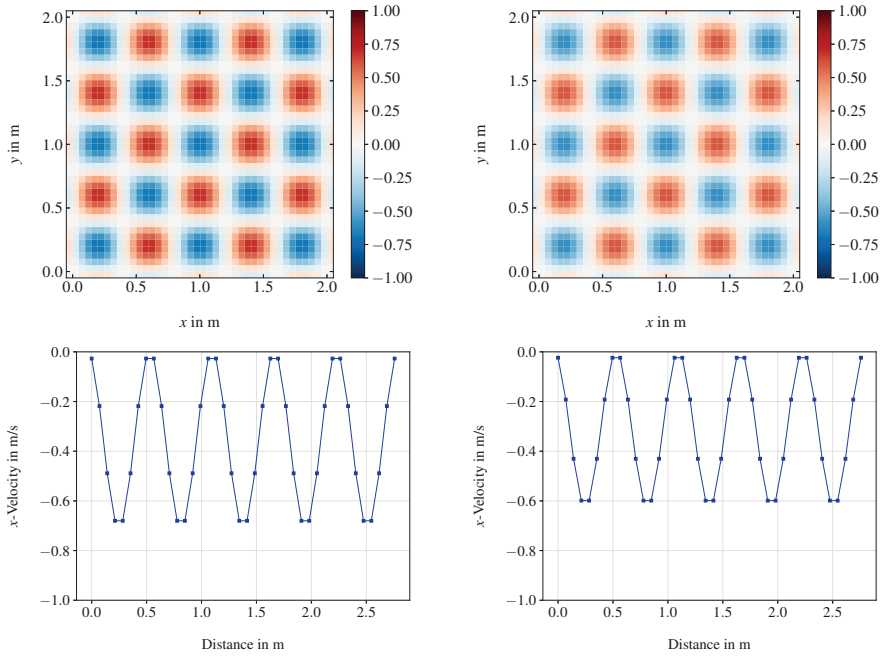


Figure 3.10: Unit test for diffusion showing x-velocity
(top: pseudocolor plot, bottom: curve along diagonal,
left: initial condition, right: numerical solution at $t = 1\,\mathrm{s}$)

Here, no-slip boundary conditions are set and the velocity takes the analytical solution at $t = 0$ as initial guess, $\mathbf{u}^{(0)}(\mathbf{x}) = \mathbf{u}^{\mathrm{ana}}(\mathbf{x}, t = 0)$ (for more details see Tab. B.3 in Appendix B). Comparing the numerical with the analytical solution at $t_{\mathrm{end}} = 1\,\mathrm{s}$ over the whole domain gives a relative error of $\epsilon_{\mathrm{rel}} \approx 6\,\%$, whereby the absolute error $\epsilon_{\mathrm{abs}} \approx 1.75 \times 10^{-2}$ is divided by the $L2$-norm of the analytical solution. The root mean square error for the horizontal velocity at the center $\mathbf{x}_c$ of the domain amounts to $\epsilon_{\mathrm{RMS}} \approx 4 \times 10^{-3}$.

b) Another (qualitative) test is to start with the piecewise constant (hat) function

$$\mathbf{u}(\mathbf{x}) \coloneqq \begin{cases} 2 & \text{, if } \mathbf{x} \in [0.5, 1]^3 \\ 1 & \text{, else} \end{cases} \tag{3.8}$$

on a uniform domain $[0, 2]^3\,\mathrm{m}^3$ with Dirichlet boundary conditions complying with the piecewise function, thus $\mathbf{u} = 1\,\mathrm{m/s}$ (cf. Tab. B.4 in Appendix B).

When highly diffused with kinematic viscosity $\nu = 0.05\,\mathrm{m}^2/\mathrm{s}$, the amplitude decreases while the width increases without any movement of the center. The numerical solution on a grid of cell size $(\Delta x, \Delta y, \Delta z) = (6.25, 6.25, 6.25)\,\mathrm{cm}$ and time step $\Delta t = 0.02\,\mathrm{s}$ shows exactly this expected behavior (cf. Fig. 3.11).

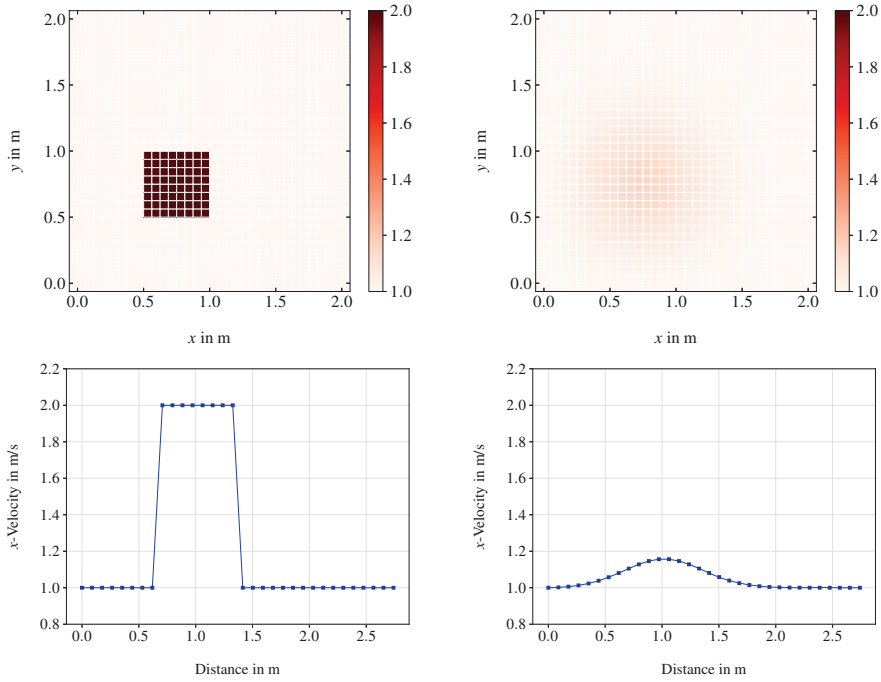


Figure 3.11: Qualitative test for diffusion showing x-velocity (top: pseudocolor plot, bottom: curve along diagonal, left: initial condition, right: numerical solution at $t = 1\,\mathrm{s}$)

3.3.3 Unit Test Pressure

Testing the multigrid scheme with pre-conditioning at the first time step is applied on the setup of a combined sinus function as right-hand side of $\nabla^2 p = \nabla \cdot \mathbf{u}^{(3)}$, where

$$\nabla \cdot \mathbf{u}^{(3)} := \sin(lx)\sin(ly)\sin(lz) \tag{3.9}$$

results in the analytical solution for pressure

$$p^{\text{ana}} = -\frac{1}{3l^2}\sin(lx)\sin(ly)\sin(lz) \tag{3.10}$$

with $p^{(0)} = 0\,\text{Pa}$ as the initial guess and wave number $l = 2\pi$. On the computational domain $[0, 2]^3\,\text{m}^3$ the pressure Poisson equation is solved in one time step with $\Delta t = 0.1\,\text{s}$ on a grid with $N_x - 2 = N_y - 2 = N_z - 2 = 64$ inner cells in each direction (i.e., $\Delta x = \Delta y = \Delta z \approx 3.13\,\text{cm}$). The Dirichlet boundary condition, $p = 0\,\text{Pa}$, complies with the analytical solution at the boundaries. The pre-conditioning in the first time step starts with two V-cycles and goes up to 100 cycles (stopping when the $L2$-norm of the residual is less than $\delta_{\text{tol}} = 1 \times 10^{-7}$). Within the cycles the Jacobi method solves the residual equation on the coarsest grid (at fifth level with $N_x - 2 = N_y - 2 = N_z - 2 = 2$ inner cells in each direction) iterating for a minimum of four and a maximum of 100 iterations, but no longer than the $L2$-norm of the residual reaches $\delta_{\text{tol}} = 1 \times 10^{-7}$ (cf. Tab. B.5 in Appendix B).

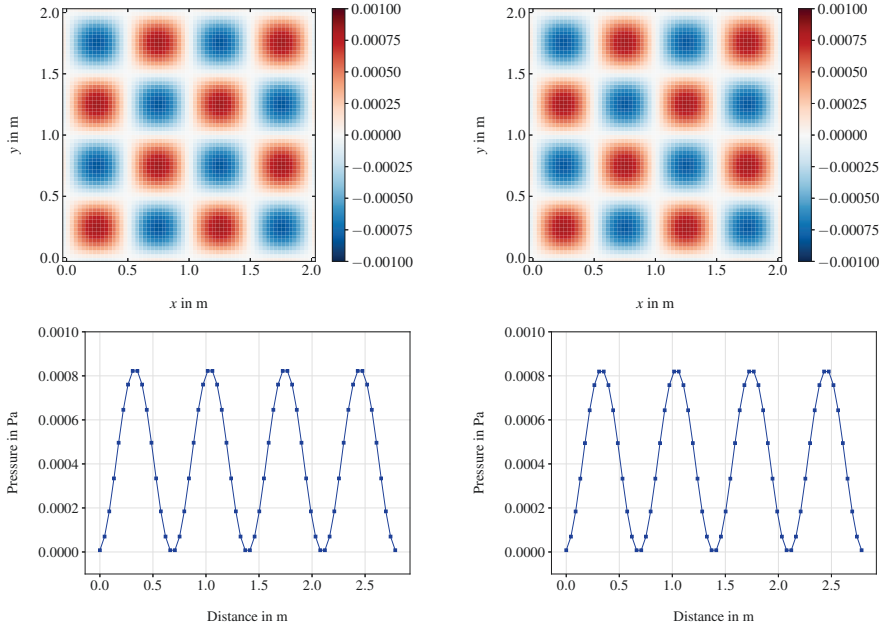


Figure 3.12: Unit test for pressure at $t = 0.1\,\text{s}$
(top: pseudocolor plot, bottom: curve along diagonal,
left: numerical, right: analytical)

Figure 3.12 shows the qualitative comparison between the numerical and analytical result. The pre-conditioning succeeded since the initial guess $p^{(0)} = 0\,\mathrm{Pa}$ proceeds in only one time step into a numerical solution with an absolute error of order $\epsilon_{\mathrm{abs}} \approx 1 \times 10^{-5}$ and resulting relative error of $\epsilon_{\mathrm{rel}} \approx 0.3\,\%$ compared to its analytical solution. The root mean square error for the pressure at the center $\mathbf{x}_c$ of the domain is of order $\epsilon_{\mathrm{RMS}} \approx 2.5 \times 10^{-8}$.

The unit tests for advection, diffusion and pressure show good results in terms of expected behavior and accuracy. Testing the combination(s) of the fractional steps for correctness is left to the verification of the code with (semi-) analytical test cases, and in addition, the underlying model needs to be validated using experimental studies.

Chapter 4

Verification and Validation of the Prognosis Software

During and after code development, it is essential to be aware of errors and if possible eliminate these errors to enhance the accuracy of the application (cf. Section 2.3). By systematically verifying the code (e.g., by unit tests as in Section 3.3) computer programming errors can be eliminated. Whereas unit testing handles isolated parts of the solver, verification determines if the implemented computer model accurately represents the mathematical model aimed to be solved. After code verification, model validation is investigated. With validation, the physical modeling errors can be estimated and the degree to which the underlying physical model accurately represents the real world can be determined. This process of model development, verification, and validation (cf. Thacker et al. (2004)) is illustrated in Figure 4.1. Accordingly, the verification of the CFD application, JuROr, is discussed in Section 4.1 using ($2D$) analytical and ($3D$) semi-analytical test cases solving the Navier-Stokes equations. Thereafter, in Section 4.2 the validation of JuROr is demonstrated using a large-scale and a small-scale experiment.

4.1 Verification of the Implementation

Analytical test cases are designed to assess the convergence (and therefore, accuracy) of the numerical to the analytical solution. Hence, they can be used for code verification. During the development of the code (especially in $2D$), two test cases from McGrattan et al. (2017a) were set up solving for the unsteady advective, viscous, and pressure terms for non-turbulent flows without external sources.

After adding the third dimension and, more importantly, inner boundaries, also

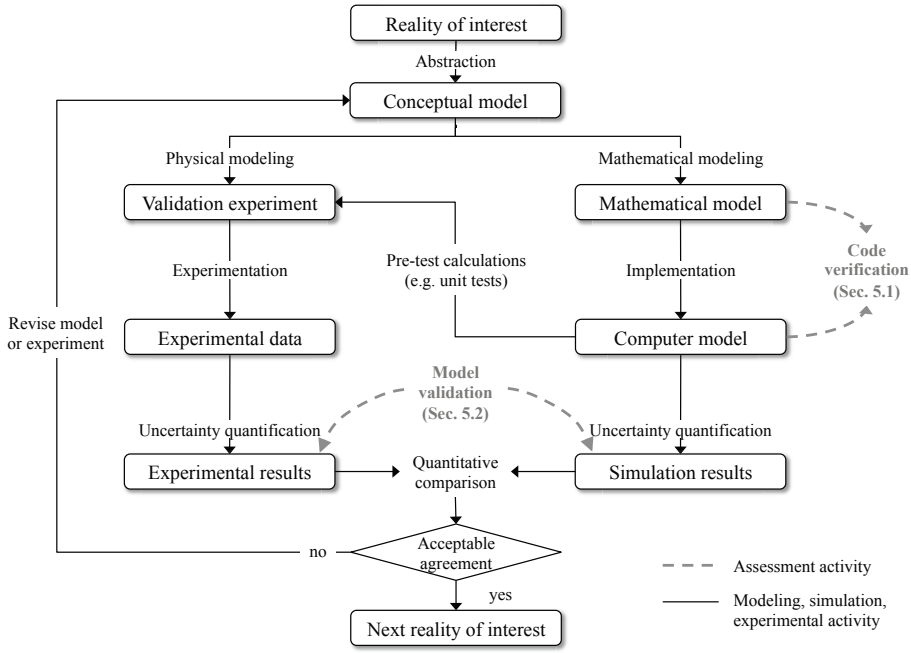


Figure 4.1: Illustrative approach to modeling, verification and validation (based on Thacker et al. (2004))

more complex test cases in $3D$ such as in Ghia et al. (1982) or Martinuzzi and Tropea (1993) and Breuer et al. (1996) are tested to verify JuROr.

4.1.1 Analytical Verification Scenarios

In $2D$, there exist several analytical test cases which are useful for confirming the convergence rates of the truncation errors in the spatial and temporal discretization of the governing incompressible Navier-Stokes equations (2.5) - (2.6)

$$\nabla \cdot \mathbf{u} = 0$$
$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\, \mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho_0} \nabla p = \mathbf{0}\,.$$

In order to assure the same conditions when increasing the time and grid resolution, the number of iterations for the iterative solver is fixed (and not adjusted automatically based on the residual).

4.1.1.1 McDermott Test Case

An analytical solution of these equations is given by McDermott (2003) as

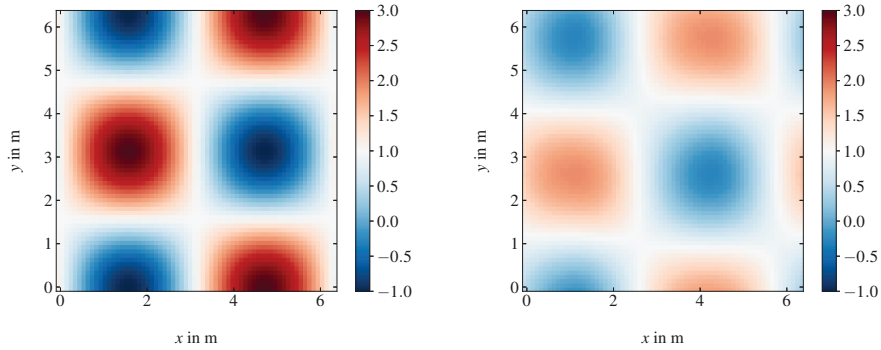
$$u(x, y, t) := 1 - A\cos(x - t)\sin(y - t)\exp(-2\nu t)\,, \tag{4.1}$$

$$v(x, y, t) := 1 + A\sin(x - t)\cos(y - t)\exp(-2\nu t)\,, \tag{4.2}$$

$$p(x, y, t) := -\frac{A^2}{4}\left[\cos(2(x - t)) + \cos(2(y - t))\right]\exp(-4\nu t) \tag{4.3}$$

with amplitude $A = 2$, kinematic viscosity $\nu \in \{0, 0.1\}\,\mathrm{m}^2/\mathrm{s}$, total simulation time $t_{\mathrm{end}} = 2\pi\,\mathrm{s}$ and periodic boundary conditions. The underlying uniform and collocated grid for the computational domain of a $[0, 2\pi]^2\,\mathrm{m}^2$ square varies for being coarse (with 8×8 inner cells) to fine (with 64×64 inner cells). Further physical, numerical and model parameters can be obtained from Tables B.6 and B.7 in Appendix B.

In case of zero viscosity, the analytical solution is periodic in time, hence after $t_{\mathrm{end}} = 2\pi\,\mathrm{s}$ the solution equals the initial setup. Figure 4.2 shows the numerical solution for the x-velocity in m/s with time step $\Delta t = 0.01\,\mathrm{s}$, where the time periodicity for $\nu = 0\,\mathrm{m}^2/\mathrm{s}$ cannot be verified, since the initial and final numerical solution at the left and right, respectively, differ.



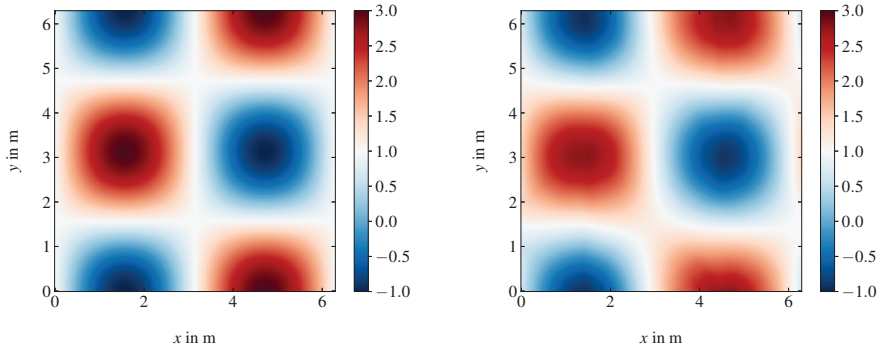
(a) Initial condition at $t = 0\,\mathrm{s}$ for x-velocity

(b) Numerical solution at $t = 2\pi\,\mathrm{s}$ for x-velocity

Figure 4.2: McDermott (2003) test case: JuROr's initial and final states of the u-component of velocity (in m/s) for 64×64 inner cells with $\Delta t = 0.01\,\mathrm{s}$

Due to artificial diffusion (cf. Section 3.3.1) induced by the Semi-Lagrangian advection scheme, the amplitude decreases even if diffusion is set to zero and the periodicity declines. When the grid resolution is increased, the effect of artificial diffusion is reduced while the amplitude stays at the same initial level and the periodicity improves

(cf. Fig. 4.3 showing the x-velocity in m/s).



(a) Initial condition at $t = 0\,\mathrm{s}$ for x-velocity

(b) Numerical solution at $t = 2\pi\,\mathrm{s}$ for x-velocity

Figure 4.3: McDermott (2003) test case: JuROr's initial and final states of the u-component of velocity (in m/s) for 512×512 inner cells with $\Delta t = 0.01\,\mathrm{s}$

Intendedly including diffusion by setting $\nu = 0.1\,\mathrm{m}^2/\mathrm{s}$, the amplitude of the velocity now decreases in time (cf. Fig. 4.4). From top left to bottom right of Figure 4.4, the profiles also show the convergence of JuROr to the analytical solution when increasing the grid resolution $N_x - 2 = N_y - 2 \in \{8, 16, 32, 64\}$.

To quantify the (spatial) convergence rate, and therefore, the order of accuracy of the advective and viscous terms in JuROr, the RMS error

$$\epsilon_{RMS} = \sqrt{\frac{1}{N_t} \sum_{n=1}^{N_t} \left[\mathbf{u}_{ijk}^{(n)} - \mathbf{u}(\mathbf{x}_c, t^{(n)}) \right]^2} \tag{4.4}$$

of the u-velocity at the center $\mathbf{x}_c$ of the grid is plotted against the grid spacing as log-log-plot in Figure 4.5. It demonstrates that the advective, viscous and pressure terms in JuROr are convergent and first-order accurate independent from the kinematic viscosity. This result is in line with the theoretical first-order accuracy of the implemented numerical solution methods for advection, diffusion and pressure (cf. Section 2.3).

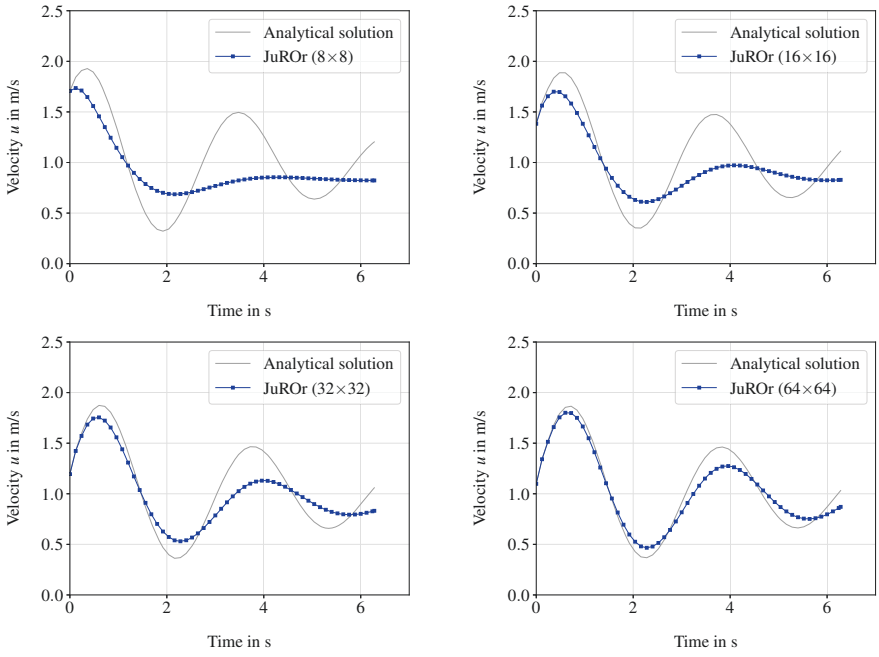


Figure 4.4: McDermott (2003) test case: Curve of horizontal velocity over time at grid center for resolutions $N_x - 2 = N_y - 2 \in \{8, 16, 32, 64\}$

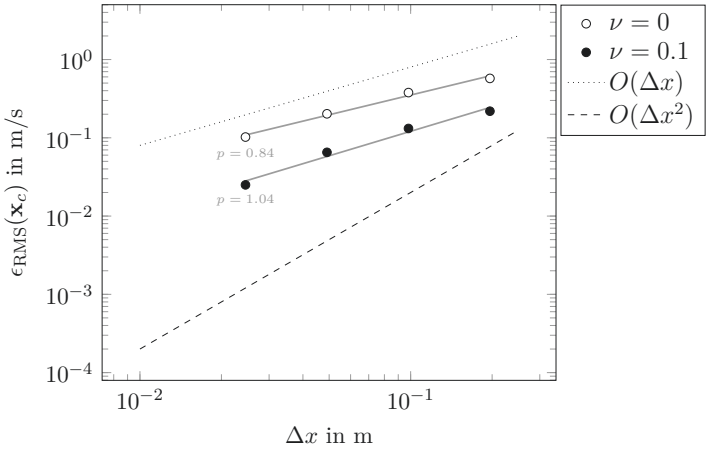


Figure 4.5: McDermott (2003) test case: JuROr's spatial convergence rate for horizontal velocity with $N_x - 2 = N_y - 2 \in \{32, 64, 128, 256\}$ and $\Delta t = 0.01\,\mathrm{s}$

To calculate the temporal convergence order, Richardson's extrapolation (cf. Moin (2001)) is applied to the RMS error, $\epsilon = \epsilon_{\mathrm{RMS}}(\mathbf{x}_c)$, of the horizontal velocity at the grid center

$$p_{\mathrm{Rich}} = \frac{\ln\left(\frac{\epsilon_3 - \epsilon_2}{\epsilon_2 - \epsilon_1}\right)}{\ln 0.5} \tag{4.5}$$

showing an order of $p = 1$ in Table 4.1, which is also in line with the theoretical order of convergence in time.

Table 4.1: McDermott (2003) test case: JuROr's temporal convergence rate for horizontal velocity with Richardson extrapolation

		$\nu = 0\,\mathrm{m}^2/\mathrm{s}$		$\nu = 0.1\,\mathrm{m}^2/\mathrm{s}$	
Inner cells	Δt in s	$\epsilon_{\mathrm{RMS}}(\mathbf{x}_c)$	p_{Rich}	$\epsilon_{\mathrm{RMS}}(\mathbf{x}_c)$	p_{Rich}
64×64	0.01	0.378	1.05	0.132	1.04
64×64	0.005	0.399	1.02	0.140	1.00
64×64	0.0025	0.409	1.02	0.143	1.02
64×64	0.00125	0.414	-	0.145	-
64×64	0.000625	0.416	-	0.146	-

4.1.1.2 Vortex Test Case

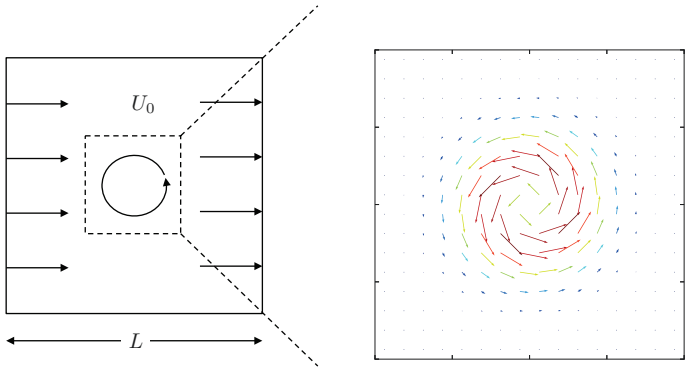
Another test case demonstrating the order of accuracy of JuROr is a flow field of a single vortex advected by a uniform flow, $\mathbf{U}_0$, in a square of width L with periodic boundary conditions and no diffusion, i.e., $\nu = 0\,\mathrm{m}^2/\mathrm{s}$ (cf. Fig. 4.6, Tab. B.8, Tab. B.9 in Appendix B). The vortex test case was developed by Jouhaud (2010) with an analytical solution of

$$u(x, y) \coloneqq U_0 - \frac{\Gamma y}{R_c^2} \exp\left(-\frac{x^2 + y^2}{2R_c^2}\right), \tag{4.6}$$

$$v(x, y) \coloneqq V_0 + \frac{\Gamma x}{R_c^2} \exp\left(-\frac{x^2 + y^2}{2R_c^2}\right), \tag{4.7}$$

maintaining the geometry of the vortex over time providing a measure of the order of accuracy of the advection scheme (and time integration). Thereby, $R_c \coloneqq L/20$ determines the characteristic size of the vortex and $\Gamma \coloneqq 0.04\, U_0 R_c \sqrt{e}$ describes its intensity. Due to the uniform flow field and periodic boundary conditions, the vortex repeatedly passes through the domain. Thereby, the time of passing through is defined as the time period required for the vortex to return to its original position, hence

$$t_f = L/U_0\,. \tag{4.8}$$



(a) Exemplary illustration of the geometry setup

(b) Exemplary illustration of the vector field

Figure 4.6: Jouhaud (2010) test case: Two-dimensional vortex in a constant flow field

Setting the domain width to $L = 1\,\mathrm{m}$ and the uniform flow velocity pointing into x-direction to $\mathbf{U}_0 = (0.1, 0)^\top\,\mathrm{m/s}$, results in $t_f = 10\,\mathrm{s}$. For the numerical result with $\Delta t = 0.01\,\mathrm{s}$ it holds that the more often the vortex passes its original position, the further its u-velocity profile diverges from the profile of the analytical solution (cf. Fig. 4.7, Fig. 4.8). To assess the order of accuracy, again the RMS error of the u-velocity at the first three pass-through times is plotted as log-log-plot in Figure 4.7.

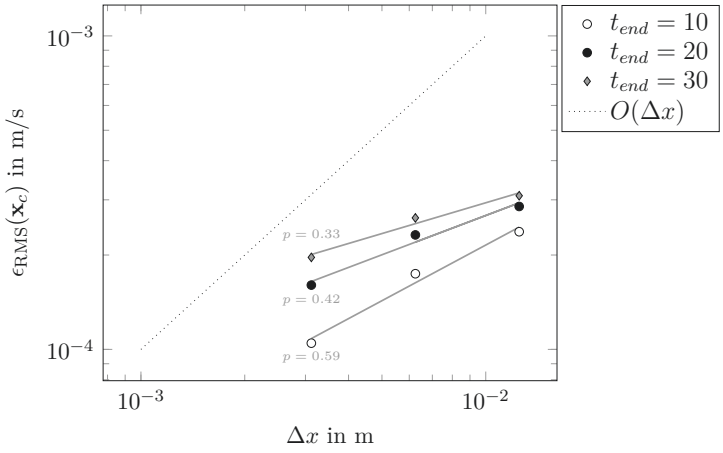


Figure 4.7: Jouhaud (2010) test case: JuROr's spatial convergence rate for x-velocity with $N_x - 2 \in \{80, 160, 320\}$ and $\Delta t = 0.01\,\mathrm{s}$

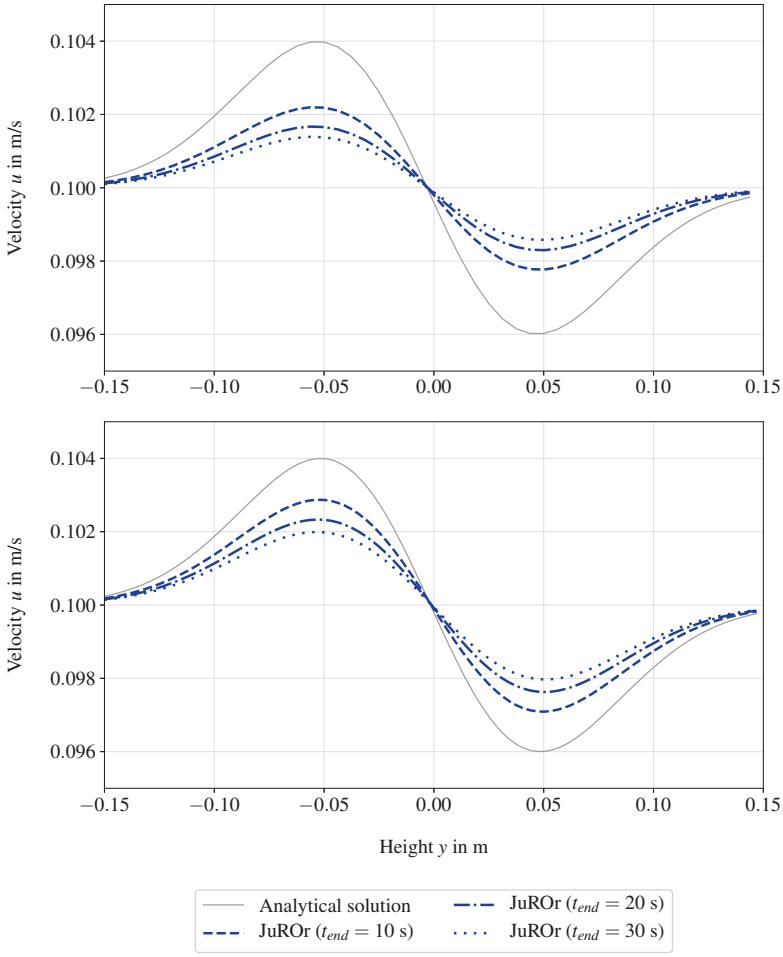


Figure 4.8: Jouhaud (2010) test case: Horizontal velocity u along a section of the domain height y at width $x = 0\,\text{m}$ for the first three complete loops (top: 160×160 inner grid cells, bottom: 320×320 inner grid cells)

The gradients of the (spatial) RMS error lines in Figure 4.7 are only very roughly parallel to the first-order line for the first pass through. The second and third passes suffer even more from artificial diffusion and therefore smoothing, which leads to decreasing convergence orders with increasing pass-through times (as also indicated in Figure 4.8 for different grid resolutions). The theoretical convergence order ($p = 1$) for the numerical solution scheme can only slightly be maintained by the first pass through.

Also of first-order should be the temporal convergence rate. Again the Richardson extrapolation is consulted to calculate the temporal convergence rate, which is indeed of first-order as can be seen in Table 4.2.

Table 4.2: Jouhaud (2010) test case: JuROr's temporal convergence rate for horizontal velocity with Richardson extrapolation

		$t_{\text{end}} = 10\,\text{s}$		$t_{\text{end}} = 20\,\text{s}$		$t_{\text{end}} = 30\,\text{s}$	
Inner cells	Δt in s	$\epsilon_{\text{RMS}}(\mathbf{x}_c)$	p_{Rich}	$\epsilon_{\text{RMS}}(\mathbf{x}_c)$	p_{Rich}	$\epsilon_{\text{RMS}}(\mathbf{x}_c)$	p_{Rich}
80×80	0.01	2.37×10^{-4}	1.05	2.86×10^{-4}	1.06	3.09×10^{-4}	1.05
80×80	0.005	2.40×10^{-4}	1.03	2.88×10^{-4}	1.03	3.11×10^{-4}	1.04
80×80	0.0025	2.42×10^{-4}	1.02	2.89×10^{-4}	1.01	3.12×10^{-4}	1.01
80×80	0.00125	2.43×10^{-4}	-	2.90×10^{-4}	-	3.13×10^{-4}	-
80×80	0.000625	2.43×10^{-4}	-	2.90×10^{-4}	-	3.13×10^{-4}	-

The $2D$ test cases of McDermott (2003) and Jouhaud (2010) confirm that the numerical solution of the incompressible Navier-Stokes equations converges in space and time to the analytical solution with rate $p = 1$ using JuROr, but nevertheless suffering from artificial diffusion.

4.1.2 Semi-Analytical Verification Scenarios

Next, JuROr is tested using semi-analytical test cases, where no analytical solution is available, but relevant benchmarks such as drift points (for separation and reattachment) can be compared to simulations and experimental data from literature.

Thereby, the first test case of lid-driven cavity flow is numerically solved by solving the incompressible Navier-Stokes equations (2.5) - (2.6) without external forces, whereas the second test case of flow around a cube is solved using the Constant Smagorinsky-Lilly LES turbulence model for

$$\nabla \cdot \mathbf{u} = 0 \tag{4.9}$$

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - (\nu + \nu_T)\nabla^2 \mathbf{u} + \frac{1}{\rho_0}\nabla p = \mathbf{0} \tag{4.10}$$

with

$$\nu_T = C_S^2 \Delta_f^2 \left\|\overline{\mathbf{S}}\right\| \quad \text{with} \quad \left\|\overline{\mathbf{S}}\right\| = \sqrt{2 \sum_{i=1}^{3}\sum_{j=1}^{3} \overline{S}_{i,j}\overline{S}_{i,j}}\,, \tag{4.11}$$

where $\overline{S}_{i,j} = \frac{1}{2}(\partial_{x_j}\overline{u}_i + \partial_{x_i}\overline{u}_j)$ for $i, j = 1, 2, 3$ and $i \neq j$ and the Smagorinsky constant C_S is set to be $C_S = 0.2$.

4.1.2.1 Lid-Driven Cavity Flow

The classical $2D$ lid-driven cavity problem simulating a continuously moving lid with constant velocity on top of a square cavity as illustrated in Figure 4.9 is well documented and has been investigated by many authors (cf. Bruneau and Saad (2006); Ghia et al. (1982); Glimberg et al. (2009); Marchi et al. (2009)).

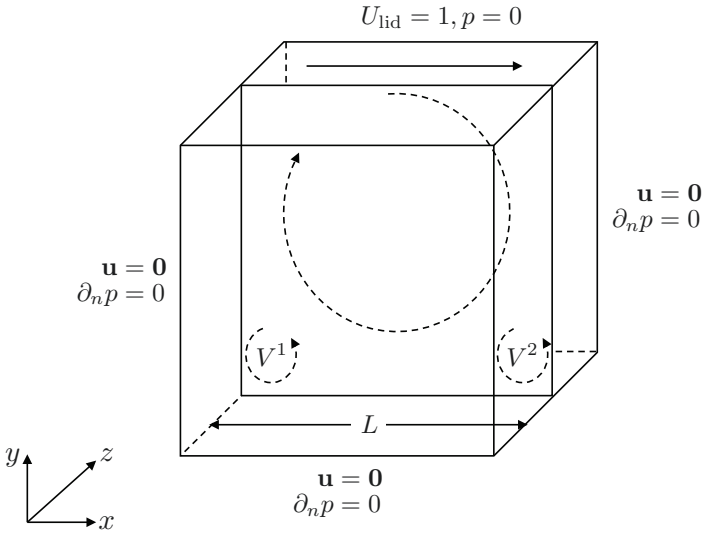


Figure 4.9: Setup of lid-driven cavity flow with boundary conditions

The key characteristics of the resulting flow are three vortices: the primary vortex in the middle of the cavity, and two secondary vortices, V^1 and V^2, in the lower corners (cf. Fig. 4.9). Thereby, the separation and reattachment points of the numerical solution (i.e., the widths, V_x^1, V_x^2 and heights, V_y^1, V_y^2, of the secondary vortices as indicated in the streamline plot 4.10) are taken as quality reference in literature for a wide range of Reynolds numbers. Here, a Reynolds number of

$$Re = \frac{U_{\text{lid}} L}{\nu} = 1000 \tag{4.12}$$

is set by prescribing $U_{\text{lid}} = 1\,\text{m/s}$, $\nu = 1 \times 10^{-3}\,\text{m}^2/\text{s}$ for a square cavity of width $L = 1\,\text{m}$ resulting in a laminar flow. Further, all velocities except the lid velocity in x-direction comply with the no-slip boundary condition $\mathbf{u} = \mathbf{0}\,\text{m/s}$. Additionally, the pressure at the lid is set to be zero, $p = 0\,\text{Pa}$, with no change in gradient, i.e., zero Neumann condition $\partial_n p = 0\,\text{Pa/m}$, everywhere else. Further physical, numerical and solution parameters can be obtained from Tables B.10 and B.11 in Appendix B.

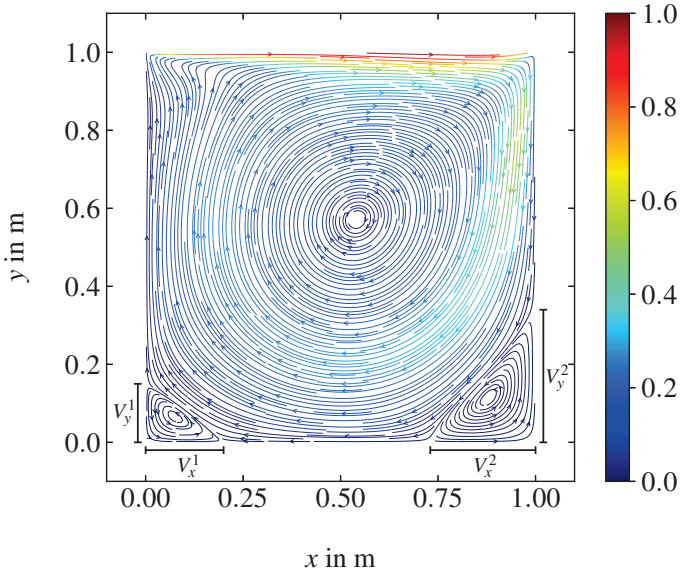
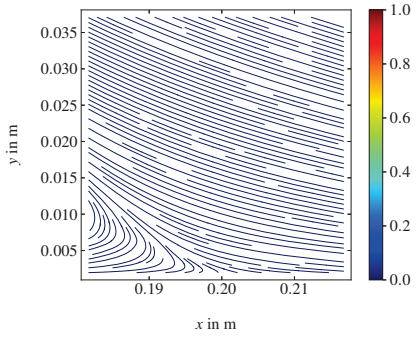


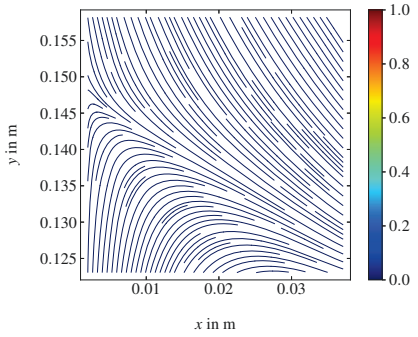
Figure 4.10: JuROr's streamlines colored by the speed $\sqrt{u^2 + v^2}$ (in m/s) of the cavity flow for $N_x - 2 = N_y - 2 = 256$ showing the primary vortex and two secondary vortices V^1 (bottom left) and V^2 (bottom right)

The vertical and horizontal spreads of the secondary vortices, V_x^1, V_y^1, V_x^2, and V_y^2, are measured by taking the separation points of the speed streamlines into account (cf. Fig. 4.11 for $N_x - 2 = N_y - 2 = 256$ from top left to bottom right). The *separation point* is thereby defined as the point in space where following two adjacent streamlines leads to separate directions.

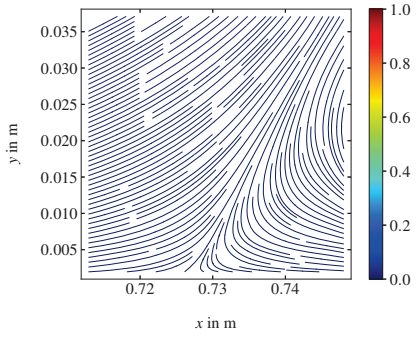
Simulating the cavity flow for $t_{\text{end}} = 300\,\text{s}$ with time resolution $\Delta t = 0.001\,\text{s}$ and grid resolutions, $N_x - 2 = N_y - 2 \in \{64, 128, 256\}$, the results (relative to L) are compared to the two-dimensional simulations of Ghia et al. (1982) using a 129×129 grid. Table 4.3 and Figure 4.12 indicate that JuROr's results are closer to the reference results of Ghia et al. (1982) the higher the grid resolution is. Thereby, the finest grid result shows sufficient agreement, whereas the coarse and medium grid results suffer from smoothing due to artificial diffusion and discretization errors. Further, the grid sizes for $N_x - 2 = N_y - 2 \in \{64, 128\}$ are larger than the Kolmogorov scale, estimated by $\eta \approx (\nu^3 L/U^3)^{1/4}$, which results in possibly non-dissolved eddies.



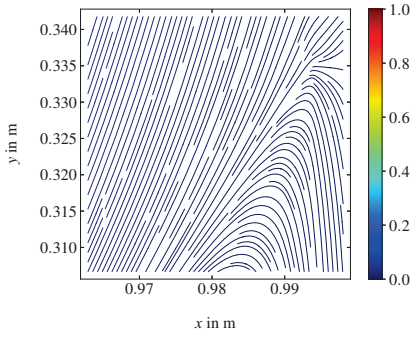
(a) Width $V_x^1 = 0.1975\,\text{m}$ of vortex V^1



(b) Height $V_y^1 = 0.1455\,\text{m}$ of vortex V^1



(c) Width $V_x^2 = 1 - 0.7275\,\text{m}$ of vortex V^2



(d) Height $V_y^2 = 0.3350\,\text{m}$ of vortex V^2

Figure 4.11: Detailed view of the separation points of the secondary vortices of the cavity flow for $N_x - 2 = N_y - 2 = 256$ (in m/s)

Table 4.3: Vertical and horizontal spread of secondary vortices V^1 and V^2 for the lid-driven cavity flow compared to Ghia et al. (1982)

	JuROr			Ghia et al. (1982)
Vortex size in m	64×64	128×128	256×256	129×129
V_x^1	0.15	0.18	0.20	0.22
V_y^1	0.12	0.14	0.15	0.18
V_x^2	0.25	0.26	0.27	0.30
V_y^2	0.29	0.32	0.34	0.35

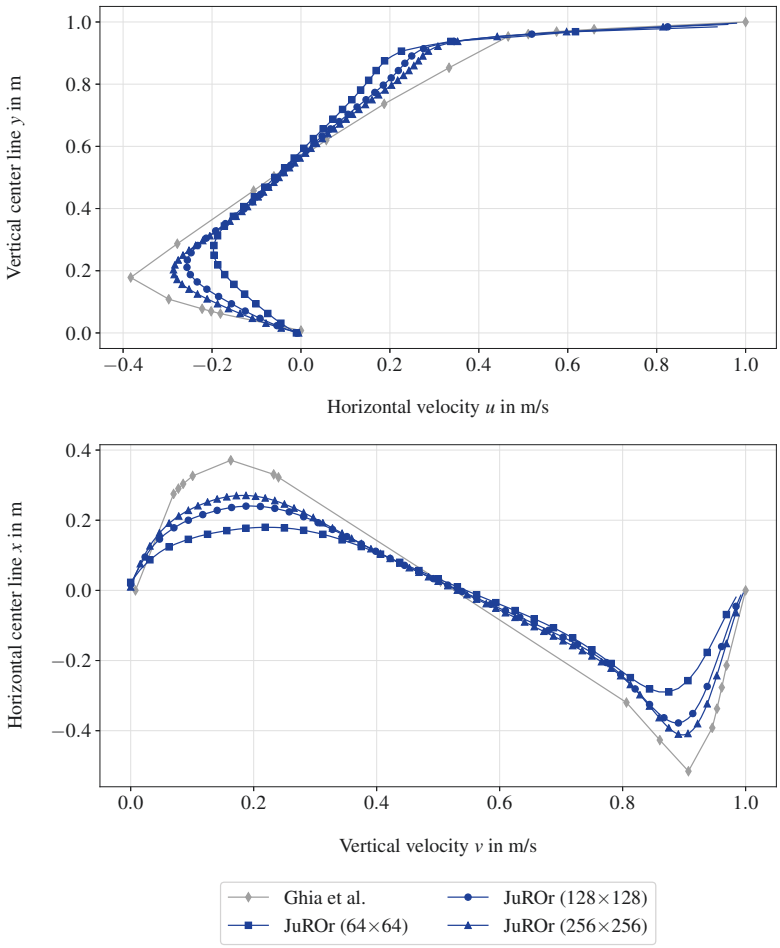


Figure 4.12: Velocity profiles along the center lines of the cavity flow for $N_x - 2 = N_y - 2 \in \{64, 128, 256\}$

4.1.2.2 Flow Around a Cube

With the scenario of turbulent flow around a cube, the turbulence model introduced in Section 2.1.2 as well as inner boundaries in $3D$ defined in Section 3.2.1 are tested. Therefore, the numerical results of JuROr are compared to the simulation data from Breuer et al. (1996); Rodi et al. (1995, 1997); Yakhot et al. (2006) and experimental data from Martinuzzi and Tropea (1993), again using the reattachment length of the occurring vortex as relevant benchmark for quality.

The setup consists of a fixed cube of width L in an open flow channel with dimensions width×height×depth of $10\,L \times 2\,L \times 7\,L$ as depicted in Figure 4.13 and parameters as summarized in Tables B.12 and B.13 in Appendix B. In order to reproduce an open flow at the outlet and a fully developed flow at the inlet (on the left-hand side), the domain boundary conditions for the velocity are set to a (horizontal) inflow velocity $\mathbf{U}_{\text{inflow}} = (u_{\text{in}}, 0, 0)^\top$ at the left with the same velocity, $\mathbf{U}_{\text{outflow}} = (u_{\text{in}}, 0, 0)^\top$, at the outflow on the right, whereas at all other domain boundaries the velocity is set to have a zero gradient using the Neumann condition. The pressure is set to have zero gradients at all domain as well as obstacle boundaries, where the velocity obeys the no-slip condition $\mathbf{u} = \mathbf{0}\,\text{m/s}$. In order to assure a turbulent flow, the inlet velocity, the height of the cube and the kinematic viscosity are chosen to yield a high Reynolds number of

$$Re = \frac{u_{\text{in}} L}{\nu} = 40\,000. \tag{4.13}$$

For a cube of height $L = 1$ m, and viscosity $\nu = 1 \times 10^{-5}$ m^2/s, the inflow velocity is set to $u_{\text{in}} = 0.4$ m/s.

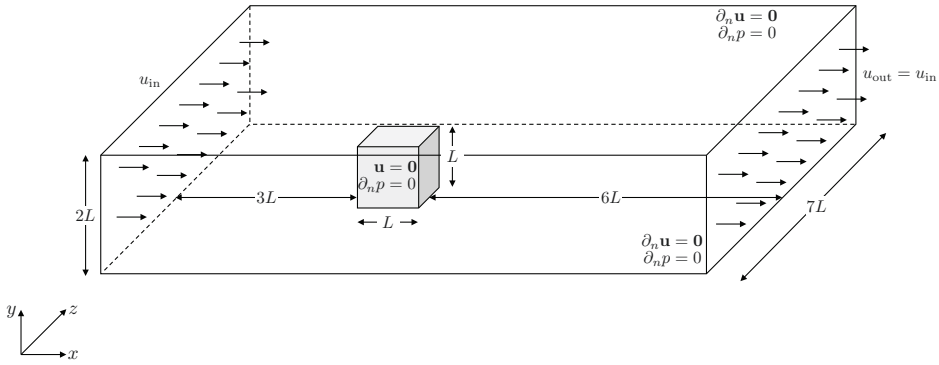


Figure 4.13: Setup of flow around cube with inner and outer boundary conditions

Since it is not possible with JuROr to set boundary conditions via functions (e.g., a parabolic velocity profile as it is the case in Breuer et al. (1996)), the open flow channel is reproduced with a constant background velocity of $\mathbf{u}^{(0)} = \mathbf{U}_{\text{inflow}}$ as initial condition resulting in a symmetric flow field. This approach already produces an error source for the numerical result compared to the expected flow behavior. The flow at hand is expected to build a primary vortex downstream the cube and a horseshoe-shaped flow alongside the cube since the flow upstream the cube is split when striking the wall.

As the flow is turbulent and hence not steady, the numerical solution is averaged over the whole simulation time (with 50 instantaneous records from a simulation time of $t_{\text{end}} = 10\,\text{s}$ and time resolution of $\Delta t = 0.01\,\text{s}$, thus every 20th time step is taken into account). The top of Figure 4.14 shows the speed streamlines in m/s in the x-y-plane at the center $z = 0$ m. Here, the primary vortex downstream the cube is visible. The flow around the cube in the x-z-plane at $y = 0.01$ m is depicted at the bottom of Figure 4.14.

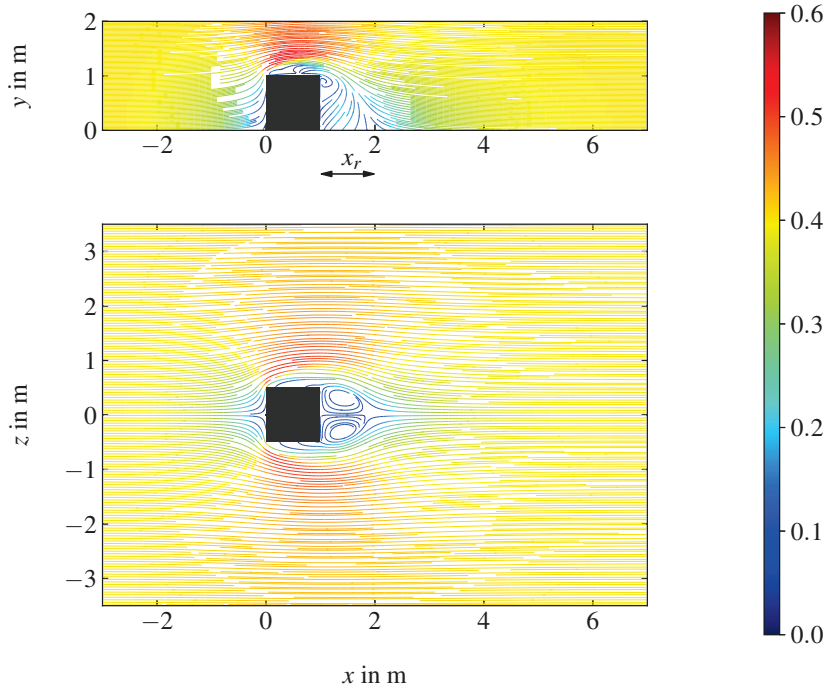


Figure 4.14: JuROr's speed streamlines in m/s of the flow around the cube in the x-y-plane $z = 0$ m (top) and in the x-z-plane at $y = 0.01$ m (bottom)

Again the reattachment length, x_r, of the vortex is defined as separation point of the streamlines at downstream location. The speed streamlines in m/s calculated by JuROr are shown in detail in Figure 4.15. The reattachment length is then compared to the simulation results of Breuer et al. (1996); Rodi et al. (1995, 1997) and Yakhot et al. (2006) as well as to the experimental result of Martinuzzi and Tropea (1993) (taken from Rodi (1997)). Table 4.4 shows that JuROr underestimates the reattachment point. Taking into consideration that the parabolic velocity profile at

the inflow and the wall boundaries are not explicitly modeled affecting the length of the reattachment point, the characteristics of the flow are maintained in sufficient agreement with the theoretical flow behavior.

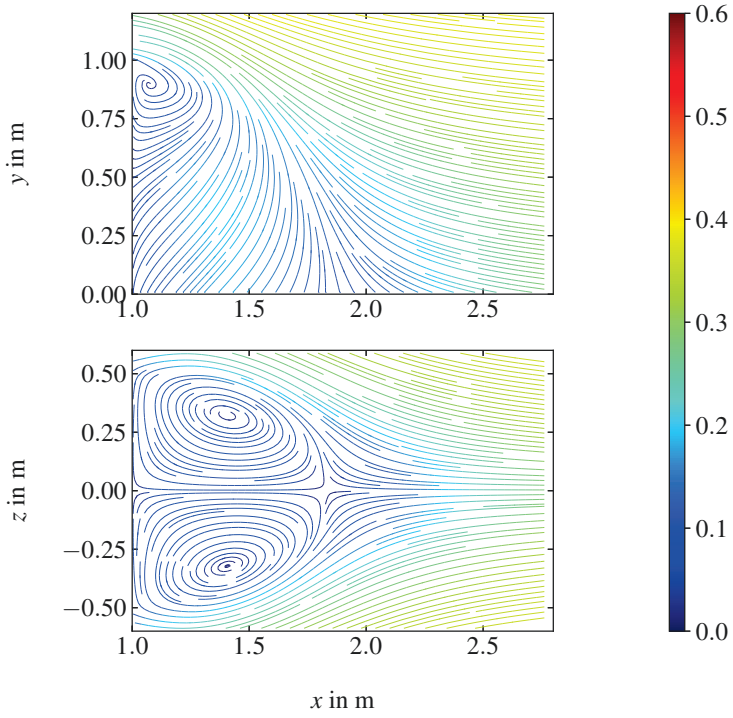


Figure 4.15: Detailed view of the reattachment point at $x = 1.85\,\text{m}$, downstream the cube in the x-y-plane at $z = 0$ m (top) and in the x-z-plane at $y = 0.01$ m (bottom) (taken from the speed streamlines in m/s)

Table 4.4: Comparison of the reattachment length, x_r, for the flow around a cube

Reference	Calculation method	x_r in m
JuROr	LES, Smagorinsky	0.85
Breuer et al. (1996)	LES, Smagorinsky	1.69
	LES, Dynamic	1.43
Rodi et al. (1995)	LES, Smagorinsky	1.70
Rodi et al. (1997)	LES, Dynamic	1.43
Yakhot et al. (2006)	DNS	1.50
Martinuzzi and Tropea (1993)	Experiment	1.61

It is not only the goal to prove that the numerical methods are correctly implemented, but also to evaluate if the code maps the physics (as in Section 2.1) correctly. Therefore, simulation results are further compared to experimental data in the next section. Sensitivity and uncertainty quantification (e.g., with Latin Hypercube Sampling or Monte Carlo) are out of scope for the work at hand, but by considering an increasing set of validation experiments, it is possible to estimate the quality of the model's prognosis in the future. This set of validation scenarios (and its execution) still needs to be build up for JuROr.

4.2 Validation of the Underlying Model

In order to validate JuROr, single room scenarios in $3D$ with thermally driven flows are simulated and compared to experimental data. First, the well-documented fire induced flow experiment of Steckler et al. (1982) in a compartment is evaluated with regards to various benchmarks. After this scenario, a comparison of JuROr's simulation data to a small-scale open plume experiment is assessed modeling the flow due to an electrically heated copper block.

4.2.1 Fire Induced Flow Experiment in a Compartment

Steckler et al. (1982) conducted 55 full-scale steady state fire experiments in a 2.8 m by 2.8 m by 2.13 m compartment with a single door of various widths, or a single window with various heights (see also Steckler et al. (1985)). At varying locations on the floor of the compartment, a methane burner with a 0.3 m diameter was placed to generate fires with heat release rates of $31.6\,\mathrm{kW}, 62.9\,\mathrm{kW}, 105.3\,\mathrm{kW}$ and $158\,\mathrm{kW}$ (cf. Fig. 4.16). The insulation of the compartment consisted of calcium silicate on top of plywood at the bottom and ceramic fiber over aluminium on the walls and the ceiling (cf. Steckler et al. (1985)).

Since the temperature difference between the room and the outside with ambient temperature T_0 creates a pressure difference, the gas flows through the opening. This effect can be seen in velocity profiles in the doorway, where the *neutral plane location* determines the height at which the velocity changes directions. Further, gas layers develop due to the nearby walls restricting the flow with high average temperatures below the ceiling and lower temperatures at the bottom (separated by the so-called *interface height*) as can be seen in temperature profiles in the room and in the opening.

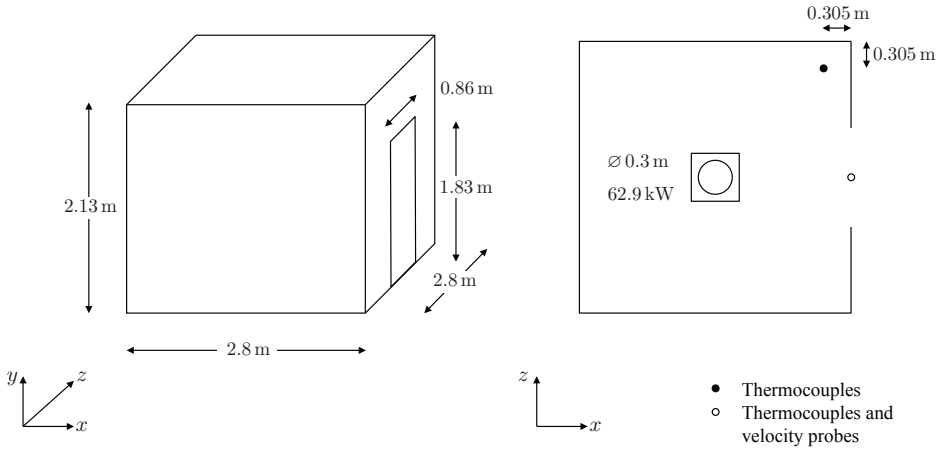


Figure 4.16: Illustrative setup of Steckler et al. (1982)'s experiment N° 16: outside $3D$ view (left) and inside $2D$ view (right)

Hence, the horizontal velocity, u, and the temperature, T, were measured vertically in the doorway (at center), along with temperature measurements inside the compartment at the front right corner. These measurements were conducted with bi-directional velocity probes and stationary bare-wire thermocouples placed 5.7 cm above the ground with a distance of 11.4 cm. The thermocouple wires, the data recording system as well as the installation (e.g., through radiation errors at the junctions) at the door opening were associated with a difference between the gas temperature, T_g, and thermocouple temperature, T_t. Table 4.5 shows relevant corrections as a function of the thermocouple temperature and gas velocity near or above the neutral plane. Further, differences of −9 °C to −1 °C occurred below the neutral plane.

Table 4.5: Corrections $(T_g - T_t)$ for opening thermocouples near or above the neutral plane in Steckler et al. (1982)'s experiments

T_t in °C \ u in m/s	< 0.1	$0.1 - 0.6$	$0.6 - 1.2$	> 1.2
< 50	$-20/+2$			
$50 - 100$		$-14/+4$	$-9/+4$	
$100 - 150$			$-9/+7$	$-2/+6$

Within the room, the error limits of the thermocouple wires and the data recording system are found to be ±1 °C and those associated with the thermocouple junctions (including radiation errors) are estimated at −3 °C to +1 °C. The velocity probes

were aligned such that their heads were directed parallel to the flow with their axes being horizontal. The associated errors due to uncertainties in the pressure difference across the probe heads, absolute gas temperature and in the calibration factor are stated to be 10 % at maximum for $u < 0.1$ m/s and at least 13 % for $u > 0.1$ m/s.

In order to simulate Steckler et al. (1982)'s experiment N° 16 with the burner placed in the center of the compartment releasing total heat of 62.9 kW and with a door of height × width of 1.83 m × 0.86 m, the physical, numerical and model parameters from Tables B.14, B.15 and B.16 in Appendix B were set. Further, to be able to capture the outflow of the door and the flow into the surrounding area with outflow boundaries, the computational domain was enhanced in each direction to 7.00 m × 4.26 m × 5.60 m with a grid of 160 × 128 × 128 inner cells resulting in a spatial resolution of roughly 0.04 m × 0.03 m × 0.04 m. The domain boundaries were set to an ambient temperature of $T_w = T_0 = 26$ °C with a no-slip boundary for the velocities and a no-flow pressure boundary. The compartment was constructed by an assembly of seven obstacles each of 0.2 m thickness with no-slip solid walls and no-flow pressure condition as well as adiabatic surfaces.

Since JuROr is not designed to simulate either radiation or combustion, the total heat release rate of 62.9 kW is reduced by $\chi_{\text{rad}} = 20\,\%$ to 50.3 kW (cf. McGrattan et al. (2017b)). Further, the volumetric heat source is modeled by a Gaussian curve of full width at half maximum of 0.25 m × 0.60 m × 0.25 m to ensure that the heat source is discretized with at least eight cells in x-z-plane and the fire height is modeled correctly. The ramp-up time is set to $\tau = 5$ s until the maximum heat is released. Additionally, the flow is turbulent and therefore non-steady. Hence, the average of 100 instantaneous numerical results of a total simulation period of $t_{\text{end}} = 1800$ s integrated with a time step size of $\Delta t = 0.05$ s was taken to assess the vertical profiles in the doorway and the inside of the compartment. Also, the results of Steckler et al. (1982) are averaged over multiple readings taken at certain time intervals.

With this setup, the qualitative effect of gas flowing through the opening can be seen at the top of Figure 4.17 showing the speed streamlines in m/s colored by $\sqrt{u^2 + v^2}$ in the center of the x-y-plane. At the bottom, Figure 4.17 shows the corresponding upper hot and lower cold gas layers (as filled contour lines in °C). Figure 4.18 shows the velocity profile (on the top) and the vertical temperature profile (on the bottom) at the center of the doorway. Here, JuROr's numerical results are compared to the experimental result of Steckler et al. (1982) (including reported error ranges) as well as the simulation result of NIST's Fire Dynamics Simulator (FDS v6.5.3, cf. McGrattan et al. (2017c)).

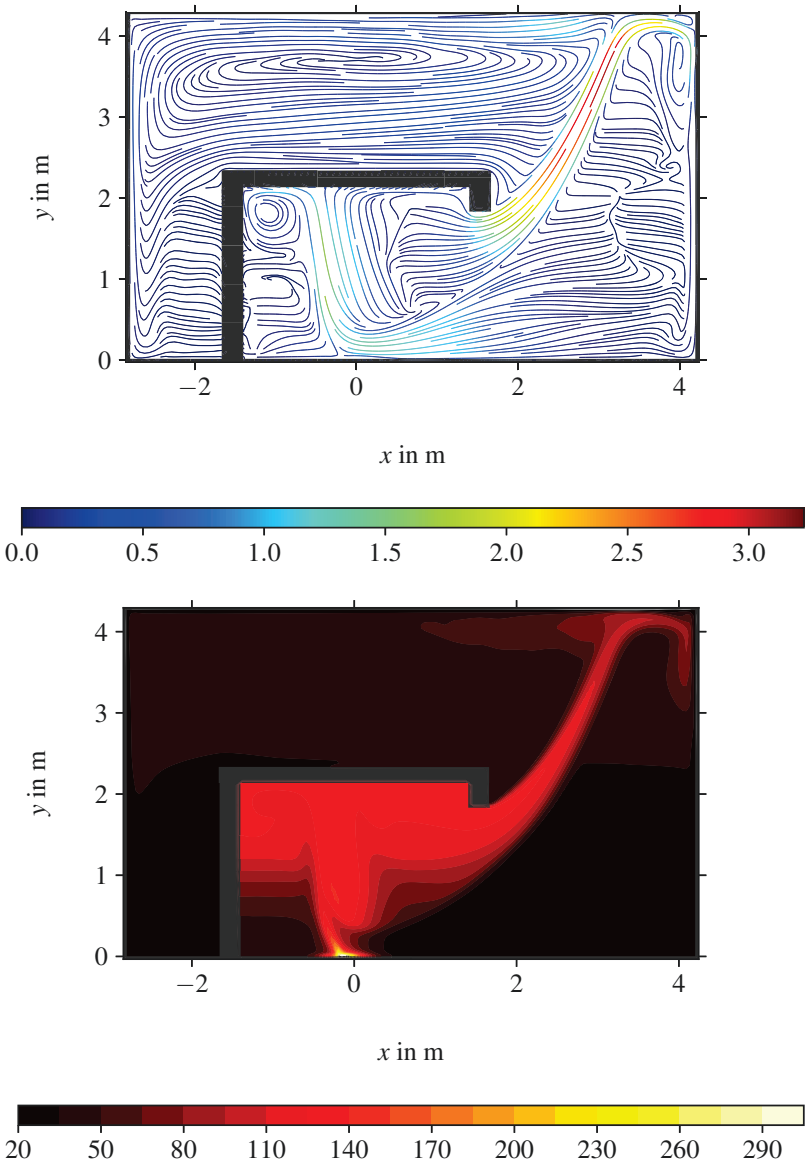


Figure 4.17: JuROr's speed streamlines (in m/s, at the top) and temperature contour lines (in °C, at the bottom) in the central x-y-plane averaged in time for Steckler et al. (1982)'s experiment N° 16

Thereby, the temperature at the top of the doorway is slightly overestimated by JuROr (with a relative error in $L2$-norm of 16 % over all data points), whereas the velocity at the bottom of the door is underestimated (with a relative error in $L2$-norm of 26 % over all data points). Since the walls are assumed to have no-slip boundaries, the velocity takes a sharp turn on the floor and beneath the door soffit. From these velocity profiles the neutral plane location, N, relative to the door height, H_0, can be determined and shows good agreement with Steckler et al. (1982)'s experimental and FDS's simulation results (cf. Tab. 4.6).

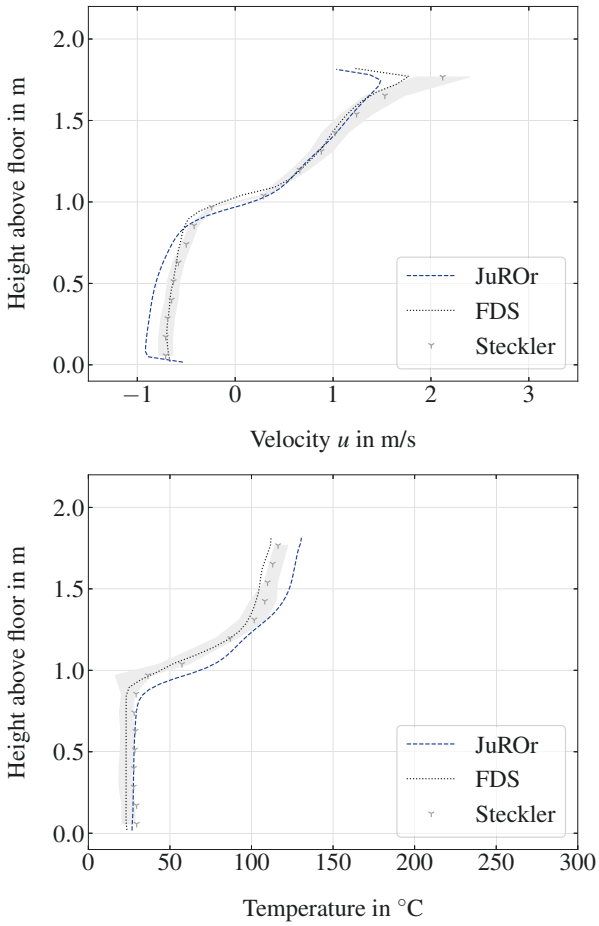


Figure 4.18: Vertical velocity (at the top) and temperature profiles (at the bottom) at the center of the doorway for Steckler et al. (1982)'s experiment N° 16 including measurement errors

Table 4.6: Comparison of neutral plane locations for Steckler et al. (1982)'s experiment N° 16

Reference	Calculation method	N/H_0 in m
JuROr	Solely transport of hot gas	0.537
FDS (v6.5.3)	With radiation, combustion, and insulation, no soot	0.569
Steckler et al. (1982)	Experiment	0.573

Within the room, the temperature profile is depicted in Figure 4.19. Again, the result of JuROr is compared to the experimental result of Steckler et al. (1982)'s experiment N° 16 (including reported error ranges) and the simulation result of FDS. As it is the case for FDS' simulation, the thermal interface height of JuROr's simulation marked by a steep temperature gradient is not as distinctive as for the experimental results. Towards the ceiling of the room, the temperature is mostly overestimated by JuROr (with a relative error in $L2$-norm of 15 % over all data points).

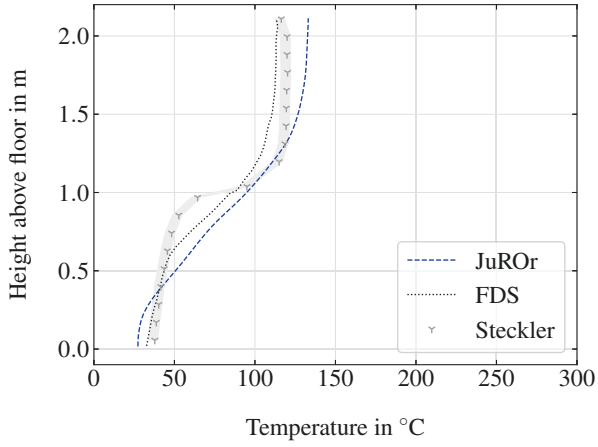


Figure 4.19: Vertical temperature profile at the front right corner of the room for Steckler et al. (1982)'s experiment N° 16 including errors

As an additional benchmark the thermal interface height, Y_i, is taken into consideration. Thereby, Y_i is calculated with the method of the steepest gradient (approximated with second-order central differences) based on the assumption of a two-zone separation. The upper and lower average temperatures, T_u and T_l, are then determined based on the thermal interface height as separation point between T_u and T_l. Table 4.7 shows good agreement of the thermal interface height and upper and lower average temperature for both, JuROr's simulation result and Steckler et al. (1982)'s

experiment N° 16 stating a ±8 % to ±50 % accuracy range. Nevertheless, JuROr (and FDS) fail to accurately predict the sharp transition between the upper hot layer and lower cold layer (cf. Fig. 4.19). In JuROr's case this imprecision could be explained by the smearing effect of artificial diffusion resulting from the use of the Semi-Lagrangian advection scheme, deficiencies in the LES (Constant Smagorinsky-Lilly) turbulence scheme and/or the lack of a radiation model.

Table 4.7: Comparison of thermal interface heights for Steckler et al. (1982)'s experiment N° 16

Reference	Y_i in m	T_l in °C	T_u in °C
JuROr	0.915	50.23	121.53
FDS (v6.5.3)	0.798	43.54	103.62
Steckler et al. (1982)	1.038	50.82	118.73

Taking into consideration that neither radiation, combustion nor conduction into walls are modeled (other than in FDS) and the Steckler experiments themselves are associated with measurement errors (cf. Tab. 4.5), JuROr's results are highly satisfying.

4.2.2 Open Plume Experiment Using Particle Image Velocimetry

In order to validate JuROr with a more suitable experiment that lies within the bounds of JuROr's technical limitations, an open plume experiment of a buoyancy-driven flow resulting from an electrically heated copper block (instead of burning methane) is used. In his dissertation, Meunders (2016) designed and conducted small-scale laboratory experiments specially designed for the validation of buoyancy-driven flows while neglecting pyrolysis and combustion due to simplification, high precision, and reproducibility. Two different setups were investigated: an undisturbed open buoyant plume above the heat source (cf. Fig. 4.20), and a buoyant spill plume emerging from a compartment opening, inside of an enclosure of width × height × depth of 735 mm × 650 mm × 575 mm.

The enclosure made of polymethyl methacrylate is needed to confine tracer particles which are used for a non-intrusive measurement technique, called *Particle Image Velocimetry* (PIV). With PIV the flow field as well as the concentration field of buoyancy-driven plumes can be investigated. First, the centered copper block of size 60 mm × 40 mm × 60 mm is electrically heated (on top of a 20 mm thick plate

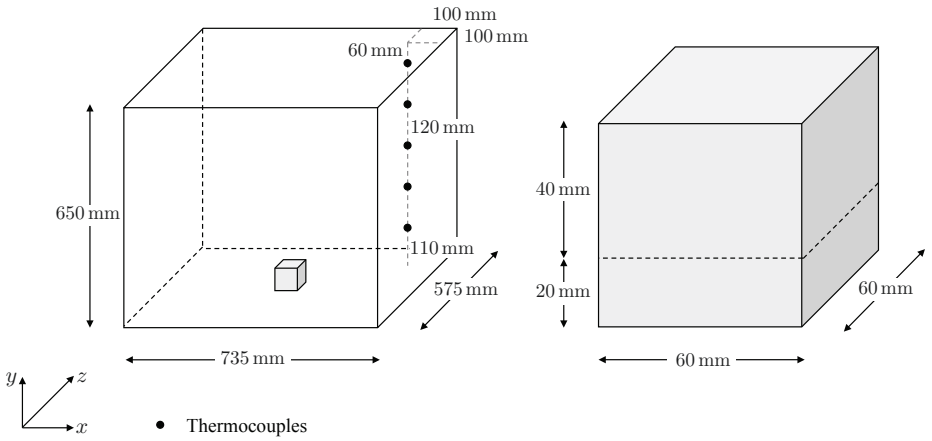


Figure 4.20: Illustrative setup of Meunders (2016)'s PIV experiment for an open plume: outside $3D$ view (left) and enlarged heated copper block (right)

out of calcium silicate to reduce heat loss to the floor and covered with chalk to keep the radiative and convective heat transfer constant). While reaching stationary block as well as gas and enclosure temperatures, the tracer particles are injected into the enclosure spreading quickly due to the buoyancy-driven air movement. Then, after reaching a quasi-steady state, the flow velocity is derived from particle images taken by adjustable cameras at a defined time interval. The particles are thereby illuminated by a laser light sheet, which is placed vertically along the plumes axis (cf. Meunders (2016); Meunders et al. (2018)). Therewith, the flow velocities can be accurately measured due to the non-intrusive nature of PIV leaving the investigated flow undisturbed. Hence, flow velocities are not only derived at a single location but in an entire plane almost instantaneously.

Additionally to the flow velocities, gas temperatures are measured by multiple thermometers in- and around the enclosure. Thereby, five (resistance) thermometers are placed at the front right corner inside of the enclosure (100 mm away from the adjacent walls) proceeding vertically beneath each other with a distance of 120 mm starting at 60 mm beneath the ceiling and ending 110 mm above the floor of the enclosure. The setup of the undisturbed buoyant plume developing over an electrically heated block of copper, as it is evaluated in Meunders et al. (2018), is now used to validate JuROr. The whole measurement setup including the laser sheet and cameras is shown in Figure 4.21.

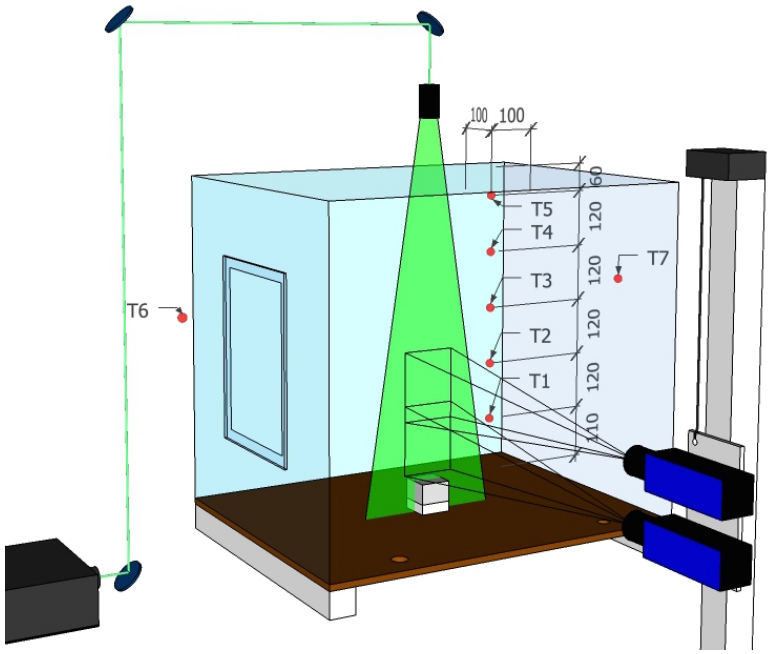


Figure 4.21: Experimental geometry of Meunders (2016) with an heat source placed in the middle of an enclosure, various thermometers in- and outside the enclosure, and adjustable cameras to capture the particles illuminated by a laser light sheet

What is expected as outcome of the open plume experiment is an undisturbed buoyant plume above the heat source and the formation of a hot gas layer in the upper part of the enclosure. The temperature will continuously increase and, thus, no distinct interface between the hot upper layer and cold lower layer can be observed (cf. Meunders et al. (2018)). Relevant benchmarks for comparing the experimental and simulation results are (besides others) time-averaged horizontal profiles of the vertical velocity component 100 mm, 250 mm and 400 mm above the copper block, as well as a time-averaged vertical temperature profile in the corner of the enclosure. Further, the convective heat transfer as sole driver of the plume is estimated by calculating the radiative fraction based on available temperature data and subtracting it from the total energy input (cf. Meunders et al. (2018)). In order to generally compare the laminar and turbulent characteristics of the flow across various experiments, the vertical velocity is plotted as a function of the dimensionless (local) Grashof number at

the symmetry centerline (at block center in z-direction) following Noto et al. (1999)

$$Gr = \frac{g\beta(T_{\text{HS, surf}} - T_0)y^3}{\nu^2} \tag{4.14}$$

with gravitational acceleration g, thermal expansion coefficient β and kinematic viscosity ν of air at ambient temperature T_0.

Out of four different heating powers, P_{HS}, of the heat source (HS), a medium high setting is chosen as benchmark with $P_{\text{HS}} = 78.0 \pm 0.3\,\text{W}$ (and associated standard deviation). This power setting results in an internal heat source temperature of $T_{\text{HS}} = 270.7 \pm 0.5\,°\text{C}$ measured by an embedded thermocouple and a surface temperature of $T_{\text{HS, surf}} = 264.8\,°\text{C}$ measured by an infrared camera with a measurement uncertainty of 1% of the measured temperature. The surface temperature of the heat source is thereby lower than its internal temperature since cooling effects of convection and radiation on the outside occur. Meunders et al. (2018) estimated the radiative heat transfer based on the Stefan-Boltzmann law (cf. Stefan (1879); Boltzmann (1884))

$$P_{\text{rad}} = A\epsilon\sigma(T^4 - T_0^4)\,, \tag{4.15}$$

where A denotes the area between which the radiative heat transport occurs, ϵ is the emissivity (here of the block coating), $\sigma \approx 5.67 \times 10^{-8}\,\text{W}/(\text{m}^2\,\text{K}^4)$ is the Stefan-Boltzmann constant and T_0 is the reference temperature (here of the enclosure walls). For the radiative fraction emitted from the heat source, they derive an estimate of $P_{\text{rad, HS}} = 46.0\,\text{W}$, and from the insulating plate underneath the heat source the radiative heat flux is calculated to be $P_{\text{rad, ins}} = 7.3\,\text{W}$. Under the assumption that the conductive heat losses are negligible, the convective heat transfer can be estimated as difference between the measured total power input P_{HS} and the total radiative heat fluxes, i.e.,

$$P_{\text{conv},\Delta} = P_{\text{HS}} - (P_{\text{rad, HS}} + P_{\text{rad, ins}}) = 24.7\,\text{W}\,. \tag{4.16}$$

An alternative estimation of the convective heat transfer is based on VDI (2013) where heat transfer coefficients, h_i, for each surface i are derived from Nusselt correlations for free convection (cf. Def. 2.1.13). Thereby, the Nusselt number is expressed as a function of the Rayleigh number and the Prandtl number such that the heat transfer

coefficients can be calculated from the ratio of convective to conductive heat transfer

$$Nu_i = \frac{\text{convective heat transfer}}{\text{conductive heat transfer}} = \frac{h_i}{k/L} = \frac{h_i L}{k}\,, \tag{4.17}$$

where k denotes the thermal conductivity and L the characteristic length. The convective heat flux in the setup at hand can then be computed as

$$\begin{aligned} P_{\text{conv}} &= h_{\text{side}}\Big(4 \cdot x_{\text{ins}} \cdot y_{\text{ins}} \cdot (T_{\text{ins}} - T_0) + 4 \cdot x_{\text{HS}} \cdot y_{\text{HS}} \cdot (T_{\text{HS, surf}} - T_0)\Big) \\ &\quad + h_{\text{top}} \cdot x_{\text{HS}}^2 \cdot (T_{\text{HS, surf}} - T_0) \\ &= 28.5\,\text{W} \end{aligned} \tag{4.18}$$

with Rayleigh numbers of $Ra_{\text{side}} = 40.8 \times 10^4$ and $Ra_{\text{top}} = 2.15 \times 10^4$, and resulting heat transfer coefficients $h_{\text{side}} = 11.13\,\text{W}/(\text{m}^2\,\text{K})$ and $h_{\text{top}} = 2.66\,\text{W}/(\text{m}^2\,\text{K})$. The respective widths and heights of the surfaces are denoted by x and y. Since JuROr does not model the radiative heat transfer, (4.19) is now used in the simulation setup to approximate the surface temperature of the heated block solely based on convection:

$$T^{-}_{\text{HS, surf}} = \frac{P_{\text{conv}} - 4 \cdot h_{\text{side}} \cdot x_{\text{ins}} \cdot y_{\text{ins}} \cdot (T_{\text{ins}} - T_0)}{4 \cdot h_{\text{side}} \cdot x_{\text{HS}} \cdot y_{\text{HS}} + h_{\text{top}} \cdot x_{\text{HS}}^2} + T_0 = 150.03\,^{\circ}\text{C} \tag{4.19}$$

assuming that $P_{\text{conv}} = 26.6\,\text{W}$ (taken as the average of both, $P_{\text{conv},\Delta} = 24.7\,\text{W}$ and $P_{\text{conv}} = 28.5\,\text{W}$), $T_{\text{ins}} = T_{\text{HS}} = 270.7\,^{\circ}\text{C}$ and $T_0 = 31.5\,^{\circ}\text{C}$. This reduced temperature, $T_{\text{HS, surf}}$ being 56.7 % of the measured temperature including radiation, serves as wall temperature boundary condition in the simulation setup with point of origin centered at the top of the heated block.

In order to quickly obtain a (quasi-) steady state (obtained in the experiments after 1.5 hours), the initial conditions for the gas temperatures inside the enclosure are prescribed as layers, while the velocity and pressure are set to zero. The domain boundaries are set to $T = 31.5\,^{\circ}\text{C}$ since the inner walls of the enclosure heat up over time due to higher gas temperatures than the ambient temperature. A no-slip boundary condition for velocity ($\mathbf{u} = \mathbf{0}\,\text{m/s}$) is set and a no-flow boundary for the pressure ($\partial_n p = 0\,\text{Pa/m}$) ensures that no fluid enters or exits the domain. Since no material properties, such as emissivity of wood, can be set in JuROr and there exists no model for the heat transfer to or from solid walls but to set the wall temperature, the heated block is constructed as one coherent obstacle with an obstacle wall temperature of $T^{-}_{\text{HS, surf}} = 150.03\,^{\circ}\text{C}$ (except at the bottom), a no-slip

velocity and again a no-flow pressure condition. Based on the grid sensitivity analysis of Meunders (2016), the domain is discretized using two resolutions, $256 \times 256 \times 256$ inner cells, which is considered appropriate for this setup and $128 \times 128 \times 128$ inner cells as a rather coarse grid. However, to achieve low calculation times, the coarser grid with $128 \times 128 \times 128$ inner cells is initially applied and the time stepping is set to $\Delta t = 0.01\,\mathrm{s}$ (possible due to the unconditionally stable nature of JuROr) with a total simulation time of $t_{\mathrm{end}} = 300\,\mathrm{s}$. As physical parameters, the kinematic viscosity is set to $\nu \approx 2.44 \times 10^{-5}\,\mathrm{m^2/s}$ and the thermal diffusivity to $\alpha = 3.31 \times 10^{-5}\,\mathrm{m^2/s}$. For the turbulence model, a turbulent Prandtl number of $Pr_T = 0.9$ is used and the Smagorinsky Constant is set to $C_S = 0.2$. Details of the simulation setup with JuROr can be found in Tables B.17 and B.18 in Appendix B. All following figures show JuROr's time-averaged simulation results of 100 instantaneous recordings for a total simulation time of $t_{\mathrm{end}} = 300\,\mathrm{s}$ with $\Delta t = 0.01\,\mathrm{s}$.

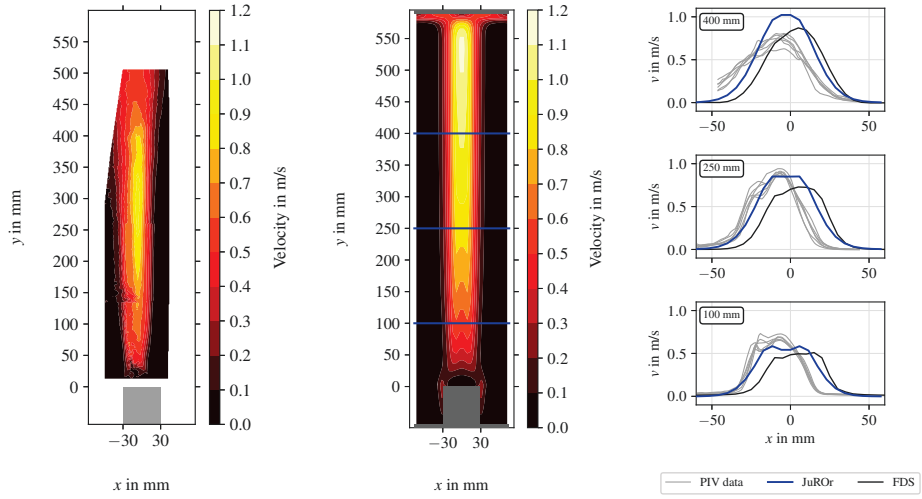


Figure 4.22: Meunders (2016)'s experimental result for one specific run showing speed contour lines (left), JuROr's speed contour lines (middle) and horizontal velocity profiles (right) in the central x-y-plane for Meunders (2016)'s open plume experiment with $128 \times 128 \times 128$ inner cells and $\Delta t = 0.01\,\mathrm{s}$. Blue horizontal lines indicate the heights at which the horizontal profiles of the vertical velocity are taken.

On the left of Figure 4.22, the magnitude, $\sqrt{u^2 + v^2 + w^2}$, of the velocity is shown in the region around the heated block for one specific run of Meunders (2016)'s open plume experiment. In JuROr's simulation results in the middle of Figure 4.22, it can

be observed that the gas velocity increases in height until the gas reaches the ceiling of the enclosure. This increase is also indicated by the horizontal profiles of the vertical velocity at 100 mm, 250 mm and 400 mm above the copper block (see right side of Figure 4.22). Around the surface of the copper block, the velocity builds up at the left and right side, while it is reduced at its top (see middle of Figure 4.22). Here, a slipstream of the heat source is produced, where the temperature is the highest on top of the heated block. This slipstream can also be observed in Figure 4.23 showing JuROr's temperature contours.

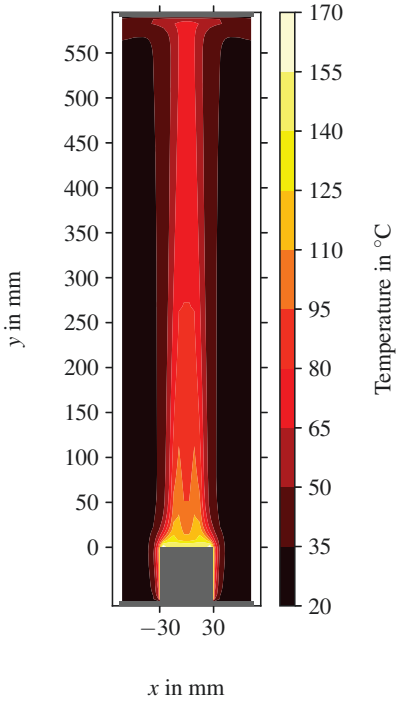


Figure 4.23: JuROr's temperature contour lines in the central x-y-plane for Meunders (2016)'s open plume experiment with $128 \times 128 \times 128$ inner cells and $\Delta t = 0.01\,\mathrm{s}$

These observations are qualitatively in line with Meunders (2016)'s results. However, the observed widening of the plume in the upper half and the narrowing above the heat source (cf. Fig. 4.22 on the left) cannot be validated with JuROr due to the unincisive turbulent behavior of the flow in JuROr's simulation results. When explicitly comparing all velocity profiles on the right side of Figure 4.22, JuROr's

results (in blue) still satisfactorily agree with the experimental results marked in gray indicating multiple measurement runs. JuROr and FDS (in black) overestimate the velocity at 400 mm, while FDS's simulation results (with a 5 mm grid, including radiation) additionally underestimate the gas velocity at the lower heights. Further, the phenomenon of a double plume in the lower part is more dominant in JuROr's simulations than it is in FDS' results. Whereas in JuROr's (and FDS') simulation the plume rises vertically above the heat source, the plume in the experiment bends to the left causing a shift of the velocity profiles. According to Meunders (2016), this bend is (possibly) due to an uneven heating of the source (which is not simulated by JuROr or FDS).

Regarding the temperature distribution (in the front right corner of the enclosure), JuROr gives less satisfactory results as indicated in Figure 4.24. Although the effective convective heat flux (with $29.4\,\mathrm{W} = \int_A \rho_0 c_p v(T - T_0) dA \approx \sum \rho_0 c_p v(T - T_0) \Delta x \Delta z$ at $y = 10\,\mathrm{mm}$) is slightly higher than the estimated convective heat flux of $P_{\mathrm{conv}} = 26.6\,\mathrm{W}$ in average, the temperature is underestimated in the lower and upper layers (cf. profile in Fig. 4.24 for $T^{-}_{\mathrm{HS,\,surf,\,57\%}} = 150.03\,^{\circ}\mathrm{C}$). Only at the center thermocouple, T_3 at $y = 300\,\mathrm{mm}$, JuROr approximates the temperature in the range of the experimental results. Consequently, the sharp interface between the upper and lower layer is less distinct.

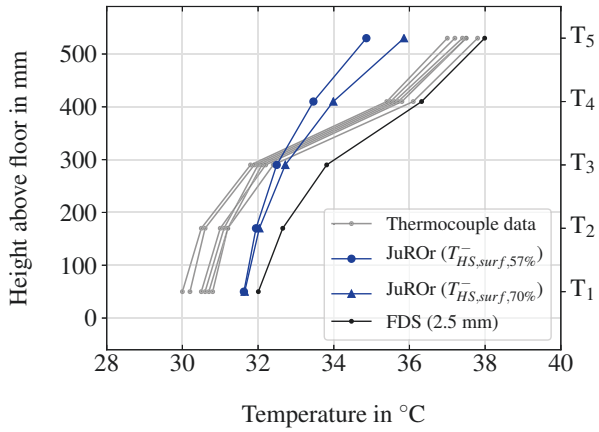


Figure 4.24: Vertical temperature profiles compared for $T^{-}_{\mathrm{HS,\,surf,\,57\%}} = 150.03\,^{\circ}\mathrm{C}$ and $T^{-}_{\mathrm{HS,\,surf,\,70\%}} = 185.36\,^{\circ}\mathrm{C}$ for Meunders (2016)'s open plume experiment

The differences can be explained through various approaches. A missing radiation (and emissivity) model and the crude approximation of the convective heat flux in

JuROr (together with a missing conduction model) lead to uncertainties since radiation is the dominant mode of heat transfer in this setup regarding Meunders (2016). Further, the heat transfer into solids is not taken into account in JuROr, which would effect the temperature. Also, the missing turbulent character of the flow is an indicator for the simulation's uncertainties as already indicated by the unfitting width of the plume (measured in Meunders (2016) with 0.2 m/s as arbitrary velocity threshold for the plume's boundary). Moreover, errors occur due to the simplification assumption of constant material parameters of the gas such as a unity density or a certain thermal diffusivity. The Boussinesq assumption ignoring density differences except in the buoyancy force could be causing errors as well. From the experimental view, also measurement uncertainties exist.

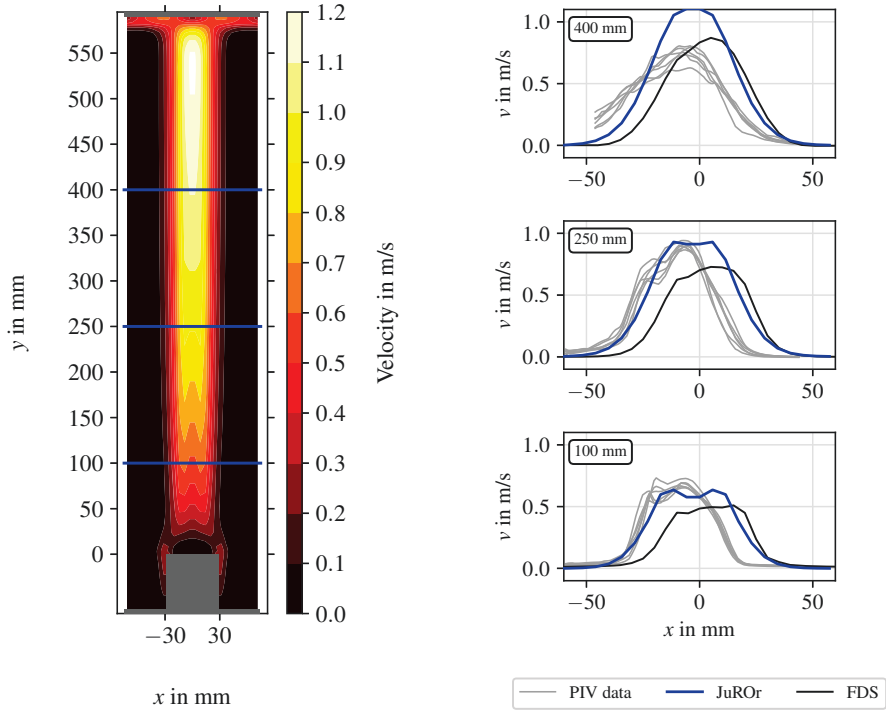


Figure 4.25: JuROr's speed (left) contour lines and horizontal velocity profiles (right) in the central x-y-plane for Meunders (2016)'s open plume experiment with $128 \times 128 \times 128$ inner cells and $\Delta t = 0.01\,\mathrm{s}$, but $\overline{T}_{\mathrm{HS,\,surf,\,70\%}} = 185.36\,^{\circ}\mathrm{C}$. Blue lines indicate the heights at which the horizontal profiles of the vertical velocity are taken.

Whereas FDS' results show better accordance to the experimental results with a finer grid (and smaller time step size), JuROr's results remain basically unchanged when refining the grid and reducing the time spacing to $256 \times 256 \times 256$ cells and $\Delta t = 0.005\,\mathrm{s}$, respectively. Also only increasing the surface temperature of the heated block to 70% of the experimentally measured temperature ($T_{\mathrm{HS,\,surf}} = 185.36\,^{\circ}\mathrm{C}$) does not give satisfactory results. Although the temperature profiles improve as shown in Figure 4.24, JuROr overestimates the velocity profiles especially at 400 mm even more as can be seen in Figure 4.25. These analyses make the missing radiation and boundary models as well as the inadequate turbulence model and Boussinesq assumption even more evident.

Nevertheless, the dimensionless benchmark of the velocity profile compared to the Grashof number again shows good agreement (for all applied setups). At maximum velocity, the Grashof number in JuROr's simulation (with a $128 \times 128 \times 128$ grid, $\Delta t = 0.01\,\mathrm{s}$ time spacing and $T^{-}_{\mathrm{HS,\,surf,\,57\%}} = 150.03\,^{\circ}\mathrm{C}$) is with $2 \times 10^8 < Gr < 1 \times 10^9$ in the range of Noto et al. (1999)'s presented range of $2 \times 10^8 < Gr < 2 \times 10^9$ (cf. Fig. 4.26).

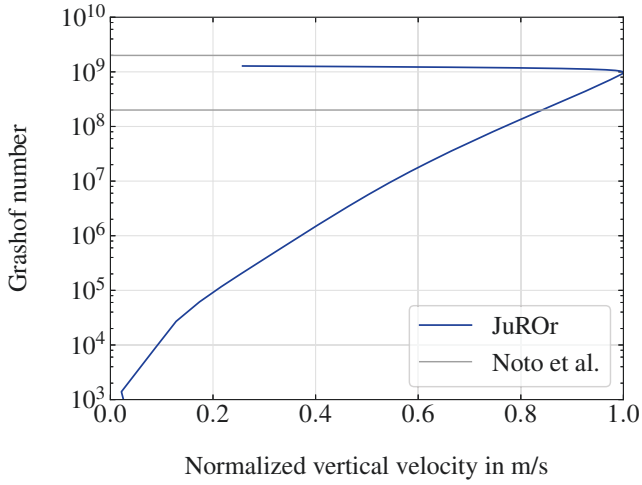


Figure 4.26: Normalized vertical velocity as a function of Gr along the centerline for Meunders (2016)'s open plume experiment with $128 \times 128 \times 128$ inner cells, $\Delta t = 0.01\,\mathrm{s}$ and $T^{-}_{\mathrm{HS,\,surf,\,57\%}} = 150.03\,^{\circ}\mathrm{C}$

In summary, JuROr also performs well enough for the second validation test case based on the previous benchmark assessments.

4.3 Summary

Keeping in mind that JuROr solves the turbulent, incompressible Navier-Stokes equations (2.28) - (2.30) with first-order accuracy as shown in Section 4.1 and neither models radiation nor heat transfer into or from solid walls, the analyses of JuROr in Section 4.2 show acceptable agreement with the experimental results of the shown validation cases. These cases ranged from a small-scale open plume experiment thermodynamically driven by an electrically heated copper block to a real-scale spill plume experiment with a heat source induced by fire.

Now, parallelizing the CPU code and porting it to graphics processing units is necessary in order to achieve a simulation towards real-time or faster than real-time.

Chapter 5

Towards Real-Time and Prognosis Simulation Using a GPU

To speed up the CFD application, JuROr, GPU-accelerated computing is deployed. *GPU-accelerated computing* is the use of a graphics processing unit together with a central processing unit. Thereby, the compute-intensive portions of the application are ported to the GPU to run in parallel, whereas the remainder runs sequentially on the CPU. Consisting of only a few cores including *arithmetic logic units* (ALUs), CPUs are optimized for the sequential processing of tasks. In contrast, GPUs are designed for handling multiple tasks simultaneously due to their parallel architecture consisting of thousands of smaller cores assembled in several streaming multiprocessors (SMs) (cf. Fig. 5.1).

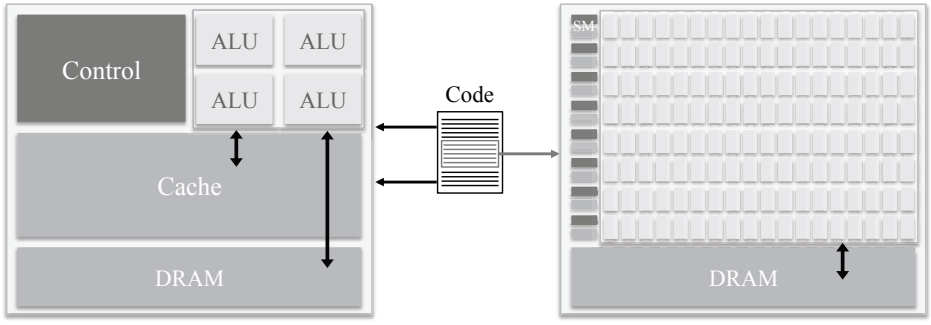


Figure 5.1: Illustrative comparison of a CPU chip (left) and a GPU chip (right)

Further, the memory and memory interfaces differ on CPUs and GPUs. CPUs contain a large system memory (i.e., dynamic random access memory (DRAM)) with medium bandwidth (BW, measured as a rate of data transfer in GB/s) as well as

cache memory close to the control unit and therefore faster access (cf. Fig. 5.1). However, GPUs contain their own high bandwidth graphics memory being smaller than the CPU memory but again consisting of DRAM and cache. CPU and GPU memory is connected to send data from the host (CPU) to the device (GPU) and back or have memory available on both, CPU and GPU (called *unified memory*).

The choice of deploying a GPU not only depends on the application but also on the hardware. In the case of the present work, the choice of using a GPU was additionally driven by economic factors such as acquisition, maintenance and running costs (compared to multiple CPUs, e.g., as in supercomputers Top500 (2017), to get parallelism) as well as the practicality in transportation and installing the hardware on site due to the compact size of GPUs.

Nevertheless, the CPU is necessary since the device does not run without its host (i.e., the program execution on the accelerator is host-driven) and together they combine both their advantages. CPUs are designed to serve a serial computing thread with low latency, whereas GPUs are optimized for throughput. Thereby, *low latency* means that it takes a low number of processor clocks for an instruction to have its data available for use of another instruction. Whereas *high throughput* denotes that it takes a low number of processor clocks for an instruction to perform calculations.

Since a CPU code cannot be run on a GPU straight away, changes and adjustments need to be carried out even if the numerical schemes are already chosen to match the target hardware (cf. Section 2.2).

In the following, the porting process from CPU to GPU as well as optimizations thereof are described. Then a performance portability analysis is conducted. Partial results of the presented work have already been published in Küsters et al. (2017).

5.1 Porting to GPU Using OpenACC

The pragma-oriented OpenACC programming model was deliberately deployed to port the CPU code, JuROr, to the GPU architecture since it has various benefits over GPU programming languages such as CUDA or OpenCL. Using a special GPU programming language results in extra source code for the GPU implementation and different languages are necessary for different brands of GPUs (CUDA for NVIDIA and OpenCL for AMD). An OpenACC-compatible compiler, though, translates the code to a multicore version with OpenMP and to GPU code with CUDA or OpenCL (dependent on the target) – ignoring pragmas for the serial CPU version. Nevertheless, the GPU programming languages can be more performant since special optimiz-

ing can be fully exhausted. Whereas with using OpenACC, various decisions regarding optimization are (deliberately) left to the compiler (e.g., commercial compilers such as PGI, Cray, CAPS, and PathScale or GNU gcc7 as an open-source compiler). Overall, the advantages of OpenACC outweigh its disadvantages for the real-world application at hand. OpenACC's benefits thereby range from ease of implementation, development, and maintenance of only one source code, over performance portability on numerous (multicore) CPU and GPU architectures making the application independent on the hardware for the end-user, to the speedup towards real-time.

Independent on the programming model, the parallel design process remains the same. In order to identify parts of the code which would benefit from GPU acceleration, realize the acceleration, and leverage the speedup in production as early as possible, the so-called APOD (Assess, Parallelize, Optimize, Deploy) process consists of four stages (cf. NVIDIA Corporation (2018b), Fig. 5.2). Thereby, APOD describes a cyclical process.

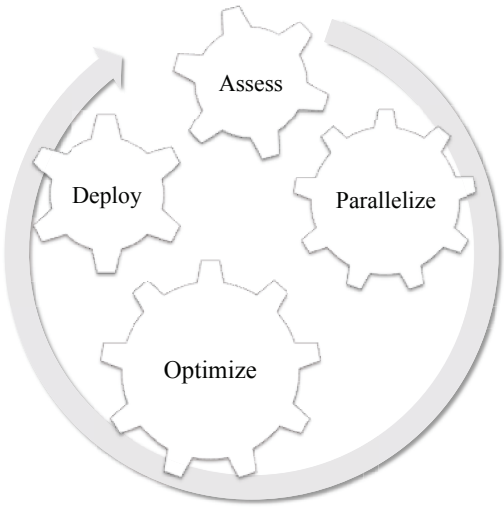


Figure 5.2: Four stages of the APOD process

Assess: The first stage is the assessment of the application locating the most time-consuming parts of the code in order to focus on the most beneficial sections.

Profiling tools (such as Linux GNU GCC Profiling Tool gprof or Intel's VTune for CPU code) can help to identify the hotspot of the code. Also in subsequent iterations of the development cycle, the assessment step is crucial for progress evaluation.

Here, NVIDIA's profiler nvvp (or nvprof as command line tool) can be used for the GPU code. Adding tracers in the code (e.g., with NVIDIA's Tools Extension Library (NVTX)) can additionally help to evaluate the code.

Parallelize: After identifying the code's hotspot and setting acceleration goals or expectations, the hotspot needs to be parallelized. Thereby, OpenACC's directives (for parallelizing, data management, and optimization) apply to the immediately following structured block of code and are constructed in C++ as

`#pragma acc` *directive [clause clause ...] newline*

Thereby, compute and data constructs are differentiated. "A *compute construct* is a `parallel`, `kernels`, or `serial` construct", whereas "a device *data construct* defines a region of the program within which `data` is accessible by the device" (see The Portland Group (2015)).

Optimize: To further improve performance, the implementation can be optimized after completing the parallelization step. Therefore, an opportunity for optimization first needs to be identified, then optimization needs to be applied and tested before the achieved speedup can be measured and repeated. Again profilers support the process of optimization, which can be applied at various levels, such as merging parallel regions, minimizing data transfers, asynchronous launches, or fine-tuning loop schedules and memory accesses.

Deploy: Taking the development to deployment early on, minimizes the risk of any integration issues and making it possible to quantify the achieved speedup.

5.1.1 Parallelization of the CPU Implementation

Based on the APOD cycle, the CPU-based code, JuROr, is now parallelized using OpenACC's 2.5 API and PGI 17.4 (unless otherwise stated). Thereby, the responsibility of producing performance portable code is delegated to the PGI compiler.

For all performance measurements (unless otherwise stated), a benchmark test case in $2D$ is run in double precision with JuROr (cf. Küsters et al. (2017)). This test case was introduced in Chapter 4 and describes a simple analytical solution to the Navier-Stokes equations comprising advection, diffusion and pressure without forces,

energy or turbulence (cf. McDermott (2003))

$$u(x,y,t) \coloneqq 1 - A\cos(x-t)\sin(y-t)\exp(-2\nu t)\,, \tag{5.1}$$

$$v(x,y,t) \coloneqq 1 + A\sin(x-t)\cos(y-t)\exp(-2\nu t)\,, \tag{5.2}$$

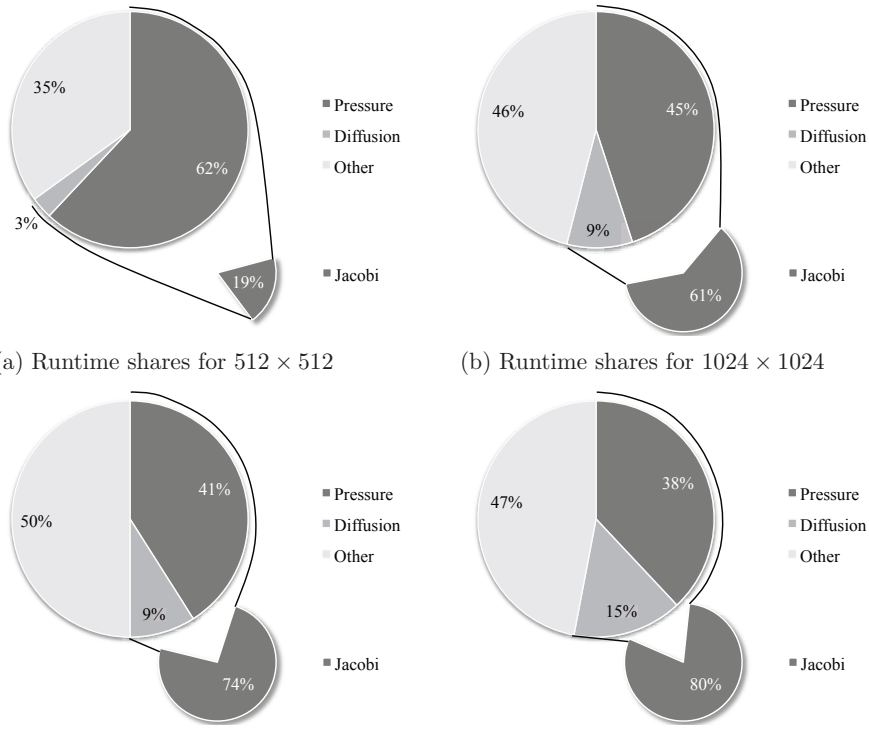
$$p(x,y,t) \coloneqq -\frac{A^2}{4}\left[\cos(2(x-t)+\cos(2(y-t)))\right]\exp(-4\nu t) \tag{5.3}$$

with amplitude $A = 2$, kinematic viscosity $\nu = 1 \times 10^{-3}\,\mathrm{m}^2/\mathrm{s}$ and periodic boundary conditions.

The underlying uniform and collocated grid for the computational domain of a $[0, 2\pi]^2\,\mathrm{m}^2$ square varies for being coarse (with 8×8 inner cells) to very fine (with 4096×4096 inner cells). Therewith, the memory size of the largest dataset comprises approximately $4098 \times 4098 \times 8$ bytes $\approx 135\,\mathrm{MB}$ (including the ghost cells at the boundary). This dataset exceeds the CPU and GPU cache sizes but fits into the CPU main memory and GPU global memory.

After assessing the test case run with JuROr using Intel's VTune profiler, the OpenACC parallelization procedure of JuROr is based on the serial runtime profile in Figure 5.3. Thereby, the diffusion and pressure methods take the majority of the runtime on an Intel Xeon Sandy Bridge E5-2650 0 CPU (other methods include initialization, boundary handling, advection, etc.). The shares for diffusion and pressure highly depend on the problem size, here in $2D$ for $N_x - 2 = N_y - 2 \in \{512, \ldots, 4096\}$. Within these two methods, the 5-point Jacobian stencil operation takes 20 % to 80 % of the serial runtime when measured by Intel VTune's hotspot analysis (again dependent on the problem size).

Thus, the Jacobian stencil describes the hotspot of the CPU code and has been parallelized first (in $2D$) using OpenACC's `kernel`, `parallel` and `data` regions (cf. Chandrasekaran and Juckeland (2018)). In order to maintain the knowledge of directional parameters such as N_x, N_y, N_z or L_x, L_y, L_z instead of losing them when using one-dimensional `arrays`, a new data structure, a `Field` pointer, is defined since `vector`s are not supported in OpenACC (API 2.5). Consequently, the data pointers to the variable fields (with data of length `size`) are used for data transfer and calculation on the GPU (cf. Listing 5.1).



(a) Runtime shares for 512×512

(b) Runtime shares for 1024×1024

(c) Runtime shares for 2048×2048

(d) Runtime shares for 4096×4096

Figure 5.3: JuROr's runtime shares for fractional steps for various grid resolutions (complete circle: shares of total runtime, adjacent section: share of pressure and diffusion runtime)

Listing 5.1: OpenACC directives for $2D$ loops

```
1 // send variable pointers from CPU to GPU
2 #pragma acc enter data copyin(out[:size], in[:size])
3 // parallel directives for nested loops
4 #pragma acc parallel loop independent
5 for (size_t i = 1; i < Nx-1; ++i){
6   for (size_t j = 1; j < Ny-1; ++j){
7     out[...] = in[...]}}
8 // send variables from GPU back to CPU
9 #pragma acc exit data delete(in[:size])
10 #pragma acc exit data copyout(out[:size])
```

After parallelizing the hotspot, all outstanding parallelizable methods – called *kernels* – (e.g., advection, pressure, boundary conditions) are step-by-step ported to the GPU always measuring the speedup in between. Whereas all applicable loops are marked as parallelizable `independent loop`s, a certain loop schedule (number of threads executed by a scalar processor as `vector` or number of thread blocks in a multiprocessor via `gangs`) for NVIDIA's GPUs is not specified in order to leave it up to the compiler to choose an appropriate loop schedule for the corresponding target architecture (cf. Fig. 5.4). This decision contributes to reaching performance portability.

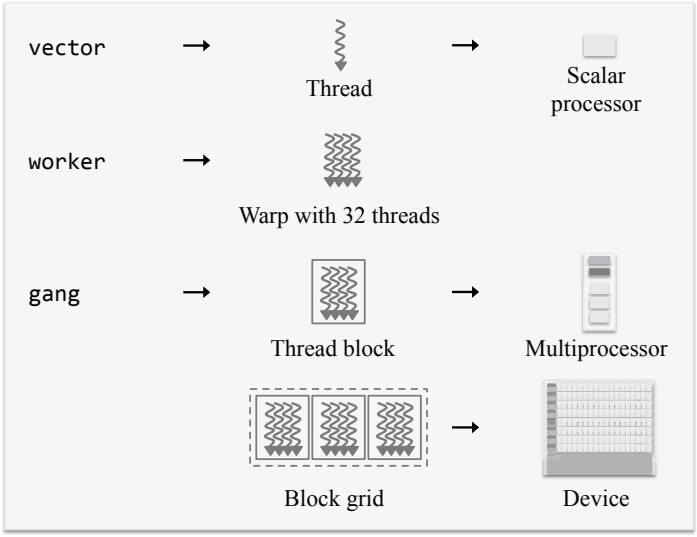


Figure 5.4: Loop schedules when using NVIDIA architectures

Further, the data regions responsible for copying in the variables via `enter data copyin` as well as copying out the variables via `exit data copyout` are expanded to ensure that all data needed for the calculation reside on the GPU. For that, the variables need to be `present` within the parallel regions. In case no data regions are set, the data is sent unnecessarily back and forth from and to the GPU slowing down the runtime (unless the compiler flag `-ta=tesla:managed` is set for NVIDIA GPUs). In the OpenACC CPU versions (serial and multicore), all data transfers are simply ignored by the compiler resulting in the same code base for GPU and CPU execution. Further, all routines with data dependencies between iterations, such as the time integration scheme, stay sequential. Also, analyzing and saving the results for visualization during the time iteration need to be done on CPU. Hence, in intermediate steps the current numerical solution needs to get `update`d on the CPU,

otherwise, the variables still contain their initial values (cf. Fig. 5.5 for a schematic execution model).

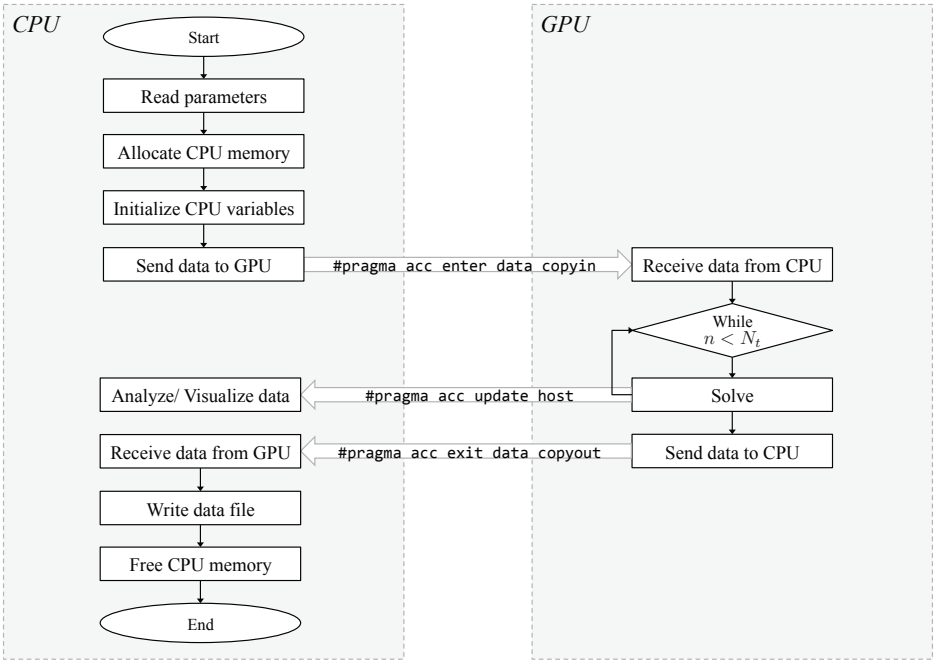


Figure 5.5: Schematic OpenACC execution model

5.1.2 Optimization of the GPU Implementation

After porting all outstanding parallelizable methods to the GPU and spreading the data region as wide as possible such that all computations (besides initialization, analysis, and data output) are performed by the GPU, a number of optimizations are implemented.

First, the parallelism across ($2D$) loops is maximized by merging smaller loops into one kernel. Further, the access to C++ member attributes in parallelized subroutines caused unnecessary CPU-to-GPU as well as GPU-to-CPU transfers. Hence, data management optimizations include the minimization of data transfers, which are avoided by introducing local parameters stored in the stack instead of the heap memory (cf. Listing 5.2 line 3). Regarding the $3D$ implementation of JuROr, the slow running index in the z-dimension is also avoided by using the index lists introduced in Section 3.2 (cf. Listing 5.2 line 7).

Listing 5.2: OpenACC directives for $1D$ row-major lists

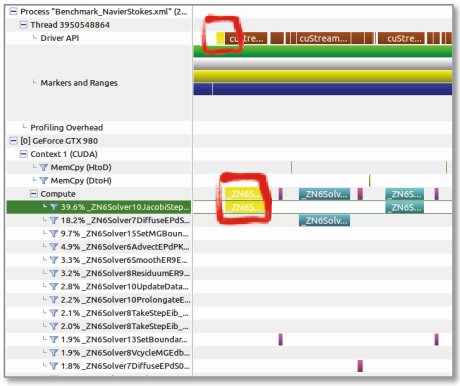
```
1 // get member data as local parameters
2 size_t* d_iList = iList.data()
3 size_t bsize_i = iList.size()
4
5 // parallel directives for inner indices
6 #pragma acc parallel loop independent
7 for(size_t j = 0; j < bsize_i; ++j){
8   const size_t i = d_iList[j];
9   ...
10 }
```

Lastly, pipelining is enabled by `async`hronous kernel launching (cf. Listing 5.3 line 2) from the CPU which reduces the kernel launch latency.

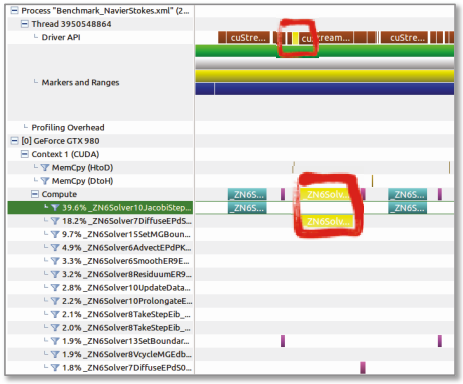
Listing 5.3: OpenACC's asynchronous clause

```
1 // launch kernel asynchronously on the CPU
2 #pragma acc parallel loop independent async
3 for(size_t j = 0; j < bsize_i; ++j){
4   const size_t i = d_iList[j];
5   ...
6 }
7
8 // wait for completion, if necessary
9 #pragma acc wait
```

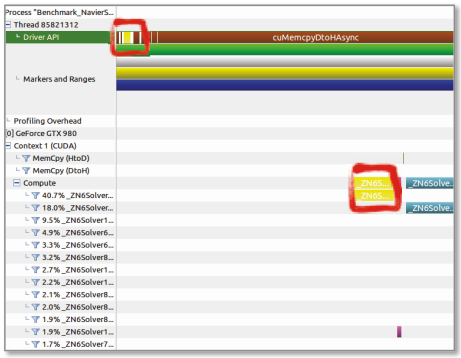
Figure 5.6 shows JuROr's nvvp timeline for kernel launching on the CPU (in the row 'Driver API') as well as the kernel execution on GPU (in the row 'Compute'). For the top Figures 5.6a and 5.6b, the code was started with synchronous kernel launching causing serial kernel launches, whereas the bottom Figures 5.6c and 5.6d show asynchronous launching leading to a reduced kernel launch latency. The left figures of Figure 5.6 show the first considered kernel marked yellow, whereas at the right a subsequent kernel is marked yellow to compare its launch and execution times. When comparing the top (synchronous) from left to right, the second kernel is only launched after the first kernel was executed. For the asynchronous launch (bottom), the kernel launches start way before the kernel executions start.



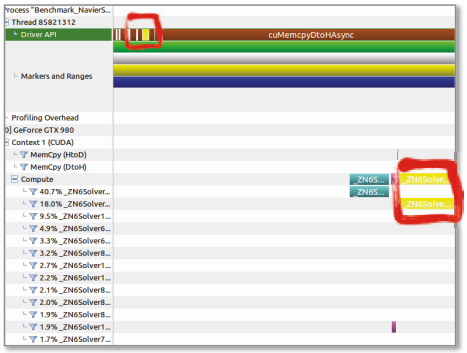
(a) Kernels launched serially by CPU



(b) Second kernel launched after execution of first kernel



(c) Kernels launched asynchronously by CPU



(d) Second kernel launch independent from termination of first kernel

Figure 5.6: Pipelining visualized with NVIDIA's nvvp profiler (in timeline view) using NVTX tracers showing different code fractions in different colors. Yellow background highlighted in a red box shows the chosen kernel.

Latency could be further reduced if the data is small enough to fit into the processor's cache memory but for large datasets, such as the test case, the memory does not fit into cache. Thus, the whole application is likely to be bandwidth bound, since the problem is also of low computational intensity (i.e., the number of operations performed per memory access is low). Investigations regarding memory-boundedness (where the limits of the system's bandwidth are reached) are shown in Section 5.2. Also, the performance portability of the ($2D$) application is tested, before assessing the speedup results in Chapter 6.

5.2 Performance Portability Analysis

To measure if the real-world application, JuROr, performs equally well on different architectures, the performance portability of PGI's OpenACC implementation of JuROr (for the $2D$ case using loops, cf. Küsters et al. (2017)) is investigated across various hardware architectures: NVIDIA's Kepler and Pascal GPUs, and Intel Xeon's Sandy Bridge, Ivy Bridge, Haswell and Broadwell CPUs (using PGI's multicore target). Therefore, roofline models are built for the different architectures, i.e., performance limiters such as Flop/s and memory bandwidth are modeled on the base of the application's (manually-computed) theoretical arithmetic intensity (A.I.) versus the measured intensity by hardware performance counters. Then, the performance portability given as a percentage of sustainable peak performance is analyzed using these roofline models.

5.2.1 Roofline Model

To investigate the performance portability of JuROr's parallelization using the PGI compiler, a roofline performance model is set up allowing the comparison of achieved performance as a percentage share of (sustainable) peak performance. The *roofline model* builds upon peak floating point performance and sustainable memory bandwidth (cf. Williams et al. (2009)). It assumes that computation and communication can be completely overlapped and takes only the slowest data path into account.

Based on these assumptions, the roofline model for JuROr is built for seven different hardware architectures (listed in Tab. 5.1): four Intel CPUs and three NVIDIA GPUs. These computing resources were provided by the Research Center Jülich GmbH (FZJ), and by RWTH Aachen University under the project rwth0207. For the investigations either one CPU socket or one GPU chip of the given hardware is used. GPU-CPU hybrid computations are not considered. Correspondingly, the performance bounds are modeled for either the CPU or the GPU chip, even though the host of a GPU-based system actually adds theoretical peak performance to the GPU performance limiters. The latter would require a corresponding two-device roofline model with inclusion of data transfers. Further, the theoretical arithmetic intensity of JuROr is computed and compared to the measured value by its performance counters. Thereby, the following terminology for performance numbers is used:

- *theoretical:* values defined in or computed from technical hardware specifications or from manual code investigations

- *sustainable:* upper performance values that might be obtained in real-world usually using benchmarks
- *measured/ achieved:* actual measured performance values of real codes on real hardware.

Table 5.1: Used hardware architectures and compilers

Name	Hardware	Used	Compiler & Flags
BDW	2-socket Intel Xeon Broadwell E5-2650 v4 @ 2.2 GHz, 2 × 12 cores	1 socket	PGI 16.10 -ta=multicore
HSW	2-socket Intel Xeon Haswell E5-2680 v3 @ 2.5 GHz, 2 × 12 cores	1 socket	PGI 16.1 -ta=multicore
SNB	2-socket Intel Xeon Sandy Bridge E5-2650 0 @ 2.0 GHz, 2 × 8 cores	1 socket	PGI 16.1 -ta=multicore
IVB	2-socket Intel Xeon Ivy Bridge E5-2640 v2 @ 2.0 GHz, 2 × 8 cores	1 socket	PGI 16.1 -ta=multicore
P100	NVIDIA Pascal P100 SMX2 GPU, 1328 MHz, 16 GB, autoboost off, ECC on, BDW host	1 GPU	PGI 16.10 -ta=tesla:cc60
K80	NVIDIA Kepler K80 with 2 GPUs, 562 MHz, 2×12 GB, autoboost off, ECC on, HSW host	1 GPU	PGI 16.1 -ta=tesla:cc35
K40	NVIDIA Kepler K40 GPU, 745 MHz, 12 GB, autoboost N/A, ECC on, SNB host	1 GPU	PGI 16.1 -ta=tesla:cc35

To get the architectural performance limiters, the peak double precision floating-point performance needs to be computed and the bandwidth measured using (micro) benchmarks. Most architectures nowadays provide boosting capabilities of the clock frequency that are applied if thermal processor conditions allow it. Since this limiter is difficult to track when calculating Flop/s, auto boosting is disabled (where possible) and Flop/s calculations are based on the base operational frequency of the CPU or GPU as reported in Table 5.1. This approach is in line with the reporting rules of the Rpeak value of the Top 500 list (cf. Top500 (2017)). Regarding the memory bandwidth measurement, it holds that achievable memory bandwidth can be significantly lower than the theoretical peak bandwidth. This discrepancy is especially true for systems that employ error correcting code (ECC) such as the given architectures do. Therefore, benchmarks are used to obtain the sustainable memory bandwidth. For the GPU systems, the CUDA version of the GPU-STREAM benchmark is applied and the bandwidth of the triad kernel is evaluated (cf. Deakin and McIntosh-Smith (2017), Deakin et al. (2016)). The measurements are verified using the SHOC benchmark (cf. Danalis et al. (2010)) as well as by comparing them with the published results on the GPU-STREAM website (where possible). For the CPU

systems using the Intel compiler with the flag `-qopt-streaming-stores=always`, the triad results of the OpenMP STREAM benchmark of McCalpin (1995) are evaluated. These results are in turn verified using Intel VTune's memory access analysis that automatically evaluates the local DRAM single-package bandwidth using a (not further specified) micro benchmark. This micro benchmark delivers slightly higher bandwidth numbers. Thus, the CPU performance portability investigations are based on these values. All floating-point performance and memory bandwidth results for CPU and GPU can be found in Table 5.2.

Table 5.2: Floating-point performance and memory bandwidth of the hardware architectures under investigation

Machine	Peak GFlop/s	Peak GB/s	STREAM GB/s	VTune GB/s
BDW	422.40	76.80	60.71	68.00
HSW	240.00	68.00	55.76	61.00
SNB	128.00	51.20	35.88	43.00
IVB	128.00	51.20	40.43	43.00
P100	4759.55	720.00	550.35	N/A
K80	935.17	240.00	149.70	N/A
K40	1430.40	288.00	191.20	N/A

High performance applications are mostly limited by either memory or computational speed. Thereby, memory-boundedness reaches the limits of the system bandwidth, whereas compute bound applications exploit the compute capabilities of the processor. Now, to evaluate which performance boundary (memory or compute capabilities) is hit by JuROr, its arithmetic intensity in Flop per Byte (in Flop/B) is investigated. Since the concept of arithmetic intensity can only be applied on individual kernels, JuROr's hotspot – the Jacobian stencil – is taken into account. Whereas the Jacobian stencil takes up to 80 % of the runtime of the diffusion and pressure methods in serial execution when measured by Intel VTune's hotspot analysis (cf. Fig. 5.3), its parallelized version still takes up to 50 % of the runtime on a K40 for the $2D$ test case with $N_x - 2 = N_y - 2 = 4096$ grid cells in each direction. Thus, again it describes the hotspot and (5.4) can be applied to first compute the stencil's sustainable performance with respect to its performance limiters:

$$\begin{aligned}\text{sustainable performance(in GFlop/s)} \\ = \min(&\text{sustainable BW (in GB/s)} \cdot \text{A.I. (in Flop/B)}, \\ &\text{peak Flop/s performance (in GFlop/s)})\,. \end{aligned} \tag{5.4}$$

In a second step, the achievable performance is measured by using performance counters and its percentage share from the sustainable peak is computed via

$$\text{performance share (in \%)} = \frac{\text{measured performance of the hotspot (in GFlop/s)}}{\text{sustainable performance of the hotspot (in GFlop/s)}} \,. \tag{5.5}$$

For determining the arithmetic intensity of the Jacobian stencil kernel, theoretical arithmetic intensity and measured arithmetic intensity are differentiated. Here, *theoretical arithmetic intensity* refers to the traditional approach of investigating the kernel's source code and manually counting (double) floating-point operations and transferred words. While this approach works well for small regular kernels, it is very challenging for real-world codes that also employ special built-in function calls or complex data access patterns. For example, a call of the `pow` or `sin` function does not deliver an intuitive Flop per Byte ratio and, thus, is little predictable. Therefore, a *measured arithmetic intensity* of JuROr's hotspot based on performance counters is also examined.

Theoretical Arithmetic Intensity: Besides counting floating-point operations, only the slowest data path is taken into account, i.e., access to main memory (on the CPU) or global memory (on the GPU). Therefore, the cache reuse with layer conditions is evaluated to exclude corresponding data accesses. Further, it is verified that non-temporal stores are used on the CPU systems. Overall, for JuROr's hotspot, it holds that

$$\begin{aligned} \text{A.I.} &= \frac{\text{floating-point operations}}{\text{data movement}} \\ &= \frac{12\,\text{Flops}}{(2\,\text{reads} + 1\,\text{write}) \cdot 8\,\text{Bytes}} \\ &= 0.500\,\text{Flop/B}\,. \end{aligned} \tag{5.6}$$

Measured Arithmetic Intensity: The approach of measured arithmetic intensity has the advantage of being applicable for any kind of code. However, it might not reflect the best possible arithmetic intensity, since it also tracks unnecessary data transfers or occurring inflating 'macho-Flop/s'. To get the measured arithmetic intensity, performance counters provide information about the number of double-precision floating-point operations and the transferred bytes. Since no common performance counter interface is available across the selected machines, the counters are manually tracked using different tools: NVIDIA's nvprof 7.5 on the NVIDIA GPU systems and

Intel's VTune Amplifier 2016/2017 on the Intel CPU systems. Unfortunately, a direct mapping from memory access counter values to the hotspot function is not possible, since the counters are based on uncore events. Therefore, VTune's filter capabilities are used to track the hotspot function within the timeline view. Then the values are read off that timeline. For ease of calculation, also VTune's calculated bandwidth numbers are taken. A summary of the applied setups can be found in Tables 5.3 and 5.4.

Table 5.3: Performance counters: Flops

Machine	Flops counter	Tool
BDW	FP_ARITH_INST_RETIRED.SCALAR_DOUBLE, FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE, FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE, INST_RETIRED.X87	VTune
HSW	N/A	N/A
SNB	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE, FP_COMP_OPS_EXE.SSE_PACKED_DOUBLE, SIMD_FP_256.PACKED_DOUBLE, FP_COMP_OPS_EXE.X87	VTune
IVB	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE, FP_COMP_OPS_EXE.SSE_PACKED_DOUBLE, SIMD_FP_256.PACKED_DOUBLE, FP_COMP_OPS_EXE.X87	VTune
P100	flop_count_dp	nvprof
K80	flop_count_dp	nvprof
K40	flop_count_dp	nvprof

Table 5.4: Performance counters: Bytes

Machine	Bytes counter	Tool
BDW	UNC_M_CAS_COUNT:RD, UNC_M_CAS_COUNT:WR	VTune
HSW	UNC_M_CAS_COUNT:RD, UNC_M_CAS_COUNT:WR	VTune
SNB	UNC_M_CAS_COUNT:RD, UNC_M_CAS_COUNT:WR	VTune
IVB	UNC_M_CAS_COUNT:RD, UNC_M_CAS_COUNT:WR	VTune
P100	dram_read_transactions, dram_write_transactions	nvprof
K80	dram_read_transactions, dram_write_transactions	nvprof
K40	dram_read_transactions, dram_write_transactions	nvprof

Due to known hardware restrictions on the Intel Haswell machine, Flop performance counters are not available on this architecture. Nevertheless, the Intel Advisor tool shall be able to measure arithmetic intensities of parts of the code for roofline models

automatically. From the intermediate result (before crashing), the achieved GFlop/s number on the Haswell system is used. Unfortunately, the Intel Advisor is not capable of running the real-world code successfully on all architectures due to crashes. Thus, the performance counter measurements described above are taken for the remaining architectures.

Given the counters in Tables 5.3 and 5.4, the measured arithmetic intensity is computed as follows:

$$\begin{aligned} \text{A.I.}_{\text{CPU}} &= \frac{\text{X87} + \text{SCALAR} + \text{SSE\_PACKED} \cdot 2 + 256\text{\_PACKED} \cdot 4}{(\text{RD} + \text{WR}) \cdot 64\,\text{Bytes}} \\ &= \frac{\text{X87} + \text{SCALAR} + \text{SSE\_PACKED} \cdot 2 + 256\text{\_PACKED} \cdot 4}{\text{BW} \cdot \text{runtime}_{\text{hotspot}}} \end{aligned} \tag{5.7}$$

as well as

$$\text{A.I.}_{\text{GPU}} = \frac{\text{flop\_count\_dp}}{(\text{read} + \text{write}) \cdot 32\,(\text{threads per warp})} \quad , \tag{5.8}$$

where

$$\text{read} + \text{write} = \text{dram\_read\_transactions} + \text{dram\_write\_transactions}. \tag{5.9}$$

5.2.2 Measurement Setup

Based on this methodology, the roofline model using the theoretical and measured arithmetic intensities is built and the performance shares are calculated. In addition to the hardware setups given in Table 5.1, all code versions (from serial over multicore CPU to GPU) are compiled with `-fast -O3`. All performance and counter measurements are run three times and the corresponding average value is taken with runtime deviations below 0.6 %. Furthermore, all measurements are executed on machines with exclusive access. For OpenACC runs on the available CPU systems, thread binding is enabled to ensure good data affinity by setting the environment variables

`ACC_NUM_CORES=<#cores> ACC_BIND=yes MP_BIND=yes MP_BLIST=0,1,<...#cores-1>`.

Since the selection of OpenACC loop schedules is left to the compiler, Table 5.5 gives an overview on the PGI compiler's choice for the Jacobian stencil on different hardware setups. For the CPUs, the outer loop of the Jacobian loop nest gets distributed across `gangs` (i.e., CPU cores), while the compiler attempts to vectorize the inner loop. Contrarily, the compiler chooses a two-dimensional work distribution on the GPUs. Each dimension gets distributed across the GPU's multiprocessors (`gangs`) and the double-precision logic units (`vector`). Whereas the overall thread tile size is

the same across all GPUs, i.e., 128 threads per block, the compiler selects different distributions within the tiles for Kepler and Pascal GPUs.

Table 5.5: Loop schedules for loop nests of the Jacobian stencil kernel chosen and reported by the PGI compiler

Machine	Outer loop	Inner loop
BDW	gang	vector sse + prefetching
HSW	gang	vector sse + prefetching
SNB	gang	vector sse + prefetching
IVB	gang	vector sse + prefetching
P100	gang vector(32)	gang vector(4)
K80	gang vector(4)	gang vector(32)
K40	gang vector(4)	gang vector(32)

5.2.3 Theoretical and Measured Arithmetic Intensity

Results for the theoretical and measured arithmetic intensity of the Jacobian stencil are presented in Table 5.6. Values of the measured arithmetic intensity show only little deviation with values in the range of 0.332 to 0.498 Flop/B across all architectures. In addition, they are roughly in line with the theoretical arithmetic intensity of 0.500, since the Jacobian stencil does not exhibit any special built-in functions or macho-Flop/s.

Table 5.6: Theoretical and measured A.I. of the Jacobian stencil kernel

	Arithmetic intensity in Flop/B		
Machine	Theoretical	Measured	Performance limiter
BDW	0.500	0.340	Memory bandwidth
HSW	0.500	0.332	Memory bandwidth
SNB	0.500	0.386	Memory bandwidth
IVB	0.500	0.354	Memory bandwidth
P100	0.500	0.498	Memory bandwidth
K80	0.500	0.416	Memory bandwidth
K40	0.500	0.418	Memory bandwidth

5.2.4 Performance Portability Results

As an overview, two exemplary roofline models for JuROr running on the Broadwell CPU in Figure 5.7 and the Pascal GPU in Figure 5.8 illustrate the theoretical

(vertically dashed line) and measured arithmetic intensity (circle marker) while also visualizing the performance limiters as rooflines. This representation also shows the achieved performance (circle marker) in comparison to the sustainable memory bandwidth (slope of diagonal line) indicating that the hotspot is memory-bound. The high performance share as ratio of measured to sustainable performance can also be read off the roofline representation by comparing the measured GFlop/s of the circle marker with the sustained GFlop/s of the diagonal line (theoretical A.I.).

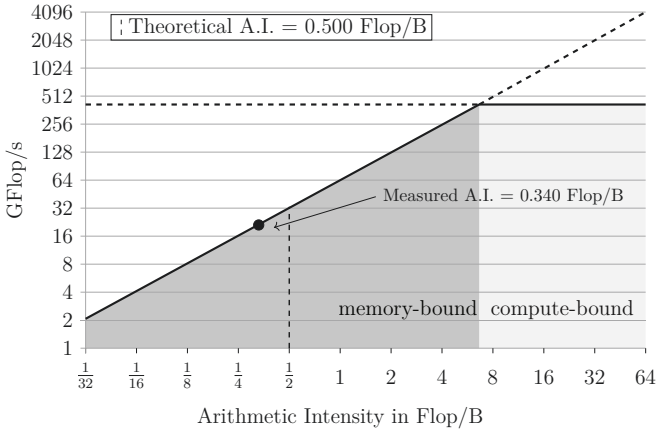


Figure 5.7: Roofline of an Intel Broadwell CPU based on dataset size of $N_x - 2 = N_y - 2 = 4096$

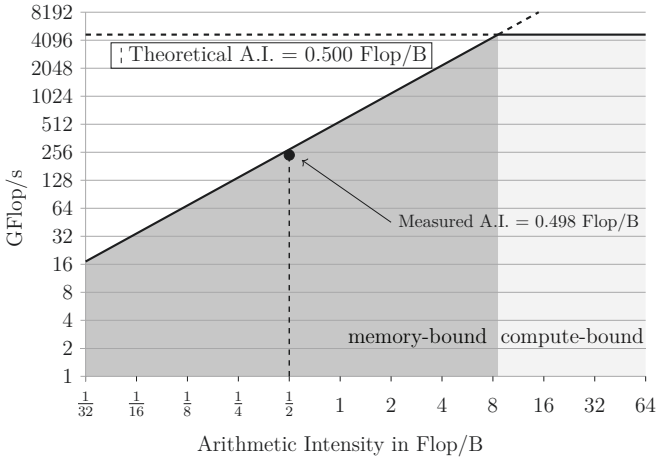


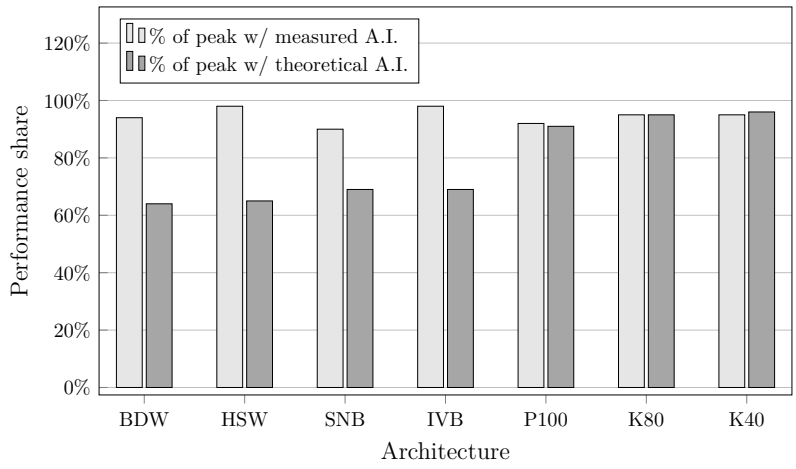
Figure 5.8: Roofline of an NVIDIA P100 GPU based on dataset size of $N_x - 2 = N_y - 2 = 4096$

The absolute performance numbers derived by performance counter measurements running JuROr are listed in Table 5.7. All these numbers, i.e., GFlop/s, GB/s, and runtime in seconds, highly differ across the architectures giving the impression of having non-portable code with respect to performance.

Table 5.7: Flop/s, memory bandwidth and runtime measurement for Jacobian stencil kernel. Bandwidths given in brackets are based on ECC overhead.

Machine	Measured GFlop/s	Measured BW in GB/s	Kernel runtime in s
BDW	21.66	63.71	4.97
HSW	19.81	59.59	5.29
SNB	14.93	38.65	8.54
IVB	14.90	42.04	7.70
P100	251.77	505.14	0.47
K80	71.17	170.91 (−29.08)	1.65
K40	91.47	218.79 (−36.30)	1.29

However, in the following, the performance portability is expressed as performance share to sustainable peak by applying Definition (5.5). The results are illustrated in Figure 5.9.



GFlop/s theor. A.I.	34.00	30.50	21.50	21.50	275.17	74.85	95.60
GFlop/s meas. A.I.	23.12	20.27	16.60	15.24	274.30	62.34	79.93
Measured GFlop/s	21.66	19.81	14.93	14.90	251.77	71.17	91.47

Figure 5.9: Performance share of all considered architectures for a dataset size of $N_x - 2 = N_y - 2 = 4096$

Looking at the theoretical arithmetic intensities, the Jacobian stencil achieves 64 % to 69 % of sustainable memory bandwidth (given by Intel VTune's micro benchmarks) across the CPUs. For the GPU systems, it achieves higher performance shares that range from 91 % to 96 % with respect to the GPU-STREAM results. Since the measured arithmetic intensities are slightly below the theoretical values, they also assume a lower sustainable peak performance in GFlop/s. Therefore, higher performance shares are achieved for the measured arithmetic intensities ranging from 90 % to 98 % on the CPUs with respect to Intel VTune's bandwidth micro benchmark and from 104 % to 108 % with respect to the OpenMP STREAM benchmark results. Thus, JuROr's hotspot delivers higher bandwidth measurements than the STREAM benchmark which may be due to additional transferred bytes for prefetching. For the GPU performance shares, initially, a similar behavior with values from 92 % to 114 % can be observed.

When investigating the appearance of the GPU performance shares above 100 % further, i.e., for the two Kepler architectures K80 and K40, it holds that NVIDIA's device memory performance counters also track transactions caused by ECC overhead (cf. Tab. 5.7). Since these extra ECC bytes do not contribute to the bandwidth achieved by the application, the corresponding values (counters `ecc_transactions`/ `ecc_throughput`) are subtracted from the measured bandwidth of the Jacobian stencil. In contrast, the Pascal architecture supports ECC natively and, hence, does not show ECC effects on bandwidth. With that, more realistic performance shares of 92 % to 95 % across the GPUs are achieved for JuROr.

Overall, although absolute performance numbers suggest otherwise, the results, that are based on the specific hardware and software characteristics, show that for the real-world OpenACC code, JuROr, the PGI compiler is capable of producing performance portable code across different target architectures with a single source code base.

5.3 Summary

Due to the similar and high valued performance shares across architectures, the OpenACC parallelization of the memory-bound hotspot of JuROr shows good performance portability (in $2D$ using loops) relying on the PGI compiler. More importantly, the performance portability can be transferred to the whole application, since the Jacobi method remains the hotspot even after parallelization (in $2D$ using loops) driving the performance.

While hand-tuned or low-level code might generally achieve higher performance, the OpenACC approach benefits from the possibility to maintain one source code base for different architectures while still delivering good performance. To achieve these results, target specific adjustments (e.g., loop schedules) are left to compiler making decisions based on the target architecture.

To further highlight JuROr's advantages, additional test cases, besides revisiting some existing cases, are set up in the next chapter. Here, JuROr's performance regarding its speedup is analyzed comparing JuROr's GPU towards its (multicore) CPU performance, but also against FDS' performance. Lastly, JuROr's ability to run in real-time and faster than real-time is evaluated.

Chapter 6

Analysis of the Real-time and Prognosis Software

After parallelizing and optimizing the CFD code, JuROr, using GPU-accelerated computing, it is of high interest to analyze JuROr's abilities to run in and faster than real-time in order to advance toward the intended support of a prognosis tool in times of emergencies. Therefore, JuROr's speedup in terms of cell updates per second (CUPS) for different grid resolutions is first determined, and JuROr's limits regarding profitability and memory – running on either CPU (serially), multicore CPU, or GPU – are tested using McDermott (2003)'s test case. Secondly, JuROr's runtime performance is compared to FDS' performance for the presented validation scenarios, and, lastly, a different performance measure, the real-time ratio (R) is consulted comparing the simulation and wallclock times of a new setup in order to analyze the real-time and prognosis capability of JuROr.

6.1 Speedup Analysis

The goal of the following speedup analysis is to provide a performance indicator comparing JuROr's performance in *cell updates per second*

$$\text{CUPS} := \frac{\text{number of cells} \times \text{number of time steps}}{\text{runtime}} \tag{6.1}$$

using a GPU as opposed to a reference architecture such as one or multiple CPU cores. This definition then results in a speedup of

$$S \coloneqq \frac{t_{\text{ref}}}{t_{\text{par}}} = \frac{\text{CUPS}_{\text{par}}}{\text{CUPS}_{\text{ref}}}\,, \tag{6.2}$$

for the parallel (here: GPU) wallclock time, t_{par}, in relation to the reference wallclock time, t_{ref}, or cell updates per second, CUPS_{par} and CUPS_{ref}, respectively. The *wallclock time* thereby is the actual amount of time taken to perform a certain task measured by a clock.

Compared is JuROr's GPU performance using an NVIDIA Pascal P100 (PCIe) GPU with 1328 MHz, 12 GB and 56 SMs (streaming multiprocessors) to the multicore and serial performance using one socket of an Intel Xeon Broadwell E5-2623 v4 (BDW) @ 2.6 GHz with 2×8 cores. JuROr is compiled with PGI 17.4 in release mode using `-fast -O3` optimization, autoboost disabled as well as ECC off by setting `nvidia-smi -e 0`. Exclusive access is assured on both, the CPU and the GPU system, and for the OpenACC runs on the CPU system, thread binding is again enabled to ensure good data affinity by setting the environment variables
`ACC_NUM_CORES=8 ACC_BIND=yes MP_BIND=yes MP_BLIST=0,1,2,3,4,5,6,7`.

Again, McDermott (2003)'s analytical solution to the $2D$ Navier-Stokes equations comprising advection, diffusion, and pressure without forces, energy and turbulence is used as benchmark test case

$$u(x, y, t) \coloneqq 1 - A\cos(x - t)\sin(y - t)\exp(-2\nu t)\,, \tag{6.3}$$

$$v(x, y, t) \coloneqq 1 + A\sin(x - t)\cos(y - t)\exp(-2\nu t)\,, \tag{6.4}$$

$$p(x, y, t) \coloneqq -\frac{A^2}{4}\left[\cos(2(x - t) + \cos(2(y - t)))\right]\exp(-4\nu t) \tag{6.5}$$

with amplitude $A = 0.1$ and kinematic viscosity $\nu = 1 \times 10^{-3}\,\text{m}^2/\text{s}$.

The simulation is run for $t_{\text{end}} = 0.002\,\text{s}$ with periodic boundary conditions and initial conditions adhering to the analytical solution at time $t = 0\,\text{s}$. The time is discretized with $\Delta t = 0.0001\,\text{s}$ sized time steps ($N_t = 20$) and the square domain $[0, 2\pi]^3\,\text{m}^3$ is divided into a grid with various resolutions ranging from coarse to fine ($N_x - 2 = N_y - 2 \in \{4, 8, 16, \ldots, 1024, 2048\}$ and $N_z - 2 = 1$ to simulate the $2D$ flow). The variance in the grid resolution shall give insights into the limits of the serial, multicore and GPU-parallel execution of JuROr. The performance measure is cell updates per second, where the number of cells consists of all cells including ghost cells and the runtime is measured during time stepping to only include the

calculation time. Again, the time measurements are averaged from three runs with a maximal deviation of 11 %.

Figure 6.1 shows JuROr's performance in MCUPS for the various resolutions comparing the serial (in light gray), multicore (in gray) and GPU-parallel versions (in black) on the Intel Xeon Broadwell E5-2623 v4 (BDW) and NVIDIA Pascal P100 (PCIe, 12 GB), respectively. The speedup of the GPU version is shown on top of the bars. For 2048×2048 inner cells, the speedup amounts $29\times$ for the GPU-parallel compared to the serial CPU version and $7\times$ when comparing to the 8-core CPU version. The P100 GPU dominates on fine grids – starting from 256×256 to 2048×2048 inner cells – regarding the performance measure of cell updates per second, while the execution of JuROr using multiple CPU cores is profitable for medium resolutions of 128×128 inner cells. For coarse grids, here 4×4 to 64×64 inner cells, the serial CPU version outperforms the multicore and GPU-parallel versions (cf. Tab. 6.1).

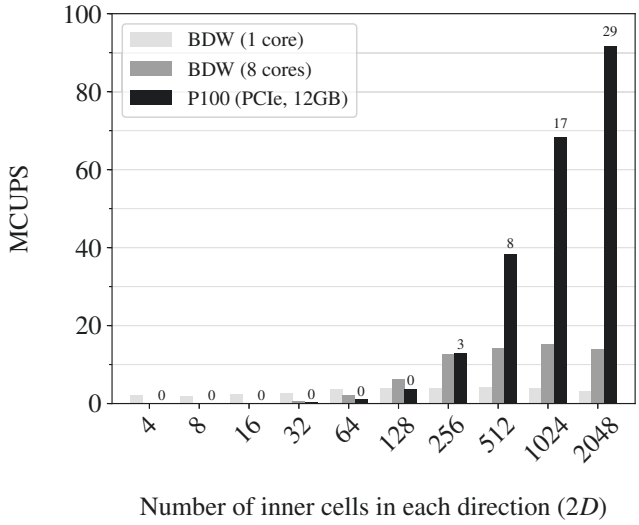


Figure 6.1: McDermott (2003) test case: JuROr's cell updates per second for various resolutions ($2D$) compared for BDW (serial and multicore) and P100 (GPU-parallel)

It is worth mentioning that the P100 PCIe reaches its memory-bound of 12 GB after roughly 40 million cells (including ghost cells). For the purpose of JuROr, though, this limitation is manageable since the simulation of smoke propagation in complex geometries does not need very fine grid resolutions. Exemplarily, Schröder (2016) simulated buoyancy-driven flows in Berlin's underground station 'Osloer Straße' with

FDS using approximately 26 million grid points with grid spacing of $\Delta x = 0.15\,\mathrm{m}$. This resolution suffices to adequately represent filigree geometry components such as stairs, lintels or beams, being relevant for the occurring fluid dynamics. Thus, P100's memory-bound of 12 GB is fully sufficient for JuROr's area of application.

After showing that JuROr runs faster and is able to update more cells per second on GPU than on (multicore) CPU for high-resolution grids and the memory of the GPU at hand suffices for the work's target geometries, the runtime performance of JuROr is compared to FDS' performance for real-world applications.

Table 6.1: McDermott (2003) test case: calculation time, cell updates per second and speedup compared for BDW (serial and multicore) and P100. Underlined architecture shows highest performance.

Inner cells	Machine	Cores/ SM	Time in s	MCUPS	Max. speedup (w.r.t.)	
4×4	<u>BDW</u>	1	0.001	2.160	54× (GPU)	46× (8-core)
	BDW	8	0.046	0.047		
	P100	56	0.054	0.040		
8×8	<u>BDW</u>	1	0.003	2.000	27× (GPU)	20× (8-core)
	BDW	8	0.061	0.098		
	P100	56	0.081	0.075		
16×16	<u>BDW</u>	1	0.008	2.340	16× (GPU)	10× (8-core)
	BDW	8	0.080	0.244		
	P100	56	0.133	0.146		
32×32	<u>BDW</u>	1	0.026	2.703	7× (GPU)	4× (8-core)
	BDW	8	0.097	0.713		
	P100	56	0.179	0.390		
64×64	<u>BDW</u>	1	0.070	3.735	3× (GPU)	2× (8-core)
	BDW	8	0.119	2.203		
	P100	56	0.218	1.206		
128×128	BDW	1	0.261	3.892		
	<u>BDW</u>	8	0.163	6.209	2× (GPU)	2× (serial)
	P100	56	0.268	3.789		
256×256	BDW	1	0.991	4.030		
	BDW	8	0.313	12.76		
	<u>P100</u>	56	0.311	12.88	3× (serial)	1× (8-core)
512×512	BDW	1	3.666	4.330		
	BDW	8	1.104	14.35		
	<u>P100</u>	56	0.416	38.28	9× (serial)	3× (8-core)
1024×1024	BDW	1	15.82	3.993		
	BDW	8	4.171	15.14		
	<u>P100</u>	56	0.924	68.38	17× (serial)	5× (8-core)
2048×2048	BDW	1	80.08	3.149		
	BDW	8	18.20	13.86		
	<u>P100</u>	56	2.754	91.57	29× (serial)	7× (8-core)

6.2 Runtime Performance of Used Cases

The validation cases of Section 4.2 also serve for the evaluation of JuROr's runtime performance, which is then compared to FDS' time-to-solution. First, Steckler et al. (1982)'s Experiment No. 16 is revisited.

6.2.1 Fire Induced Flow Experiment in a Compartment

In order to compare the wallclock times, t_{wc}, of FDS' and JuROr's simulation for Steckler et al. (1982)'s experiment N$^{\mathrm{o}}$ 16, some adjustments need to be applied. First, the radiation model in FDS is turned off. Then, no output must be generated (for both, FDS and JuROr) and the simulation time is set to $t_{\mathrm{end}} = 180\,\mathrm{s}$ (instead of 1800 s). Finally, a domain decomposition into twelve equally sized subdomains needs to be performed for the parallel computation with FDS on JURECA's 2-socket Intel Xeon Haswell E5-2680 v3 @ 2.5 GHz (HSW) and 2×12 cores (cf. Jülich Supercomputing Centre (2018), Jülich Research on Exascale Cluster Architectures). FDS is compiled with Intel ifort 18.0.0 and MPI 3.1.

For comparison purposes, JuROr's multicore simulation is also performed on JURECA compiled with PGI 17.3 and run with enabled thread binding ensuring good data affinity:

```
ACC_NUM_CORES=12 ACC_BIND=yes MP_BIND=yes MP_BLIST=0,1,2,3,4,5,6,7,8,9,10,11.
```

JuROr's GPU version is run on an NVIDIA Pascal P100 (PCIe) GPU with 1328 MHz, 12 GB, 56 SMs and a 2-socket Intel Xeon Broadwell E5-2623 v4 @ 2.6 GHz (BDW) as host. This version is compiled with PGI 17.4 and autoboost as well as ECC off. Additionally, all measurements are executed on the machines with exclusive access.

Since JuROr's time stepping is independent of CFL and Von Neumann conditions, a larger time step of $\Delta t = 0.1\,\mathrm{s}$ is set still assuring convergence and giving satisfying results (with relative errors of $\epsilon_{T_{\mathrm{door}}} = 11\%$, $\epsilon_{u_{\mathrm{door}}} = 30\%$ and $\epsilon_{T_{\mathrm{room}}} = 12\%$ running for $t_{\mathrm{end}} = 90\,\mathrm{s}$). Further, all performance measurements using JuROr are run three times and the corresponding average value is taken with runtime deviations below 1 %.

Table 6.2 and Figure 6.2 demonstrate a speedup of 44× when comparing JuROr's and FDS' 12-core parallel CPU results and a speedup of 126× for JuROr's GPU version compared to FDS' 12-core parallel CPU result regarding the wallclock time of the time stepping.

Table 6.2: Compared wallclock times for Steckler et al. (1982)'s experiment N° 16

Software	Machine	Cores/ SM	t_{wc} in s	Speedup w.r.t. FDS
FDS (v6.5.3)	HSW	12	57 701	-
JuROr	HSW	12	1312	44×
	P100	56	459	126×

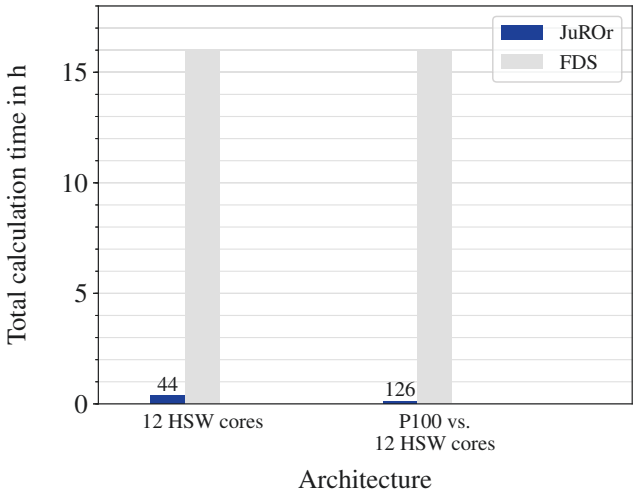


Figure 6.2: Wallclock times (in h) for Steckler et al. (1982)'s experiment N° 16

In order to compare the performance on multicore CPU rather fairly, two characteristics of the P100 GPU are mapped to the number of CPU cores. First, P100's theoretical peak performance of 4700 GFlop/s can be roughly matched to a performance of $4800 = 20 \cdot 240$ GFlop/s of 20 HSW CPUs with each 240 GFlop/s. These 20 HSW CPUs consist of 10 HSW 2-socket nodes with 24 cores each – resulting in 240 cores in total. Secondly, the acquisitions costs of an NVIDIA P100 (PCIe, 12 GB) GPU with approximately $\$ 6000.00 = 4 \cdot \$ 1500.00$ yield four HSW CPUs (two nodes with 48 cores in total) for roughly 1500.00$ each.

Thus, assuming a linear course, the time stepping wallclock time of FDS on 12 HSW cores is extrapolated to $t_{wc,\,48} = 14\,425$ s on 48 cores and $t_{wc,\,240} = 2885$ s on 240 cores. This comparison still results in a speedup of JuROr's GPU version of 31× and 6× compared to FDS' performance on 48 and 240 HSW cores, respectively. Hence, JuROr has the advantage (over FDS) of a faster time-to-solution.

6.2.2 Open Plume Experiment Using Particle Image Velocimetry

Next, Meunders (2016)'s PIV experiment is consulted again in order to evaluate JuROr's runtime performance. Thereby, JuROr's total time-to-solution is compared to the time FDS takes. The FDS simulations are once more run on one socket of JURECA's 2-socket Intel Xeon Haswell E5-2680 v3 @ 2.5 GHz (HSW) and 2×12 cores with exclusive access and FDS is compiled again with Intel ifort 18.0.0 and MPI 3.1. For comparison purposes, JuROr's multicore simulation is also performed on JURECA compiled with PGI 17.3 and run with enabled thread binding ensuring good data affinity by setting the environment variables
`ACC_NUM_CORES=24 ACC_BIND=yes MP_BIND=yes MP_BLIST=0,...,23`.
JuROr's GPU version is executed on the NVIDIA Pascal P100 (PCIe) GPU with 1328 MHz, 12 GB, 56 SMs and a 2-socket Intel Xeon Broadwell E5-2623 v4 as host with exclusive access. This version is compiled with PGI 17.4 and autoboost as well as ECC are turned off.

Here, the total time (including initialization etc.) is considered and in order to compare the same setup, the $P_{\mathrm{HS}} = 78\,\mathrm{W}$ case with the 5 mm mesh is recalculated in parallel with FDS v6.5.3 since Meunders (2016) only reported the total time for a different test case of $P_{\mathrm{HS}} = 96\,\mathrm{W}$. Therefore, it needs to be noticed that these compute times include the radiation model and the output production since these have no negative impact on the compute time in this case. For JuROr, the output is omitted since producing output results in restraining communication efforts between CPU and GPU (which is not necessary for MPI processes such as in FDS). Further, the number of cells in JuROr is reduced to the restriction of the multigrid method to be of magnitude $2^n + 2$ (in each direction) reducing FDS' time by 0.746×. Thus, the times are not fully comparable, but are taken is an indicator (cf. Tables 6.3 and 6.4). Again, the average of three runs is taken with JuROr resulting in variations of less than 0.6 %.

Table 6.3: Comparison of wallclock times for Meunders (2016)'s PIV experiment with 5.0 mm resolution and $t_{\mathrm{end}} = 300\,\mathrm{s}$

Software	Machine	Cores/ SM	Number of cells	t_{wc} in h	Speedup w.r.t. FDS
FDS (v6.5.3)	HSW	24	2.95×10^6	38.00	-
JuROr	HSW	24	2.20×10^6	1.20	32×
	P100	56		0.78	49×

Reducing the FDS times by 0.746× still results in a remarkable speedup of JuROr's multicore version of 24× and for the GPU version of 36× (cf. Tab. 6.4 and Fig. 6.3). In contrast to the speedup results of the Steckler et al. (1982) test case, these speedup numbers are not as high due to the unaltered small time stepping of $\Delta t = 0.01\,\mathrm{s}$.

Table 6.4: Corrected wallclock times for Meunders (2016)'s PIV experiment with 5.0 mm resolution and $t_{\mathrm{end}} = 300\,\mathrm{s}$

			Adjusted by penalty factor 0.746		
Software	Machine	Cores/ SM	Number of cells	t_{wc} in h	Speedup w.r.t. FDS
FDS (v6.5.3)	HSW	24	2.20×10^6	28.35	-
JuROr	HSW	24	2.20×10^6	1.20	24×
	P100	56		0.78	36×

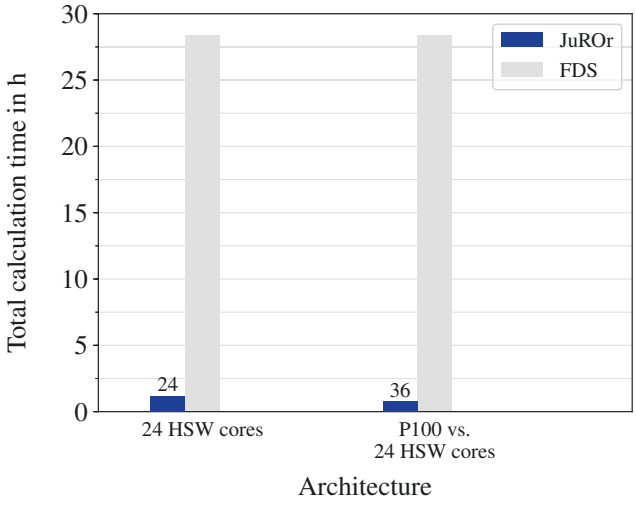


Figure 6.3: Corrected wallclock times (in h) for Meunders (2016)'s PIV experiment with 5.0 mm resolution and $t_{\mathrm{end}} = 300\,\mathrm{s}$

Again comparing JuROr's P100 performance to 48 and 240 Haswell cores by extrapolation of FDS' 24-core total wallclock time of $t_{\mathrm{wc},\,24} = 28.35\,\mathrm{h}$ to the respective wallclock times $t_{\mathrm{wc},\,48} = 14.18\,\mathrm{h}$ and $t_{\mathrm{wc},\,240} = 2.84\,\mathrm{h}$, still results in satisfying runtime performances of 18× and 4×, respectively, with respect to FDS' runtime results. Nevertheless, the simulation was not executed in real-time.

Both validation cases clearly showed the advantage of JuROr to produce results faster than the widely used fire simulation tool FDS. In addition to comparing the runtime performance of JuROr and FDS, it is analyzed in the following if JuROr's goal of a real-time and prognosis simulation can be achieved.

6.3 Real-Time and Prognosis Analysis

A measure to analyze the real-time and prognosis capability of a code is the *real-time ratio*, R, a performance indicator relating the program's wallclock time, t_{wc}, to the simulation time, t_{sim} (cf. Kempe and Hantsch (2017)):

$$R \coloneqq \frac{t_{\mathrm{wc}}}{t_{\mathrm{sim}}} = \frac{\mathrm{CUPS}_{\mathrm{sim}}}{\mathrm{CUPS}} \,. \tag{6.6}$$

It can be related to cell updates per second, where

$$\mathrm{CUPS}_{\mathrm{sim}} = \frac{\text{number of cells} \times \text{number of time steps}}{\text{simulation time}} \tag{6.7}$$

takes the simulation time instead of the actual runtime into consideration.

Now, a code is called *real-time* capable if the real-time ratio equals one, thus the calculation takes as long as the time simulated, $R = 1$. Is the real-time ration smaller than one, the code is even capable of giving a *prognosis*, $R < 1$. In the case of prognosis capability, the time difference, $t_{\mathrm{sim}} - t_{\mathrm{wc}}$, indicates the *lead time*.

JuROr's real-time and prognosis capability using NVIDIA's P100 (PCIe, 12 GB) is investigated with a simple test case relevant for fire protection. Therefore, a tunnel as in Figure 6.4 is set up with width, height, and depth of 150 m × 5 m × 8 m. The center of a volumetric heat source is placed 50 m from the left and 2 m from the front boundary at the bottom of the geometry. Its full width at half maximum extents 2.5 m in length, 4 m in depth and 4 m in height. This heat source dimension is in alignment with Schneider (2006)'s design fire scenario 02.02.003 of a burning car with a heat release rate of $\dot{Q} = 8.3\,\mathrm{MW}$, which is reduced for JuROr by a radiative fraction of 20 % to $\dot{Q} = 6.64\,\mathrm{MW}$ (cf. McGrattan et al. (2017b)) with a ramp-up time $\tau = 5\,\mathrm{s}$. As initial condition, the velocity and pressure are set to zero, whereas the temperature is initialized with five equally and vertically distributed layers, each with a temperature out of $T_0 \in \{30.5, 30.9, 32.1, 35.7, 37.4\}\,°\mathrm{C}$. The domain boundaries consist of no-slip walls for the velocity ($\mathbf{u} = \mathbf{0}\,\mathrm{m/s}$) and zero-gradient for pressure ($\partial_n p = 0\,\mathrm{Pa/m}$) at the front, back, top and bottom domain boundary.

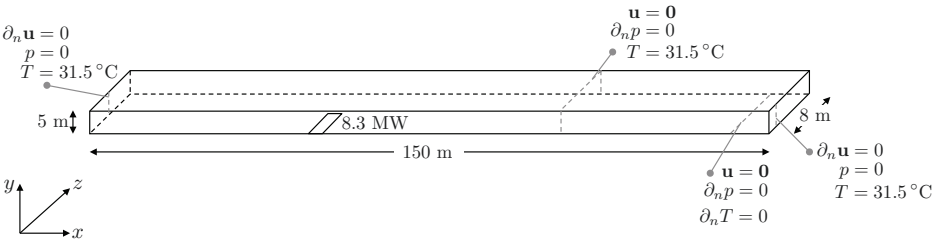


Figure 6.4: Real-time test case: tunnel setup with a volumetric heat source

Zero-gradient velocities ($\partial_n \mathbf{u} = 0\,\mathrm{m/s}$) as well as zero pressure ($p = 0\,\mathrm{Pa}$) are set on the left and right domain boundaries. The wall temperature at the boundaries is set to $T = 31.5\,°\mathrm{C}$ with an adiabatic bottom ($\partial_n T = 0$ °C/m to prevent heat transfer). Further physical, numerical and solution method parameters can be obtained from Tables B.19 and B.20 in Appendix B.

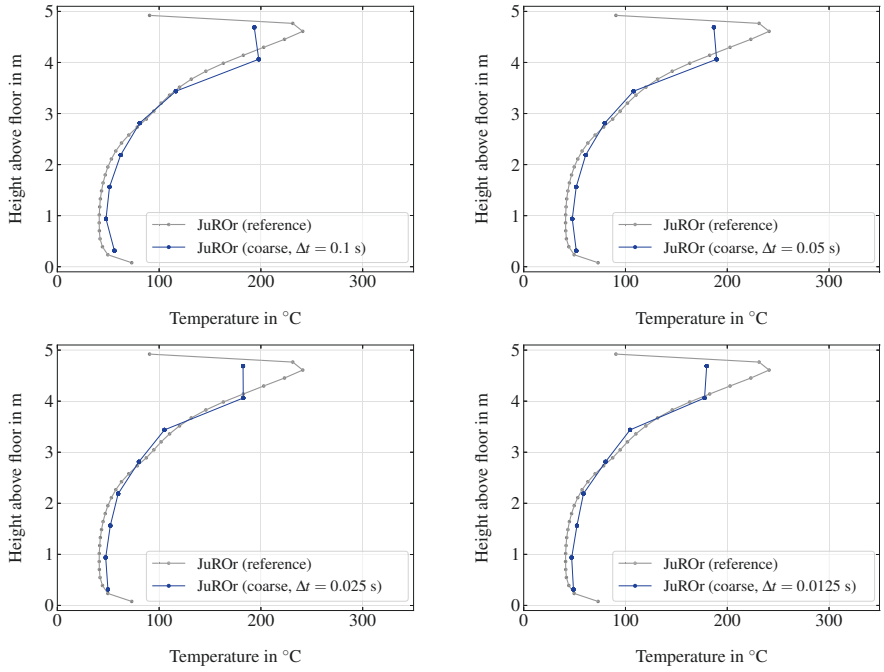


Figure 6.5: Real-time test case: accuracy analysis for coarse grid with various time stepping sizes, $\Delta t \in \{0.1, 0.05, 0.025, 0.0125\}\,\mathrm{s}$

JuROr is run with various time stepping sizes, $\Delta t \in \{0.1, 0.05, 0.025, 0.0125\}\,\mathrm{s}$, and spatial resolutions, coarse, medium, and fine, of roughly $\Delta x \in \{0.6, 0.3, 0.15\}\,\mathrm{m}$,

$\Delta y \in \{0.6, 0.3, 0.16\}$ m and $\Delta z \in \{0.5, 0.25, 0.13\}$ m complying with the results of the speedup analysis in Section 6.1 stating that JuROr's GPU version outperforms in high-resolution problems. Figures 6.5 - 6.7 show JuROr's numerical results after a simulation time of $t_{\text{sim}} = t_{\text{end}} = 500$ s. Here, the vertical temperature profile 2 m to the left of the heat source center and in 2 m depth is compared to the numerical solution with the finest spatial and temporal resolution, $\Delta t = 0.0125$ s and $\Delta x \approx 0.15$ m, respectively, as a reference (in gray).

This accuracy analysis is conducted in order to obtain the limits of the chosen spatial and temporal resolutions. As an example of the coarse time resolution, $\Delta t = 0.1$ s, Figure 6.5 reveals that the temperature above 4 m height is underestimated by JuROr. Thus, the results for the coarse time resolution need to be handled with caution keeping the temperature underestimation at the top of the tunnel in mind. All other setups provide satisfying (or slightly overestimated) results when compared with the finest (numerical) resolution result (cf. Fig. 6.6 and Fig. 6.7).

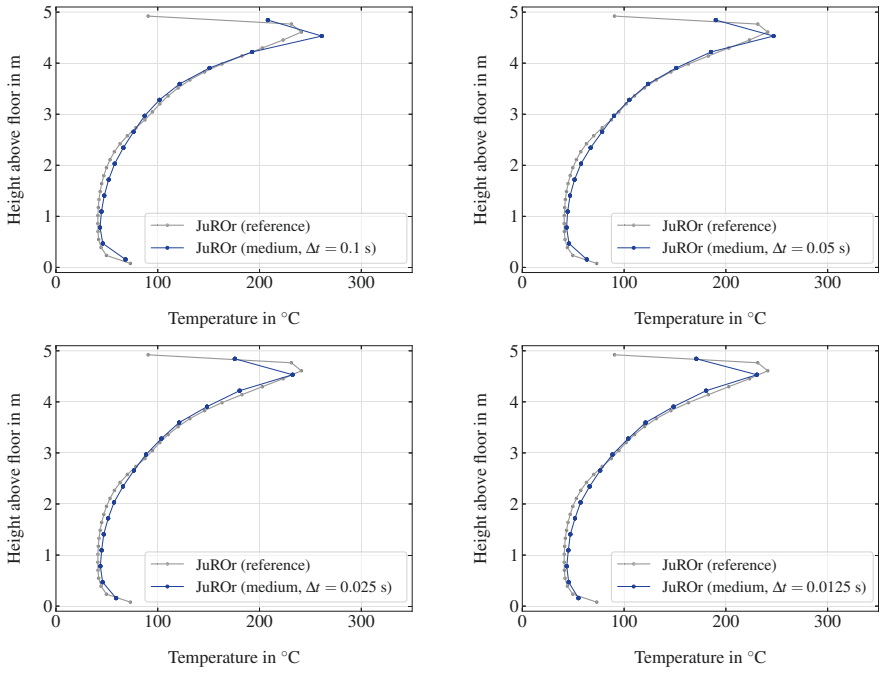


Figure 6.6: Real-time test case: accuracy analysis for medium grid with various time stepping sizes, $\Delta t \in \{0.1, 0.05, 0.025, 0.0125\}$ s

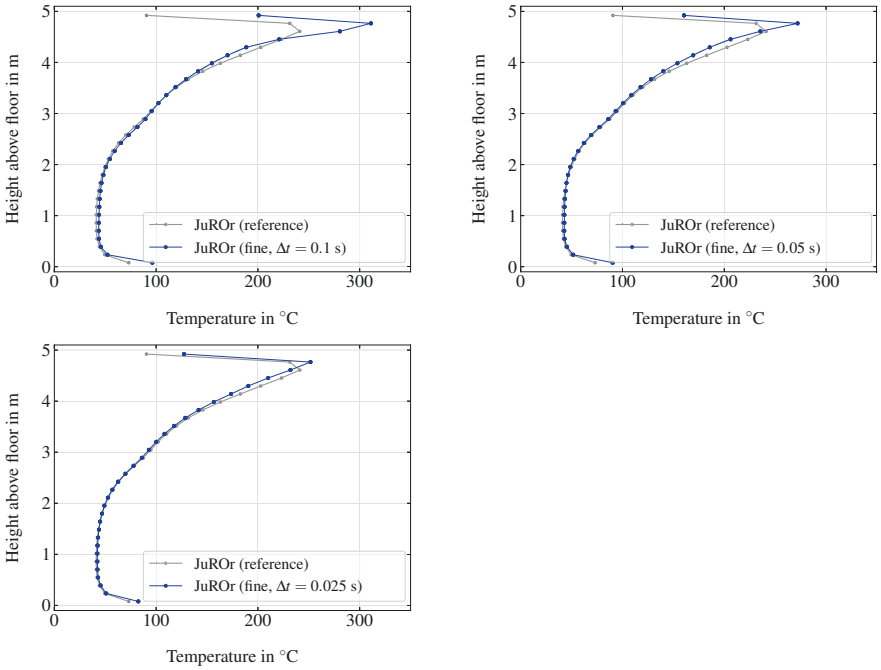


Figure 6.7: Real-time test case: accuracy analysis for fine grid with various time stepping sizes, $\Delta t \in \{0.1, 0.05, 0.025, 0.0125\}$ s

Now, when comparing the time stepping wallclock time of JuROr's GPU-parallel version (with exclusive access) with the total simulation time of $t_{\text{end}} = 500$ s or the respective cell updates per second, the real-time ratio, R, in brackets, Table 6.5 indicates that for all simulations with the coarsest spatial resolution JuROr runs in and faster than real-time. For the medium resolution, the real-time ratio yields $R < 1$ for the three coarser time steppings and just slightly above one, $R \approx 1.3$, for the finest time stepping. Regarding the finest grid resolution, only the coarsest time stepping results in a forecast.

Table 6.5: Real-time test case: JuROr's cell updates per second (MCUPS) and corresponding real-time ratio (R) for different setups using a P100 GPU

Inner cells \ Δt in s		0.1	0.05	0.025	0.0125
coarse:	$256 \times 8 \times 16$	3.48 (0.13)	3.65 (0.25)	3.73 (0.50)	3.77 (0.99)
medium:	$512 \times 16 \times 32$	16.67 (0.19)	17.67 (0.36)	18.39 (0.68)	19.08 (1.32)
fine:	$1024 \times 32 \times 64$	31.96 (0.72)	34.17 (1.35)	36.13 (2.55)	40.97 (4.50)

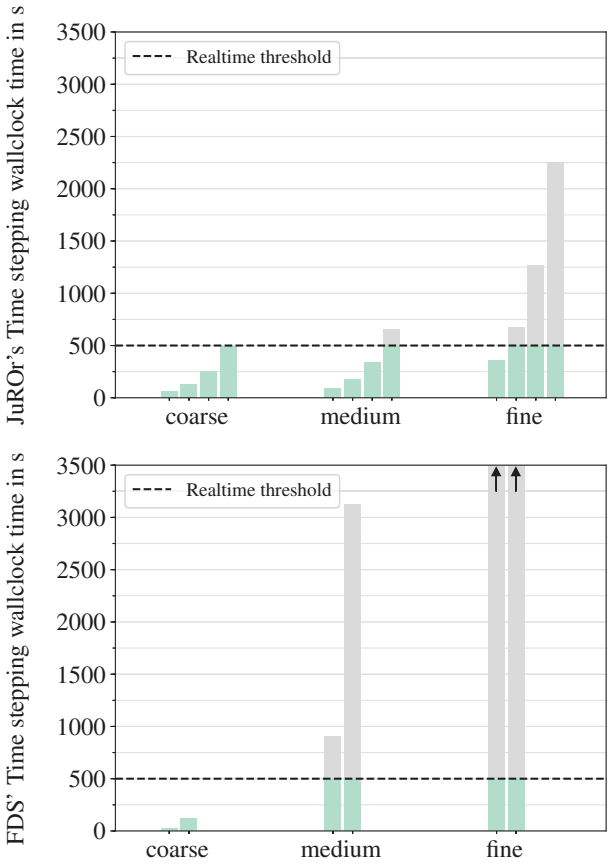


Figure 6.8: Real-time test case: time stepping wallclock times for JuROr's GPU version (at the top with time steps $\Delta t \in \{0.1, 0.05, 0.025, 0.0125\}$) and FDS' CPU parallel version (at the bottom in parallel with 240 and 48 cores, where fine grid wallclock times of $t_{\text{wc, 48}} = 8820\,\text{s}$ and $t_{\text{wc, 240}} = 44\,100\,\text{s}$ rise above the ordinate's limit indicated by black arrows)

The top of Figure 6.8 shows the corresponding time stepping wallclock times of JuROr's GPU-version compared to the total simulation time of $t_{\text{end}} = 500\,\text{s}$ (indicated as a dashed threshold line) for coarse to fine spatial resolutions with various time steps $\Delta t \in \{0.1, 0.05, 0.025, 0.0125\}$ within each spatial resolution block. For eight out of twelve simulation setups with simulation time of 500 s, JuROr runs in or even faster than real-time (indicated in green below the real-time threshold line) with lead times up to roughly 430 s meaning that the predictions are actual forecasts

while maintaining a sufficiently high accuracy. For this analysis, the average of three runs with a maximal variation of 6 % is again taken. Also, JuROr is again compiled with PGI 17.4 in release mode using `-fast -O3` optimization, autoboost disabled as well as ECC off. Worth mentioning is that, although the time stepping wallclock time is assessed, the total runtime (including parameter parsing, setup of boundary lists as well as initialization, but excluding output and visualization) is closer to the time stepping time the larger the problem size and therefore, the differences can be neglected in this analysis.

On the bottom of Figure 6.8, the time stepping wallclock times of FDS' (v6.5.3) simulation of the tunnel on JURECA's Intel Xeon Haswell E5-2680 v3 @ 2.2 GHz (HSW) are shown again for a coarse to a fine grid, but with time stepping based on the CFL-condition. The ordinate axis is deliberately cut off at $t_{\mathrm{wc}} = 3500\,\mathrm{s}$ to ensure comparability with JuROr's results. Compared here are the different amounts of processors used to run FDS in parallel while still being comparable with the characteristics of the P100 GPU: 240 cores based on a performance match (left bar in blocks) and 48 cores based on an acquisition cost comparison (right bar in blocks). Thereby, the time stepping wallclock time for 240 cores is extrapolated from a run with 192 cores assuming a linear course. Table 6.6 shows the underlying values of FDS' time stepping wallclock times and the corresponding real-time ratios (R) in brackets indicating that FDS is only able to run faster than real-time for the coarse grid in this setup, but with a faster time-to-solution than JuROr for the case of 240 cores on the coarse grid.

Table 6.6: Real-time test case: FDS' time stepping wallclock times in s and corresponding real-time ratio (R) for different setups

Inner cells \ Number of CPU cores		240	48
coarse:	$256 \times 8 \times 16$	24 (0.05)	121 (0.24)
medium:	$512 \times 16 \times 32$	911 (1.82)	3123 (6.25)
fine:	$1024 \times 32 \times 64$	8820 (17.64)	44 100 (88.20)

6.4 Summary

The speedup analysis in Section 6.1 revealed that JuROr's GPU version achieves an acceleration of 29× on an NVIDIA Pascal P100 (PCIe, 12 GB) GPU compared to JuROr's serial single-socket version on an Intel Xeon Broadwell E5-2623 v4 (BDW)

using the McDermott (2003) test case with 2048×2048 inner cells. Compared to JuROr's multicore version with eight cores on the BDW CPU, its GPU version still reaches a speedup of $7\times$. Including a larger range of spatial grid resolutions, the analysis showed that JuROr's GPU version outperforms the multi- (and single-) core version for fine grids from 256×256 to 2048×2048 inner cells and more, whereby the memory limit of 12 GB is reached simulating with roughly 40 million cells in total. The multicore version of JuROr succeeds in the specific setup with 128×128 inner cells. For the coarser grids (4×4 to 64×64 inner cells) the serial version on CPU dominates regarding runtime or cell updates per second due to high memory effort on multicore CPUs and GPU.

After assessing the speedup using the verification test case of McDermott (2003) with an analytical solution, the used validation cases were revisited in Section 6.2 to assess JuROr's runtime performance in real-world applications compared to FDS' performance. Here, a speedup of $126\times$ was achieved simulating Steckler et al. (1982)'s Experiment No. 16 with JuROr's GPU version on NVIDIA's Pascal P100 (PCIe, 12 GB) with respect to FDS' wallclock time of the time stepping for the CPU-parallel execution on 12 cores of Intel's Xeon Haswell E5-2680 v3. Regarding the second validation case of Meunders (2016)'s PIV experiment, JuROr achieves a speedup of $36\times$ using the P100 compared to 24 Intel Haswell cores.

Lastly, JuROr's real-time and prognosis capability was analyzed in Section 6.3 in order to advance toward a fire fighting support tool with fast time-to-solution while maintaining high accuracy. Here, a tunnel test case was set up with buoyancy-driven flow. The chosen spatial and temporal resolutions were successfully tested regarding accuracy based on a vertical temperature profile close to the heat source. With these compatible resolutions, the real-time ratio was determined revealing that JuROr is real-time and prognosis capable in eight out of the twelve setups at hand.

To conclude, JuROr serves as a software basis for a real-time and prognosis capable simulation of smoke propagation using a GPU. However, in order to act as a support tool in real-world emergencies, JuROr needs to be further developed including in-situ visualization and user interfaces. These and other developments of JuROr such as data assimilation are left to be handled in the future.

Chapter 7

Closing Remarks

7.1 Conclusions

The evaluation of life safety in buildings in case of fire is often based on smoke spread calculations. However, recent simulation models – in general, based on computational fluid dynamics – often require long execution times or high-performance computers to achieve simulation results in or faster than real-time.

7.1.1 Societal Benefit and Value-add for Research Community

Real-time or even faster than real-time executions of CFD models would allow for further application fields, where the predicted data may be used to steer technical systems like smoke extractions systems or dynamic escape routing for the benefit of self-rescue, besides the direct objectives of capturing the high-risk situation (in status-quo) and the dynamic and scenario-based adoption of fire fighting tactics. Here, the decision-makers, such as the operational leadership of a fire brigade or police responsible for direct emergency response measures, and the fire fighters at site, would require practicable resources. This resource should be financially feasible and space-saving for the usage at site, while being able to calculate highly parallel computations in or faster than real-time as well as visualize the results.

To provide these opportunities the study at hand took up the challenge to develop a concept for the real-time and prognosis simulation of smoke propagation in compartments, where the propagation of smoke is complex and hard to predict. Thereby, these methods were summarized in a software basis, called JuROr (*Jülich's Real-time simulation within ORPHEUS*), making use of the advantages of GPUs (being highly

performant, affordable and portable as well as space-saving). The focus of this study lies in providing a proof of JuROr's concept while handling the challenge of balancing practicality with sufficient accuracy. Therefore, the following approach was pursued:

1. Specification of smoke propagation processes and physical, mathematical as well as numerical modeling thereof fully adapted to highly parallel computer architectures (cf. Chapter 2)
2. Development of an expandable and abstract software design concept for central processing units (CPUs), inner boundary handling in $3D$ as well as implementation and testing thereof (cf. Chapter 3)
3. Verification of the code using (semi-) analytical test cases as well as validation of the model by comparison to experimental results of scenarios relevant for fire protection (cf. Chapter 4)
4. Porting of the software to a GPU providing use of various resources with high performance portability while maintaining one source code (cf. Chapter 5)
5. Analysis of JuROr's speedup (using a verification test case) and runtime performance compared to the established Fire Dynamics Simulator (based on the applied validation scenarios) as well as real-time and prognosis capability while maintaining sufficient accuracy (cf. Chapter 6)

Equipped with this software basis, the research community is then able to enhance it by different numerical methods, models, and geometries, or adopt the software for similar applications in the field of transport phenomena.

7.1.2 Main Contributions and Limitations

Following the outlined approach from modeling, over software design and implementation, verification and validation, to real-time and prognosis analysis, results in the main contribution of this study:

> A REAL-TIME AND PROGNOSIS CAPABLE CFD CODE BASE SIMULATING BUOYANCY-DRIVEN TURBULENT SMOKE SPREAD BASED ON FINITE DIFFERENCES AND A LARGE EDDY SIMULATION TURBULENCE MODEL, BEING PERFORMANCE PORTABLE ON CPU AND GPU CONTAINED IN ONE EXPANDABLE, OPEN SOURCE CODE, SUCCESSFULLY VERIFIED USING UNIT, ANALYTICAL AND SEMI-ANALYTICAL TESTS, AND SUCCESSFULLY VALIDATED WITH SCENARIOS RELEVANT FOR FIRE PROTECTION.

In detail, a reduced modeling approach was deliberately chosen and fully adapted to match the target hardware of highly parallel computer architectures such as GPUs. This modeling approach solely regards a large eddy simulation turbulence model with focus on heat and mass transfer in buoyant flows leading to incompressible Navier-Stokes equations neither including combustion, radiation nor pyrolysis. In the CFD model, first-order numerical schemes, which are of implicit nature (where needed), were applied using finite differences on a structured, regular and collocated grid with a fractional step approach decoupling the governing equations. Thereby, the pressure Poisson equation is solved by a (geometric) multigrid solver to ensure incompressibility by orthogonal projection. The model further allows for volumetric heat and mass sources and supports obstacles with individual boundary conditions in three-dimensional space to simulate compartments.

Accordingly, a software design concept was developed and implemented. It was guided by the desire for continuous testing, development, and enhancement as well as reusability by providing all the software and data in an online repository, along with user instructions. The implementation is written in C++ and uses the pragma-oriented OpenACC programming model to provide a one-source code base flexibly applicable on various (multicore) CPU and GPU systems. It was shown that this implementation, called JuROr, produces highly performance portable code with performance shares between 90 % and 95 % across various architectures ranging from CPU to GPU.

During the implementation, verification studies were conducted. They covered manufactured solutions of the fluid dynamic model as well as semi-empirical setups. The results were compared with other models and empirical correlations. They confirmed the anticipated first order of accuracy in space and time and show reasonable agreement with the results of the consulted models. Further, the simulation of relevant experiments for buoyant flows showed sufficiently high accordance with both, the experimental and simulation results of the established Fire Dynamics Simulator.

Besides validation, the speedup, computing power and memory limits in terms of time-to-solution, number of cell updates per second and maximal spatial resolution were assessed. It is worth mentioning, that for all performance analyses the post-processing (such as data output, visualization, or error assessments) was turned off. Thereby, the first assessment revealed a speedup of JuROr's GPU version (on NVIDIA's Pascal P100 PCIe 12 GB) for fine grid resolutions of 29× compared to its serial CPU version (on Intel's Xeon Broadwell E5-2623 v4) and was still up to 7× faster than its multicore version. JuROr's computing power increased up to

approximately 92 MCUPS. Its memory-bounds on the P100 with 12 GB were reached using roughly 40 million grid cells. For JuROr's purpose, though, this limitation of memory is manageable, since the simulation of smoke propagation, for instance, in the underground station 'Osloer Straße' in Berlin, only required a grid resolution of roughly 15 cm which adequately represents filigree geometry components relevant for the occurring fluid dynamics (cf. Schröder (2016)). In a second study, the runtime performance of JuROr was compared to FDS using the validation cases of buoyant flows to assess how JuROr performs in real-world applications. This study revealed a speedup of 126× when simulating a real-scale spill plume experiment in a compartment with JuROr's GPU version on NVIDIA's Pascal P100 (PCIe, 12 GB) with respect to FDS' wallclock time of the time stepping for the CPU-parallel execution on 12 cores of Intel's Xeon Haswell E5-2680 v3 @2.2 GHz. Simulating a small-scale open plume experiment, JuROr showed a speedup of 36× using the P100 compared to 24 Haswell cores. In the last study, JuROr's real-time and prognosis capabilities, while maintaining sufficient accuracy, were successfully determined with the measure of the real-time ratio. Ultimately, JuROr serves as a basis for a real-time and prognosis capable simulation of the relevant processes of smoke propagation by efficiently using the advantages of a GPU.

Nevertheless, it is needless to say that this study is bound by its limitations and scope. First, the limited range of verification and validation test cases together with limited boundary and initial conditions shows JuROr's vulnerability to quantify its accuracy and uncertainty in general. Secondly, the limited range of speedup and time-to-solution test cases reveals a shortage of a comprehensive study regarding real-time and prognosis capability. Then, the scope of real-world applications is quite narrow since the applied test case geometries only illustrate rather simple rooms or enclosures created manually with `XML`-files instead of automatically for instance from computer-aided design (CAD) software. Lastly, JuROr has not been coupled to assisting tools such as a graphical user interface (GUI) with real-time visualization in order to achieve the motivational goal of a decision-making tool utilized by the end-user in real life. These limitations serve as motivation for future developments which are worth investigating in more detail or worth implementing to enhance the presented code base JuROr.

7.2 Outlook

Based on the outlined limitations, the following enhancements would be a beneficial future augmentation for JuROr.

Accuracy: In order to further evaluate and quantify JuROr's accuracy in its results as well as sensitivity and uncertainty of parameters, a comprehensive, but aligned test suite needs to be created. This test suite could be structurally build up as a test matrix as introduced in Münch (2013) consisting of various tests (with analytical solution, numerical solution, (semi-)experimental data) to build trust into the program with chosen parameters, assumptions, and discretizations by assuring fundamental functionality, significance and quality of the prediction. In Münch (2013), the eligibility of a CFD code is examined in three steps:

1. Program qualification: compare requirements for the model to the program's documentation,
2. Program verification: test correct implementation and functionality of the program('s units) with possible first restrictions for the program
3. Program validation: run the tests in the test matrix which are comparable with the application at hand to estimate optimal execution parameters with respect to the balance between runtime and accuracy.

Here, semi-analytical test cases could be a channel flow (with and without an obstacle), flow over a backwards facing step where drift points are compared to literature (cf. Kim et al. (1980); Pronchick (1983); Driver et al. (1987); Lasher and Taulbee (1992)) or a periodic isotropic turbulence simulation in the absence of both molecular and turbulent viscosity or plume experiments as in McGrattan et al. (2017a). Further, the comparison to experimental data could range from small-scale lab experiments such as Meunders (2016)'s spill plume experiment to real-scale experiments in complex buildings such as an underground station in ORPHEUS (2018).

To allow for continuous integration of new or optimized methods or models, this test suite needs to automatically verify newly implemented methods and validate changes to the underlying model. Methodical changes could include a staggered grid approach or higher-order advection schemes such as MacCormack (cf. Selle et al. (2008)) to decrease artificial diffusion, or the Conjugate Gradient method (cf. Hestenes and Stiefel (1952)) as well as mixed boundary conditions for a far-field outflow and

more complex initial conditions such as parabolic inflow using functions or given data. Testing different models could consist of a suitable turbulence model such as Dynamic Smagorinsky (cf. Germano et al. (1991); Moin et al. (1991)), Deardorff (cf. Deardorff (1972)), Vreman (cf. Vreman (2004)) or RANS model (cf. Reynolds (1895)), with a wall function approach (cf. Bredberg (2000)), or adding combustion including a radiation model (cf. Viskanta and Mengüç (1987)).

Furthermore, it is intended to add a coupling layer to JuROr in order to assimilate data from sensors. A potential approach is based on existing models employing either an ensemble Kalman filter (cf. Evensen (2009)) or a deterministic and variational ansatz (cf. Law et al. (2015); Lahoz et al. (2010)), which have both been successfully integrated in the fire engineering context (cf. Cowlard et al. (2010); Jahn et al. (2009); Jahn (2010); Jahn et al. (2012); Jahn (2017) in enclosures and Rochoux et al. (2013a,b, 2014, 2015) for wildfire spread) including the quantification of uncertainties. These inverse modeling frameworks aim to minimize a cost function based on a combination of distances between measurements and model forecasts over time. Which sensors can be used is tested for instance in Cowlard et al. (2010); Jahn et al. (2009); Jahn (2010); Jahn et al. (2012); Jahn (2017), where smoke detectors and sprinkler heads are successfully applied as detection devices for the estimation of fire characteristics such as the growth rate of the fire together with the location of the fire origin.

Speedup: Having the balance between accuracy and practicality in mind, the computational runtime needs to be continuously evaluated and accelerated when implementing new methods or solvers. In order to assure and maintain the real-time and prognosis capability of JuROr, additional relevant test cases should be established. Further, additional analyses regarding the performance should be integrated such as the performance portability assessment of the code for $3D$ cases and quantifying the speedup (or loss) by deploying lists instead of loops to map the domain.

Also, methodological optimizations can be carried out. For instance, only parts which are affected by smoke could be simulated instead of the entire building or room, which is then dynamically extended based on set thresholds (cf. Meyer (2005)). In addition, the percentage of parallelism can be further increased by allowing independent solvers for different variables such as pressure and temperature to run in parallel within the so far sequential fractional steps. For the pressure solver itself, it could be tested to outsource the computationally inexpensive coarse levels of the multigrid method to the host CPU while keeping in mind the additional data transfers.

To improve memory management, pinned or unified memory accessible from CPU

or GPU could be utilized by `ta=tesla:pinned` and `ta=tesla:managed`, respectively. Here, parsing the parameters should be adjusted to use stack instead of heap memory. Shared memory could be declared to be used on the device using OpenACC's `cache` directive. Also mixed precision could improve data management.

Scope of application: In order to enhance the range of applications towards complex infrastructures such as underground stations, it would be beneficial to incorporate static data such as the geometry with openings and obstacles automatically by enabling data import to JuROr from CAD or BIM (building information modeling). Thereby, BIM is a process of generating and managing digital representations of physical and functional characteristics of facilities (cf. Ruffle (1986); Aish (1986); van Nederveen and Tolman (1992)). To avoid long preparation and setup times for the simulation, especially in emergencies, the static data of various existing buildings and infrastructures need to be accumulated beforehand.

In terms of reusability of simulation results, the storing of lists maintaining the static data of the domain and obstacles should be outsourced in such a way that these unchanged lists can be again used for the same static data, but with different physical or numerical parameters. Also restarting a simulation at a certain time (other than zero) with initial conditions from a previous simulation would help to broaden the range of applications.

Further, the applied computing resources could be expanded by testing HPC cloud operation systems, since parallel problems require little or no communication of results between tasks, which makes them highly suited for cloud computing with limited interconnect speeds, but high-throughput workloads. Additional advantages of using a cloud are space- and energy-savings by the end-user. Nevertheless, the limitations of clouds such as data egress and storage in the cloud, possible connection problems, and costs are not (yet) manageable for emergency cases, where the end-user needs to have instantaneous unlimited access as single user without being interrupted.

Handling: To improve JuROr's practical value from the perspective of the end-user, a graphical user interface is inevitable. With this interface, the user should be able to dynamically steer the simulation by changing parameters and to simultaneously view and analyze the simulation results during the simulation to assess the smoke spread and danger based on only a few key characteristics. The underlying framework of coupling the visualization with the simulation and visualizing the results whilst the simulation is running is called *in-situ visualization* (cf. Rivi et al. (2012) and

Buffat et al. (2015)). Therewith, monitoring of the simulation is allowed in-situ, enabling, besides visualization, the analysis of incoming data as it is generated in order to stop or modify the simulation and, hence, conserving resources. This approach also circumvents shortages regarding storing data in or retrieving data from disk. Technically, the continuous visualization can be handled by the computing or an additional GPU, while the data analyses and storage could be handled by the host CPU, which idles during the calculation on GPU anyway.

From the researcher's point of view, all sources and data developed within this work will be provided in an online repository, along with instructions for compiling, expanding, and running JuROr in order to assure continuous development, support and collaboration with researchers and users at the interface between CFD, HPC and civil safety. In addition to being reliable, accessible and expandable from a research standpoint, feedback from the intended users, such as fire fighters and decision-makers such as the operational leadership, lastly needs to be gathered, assessed, translated and then implemented to ensure that tools such as JuROr are meeting the needs of the end-user. This feedback, besides training, can, in the long run, raise the potential for the acceptance and adoption of JuROr.

Appendix A

Compiling, Developing and Running the Software

A.1 Requirements and Code Compilation

The source code will be openly available in a repository of the version control system `git`. To access the code, save the repository locally by cloning (`git clone`). The repository is structured into two folders, `Src` and `Test`. In `Src` all source code is placed, whereas in `Test` all test cases are stored subordinated into folders based on the nature of their solver.

A.1.1 Minimum requirements

The serial CPU version of JuROr can be compiled on Linux or OS X systems with very few tools, whereas the multicore and GPU version need an OpenACC capable compiler. Detailed requirements are listed in Table A.1 (general requirements for serial version, specific for multicore and GPU version).

It is worth mentioning that JuROr's $3D$ GPU version using lists has only been tested with PGI's compiler in versions 17.3 and 17.4, whereas versions $\geq$ 17.9 were not able to compile the code due to compiler bugs.

A.1.2 Compiling the code

Once the code has been checked out and all required software has been loaded, JuROr can be built from the terminal by first running `cmake` to configure the build, then running `make`. Listing A.1 summarizes the steps.

Table A.1: Minimum requirements to compile JuROr on CPU, multicore or GPU

	Purpose	Tool	Version
General	Version control system to obtain the source code	git	≥ 2.0
	Build processor using a compiler-independent method	CMake	≥ 2.8
	Compiler fully supporting C++-11	gcc	≥ 4.7
		clang	≥ 6.1
	Visualization of output	vtk	≥ 5.8
		Paraview/	VisIT
	Testing for consistency of output while developing	python	2.7
Specific	Compiler fully supporting C++-11 and OpenACC	PGI	17.3/17.4
	Profiling (using NVTX)	CUDA	≥ 8.0

Listing A.1: Cloning, building and running JuROr

```
# 1. Clone
git clone https://gitlab.version.<...>/JuROr.git
cd JuROr

# 2. Make and enter a folder for compiling the code
mkdir build
cd build

# 3. Prepare environment (for use of CUDA Tools)
export CUDA_LIB=$CUDA_ROOT/lib64
export CUDA_INC=$CUDA_ROOT/include

# 4. Use CMake to configure the build
# By default JuROr builds in release mode
# with optimizations, without warnings.
cmake ..

# 5. Build JuROr (fast with option -j <#cores>)
make

# 6. Run simulations with appropriate executable
cd Test/<test case>
../../bin/<executable> testcase.xml
```

CMake options:
By default, JuROr is build in release mode, which should be used for installing, benchmarking and producing with JuROr. To compile in debug mode with `-g -O0` flags and warnings, use the `CMAKE_BUILD_TYPE` CMake parameter. Further, CMake uses the compiler which is set by the environment variables `CC` and `CXX`. Check with `cc --version` or `c++ --version`. To change these, use the CMake parameters `CMAKE_C_COMPILER` and `CMAKE_CXX_COMPILER`. Listing A.2 summarizes these options.

Listing A.2: CMake options for building JuROr

```
# In 4. Use CMake parameters to configure the build
cmake -DCMAKE_BUILD_TYPE={Release, Debug} \
      -DCMAKE_C_COMPILER={gcc, clang, pgcc} \
      -DCMAKE_CXX_COMPILER={g++, clang++, pgc++} \
..
```

Based on the GPU's compute capability, the GPU target needs to be set as special flag, e.g., by `-DGPU_MODEL={K40,K80,P100}` resulting, for instance, in the target flag `-ta=tesla:cc60` for NVIDIA's P100 GPU, whereas `P100` is set as default. Here, also the CUDA version can be set, e.g., by `-DCUDA_VERSION={8,9}`, where `cuda8.0` is set as default.

Executables:
Since JuROr is performance portable and applicable to various architectures, there exist several targets when building JuROr (selected by `make <target>`), whereby each executable has a different purpose described in Table A.2.

Table A.2: Various executables of JuROr

Purpose and properties	Architecture	Executable/ Target
Production	CPU - serial	`juror`
- with terminal/ data output,	CPU - multicore	`juror_multicore`
- visualization and analysis	GPU	`juror_acc`
- using unified memory	GPU	`juror_acc_managed`
- using pinned memory	GPU	`juror_acc_pinned`
Convergence testing	CPU - serial	`juror_iter`
- with set number of	CPU - multicore	`juror_multicore_iter`
iterations for diffusion	GPU	`juror_acc_iter`
Benchmarking	CPU - serial	`juror_prof`
- without output or visualization	CPU - multicore	`juror_multicore_prof`
- without analysis,	GPU	`juror_acc_prof`
- with tracing for profiling		
- using unified memory	GPU	`juror_acc_managed_prof`
- using pinned memory	GPU	`juror_acc_pinned_prof`

Using a Script to Compile:

There also exists a `compile.sh` script to compile JuROr. Thereby, only the first step in Listing A.1 needs to be performed, and all other steps (including creation of the build folder, loading modules for a specified workstation and setting the compute capability or CUDA version) are executed automatically. Listing A.3 shows an example for compiling JuROr for an NVIDIA P100 GPU on a specific workstation `zam035`, for which CUDA and PGI modules are loaded and environment variables for CUDA are exported.

Listing A.3: Example of compiling JuROr with a script

```
# 1. Clone
git clone https://gitlab.version.<...>/JuROr.git
cd JuROr

# 2. Compile GPU-target on zam035 using script
./compile.sh --zam035 --juror_acc
```

In case no workstation is specified, the default compiler (which is set by the environment variables `CC` and `CXX`) is used and as default, all executables in Table A.2 are compiled. For more options, type `./compile -h`.

Checking OpenACC compiler output:
During the compilation of GPU targets, the flags `-Minfo=accel` as well as `-ta=<target>,lineinfo` set in `CMakeLists.txt` display all acceleration information such as data regions or kernel generation with loop schedules and show the corresponding lines of the source files (cf. List. A.4 for an example). Here, it is important to check the information in cases of new parallelizations or optimizations with OpenACC. Using the PGI OpenACC compiler, results such as
`Complex loop carried dependence of ... -> prevents parallelization` or
`Loop carried backward dependence of ... -> prevents vectorization`
indicate false usage of `kernel` or `parallel loop` pragmas, whereas
`upper bound for dimension 0 of array '...' is unknown`
shows missing pointer size information in a `data` pragma.

Listing A.4: Example of PGI's compiler output

```
338, Generating present(d_out[:bsize],d_in[:bsize],
   d_iList[:bsize_i],d_b[:bsize])
       Accelerator kernel generated
       Generating Tesla code
      341, #pragma acc loop gang, vector(128) /*
         blockIdx.x threadIdx.x */
```

After compilation during running a simulation, there can still occur errors such as `FATAL ERROR: variable in data clause is partially present on device`. This error indicates that a pointer used by the GPU is not `present` and was not send to the GPU via `enter data`. In order to gain more detailed insights into the data movements or accelerator kernel launches, profiling tools can be utilized or additional verbose output while running the executable can be requested (by the PGI compiler) by setting the environment variable `PGI_ACC_NOTIFY=3` before executing a program. `PGI_ACC_NOTIFY=1` will only print kernel launches, and `PGI_ACC_NOTIFY=2` will only print upload and download lines.

A.2 Code Development and Testing

For code development, request access to the git repository and create a new branch. After successful implementation and testing, file a merge request on git. Developments and testing thereof can be easily added to the existing code as described as follows:

Steps to include a new solver:

1. If necessary, add a new field
   (in `SolverI.cpp`, `TimeIntegration.cpp`, `Visual.cpp`, `Solution.cpp`)
2. Implement the solver itself (according to the interface `DoStep()`) in a new source file and add source and header files in `CMakeLists.txt`
3. Add the solver in if-statements
   (in `main.cpp`, `SolverI.cpp`, `Boundary.cpp`, `SourceI.cpp`)

Inclusion of further methods:

1. Implement the method itself in a new source file (according to the solver interface) and add the source and header files in `CMakeLists.txt`
2. Add the method in if-statements in all solvers using the method

Steps to include more functions for initialization:

1. If necessary, add the initialization to `Functions.cpp`
2. Build an `XML` file with the new initialization function
3. Where applicable, add the function
   (in `SetUp()`, `Init()`, `UpdateSources()` of `SolverI.cpp` and `Solution.cpp`)

Adding new tests in `Test` folder:

1. Build an `XML` file for the new test (e.g., by using the script `xml-builder.sh`) and save it in new folder `Test/<name of case>`
2. If an analytical solution is available, add the solution to `Functions.cpp` as well as
   `Solution.cpp` to calculate the error between the numerical and the analytical solution (in `Analysis.cpp`)
3. Run and save the output in `_ref.dat` files in the `Test/<name of case>` folder to test the consistency in cases of changing the underlying code

4. Add consistency testing via the Python script `verify.py` in the `Test/<name of case>` folder, which simply compares the output files `.dat` with the reference files `_ref.dat` via element-wise comparison using the $L2$-norm.

5. Optional for use of `ctest`:

   - Add the `run.sh` script to test the case for consistency
   - Add the test in `CMakeLists.txt` with `add_test( NAME <...> COMMAND sh ./run.sh $CMAKE_BINARY_DIR/$EXECUTABLE_OUTPUT_PATH/juror WORKING_DIRECTORY $CMAKE_BINARY_DIR/Test/<name of case>`)

Testing:

1. Test for consistency automatically via `ctest ..`, which uses the provided `run.sh` and `verify.py` scripts or manually start simulation and then test for consistency with the script `verify.py`

2. If an analytical solution is available, check the absolute and root mean square errors in the terminal output

A.3 Software Usage and Restrictions

Input File:
JuROr takes an `XML` file as input argument (cf. Listings (A.5) - (A.11) as an example). This `XML` file includes several parts which are constructed with a tree structure. First, the root element is always `JuROr`, then the filename is set and thereafter the physical and numerical parameters such as simulation end time, time stepping as well as, if needed, viscosity, thermal expansion, gravitational constant and thermal diffusion are defined (cf.Listing A.5).

Listing A.5: Example of an XML input file: part I (filename and physical parameters)

```
<?xml version="1.0" encoding="UTF-8" ?>
<JuROr>
  <xml_filename>testcase.xml</xml_filename>

  <physical_parameters>
    <t_end> 300. </t_end>        <!-- simulation end time -->
    <dt> 0.01 </dt>              <!-- time stepping -->
    <nu> 2.44139e-05 </nu>       <!-- kinematic viscosity -->
    <beta> 3.28e-3 </beta>       <!-- thermal expansion -->
    <g> -9.81 </g>               <!-- gravitation -->
    <kappa> 3.31e-5 </kappa>     <!-- thermal diffusion -->
  </physical_parameters>
```

Then, the solver and its methods are set for the respective unknowns (`field`) and suitable parameters are defined (cf. Listing A.6, exemplarily, for the velocity and pressure). These are for instance,

- the advection model,
- the maximum number of iterations, tolerances for the residual threshold, or relaxation weight for the chosen diffusion method,
- the Smagorinsky-Lilly constant for the turbulence model,
- the buoyancy force and its direction (besides the possibility to set the force to zero),
- number of levels and cycles for the multigrid method and corresponding maximal cycles and tolerance for the pre-conditioning in the first step of solving the pressure Poisson equation as well as
- the diffusion solver for the multigrid method and its parameters such as number of relaxations in finer levels and maximum number of iterations and the tolerance for solving at the coarsest level as well as the relaxation weight

Listing A.6: Example of an XML input file: part II (solver and methods for velocity and pressure)

```
14  <solver description = "NSTempTurbSolver" >
15      <advection type = "SemiLagrangian" field = "u,v,w">
16      </advection>
17      <diffusion type = "Jacobi" field = "u,v,w">
18        <max_iter> 100 </max_iter>
19        <tol_res> 1e-07 </tol_res>
20        <w> 1 </w>
21      </diffusion>
22      <turbulence type = "ConstSmagorinsky">
23        <Cs> 0.2 </Cs>
24      </turbulence>
25      <source type = "ExplicitEuler" force_fct = "Buoyancy"
           dir = "y">
26      </source>
27      <pressure type = "VCycleMG" field = "p">
28        <n_level> 6 </n_level>
29        <n_cycle> 2 </n_cycle>
30        <max_cycle> 4 </max_cycle>
31        <tol_res> 1e-07 </tol_res>
32        <diffusion type = "Jacobi" field = "p">
33          <n_relax> 4 </n_relax>
34          <max_solve> 100 </max_solve>
35          <tol_res> 1e-07 </tol_res>
36          <w> 0.6666666667 </w>
37        </diffusion>
38      </pressure>
```

For solving the temperature equation, the methods and their parameters are set, similarly (cf. Listing A.7, lines 39 – 52). Further, for verification cases, the numerical solution can be compared to an analytical solution against which the numerical solution can then be analyzed regarding numerical errors by setting the availability to `"Yes"` (cf. Listing A.7, lines 53 – 54).

Listing A.7: Example of an `XML` input file: part III (methods for temperature and analytical solution availability)

```
39        <temperature>
40          <advection type = "SemiLagrangian" field = "T">
41          </advection>
42          <diffusion type = "Jacobi" field = "T">
43            <max_iter> 100 </max_iter>
44            <tol_res> 1e-07 </tol_res>
45            <w> 1 </w>
46          </diffusion>
47          <turbulence include = "Yes">
48            <Pr_T> 0.9 </Pr_T>
49          </turbulence>
50          <source type = "ExplicitEuler" temp_fct = "Zero"
               dissipation = "No">
51          </source>
52        </temperature>
53        <solution available = "No">
54        </solution>
55    </solver>
```

In case of a volumetric heat source, the `temp_fct` in lines 50 – 51 of Listing A.7 is replaced by `"GaussST"` with certain heat release rate, specific heat capacity, center coordinates of the volume as well as full widths at half maximum (FWHM). A `ramp_fct` is set to `"RampTanh"` with specific ramp-up time (cf. Listing A.8).

Listing A.8: Example of an XML input for a volumetric heat source

```
<source type = "ExplicitEuler" temp_fct = "GaussST"
   ramp_fct = "RampTanh" dissipation = "No">
  <HRR> 50.3 </HRR>      <!-- total HRR (in kW) -->
  <cp> 1. </cp>          <!-- c_p (in kJ/kgK)-->
  <x0> 0. </x0>          <!-- center of Gaussian-->
  <y0> 0.02 </y0>
  <z0> 0. </z0>
  <sigmax> 0.25 </sigmax><!-- FWHM-->
  <sigmay> 0.6 </sigmay>
  <sigmaz> 0.25 </sigmaz>
  <tau> 5. </tau>        <!-- ramp up time -->
</source>
```

After defining the solver and its methods, the domain (the whole physical domain and the computational domain, which is always part of the first) and possible obstacles or surfaces need to be set up as well as the resolution of the computational domain (cf. Listing A.9). The domains and obstacles are set in $3D$ as rectangular boxes defined by their opposing corners, whereby the domains range from their corner cell faces (excluding boundary cells), but obstacles are defined at their corner cell centers. Surfaces are defined as $2D$ rectangles (specified at their corner cell centers) as part of the domain boundary.

Listing A.9: Example of an XML input file: part IV (domain parameters, obstacles and resolution)

```
57  <domain_parameters>
58    <X1> -0.3675 </X1>  <!-- physical domain -->
59    <X2> 0.3675 </X2>   <!-- defined by opposing corners -->
60    <Y1> -0.06 </Y1>
61    <Y2> 0.59 </Y2>
62    <Z1> -0.2875 </Z1>
63    <Z2> 0.2875 </Z2>
64    <x1> -0.3675 </x1>  <!-- computational domain -->
65    <x2> 0.3675 </x2>   <!-- defined by opposing corners -->
66    <y1> -0.06 </y1>
67    <y2> 0.59 </y2>
68    <z1> -0.2875 </z1>
69    <z2> 0.2875 </z2>
70    <obstacle ID="0"    ox1="-0.025840" ox2="0.025840"
71                        oy1="-0.057461" oy2="-0.00160"
72                        oz1="-0.029199" oz2="0.029199"/>
73                        <!-- use cell centers -->
74    <Nx> 130 </Nx>      <!-- computational grid resolution -->
75    <Ny> 130 </Ny>
76    <Nz> 130 </Nz>
77  </domain_parameters>
```

Then, the boundary conditions are defined for the various unknowns (u, v, w also possible separately, p only on its own such as T in Kelvin by itself). The patches (also possible to set separately) are `left, right` in x-direction, `top, bottom` in y-direction, and `front, back` in z-direction and the conditions are set in the fixed order `front, back, top, bottom, left, right`. The `type` can be chosen from Dirichlet, Neumann or periodic with an appropriate `value`. This approach also holds for obstacles and surfaces (cf. Listing A.10).

Listing A.10: Example of an XML input file: part V (boundary conditions)

```
79  <boundaries>
80    <boundary field="u,v,w"
81            patch="front,back,left,right,top,bottom"
82            type="dirichlet"
83            value="0.0" />
84    <boundary field="p"
85            patch="front,back,left,right,top,bottom"
86            type="neumann"
87            value="0.0" />
88    <boundary field="T"
89            patch="front,back,left,right,top,bottom"
90            type="dirichlet"
91            value="304.64" />
92    <obstacle ID="0" field="T"
93            patch="front,back,left,right,top"
94            type="dirichlet"
95            value="423.17" />
96    <obstacle ID="0" field="T"
97            patch="bottom"
98            type="dirichlet"
99            value="0.0" />
100   <obstacle ID="0" field="u,v,w"
101           patch="front,back,left,right,top,bottom"
102           type="dirichlet"
103           value="0.0" />
104   <obstacle ID="0" field="p"
105           patch="front,back,left,right,top,bottom"
106           type="neumann"
107           value="0.0" />
108 </boundaries>
```

Lastly, the initial conditions (set all to zero as default) for the unknowns are set (cf. Listing A.11). Here, various functions are already defined (see `Functions.cpp`). Further, the output format (a `CSV` (comma-separated values) file and/ or `vtk` files) is set.

Listing A.11: Example of an XML input file: part VI (initial conditions and output management)

```
110   <initial_conditions usr_fct = "LayersT" dir="y">
111     <n_layers> 5 </n_layers>
112     <border_1> 0.11 </border_1>          <!-- at cell face -->
113     <border_2> 0.23 </border_2>
114     <border_3> 0.35 </border_3>
115     <border_4> 0.47 </border_4>
116     <value_1> 303.64 </value_1>          <!-- in Kelvin -->
117     <value_2> 304.04 </value_2>
118     <value_3> 305.24 </value_3>
119     <value_4> 308.84 </value_4>
120     <value_5> 310.54 </value_5>
121   </initial_conditions>
122
123   <visualization save_csv="No">
124         <n_plots> 100 </n_plots>         <!-- # of vtk's -->
125   </visualization>
126 </JuROr>
```

Error Sources and Restrictions:
Once the input file is set up, the simulation can be started with an appropriate executable from the `build/bin` file and the `XML` file as argument. The set of executables can be found in Table A.2. In case the terminal shows a `segmentation fault`, there might be errors in the `XML` file. Thereby, the possible non-functionality of the input file can be caused by:

- Filename discrepancies: always set the correct filename in the `xml_filename`,
- Whitespaces: do not use whitespaces in attributes of the `XML` file,
- Grid sizing: use $2^n + 2$ cells for each direction when using the multigrid method,
- Grid positions: use cell centers, when setting up obstacles or surfaces,
- Obstacle dimensions: prevent using one cell thick obstacles or multiple obstacles occupying all inner cells (especially when restricting to the lowest level in the multigrid method).

Besides a `segmentation fault`, numerical instability or precision errors on different architectures in examples of chaotic systems with high turbulence (i.e., high Re) can occur. Then, try refining the grid and/ or the time stepping.

Using a Script to Build an XML Input file:
In order to keep usage errors as low as possible, a supporting script (used from the terminal) is provided for setting up `XML` files. The `xml-builder.sh` has built-in options for different solvers:

- Pure advection (`--advection`)
- Pure diffusion (`--diffusion`) and turbulent diffusion (`--diffturb`)
- Pure pressure (`--pressure`)
- Advection and diffusion (`--burgers`)
- Navier-Stokes equations for velocity (`--ns`)
  - with turbulence modeling (`--nsturb`)
  - with energy equation for temperature (`--nstemp`)
    - with turbulence modeling (`--nstempturb`)
    - with passive scalar equation for concentration (`--nstempcon`)
      - with turbulence modeling (`--nstempturbcon`)

Mandatory to provide are appropriate initial (`--initialconditions`) and boundary conditions (`--boundaryconditions`), which are declared in corresponding `txt` files. If applicable, also all domain obstacles (`--domainobstacles`) and all surface patches (`--domainsurfaces`) have to be explicitly declared via `txt` files, which the script then parses. It is worth mentioning that there is no validation of the chosen (or default) parameters (all shown with `./xml-builder.sh -h`). An example of the usage is given in Listing A.12.

Listing A.12: Example of using the script for building an XML file

```
# 1. To get an overview of the options, type
./xml-builder.sh -h

# 2. To build up a Navier-Stokes solver
# with no external forces in domain [0,1]^3
# with set time and spatial resolution, type e.g.,
./xml-builder.sh --ns \
  --forcefct Zero --forcedir xyz \
  --initialconditions ifile.txt \
  --boundaryconditions bfile.txt \
  --xstartc 0. --xstartp 0. --xendc 1. --xendp 1. \
  --ystartc 0. --ystartp 0. --yendc 1. --yendp 1. \
  --zstartc 0. --zstartp 0. --zendc 1. --zendp 1. \
  --tend 10. --dt 0.01 --nu 0.001 \
  --nx 514 --ny 514 --nz 514 \
  --nplots 100 --solavail No \
  --xml testcase.xml
```

In order to build various XML files with different simulation times, time and/ or spatial resolutions or iterations, the entry of multiple parameters for `--nx`, `--ny`, `--nz`, `--dt`, `--tend`, and `--maxiter` is possible by separation of values with commas. To assure that the different setups are saved in different XML files use the printf specifier `%d` for multiple files (as in `--xml testcase_No_%d.xml`).

Using a Script to Run:
Now, to start one or multiple XML files and save details on the input parameters such as resolution and simulation time as well as output data such as runtime, CUPS, or final errors, the `xml-starter.sh` script in the `Test` folder can be used (cf. Listing A.13 as an example). Available options are

- Choosing an executable (as in Table A.2)
- Specifying the parent directory of JuROr or the absolute path to it with `-d`
- Specifying files to execute (default: all XMLs in the directory the script is started in) with `-f`
- Specifying the output file with `-o`

- Printing all output to screen with `-v`

Listing A.13: Example of using the script for starting an `XML` file

```
# 1. To get an overview of the options, type
./xml-starter.sh -h

# 2. To start a testcase.xml with JuROr's
# GPU version in profiling mode, which lies
# in absolute path /home/software/JuROr/build/bin,
# type e.g.,
./xml-starter.sh --juror_acc_prof\
-d /home/software/JuROr/build/bin\
-f testcase.xml -o output.csv
```

The scripts `xml-builder.sh` and `xml-starter.sh` can in particular be used for analyses such as convergence or speedup testing, where multiple configurations (e.g., in time and grid resolution) need to be build and run.

Appendix B

Detailed Input Data of Test Cases

B.1 Advection Test Case

Table B.1: Advection test case: physical, numerical and solution parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	2	s
Density	ρ_0	1	kg/m^3
Computational domain	(x_1, x_2)	$(0, 2)$	m
	(y_1, y_2)	$(0, 2)$	m
	(z_1, z_2)	$(0, 1)$	m
Domain boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{0}$	m/s
Numerical parameters			
Time resolution	Δt	0.001	s
Initial condition			
- Velocity	$\mathbf{u}$	$A \exp\left(-\frac{1}{2\sigma^2}\left[\sum_i \left(\frac{\mathbf{x}_i - \mathbf{x}_{i,0}}{\mathbf{c_i}}\right)^2\right]\right)$	m/s
- Amplitude	A	1	
- Width	σ	0.031 25	m
- Center	(x_0, y_0, z_0)	$(1.025, 1.025, 0.5)$	m
- Bulk velocity	(c_x, c_y, c_z)	$(0.5, 0.5, 0.25)$	m/s
Number of inner cells	$N_x - 2$	40	
	$N_y - 2$	40	
	$N_z - 2$	1	
Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		

B.2 Artificial Diffusion Test Case

Table B.2: Artificial diffusion test case: physical, numerical and solution parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	10	s
Density	ρ_0	1	kg/m^3
Computational domain	(x_1, x_2)	$(0, 1)$	m
	(y_1, y_2)	$(0, 1)$	m
	(z_1, z_2)	$(0, 1)$	m
Domain boundary condition			
- left, top	T	0	°C
- right, bottom	T	100	°C
- front, back	$\partial_n T$	0	°C/m
Numerical parameters			
Time resolution	Δt	0.001	s
Initial condition			
- Temperature	T	0	°C
- Bulk velocity	(c_x, c_y, c_z)	$(2, 2, 0)$	m/s
Number of inner cells	$N_x - 2$	$\{64, 128, 256\}$	
	$N_y - 2$	$\{64, 128, 256\}$	
	$N_z - 2$	1	
Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		

B.3 Diffusion Test Case

Table B.3: Diffusion test case: physical, numerical and solution parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	1	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	0.001	m^2/s
Computational domain	(x_1, x_2)	$(0, 2)$	m
	(y_1, y_2)	$(0, 2)$	m
	(z_1, z_2)	$(0, 1)$	m
Domain boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{0}$	m/s
Numerical parameters			
Time resolution	Δt	0.0125	s
Initial condition			
- Velocity	$\mathbf{u}$	$A\sin(lx)\sin(ly)\sin(lz)$	m/s
- Amplitude	A	1	
- Wave number	l	$5/2\pi$	
Number of inner cells	$N_x - 2$	40	
	$N_y - 2$	40	
	$N_z - 2$	1	
Fractional step	**Method**	**Parameter**	
Diffusion	Jacobi	100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.4 Diffusion (Hat) Test Case

Table B.4: Diffusion hat test case: physical, numerical and solution parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	1	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	0.05	m^2/s
Computational domain	(x_1, x_2)	$(0, 2)$	m
	(y_1, y_2)	$(0, 2)$	m
	(z_1, z_2)	$(0, 2)$	m
Domain boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{1}$	m/s
Numerical parameters			
Time resolution	Δt	0.02	s
Initial condition			
- Velocity	$\mathbf{u}$	2 if $\mathbf{x} \in [0.5, 1]^3$	m/s
		0 else	m/s
Number of inner cells	$N_x - 2$	32	
	$N_y - 2$	32	
	$N_z - 2$	32	
Fractional step	Method	Parameter	
Diffusion	Jacobi	100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.5 Pressure Test Case

Table B.5: Pressure test case: physical, numerical and solution parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	0.1	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	0.1	m^2/s
Computational domain	(x_1, x_2)	$(0, 2)$	m
	(y_1, y_2)	$(0, 2)$	m
	(z_1, z_2)	$(0, 2)$	m
Domain boundary condition			
- all walls	p	0	m/s
Numerical parameters			
Time resolution	Δt	0.1	s
Initial condition			
- Pressure	p	0	Pa
- Wave number	l	2π	
Number of inner cells	$N_x - 2$	64	
	$N_y - 2$	64	
	$N_z - 2$	64	
Fractional step	**Method**	**Parameter**	
Pressure	Multigrid	2 cycles	5 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.6 McDermott Test Case

Table B.6: McDermott (2003) test case: physical and numerical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	2π	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	$\{0, 0.1\}$	m^2/s
Computational domain	(x_1, x_2)	$(0, 2\pi)$	m
	(y_1, y_2)	$(0, 2\pi)$	m
	(z_1, z_2)	$(0, 2\pi)$	m
Domain boundary condition			
- all walls	$\mathbf{u}$	periodic	m/s
	p	periodic	Pa/m
Numerical parameters			
Time resolution	Δt	$\{0.01, 0.005, 0.0025, 0.001\,25, 0.000\,625\}$	s
Initial condition			
- Velocity	u	$1 - A\cos(x)\sin(y)$	m/s
	v	$1 + 2\sin(x)\cos(y)$	m/s
	w	0	m/s
- Pressure	p	$-(\cos(2x) + \cos(2y))$	Pa
Number of inner cells	$N_x - 2$	$\{8, 16, 32, 64\}$	
	$N_y - 2$	$\{8, 16, 32, 64\}$	
	$N_z - 2$	1	

Table B.7: McDermott (2003) test case: solution method and parameters

Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		
Diffusion	Jacobi	100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Pressure	Multigrid	2 cycles	4 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.7 Vortex Test Case

Table B.8: Jouhaud (2010) test case: physical and numerical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	$\{10, 20, 30\}$	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	0	m^2/s
Computational domain	(x_1, x_2)	$(-0.5, 0.5)$	m
	(y_1, y_2)	$(-0.5, 0.5)$	m
	(z_1, z_2)	$(-0.5, 0.5)$	m
Domain boundary condition			
- all walls	$\mathbf{u}$	periodic	m/s
	p	periodic	Pa/m
Numerical parameters			
Time resolution	Δt	$\{0.01, 0.005, 0.0025, 0.001\,25, 0.000\,625\}$	s
Initial condition			
- Velocity	u	$0.1 - \frac{\Gamma y}{R_c^2} \exp(-\frac{x^2+y^2}{2R_c^2})$	m/s
	v	$\frac{\Gamma x}{R_c^2} \exp(-\frac{x^2+y^2}{2R_c^2})$	m/s
	w	0	m/s
- Pressure	p	0	Pa
Number of inner cells	$N_x - 2$	$\{8, 16, 32, 64\}$	
	$N_y - 2$	$\{8, 16, 32, 64\}$	
	$N_z - 2$	1	

Table B.9: Jouhaud (2010) test case: solution method and parameters

Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		
Pressure	Multigrid	2 cycles	4 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.8 Lid-Driven Cavity Flow

Table B.10: Cavity flow test case: physical and numerical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	300	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	0.001	m^2/s
Computational domain	(x_1, x_2)	$(0, 1)$	m
	(y_1, y_2)	$(0, 1)$	m
	(z_1, z_2)	$(0, 1)$	m
Domain boundary condition			
- top	$\mathbf{u}$	$(1, 0, 0)^\top$	m/s
	p	0	Pa
- bottom, left, right, front, back	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
Numerical parameters			
Time resolution	Δt	0.001	s
Initial condition			
- Velocity	$\mathbf{u}$	$\mathbf{0}$	m/s
- Pressure	p	0	Pa
Number of inner cells	$N_x - 2$	256	
	$N_y - 2$	256	
	$N_z - 2$	1	

Table B.11: Cavity flow test case: solution method and parameters

Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		
Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Pressure	Multigrid	2 cycles	5 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.9 Flow Around a Cube

Table B.12: Flow around cube test case: physical and numerical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	10	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	1×10^{-5}	m^2/s
Computational domain	(x_1, x_2)	$(-3.0, 7.0)$	m
	(y_1, y_2)	$(\ 0.0, 2.0)$	m
	(z_1, z_2)	$(-3.5, 3.5)$	m
Domain boundary condition			
- left , right	$\mathbf{u}$	$(0.4, 0.0, 0.0)^\top$	m/s
	$\partial_n p$	0	Pa/m
- top, bottom	$\partial_n \mathbf{u}$	$\mathbf{0}$	1/s
	$\partial_n p$	0	Pa/m
- front, back	$\partial_n \mathbf{u}$	$\mathbf{0}$	1/s
	$\partial_n p$	0	Pa/m
Obstacle (cell centers)	(x_1, x_2)	$(\ 0.0273440, 0.964844)$	m
	(y_1, y_2)	$(\ 0.0078125, 0.992188)$	m
	(z_1, z_2)	$(-0.4785160, 0.478516)$	m
Obstacle boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
Numerical parameters			
Time resolution	Δt	0.01	s
Initial condition			
- Velocity	$\mathbf{u}$	$(0.4, 0.0, 0.0)^\top$	m/s
- Pressure	p	0	Pa
Number of inner cells	$N_x - 2$	256	
	$N_y - 2$	128	
	$N_z - 2$	256	

Table B.13: Flow around cube test case: solution method and parameters

Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		
Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Turbulence	Smagorinsky-Lilly	constant	$C_s = 0.2$
Pressure	Multigrid	2 cycles	5 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

B.10 Fire Induced Flow Experiment in a Compartment

Table B.14: Steckler et al. (1982)'s experiment N° 16: physical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	1800	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	3.10×10^{-5}	m^2/s
Gravitation	$\mathbf{g}$	$(0, -9.81, 0)^\top$	m/s^2
Thermal expansion coefficient	β	3.34×10^{-3}	1/K
Thermal diffusivity	α	4.10×10^{-5}	m^2/s
Effective heat release rate	$\dot{Q}$	50.3 (80% from 62.9)	kW
Specific heat capacity	c_p	1	kJ K/kg
Computational domain	(x_1, x_2)	$(-2.80, 4.20)$	m
	(y_1, y_2)	$(\ \ 0.00, 4.26)$	m
	(z_1, z_2)	$(-2.80, 2.80)$	m
Domain boundary condition			
- front, back, left, right, top, bottom	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
- front, back, left, right, top	T	26	°C
- bottom	$\partial_n T$	0	°C/m

Table B.15: Steckler et al. (1982)'s experiment N° 16: physical (cont'd) and numerical parameters

Physical parameters		Value	Unit
Obstacle (cell centers)			
- left wall	(x_1, x_2)	$(-1.640625000, -1.421875000)$	m
	(y_1, y_2)	$(\ 0.016640625,\ 2.113359375)$	m
	(z_1, z_2)	$(-1.378125000,\ 1.378125000)$	m
- top wall	(x_1, x_2)	$(-1.640625000,\ 1.640625000)$	m
	(y_1, y_2)	$(\ 2.146640625,\ 2.313046875)$	m
	(z_1, z_2)	$(-1.640625000,\ 1.640625000)$	m
- back wall	(x_1, x_2)	$(-1.640625000,\ 1.640625000)$	m
	(y_1, y_2)	$(\ 0.016640625,\ 2.113359375)$	m
	(z_1, z_2)	$(\ 1.421875000,\ 1.640625000)$	m
- front wall	(x_1, x_2)	$(-1.640625000,\ 1.640625000)$	m
	(y_1, y_2)	$(\ 0.016640625,\ 2.113359375)$	m
	(z_1, z_2)	$(-1.640625000, -1.421875000)$	m
- right front wall	(x_1, x_2)	$(\ 1.421875000,\ 1.640625000)$	m
	(y_1, y_2)	$(\ 0.016640625,\ 2.113359375)$	m
	(z_1, z_2)	$(-1.378125000, -0.459375000)$	m
- right middle wall	(x_1, x_2)	$(\ 1.421875000,\ 1.640625000)$	m
	(y_1, y_2)	$(\ 1.847109375,\ 2.113359375)$	m
	(z_1, z_2)	$(-0.415625000,\ 0.415625000)$	m
- right back wall	(x_1, x_2)	$(\ 1.421875000,\ 1.640625000)$	m
	(y_1, y_2)	$(\ 0.016640625,\ 2.113359375)$	m
	(z_1, z_2)	$(\ 0.459375000,\ 1.378125000)$	m
Obstacle boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
	$\partial_n T$	0	°C/m
Numerical parameters			
Time resolution	Δt	0.05	s
Initial condition			
- Velocity	$\mathbf{u}$	$\mathbf{0}$	m/s
- Pressure	p	0	Pa
	T	26	°C
Number of inner cells	$N_x - 2$	160	
	$N_y - 2$	128	
	$N_z - 2$	128	

Table B.16: Steckler et al. (1982)'s experiment N° 16: solution method and parameters

Fractional step	Method	Parameter	
Velocity & Pressure			
- Advection	Semi-Lagrangian		
- Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
- Turbulence	Smagorinsky-Lilly	constant	$C_s = 0.2$
- Source: buoyancy force	Explicit Euler		
- Pressure	Multigrid	2 cycles	5 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Temperature			
- Advection	Semi-Lagrangian		
- Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
- Turbulence	Smagorinsky-Lilly	Prandtl number	$Pr_T = 0.5$
- Source: volumetric heat	Explicit Euler	Gaussian	$x_0 = 0$
			$y_0 = 0.016640625$
			$z_0 = 0$
			$\sigma_x = 0.25$
			$\sigma_y = 0.6$
			$\sigma_z = 0.25$
		Ramp-up	$\tau = 5\,\text{s}$

B.11 Open Plume Experiment Using Particle Image Velocimetry

Table B.17: Meunders (2016)'s PIV experiment: physical and numerical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	300	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	$2.441\,39 \times 10^{-5}$	m^2/s
Gravitation	$\mathbf{g}$	$(0, -9.81, 0)^\top$	m/s^2
Thermal expansion coefficient	β	3.28×10^{-3}	1/K
Thermal diffusivity	α	3.31×10^{-5}	m^2/s
Computational domain	(x_1, x_2)	$(-0.3675, 0.3675)$	m
	(y_1, y_2)	$(-0.0600, 0.5900)$	m
	(z_1, z_2)	$(-0.2875, 0.2875)$	m
Domain boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
	T	31.5	°C
Obstacle (cell centers)	(x_1, x_2)	$(-0.025840, \quad 0.025840)$	m
	(y_1, y_2)	$(-0.057461, -0.001600)$	m
	(z_1, z_2)	$(-0.029199, \quad 0.029199)$	m
Obstacle boundary condition			
- all walls	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
- all walls except bottom	T	150.03 (55% from 270.7)	°C
Numerical parameters			
Time resolution	Δt	0.01	s
Initial condition			
- Velocity	$\mathbf{u}$	$\mathbf{0}$	m/s
- Pressure	p	0	Pa
- Temperature			
- lower layer	T	30.5	°C
	T	30.9	°C
	T	32.1	°C
	T	35.7	°C
- upper layer	T	37.4	°C
Number of inner cells	$N_x - 2$	$\{128, 256\}$	
	$N_y - 2$	$\{128, 256\}$	
	$N_z - 2$	$\{128, 256\}$	

Table B.18: Meunders (2016)'s PIV experiment: solution method and parameters

Fractional step	Method	Parameter	
Velocity & Pressure			
- Advection	Semi-Lagrangian		
- Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
- Turbulence	Smagorinsky-Lilly	constant	$C_s = 0.2$
- Source: buoyancy force	Explicit Euler		
- Pressure	Multigrid	2 cycles	$\{6, 7\}$ levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Temperature			
- Advection	Semi-Lagrangian		
- Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
- Turbulence	Smagorinsky-Lilly	Prandtl number	$Pr_T = 0.9$

B.12 Tunnel Simulation for Real-time and Prognosis Analysis

Table B.19: Tunnel test case: physical and numerical parameters

Physical parameters		Value	Unit
Simulation time	t_{end}	500	s
Density	ρ_0	1	kg/m^3
Kinematic viscosity	ν	$2.441\,39 \times 10^{-5}$	m^2/s
Gravitation	$\mathbf{g}$	$(0, -9.81, 0)^\top$	m/s^2
Thermal expansion coefficient	β	3.28×10^{-3}	1/K
Thermal diffusivity	α	3.31×10^{-5}	m^2/s
Computational domain	(x_1, x_2)	$(-50, 100)$	m
	(y_1, y_2)	$(0, 5)$	m
	(z_1, z_2)	$(-4, 4)$	m
Domain boundary condition			
- front, back, top, bottom	$\mathbf{u}$	$\mathbf{0}$	m/s
	$\partial_n p$	0	Pa/m
- left, right	$\partial_n \mathbf{u}$	$\mathbf{0}$	1/s
	p	0	Pa
- all walls except bottom	T	31.5	°C
- bottom	$\partial_n T$	0	°C/m
Numerical parameters			
Time resolution	Δt	$\{0.1, 0.05, 0.025, 0.0125\}$	s
Initial condition			
- Velocity	$\mathbf{u}$	$\mathbf{0}$	m/s
- Pressure	p	0	Pa
- Temperature			
- lower layer	T	30.5	°C
	T	30.9	°C
	T	32.1	°C
	T	35.7	°C
- upper layer	T	37.4	°C
Number of inner cells	$N_x - 2$	$\{256, 512, 1024\}$	
	$N_y - 2$	$\{8, 16, 32\}$	
	$N_z - 2$	$\{16, 32, 64\}$	

Table B.20: Tunnel test case: solution method and parameters

Fractional step	Method	Parameter	
Velocity & Pressure			
- Advection	Semi-Lagrangian		
- Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
- Turbulence	Smagorinsky-Lilly	constant	$C_s = 0.2$
- Source: buoyancy force	Explicit Euler		
- Pressure	Multigrid	2 cycles	$\{7, 8, 9\}$ levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Temperature			
- Advection	Semi-Lagrangian		
- Diffusion	Jacobi	iterations until	$\delta_{\text{tol}} = 1 \times 10^{-7}$
- Turbulence	Smagorinsky-Lilly	Prandtl number	$Pr_T = 0.5$

Nomenclature

Latin Symbols

A	Area	m^2
C	Species concentration	$\mathrm{kg/m}^3$
c	Constant value	[–]
c_p	Specific heat capacity at constant pressure	$\mathrm{J/(kg\,K)}$
D	Mass diffusivity	m^2/s
e	Specific internal energy	J/kg
$\mathbf{f}$	Force density	$\mathrm{N/m}^3$
$\mathbf{g}$	Gravitational acceleration	$\mathrm{m/s}^2$
G	Convolution kernel (filter function)	[–]
h	Elevation	m
h	Grid resolution, h finest and Lh coarsest	m
h	Heat transfer coefficient	$\mathrm{W/(m}^2\,\mathrm{K)}$
h	Specific enthalpy, $h \coloneqq e + p/\rho$	J/kg
H_c	Heat of combustion	J/kg
(i, j, k)	3-tuple of cell indices	[–]
ix	Global cell index $ix \coloneqq i + N_x \cdot j + N_x \cdot N_y \cdot k$	[–]
k	Thermal conductivity	$\mathrm{W/(m\,K)}$
l	Iteration step	[–]
l	Number of levels	[–]
L	Width of domain; large scale; characteristic length	m
L_x	Width, $L_x \coloneqq x_2 - x_1$, of physical domain in x-direction	m
L_y	Height, $L_y \coloneqq y_2 - y_1$, of physical domain in y-direction	m
L_z	Depth, $L_z \coloneqq z_2 - z_1$, of physical domain in z-direction	m
M	Molar mass	kg/mol
N	Number of all inner cells, $N \coloneqq (N_x - 2)(N_y - 2)(N_z - 2)$	[–]
n	Time step of current discrete time $t^{(n)}$, $n \coloneqq t^{(n)}/\Delta t$	[–]
$N(i)$	Set of neighbor indices	[–]

N_t	Number of time steps, $N_t \coloneqq t_{\text{end}}/\Delta t$	[−]
N_x	Number of cells in x-direction (including ghost cells)	[−]
N_y	Number of cells in y-direction (including ghost cells)	[−]
N_z	Number of cells in z-direction (including ghost cells)	[−]
p	Order of a numerical method	[−]
$\mathbf{p}$	Path of a fluid element	m
p	Pressure	Pa
P	Dynamic pressure	Pa
P	Power	W
$\dot{Q}$	Heat release rate	J/s
r	Refinement ratio	[−]
$\mathbf{r}$	Residual	[−]
R	Universal gas constant, $R \approx 8.134\,\text{J}/(\text{mol}\,\text{K})$	J/(mol K)
R	Real-time ratio	[−]
S	Speedup, $S \coloneqq t_{\text{ref}}/t_{\text{par}}$	[−]
S_C	Specific mass source for species	kg/(m^3 s)
S_T	Energy source	K/s
t	Time	s
t_{end}	Total simulation time, $t_{\text{end}} \coloneqq N_t \Delta t$	s
$t^{(n)}$	Discrete time, $t^{(n)} \coloneqq n\Delta t$	s
T	Temperature field $T(\mathbf{x}, t)$ inside a fluid	°C
$\mathbf{u}$	Velocity $\mathbf{u}(\mathbf{x}, t)$ of an element of fluid moving through $\mathbf{x}$ at time t	m/s
u	Velocity in x-direction	m/s
U	Characteristic velocity	m/s
v	Velocity in y-direction	m/s
V	Volume	m^3
V	Vortex	[−]
w	Velocity in z-direction	m/s
$\mathbf{x}$	Point in space $(x, y, z)^\top \in \mathbb{R}^3$	m
$\mathbf{x}_d$	Departure point $\mathbf{x}_d \coloneqq \mathbf{p}(\mathbf{x}, -\Delta t) \coloneqq \mathbf{x} - \Delta t \mathbf{u}$ along path $\mathbf{p}$	m
(x_1, x_2)	Physical domain borders in x-direction	m
x_i	Cell faces, $x_i \coloneqq x_1 + (i-1)\Delta x$, of numerical grid in x-direction	m
$\tilde{x}_i$	Cell centers, $\tilde{x}_i \coloneqq x_1 + (i-0.5)\Delta x$, of numerical grid in x-direction	m
(y_1, y_2)	Physical domain borders in y-direction	m
y_j	Cell faces, $y_j \coloneqq y_1 + (j-1)\Delta y$, of numerical grid in y-direction	m
$\tilde{y}_j$	Cell centers, $\tilde{y}_j \coloneqq y_1 + (j-0.5)\Delta y$, of numerical grid in y-direction	m

Y_s	Soot yield	g/g
(z_1, z_2)	Physical domain borders in z-direction	m
z_k	Cell faces, $z_k \coloneqq z_1 + (k-1)\Delta z$, of numerical grid in z-direction	m
$\tilde{z}_k$	Cell centers, $\tilde{z}_k \coloneqq z_1 + (k-0.5)\Delta z$, of numerical grid in z-direction	m

Greek Symbols

α	Thermal diffusivity	m^2/s
β	Volumetric thermal expansion coefficient	$1/\mathrm{K}$
$\mathbf{\Phi}$	Viscous dissipation	W/m^3
ϵ	Emissivity	[−]
ϵ	Truncation error	[−]
$\epsilon^{(l)}$	Iteration error at iteration step l	[−]
η	Kolmogorov scale	m
μ	Dynamic (shear) viscosity	$\mathrm{Pa} \cdot \mathrm{s}$
ν	Kinematic viscosity $\nu \coloneqq \mu/\rho$	m^2/s
ω	Weight	[−]
ϕ	Arbitrary quantity	[−]
ρ	Mass density $\rho(\mathbf{x}, t)$ for each time t and space $\mathbf{x}$	kg/m^3
σ	Full width at half maximum of a Gaussian curve	m
σ	Normal stress	Pa
σ	Stefan-Boltzmann constant, $\sigma \approx 5.670 \times 10^{-8}\ \mathrm{W}/(\mathrm{m}^2\,\mathrm{K}^4)$	$\mathrm{W}/(\mathrm{m}^2\,\mathrm{K}^4)$
δ_{tol}	Tolerance value	[−]
τ	Ramp-up time	s
τ	Shear stress	Pa
τ^{SGS}	Residual stress tensor in subgrid-scale	Pa
χ	Radiative fraction	[−]

Mathematical Symbols

$\mathbf{A}$	Matrix notation (upper case, bold)	[−]
$\mathbf{a}$	Vector notation (lower case, bold)	[−]
$\mathbf{a} \cdot \mathbf{b}$	Scalar or inner product of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, $\mathbf{a} \cdot \mathbf{b} \coloneqq \sum_{i=1}^{n} a_i b_i$	[−]
a	Scalar notation (lower case)	[−]
$\mathbb{C}$	Convection operator	[−]
Δ	Change of any varying quantity	[−]
Δ_f	Filter width	[−]
$\mathbb{D}$	Diffusion operator	[−]
∂_t	Time derivative	[−]

∂_{x_i}	Partial derivative in space, where $x_i \in \{x, y, z\}$	$[-]$
$\frac{d}{dt}$	Material derivative $\frac{d}{dt} \coloneqq \partial_t + \mathbf{u} \cdot \nabla$	$[-]$
$d(\cdot)$	Infinitesimal (infinitely small) change in some varying quantity	$[-]$
Δt	Time step $\Delta t \coloneqq t_{\text{end}}/N_t$	s
Δx	Grid spacing, $\Delta x \coloneqq L_x/(N_x - 2)$, in x-direction	m
Δy	Grid spacing, $\Delta y \coloneqq L_y/(N_y - 2)$, in y-direction	m
Δz	Grid spacing, $\Delta z \coloneqq L_z/(N_z - 2)$, in z-direction	m
$\mathbb{F}$	Force operator	$[-]$
$\mathbf{I}$	Identity matrix	$[-]$
$\mathbb{L}$	Differential operator for respective conservation law	$[-]$
∇^2	Laplace operator $\nabla^2 \coloneqq \nabla \cdot \nabla \coloneqq \sum_{i=1}^{n} \partial_{x_i}^2$	$[-]$
∇	Nabla operator $\nabla \coloneqq (\partial_x, \partial_y, \partial_z)^\top$	$[-]$
$\mathbb{N}^0$	Natural numbers including zero	$[-]$
$\|\cdot\|$	Norm	$[-]$
$\mathcal{O}$	Big O notation	$[-]$
$\mathbb{P}$	Pressure operator	$[-]$
$\mathbf{P}_{2h}^{h}$	Prolongation operator from coarse grid $2h$ to fine grid h	$[-]$
$\mathbb{R}$	Real numbers	$[-]$
$\mathbf{R}_{h}^{2h}$	Restriction operator from fine grid h to coarse grid $2h$	$[-]$
$[\cdot]$	Rounding operator	$[-]$
$[[\cdot][\cdot]]$	Tensor operator	$[-]$
$\text{Tr}()$	Trace of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, $Tr(\mathbf{A}) \coloneqq \sum_{i=1}^{n} a_{ii}$	$[-]$

Subscripts

$(\cdot)_0$	Reference, ambient quantity or quantity at time $t = 0$ s
$(\cdot)_B$	Body or buoyancy related quantity
$(\cdot)_{\text{b}}$	Black iterates in red-black ordering
$(\cdot)_{\mathbf{c}}$	Quantity at domain center
$(\cdot)_{\text{coarse}}$	Quantity related to a coarse resolution
$(\cdot)_{\text{conv}}$	Convection related quantity
$(\cdot)_{\text{eff}}$	Effective quantity
$(\cdot)_{\text{fine}}$	Quantity related to a fine resolution
$(\cdot)_g$	Gas related quantity
$(\cdot)_h$	Quantity at grid with resolution h
$(\cdot)_{\text{ins}}$	Insulation related
$(\cdot)_{ijk}$	Quantity at discrete location (i, j, k) in numerical grid
$(\cdot)_l$	Quantity at lower level

$(\cdot)_{\text{par}}$	Parallel version
$(\cdot)_{\text{rad}}$	Quantity related to radiation
$(\cdot)_{\text{ref}}$	Reference
$(\cdot)_{\text{Rich}}$	Richardson's Extrapolation
$(\cdot)_{\text{r}}$	Red iterates in red-black ordering
$(\cdot)_{\text{surf}}$	Surface related
$(\cdot)_{\text{sim}}$	Simulation related
$(\cdot)_T$	Turbulent quantity; quantity related to temperature
$(\cdot)_t$	Quantity of thermocouple
$(\cdot)_u$	Quantity at upper level
$(\cdot)_w$	Quantity at wall
$(\cdot)_{\text{wc}}$	Wallclock time related
$(\cdot)_{\text{wc, \#cores}}$	Wallclock time related with applied number of cores
$(\cdot)_x$	Quantity or operator in x-direction
$(\cdot)_y$	Quantity or operator in y-direction
$(\cdot)_z$	Quantity or operator in z-direction

Superscripts

$(\cdot)^{\text{ana}}$	Analytical solution
$\tilde{(\cdot)}$	Quantity at cell center
$(\cdot)'$	Eddy (turbulent) motion of an instantaneous property
$(\cdot)^{(l)}$	Quantity at iteration step l
$\overline{(\cdot)}$	Mean motion of an instantaneous property
$(\cdot)^{(n)}$	Quantity at discrete time $t^{(n)}$
$(\cdot)^{\text{ramp}}$	Ramp-up function
$\dot{(\cdot)}$	Rate
$(\cdot)^{\text{total}}$	Total quantity
$(\cdot)^{\text{vol}}$	Volumetric function

Abbreviations

$1D$	One-Dimensional
$2D$	Two-Dimensional
$3D$	Three-Dimensional
A.I.	Arithmetic Intensity
ADI	Alternating Direction Implicit
ALU	Arithmetic Logic Unit
APOD	Assess, Parallelize, Optimize, Deploy

BD	Backward Difference
BDF	Backward Differentiation Formulas
BDW	Intel Xeon Broadwell E5-2650 v4, 2.5 GHz, 2×12 cores or
	Intel Xeon Broadwell E5-2623 v4, 2.6 GHz, 2×8 cores
BE	Backward Euler method
BiCGStab	BiConjugate Gradient method with Stabilization
BIM	Building Information Modeling
BMBF	German Federal Ministry of Education and Research
BW	Bandwidth
CA	Cellular Automata method
CAD	Computer-Aided Design
CD	Central Difference
CFD	Computational Fluid Dynamics
CG	Conjugate Gradient method
CGS	Colored Gauss-Seidel method
CPU	Central Processing Unit
CSV	Comma Separated Values
CUPS	Cell Updates Per Second
DAAD	Deutscher Akademischer Austauschdienst
DNS	Direct Numerical Simulation
ECC	Error Correcting Code
FD	Forward Difference
FDM	Finite Difference Method
FDS	Fire Dynamics Simulator
FE	Forward Euler method
FEM	Finite Element Method
Flop	Floating-point operation
FVM	Finite Volume Method
Gr	Grashof number
GMRES	Generalized Minimal Residual method
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HPC	High Performance Computing
HRR	Heat Release Rate
HS	Heat source
HSW	Intel Xeon Haswell E5-2680 v3, 2.2 GHz, 2×12 cores

IVB	Intel Xeon Ivy Bridge E5-2640 v2, 2.0 GHz, 2×8 cores
JSC	Jülich Supercomputing Center
JURECA	Jülich Research on Exascale Cluster Architectures
JuROr	Jülich's Real-time simulation within Orpheus
K40	NVIDIA Kepler K40 GPU with 12 GB
K80	NVIDIA Kepler K80 with 2 GPUs and 2×12 GB
LBM	Lattice Boltzmann Method
LES	Large Eddy Simulation
Ma	Mach number
MAC	Marker And Cell method
MG	Multigrid method
MPI	Message Passing Interface
Nu	Nusselt number
OpenACC	Open Accelerators
OpenMP	Open Multi-Processing
ORPHEUS	Optimierung der Rauchableitung und Personenführung in U-Bahnhöfen: Experimente und Simulationen
Pr	Prandtl number
P100	NVIDIA Pascal P100 SMX2 GPU with 16 GB or PCIe with 12 GB
PISO	Pressure-Implicit with Splitting of Operators
PIV	Particle Image Velocimetry
Ra	Rayleigh number
Re	Reynolds number
RK	Runge-Kutta method
Sc	Schmidt number
SGS	Subgrid-scale
SIMPLE	Semi-Implicit Methods for Pressure-Linked Equations
SIMPLEC	Semi-Implicit Methods for Pressure-Linked Equations – Consistent
SIMPLER	Semi-Implicit Methods for Pressure-Linked Equations – Revised
SL	Semi-Lagrangian method
SM	Streaming Multiprocessor
SNB	Intel Xeon Sandy Bridge E5-2650 0, 2.0 GHz, 2×8 cores
SOR	Successive Over-Relaxation
vtk	Visualization Toolkit
w.r.t	With respect to
XML	Extensible Markup Language

Bibliography

AIAA (1998). *Guide: Guide for the Verification and Validation of Computational Fluid Dynamics Simulations (AIAA G-077-1998(2002))*. AIAA Standards. American Institute of Aeronautics and Astronautics, Inc.

Aish, R. (1986). *Building Modelling: The Key to Integrated Construction CAD*. In CIB 5th International Symposium on the Use of Computers for Environmental Engineering related to Building, Guildhall, Bath.

ANSYS (2013). *ANSYS CFX Reference Guide*. Technical report, American Society for Testing and Materials (ASTM). URL: `http://www.ansys.com/products/fluids/ansys-cfx`.

ANSYS (2015). *ANSYS Fluent Theory Guide (Release 16.2)*. Technical report, ANSYS.

Anzt, H., Baboulin, M., Dongarra, J., Fournier, Y., Hulsemann, F., Khabou, A., and Wang, Y. (2017). *Accelerating the Conjugate Gradient Algorithm with GPUs in CFD Simulations*. In Dutra, I., Camacho, R., Barbosa, J., and Marques, O., editors, High Performance Computing for Computational Science – VECPAR 2016, pages 35–43, Cham. Springer International Publishing.

ASTM (2005). *Standard Guide for Evaluating Predictive Capability of Deterministic Fire Models*. American Society for Testing and Materials (ASTM), West Conshohoken, PA.

Autodesk Inc. (2017). *SimCFD - User's Guide*. Technical report. URL: `https://knowledge.autodesk.com/support/cfd/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/SimCFD-UsersGuide/files/GUID-3FCD44C0-8AAC-4826-B199-9D1C299F7165-htm.html?v=2018`.

AVL LIST GmbH (2014). *AVL FIRE VERSION 2014*. Technical report. URL: `http://web.itu.edu.tr/~sorusbay/SI/AVL2.pdf`.

Bakhvalov, N. (1966). *On the convergence of a relaxation method with natural constraints on the elliptic operator*. USSR Computational Mathematics and Mathematical Physics, 6(5):101–135.

Beata, P. A., Jeffers, A. E., and Kamat, V. R. (2018). *Real-Time Fire Monitoring and Visualization for the Post-Ignition Fire State in a Building*. Fire Technology.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, Englewood Cliffs, NJ.

Boltzmann, L. (1884). *Ableitung des Stefan'schen Gesetzes, betreffend die Abhängigkeit der Wärmestrahlung von der Temperatur aus der electromagnetischen Lichttheorie*. Annalen der Physik und Chemie, 22:291–294.

Bonaventura, L. (2004). *An introduction to semi-Lagrangian methods for geophysical scale flows*.

Boussinesq, M. J. (1877). *Essai sur la théorie des eaux courantes*. Mém. Présentés par divers savants à l'Académie des Sciences Inst. France, 17:1–680.

Brandt, A. (1973). *Multi-Level Adaptive Technique (MLAT) for Fast Numerical Solution to Boundary-Value Problems*. In Cabannes, H. and Temam, R., editors, Third International Conference on Numerical Methods in Fluid Mechanics, Lecture Notes in Physics, pages 82–89, Paris 1972. Springer-Verlag.

Brandt, A. (1977). *Multi-Level Adaptive Solutions to Boundary-Value Problems*. Mathematics of Computation, 31:333–390.

Bredberg, J. (2000). *On the Wall Boundary Condition for Turbulence Models*. Technical report, Department of Thermo and Fluid Dynamics.

Breuer, M., Lakehal, D., and Rodi, W. (1996). *Flow around a Surface Mounted Cubical Obstacle: Comparison of LES and RANS-Results*. In Deville, M., Gavrilakis, S., and Ryhming, I. L., editors, IMACS-COST Conference on Computational Fluid Dynamics, Computation of Three-Dimensional Complex Flows, pages 22–30, Lausanne. Vieweg+Teubner Verlag.

Brieda, L. (2015). *Finite Element Particle in Cell (FEM-PIC)*. Technical report, Particle In Cell Consulting LLC. URL: `https://www.particleincell.com/2015/fem-pic/`.

Bruneau, C. H. and Saad, M. (2006). *The 2D lid-driven cavity problem revisited.* Computers & Fluids, 35:326–348.

Buffat, M., Cadiou, A., Le Penven, L., and Pera, C. (2015). *In situ analysis and visualization of massively parallel computations.* The International Journal of High Performance Computing Applications, 31(1):83–90.

Calore, E., Gabbana, A., Kraus, J., Schifano, S. F., and Tripiccione, R. (2016). *Performance and portability of accelerated lattice Boltzmann applications with OpenACC.* Concurr. Comput. Pract. Exper., 28(12):3485–3502.

Chandrasekaran, S. and Juckeland, G. (2018). *OpenACC for programmers: concepts and strategies.* Pearson Education Inc., Addison-Wesley, Boston, 1st edition.

Charney, J. G., Fjørtoft, R., and von Neumann, J. (1950). *Numerical Integration of the Barotropic Vorticity Equation.* Tellus, 2:237–254.

Choi, H. and Moin, P. (1994). *Effects of the Computational Time Step on Numerical Solutions of Turbulent Flow.* Journal of Computational Physics, 113(1):1–4.

Chorin, A. J. (1967). *The numerical solution of the Navier-Stokes equations for an incompressible fluid.* Bulletin of the American Mathematical Society, 73(6):928–931.

Chorin, A. J. (1968). *Numerical solution of Navier–Stokes equations.* Mathematics of Computation, 22(104):745–762.

Chorin, A. J. and Marsden, J. E. (1979). *A mathematical introduction to fluid mechanics.* Springer-Verlag.

Cohen, J. M. and Molemaker, J. M. (2009). *A fast double precision CFD code using CUDA.* In Parallel Computational Fluid Dynamics: Recent Advances and Future Directions.

Courant, R., Friedrichs, K., and Lewy, H. (1928). *Über die partiellen Differenzengleichungen der mathematischen Physik.* Mathematische Annalen, 100:32–74.

Cowlard, A., Jahn, W., Abecassis-Empis, C., Rein, G., and Torero-Cullen, J. (2010). *Sensor Assisted Fire Fighting.* Fire Technology, 46(3):719–741.

Crane, K., Llamas, I., and Tariq, S. (2007). *GPU Gems 3: Chapter 30 - Real-Time Simulation and Rendering of 3D Fluids.* Addison-Wesley Professional.

Crank, J. and Nicolson, P. (1947). *A Practical Method for Numerical Evaluation of Solutions of Partial Differential Equations of Heat Conduction Type*. Mathematical Proceedings of the Cambridge Philosophical Society, 40:50–67.

Danalis, A., Marin, G., McCurdy, C., Meredith, J., Roth, P., Spafford, K., Tipparaju, V., and Vetter, J. (2010). *The scalable heterogeneous computing (SHOC) benchmark suite*. In Proceedings of the Third Workshop on General-Purpose Computation on Graphics Processors (GPGPU 2010), pages 63–74.

Daniel, N. and Rein, G. (2016). *The Fire Navigator: forecasting the spread of building fires on the basis of sensor data*. Technical report. URL: `http://www.sfpe.org/page/FPEExtraIssue3`.

Deakin, T. and McIntosh-Smith, S. (2017). *GPU-STREAM v1.0/ v3.1*. Technical report. URL: `https://github.com/UoB-HPC/BabelStream`.

Deakin, T., Price, J., Martineau, M., and McIntosh-Smith, S. (2016). *GPU-STREAM v2.0: benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming model*. In Taufer, M., Mohr, B., and Kunkel, J. M., editors, ISC High Performance 2016. LNCS, volume 9945, pages 489–507. Springer.

Deardorff, J. W. (1972). *Numerical Investigation of Neutral and Unstable Planetary Boundary Layers*. Journal of Atmospheric Sciences, 29:91–115.

Driver, D. M., Seegmiller, H. L., and Marvin, J. G. (1987). *Time-dependent behavior of a reattaching shear layer*. AIAA Journal, 25(7):914–919.

Drysdale, D. (1999). *An Introduction to Fire Dynamics*. John Wiley & Sons Ltd, 2nd edition.

Drzycimski, K. and Arnold, L. (2015). *Smoke and Fire Simulations with Adaptive FEM*. In Parallel Computational Fluid Dynamics (ParCFD), Montreal, Canada.

EDF Group (2017). *Code Saturne version 3.3.0 practical user's guide*. Technical report, EDF Group. URL: `https://www.code-saturne.org/cms/`.

Evensen, G. (2009). *Data Assimilation - The Ensemble Kalman Filter*. Springer, 2nd edition.

Ewer, J., Galea, E., Patel, M. K., Jia, F., Grandison, A., and Wang, Z. (2010). *SMARTFIRE – The Fire Field Modelling Environment*. In Sequeira, J. C. F. P. and A., editors, V European Conference on Computational Fluid Dynamics (ECCOMAS CFD 2010), Lisbon, Portugal. Fire Safety Engineering Group (FSEG) of the University of Greenwich.

Ewer, J., Jia, F., Grandison, A., Galea, E., and Patel, M. K. (2008). *SMARTFIRE v4.1 Technical Reference Manual*. Technical report, Fire Safety Engineering Group (FSEG) of the University of Greenwich.

Ewer, J., Jia, F., Grandison, A., Galea, E., and Patel, M. K. (2013). *SMARTFIRE v4.3 User Guide and Technical Manual*. Technical report, Fire Safety Engineering Group (FSEG) of the University of Greenwich.

Fedorenko, R. (1962). *A relaxation method for solving elliptic difference equations*. USSR Computational Mathematics and Mathematical Physics, 1(4):1092–1096.

Fedorenko, R. (1964). *The rate of convergence of an iterative process*. USSR Computational Mathematics and Mathematical Physics, 4(3):227–235.

Fehling, M., Boltersdorf, J., and Arnold, L. (2017). *Towards smoke and fire simulation with grid adaptive FEM: Verification of the flow solver (Poster)*. In 12th International Symposium on Fire Safety Science (IAFSS), Lund University, Lund Sweden. IAFSS.

Ferziger, J. H. and Peric, M. (2002). *Computational Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg, 3 edition.

Gant, S. and Hoyes, J. (2010). *Review of FLACS version 9.0: Dispersion modelling capabilities*. Technical report, Health and Safety Laboratory for the Health and Safety Executive (HSE) 2010.

Garg, S., Forbes-Smith, N., Hilton, J., and Prakash, M. (2018). *SparkCloud: A Cloud-Based Elastic Bushfire Simulation Service*. Remote Sensing, 10(74).

Germano, M., Piomelli, U., Moin, P., and Cabot, W. H. (1991). *A dynamic subgrid-scale eddy viscosity model*. Physics of Fluids A, 3(7):1760–1765.

Gexcon, F. (2015). *FLACS High Performance Computing Service*. Technical report. URL: `http://www.gexcon.com/search_results/free-trial-of-high-performance-computing-services`.

Ghia, U., Ghia, K. N., and Shin, C. T. (1982). *High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method*. Journal of Computational Physics, 48(3):387–411.

Glimberg, S. L. (2009). *Smoke Simulation for Fire Engineering using CUDA*. PhD thesis, University of Copenhagen.

Glimberg, S. L., Erleben, K., and Bennetsen, J. C. (2009). *Smoke Simulation for Fire Engineering using a Multigrid Method on Graphics Hardware*. In Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS).

Hackbusch, W. (1977). *A Multi-grid Method Applied to a Boundary Value Problem with Variable Coefficients in a Rectangle*. Report 77-17. Institut für Angewandte Mathematik, Universität Köln.

Hamano, J. C., Pearce, S. O., and Torvalds, L. (2005). *git – everything-is-local*. Technical report. URL: `https://git-scm.com`.

Hamins, A., Bryner, N., Jones, A., Koepke, G., Grant, C., and Raghunathan, A. (2014). *Smart Firefighting Workshop Summary Report*. Technical report, U. S. Department of Commerce anf National Institute of Standards and Technology (NIST) Special Publication.

Hamins, A., Grant, C., Bryner, N., Jones, A., and Koepke, G. (2015). *Research Roadmap for Smart Fire Fighting*. Technical report, U. S. Department of Commerce anf National Institute of Standards and Technology (NIST) Special Publication.

Hamins, A., Kashiwagi, T., and Buch, R. (1996). *Characteristics of pool fire burning*.

Han, L., Potter, S., Beckett, G., Pringle, G., Welch, S., Koo, S. H., Wickler, G., Usmani, A., Torero, J. L., and Tate, A. (2010). *FireGrid: An e-infrastructure for next-generation emergency response support*. J. Parallel Distrib. Comput., 70:1128–1141.

Harlow, F. H. and Welch, J. E. (1965). *Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface*. The Physics of Fluids, 8(12):2182–2189.

Harris, M. (2016). *Inside Pascal: NVIDIA's Newest Computing Platform*. Technical report. URL: `https://devblogs.nvidia.com/parallelforall/inside-pascal/`.

Harris, M. J. (2004). *GPU GEMS*. Chapter 38, Fast Fluid Dynamics Simulation on the GPU. University of North Carolina, Chapel Hill.

Hejn, K. and Rosenkvist, J. P. (2009). *Distributed Fluid Simulation on Multiple GPUs*. PhD thesis, University of Copenhagen.

Herdman, J. A., Gaudin, W. P., Perks, O., Beckingsale, D. A., Mallinson, A. C., and Jarvis, S. A. (2014). *Achieving Portability and Performance through OpenACC*.

Hestenes, M. R. and Stiefel, E. (1952). *Methods of Conjugate Gradients for Solving Linear Systems*. Journal of Research of the National Bureau of Standards, 49(6).

Huang, Z., Kavan, L., Li, W., Hui, P., and Gong, G. (2015). *Reducing numerical dissipation in smoke simulation*. Graphical Models, 78:10–25.

Hunt, A. and Thomas, D. (1999). *The Pragmatic Programmer*. Pearson Education.

Hurley, M. J., Gottuk, D. T., Hall Jr., J. R., Harada, K., Kuligowski, E. D., Puchovsky, M., Torero, J. L., Watts Jr., J. M., and Wieczorek, C. J. (2016). *SPFE Handbook of Fire Protection Engineering*. Springer-Verlag New York, 5th edition.

Husted, B., Li, Y. Z., Huang, C., Anderson, J., Svensson, R., Ingason, H., Runefors, M., and Wahlqvist, J. (2017). *Verification, validation and evaluation of FireFOAM as a tool for performance based design*. Technical report. URL: `https://www.brandskyddsforeningen.se/globalassets/brandforsk/rapporter-2017/brandforsk_rapport_309_131_firefoam_new.pdf`.

Issa, R. I. (1986). *Solution of the implicitly discretised fluid flow equations by operator-splitting*. Journal of Computational Physics, 62(1):40–65.

Jacobi, C. G. J. (1845). *Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadraten vorkommenden lineare Gleichungen*. Astronomische Nachrichten, 22(20):297–306.

Jahn, W. (2010). *Inverse Modeling to Forecast Enclosure Fire Dynamics*. PhD thesis, University of Edinburg, UK.

Jahn, W. (2017). *Using suppression and detection devices to steer CFD fire forecast simulations*. Fire Safety Journal, 91:284–290.

Jahn, W., Rein, G., and Torero, J. L. (2009). *Data Assimilation in Enclosure Fire Dynamics - Towards Adjoint Modelling*.

Jahn, W., Rein, G., and Torero, J. L. (2012). *Forecasting fire dynamics using inverse computational fluid dynamics and tangent linearisation*. Advances in Engineering Software, 47(1):114–126.

Jeffreys, H. and Jeffreys, B. S. (1956). *Methods of Mathematical Physics*. Cambridge University Press, Cambridge, England, 3rd edition.

Jin, M., Zuo, W., and Chen, Q. (2012). *Improvements of fast fluid dynamics for simulating airflow in buildings*. Numerical Heat Transfer, Part B: Fundamentals, 62(6):419–438.

Jouhaud, J. C. (2010). *Benchmark on the vortex preservation*. Technical report. URL: `http://elearning.cerfacs.fr/pdfs/numerical/TestCaseVortex2D.pdf`.

Jülich Supercomputing Centre (2018). *JURECA: Modular supercomputer at Jülich Supercomputing Centre*. Journal of large-scale research facilities, 4(A132).

Kemloh, A. U. W., Steffen, B., Seyfried, A., and Chraibi, M. (2013). *Parallel real time computation of large scale pedestrian evacuations*. Advances in Engineering Software, 60-61:98–103.

Kempe, T. and Hantsch, A. (2017). *Large-eddy simulation of indoor air flow using an efficient finite-volume method*. Building and Environment, 115:291–305.

Kim, J., Kline, S. J., and Johnston, J. P. (1980). *Investigation of a Reattaching Turbulent Shear Layer: Flow Over a Backward-Facing Step*. Journal of Fluids Engineering, 102(3):302–308.

Kim, J. and Moin, P. (1985). *Application of a fractional-step method to incompressible Navier-Stokes equations*. Journal of Computational Physics, 59(2):308–323.

Klote, J. H., Milke, J. A., Turnbull, P. G., Kashef, A., and Ferreira, M. J. (2012). *Handbook of smoke control engineering*. American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, GA, USA.

Kolawa, A. and Huizinga, D. (2007). *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press.

Koo, S. H. (2010). *Forecasting fire development with sensor-linked simulation*. PhD thesis, University of Edinburgh.

Koseki, H. (1989). *Combustion properties of large liquid pool fires.* Fire Technology Journal, 25(3):241–255.

Kraus, J. and Schlottke, M. (2014). *Accelerating a C++ CFD Code with OpenACC.* Technical report. URL: `https://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/nvidia-ws-2014/05-kraus-zfs.pdf?__blob=publicationFile`.

Krüger, J. H. (2006). *A GPU Framework for Interactive Simulation and Rendering of Fluid Effects.* PhD thesis, Technische Universität München (TUM).

Küsters, A., Wienke, S., and Arnold, L. (2017). *Performance Portability Analysis for Real-Time Simulations of Smoke Propagation Using OpenACC.* In Kunkel, J. M., Yokota, R., Taufer, M., and Shalf, J., editors, High Performance Computing: ISC High Performance 2017 International Workshops, pages 477–495, Cham. Springer International Publishing.

Lahoz, W., Khattatov, B., and Ménard, R. (2010). *Data Assimilation - Making Sense of Observations.* Springer.

Lasher, W. C. and Taulbee, D. B. (1992). *On the computation of turbulent backstep flow.* International Journal of Heat and Fluid Flow, 13(1):30–40.

Laurien, E. and Oertel, H. (2013). *Numerische Strömungsmechanik: Grundgleichungen und Modelle - Lösungsmethoden - Qualität und Genauigkeit.* Springer Vieweg, Springer Fachmedien Wiesbaden, 5th edition.

Law, K. J. H., Stuart, A. M., and Zygalakis, K. C. (2015). *Data Assimilation: A Mathematical Introduction*, volume 62 of Texts in Applied Mathematics. Springer International Publishing.

Lentine, M., Grétarsson, J. T., and Fedkiw, R. (2011). *An unconditionally stable fully conservative semi-Lagrangian method.* Journal of Computational Physics, 230(8):2857–2879.

LeVeque, R. J. (2005). *Finite Difference Methods for Differential Equations.*

LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA.

Lintermann, A. and Schröder, W. (2018). *Zonal flow solver (ZFS): a highly efficient multi-physics simulation framework*. In 6th European Conference on Computational Mechanics and 7th European Conference on Computational Fluid Dynamics, Glasgow, United Kingdom.

Liskov, B. (1987). *Data abstraction and hierarchy*. SIGPLAN Not., 23(5):17–34.

Liu, Y., Liu, X., and Wu, E. (2004). *Real-Time 3D Fluid Simulation on GPU with Complex Obstacles*.

Lopez, M. G., Larrea, V. V., Joubert, W., Hernandez, O., Haidar, A., Tomov, S., and Dongarra, J. (2016). *Towards achieving performance portability using directives for accelerators*. In Third Workshop on Accelerator Programming Using Directives (WACCPD), pages 13–24.

Ludwig, J. C. (2011). *PHOENICS-VR Reference Guide*. Technical report, CHAM. URL: `http://www.cham.co.uk/phoenics.php`.

Marchi, C. H., Suero, R., and Araki, L. K. (2009). *The Lid-Driven Square Cavity Flow: Numerical Solution with a 1024 x 1024 Grid*. J. of the Braz. Soc. of Mech. Sci. & Eng., XXXI, No. 3:186–198.

Martin, R. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Pearson Education.

Martinuzzi, R. and Tropea, C. (1993). *The Flow Around Surface-Mounted, Prismatic Obstacles Placed in a Fully Developed Channel Flow*. Journal of Fluids Engineering, 115(1):85–92.

McCaffrey, B. J. and Cox, G. (1982). *Entrainment and heat flux of buoyant diffusion flames*. NBSIR 82-2473. U.S. Department of Commerce, National Bureau of Standards, Washington, D.C.

McCalpin, J. D. (1995). *Memory bandwidth and machine balance in current high performance computers*. IEEE Comput. Soc. Techn. Committee Comput. Archit. (TCCA) Newsl., pages 19–25.

McDermott, R. (2003). *A non-trivial analytical solution to the 2-D incompressible Navier-Stokes equations*. Technical report. URL: `https://sites.google.com/site/randymcdermott/NS_exact_soln.pdf`.

McDonough, J. M. (2007). *Introductory Lectures on Turbulence - Physics, Mathematics and Modeling*.

McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., Vanella, M., Weinschenk, C., and Overholt, K. (2017a). *Fire Dynamics Simulator Technical Reference Guide Volume 2: Verification*. Technical report.

McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., Vanella, M., Weinschenk, C., and Overholt, K. (2017b). *Fire Dynamics Simulator User's Guide*. Technical report.

McGrattan, K., McDermott, R., Hostikka, S., Floyd, J., Vanella, M., Weinschenk, C., and Overholt, K. (2017c). *Fire Dynamics Simulator Technical Reference Guide Volume 3: Validation*. Technical report, National Institute of Standards and Technology (NIST) and VTT Technical Research Centre of Finland.

Melaaen, M. C. (1990). *Analysis of curvilinear non-orthogonal coordinates for numerical calculation of fluid flow in complex geometries*. University of Trondheim, Norwegian Institute of Technology, Division of Thermodynamics, Trondheim.

Meunders, A. (2016). *A study on buoyancy-driven flows: Using particle image velocimetry for validating the Fire Dynamics Simulator*. PhD thesis, University of Wuppertal (BUW).

Meunders, A., Arnold, L., Belt, A., and Hundhausen, A. (2018). *Velocity measurements of a bench scale buoyant plume applying particle image velocimetry*. International Journal of Heat and Mass Transfer, 123:473–488.

Meyer, J. (2005). *Interactive Real-Time Simulation of Smoke*. Master, University of Copenhagen.

Moin, P. (2001). *Fundamentals of Engineering Numerical Analysis*. Cambridge University Press.

Moin, P., Squires, K., Cabot, W., and Lee, S. (1991). *A dynamic subgrid-scale model for compressible turbulence and scalar transport*. Physics of Fluids A, 3(11):2746–2757.

Murray, L. (2011). *GPU Acceleration of Runge-Kutta Integrators*. IEEE Transactions on Parallel and Distributed Systems, 23(1):94 – 101.

Münch, M. (2013). *Konzept zur Absicherung von CFD-Simulationen im Brandschutz und in der Gefahrenabwehr*. INURI.

Nicolini, M., Miller, J., Wienke, S., Schlottke-Lakemper, M., Meinke, M., and Müller, M. S. (2016). *Software cost analysis of GPU-accelerated aeroacoustics simulations in C++ with OpenACC*. In Taufer, M., Mohr, B., and Kunkel, J. M., editors, ISC High Performance 2016. LNCS, volume 9945, pages 524–543. Springer.

NIOSH (2016). *Mining Product: MFIRE*. Technical report, The National Institute for Occupational Safety and Health (NIOSH). URL: `https://www.cdc.gov/niosh/mining/works/coversheet1816.html`.

Noto, K., Teramoto, K., and Nakajima, T. (1999). *Spectra and Critical Grashof Numbers for Turbulent Transition in a Thermal Plume*. Journal of Thermophysics and Heat Transfer, 13(1):82–90.

NVIDIA Corporation (2018a). *GPU-accelerated Applications*. Technical report. URL: `https://www.nvidia.com/content/gpu-applications/PDF/gpu-applications-catalog.pdf`.

NVIDIA Corporation (2018b). *NVIDIA CUDA C Programming Best Practices Guide*. Technical report, NVIDIA Corporation.

ORPHEUS (2015-2018). *BMBF funded research project, Optimierung der Rauchableitung und Personenführung in U-Bahnhöfen: Experimente und Simulationen (ORPHEUS) - Teilvorhaben: Brand- und Personenstromsimulationen in unterirdischen Verkehrsstationen*. Technical report, Research Center Jülich GmbH. URL: `http://www.orpheus-projekt.de`.

Patankar, S. V. (1981). *A calculation procedure for two-dimensional elliptic situations*. Numerical Heat Transfer, 4(4):409–425.

Patankar, S. V. and Spalding, D. B. (1972). *A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows*. International Journal of Heat and Mass Transfer, 15(10):1787–1806.

Peaceman, D. W. and Rachford Jr., H. H. (1955). *The Numerical Solution of Parabolic and Elliptic Differential Equations*. Journal of the Society for Industrial and Applied Mathematics, 3(1):28–41.

Peiró, J. and Sherwin, S. (2005). *Finite Difference, Finite Element and Finite Volume Methods for Partial Differential Equations.* In Yip, S., editor, Handbook of Materials Modeling: Methods, pages 2415–2446. Springer Netherlands, Dordrecht.

Pennycook, S. J., Hammond, S. D., Wright, S. A., Herdman, J. A., Miller, I., and Jarvis, S. A. (2013). *An investigation of the performance portability of OpenCL.* J. Parallel Distrib. Comput., 73(11):1439–1450.

Prandtl, L. (1949). *Führer durch die Strömungslehre (engl. Essentials of Fluid Dynamics)*, volume 3. F. Vieweg, University of California.

Pronchick, S. W. (1983). *An experimental investigation of the structure of a turbulent reattaching flow behind a backward-facing step.* PhD thesis, Stanford University, CA.

Quintiere, J. G. (2006). *Fundamentals of fire phenomena.* John Wiley & Sons, Ltd.

RCPE (2018). *Reserach Center Pharmaceutical Engineering: Computional Fluid Dynamics – Discrete Element Method (CFD-DEM).* Technical report. URL: `http://www.rcpe.at/en/competences/modeling-prediction/computional-fluid-dynamics-discrete-element-method-cfd-dem/`.

Reynolds, O. (1883). *An experimental investigation of the circumstances which determine whether the motion of water in parallel channels shall be direct or sinuous and of the law of resistance in parallel channels.* Philosophical Transactions of the Royal Society, 174:935–982.

Reynolds, O. (1895). *On the dynamical theory of incompressible viscous fluids and the determination of the criterion.* Philosophical Transactions of the Royal Society of London. (A.), 186:123–164.

Rivi, M., Calori, L., Muscianisi, G., and Slavnic, V. (2012). *In-situ visualization: State-of-the-art and some use cases.* PRACE White Paper, pages 1–18.

Rochoux, M. C., Cuenot, B., Ricci, S., Trouvé, A., Delmotte, B., Massart, S., Paoli, R., and Paugam, R. (2013a). *Data assimilation applied to combustion.* Comptes Rendus Mécanique, 341(1):266–276.

Rochoux, M. C., Delmotte, B., Cuenot, B., Ricci, S., and Trouvé, A. (2013b). *Regional-scale simulations of wildland fire spread informed by real-time flame front observations.* Proceedings of the Combustion Institute, 34(2):2641–2647.

Rochoux, M. C., Emery, C., Ricci, S., Cuenot, B., and Trouvé, A. (2015). *Towards predictive data-driven simulations of wildfire spread – Part II: Ensemble Kalman Filter for the state estimation of a front-tracking simulator of wildfire spread*. Natural Hazards Earth System Sciences, 15:1721–1739.

Rochoux, M. C., Ricci, S., Lucor, D., Cuenot, B., and Trouvé, A. (2014). *Towards predictive data-driven simulations of wildfire spread –Part I: Reduced-cost Ensemble Kalman Filter based on a Polynomial Chaos surrogate model for parameter estimation*. Natural Hazards Earth System Sciences, European Geosciences Union, 14(11):2951–2973.

Rodi, W. (1997). *Comparison of LES and RANS calculations of the flow around bluff bodies*. Journal of Wind Engineering and Industrial Aerodynamics, 69-71:55–75.

Rodi, W., Ferziger, J., Breuer, M., and Pourquié, M. (1995). *Workshop on Large-Eddy Simulation of Flows past Bluff Bodies*. Technical report, University of Karlsruhe.

Rodi, W., Ferziger, J., Breuer, M., and Pourquié, M. (1997). *Status of large-eddy simulation: Results of a workshop*. ASME Transaction Journal of Fluids Engineering, 119:248–262.

Ruffle, S. (1986). *Architectural Design Exposed: From Computer-Aided Drawing to Computer-Aided Design*. Environment and Planning B: Planning and Design, 13(4):385–389.

Rupp, K. (2016). *CPU, GPU and MIC Hardware Characteristics over Time*. Technical report. URL: `https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/`.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2nd edition.

Saad, Y. and Schultz, M. H. (1986). *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*. SIAM Journal on Scientific and Statistical Computing, 7(3):856–869.

Sabne, A., Sakdhnagool, P., Lee, S., and Vetter, J. S. (2014). *Evaluating Performance Portability of OpenACC*. In International Workshop on Languages and Compilers for Parallel Computing (LCPC).

Sagaut, P. (2006). *Large Eddy Simulation for Incompressible Flows: An Introduction*. Scientific Computation. Springer-Verlag Berlin Heidelberg, 2nd edition.

Scharfetter, D. L. and Gummel, D. L. (1969). *Large signal analysis of a Silicon Read diode oscillator*. IEEE Transaction on Electron Devices, 16(1):64–77.

Schneider, U. (2006). *Systematische Zusammenstellung von Bemessungsbrandszenarien für den Brandschutzentwurf*. Technical report, Institut für Hochbau und Technologie.

Schnipke, R. J. (1986). *Streamline upwind finite-element method for laminar and turbulent flow*. PhD thesis, University of Virginia, Charlottesville, VA.

Schröder, B. (2016). *Multivariate Methods for Life Safety Analysis in Case of Fire*. PhD thesis, University of Wuppertal (BUW).

Selle, A., Fedkiw, R., Kim, B. M., Liu, Y., and Rossignac, J. (2008). *An Unconditionally Stable MacCormack Method*. Journal of Scientific Computing, 35(2-3):350–371.

Siemens PLM software (2017). *Star-CCM+*. Technical report. URL: `https://mdx.plm.automation.siemens.com/star-ccm-plus`.

Smagorinsky, J. (1963). *General Circulation Experiments with the Primitive Equations*. Monthly Weather Review, 91:99.

Smart, D. and White, J. (1988). *Reducing the Parallel Solution Time of Sparse Circuit Matrices Using Reordered Gaussian Elimination and Relaxation*.

Sobachkin, A. and Dumnov, G. (2013). *White Paper: Numerical Basis of CAD-Embedded CFD*. Technical report, Dassault Systems SOLIDWORKS. URL: `https://www.solidworks.com/sw/docs/Flow_Basis_of_CAD_Embedded_CFD_Whitepaper.pdf`.

Stam, J. (1999). *Stable fluids*. SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 121–128.

Steckler, K. D., Baum, H. R., and Quintiere, J. G. (1985). *Fire induced flows through room openings-flow coefficients*. Symposium (International) on Combustion, 20(1):1591–1600.

Steckler, K. D., Quintiere, J. G., and Rinkinen, W. J. (1982). *Flow induced by fire in a compartment*. Symposium (International) on Combustion, 19(1):913–920.

Stefan, J. (1879). *Über die Beziehung zwischen der Wärmestrahlung und der Temperatur*. Sitzungsberichte der mathematisch-naturwissenschaftlichen Classe der kaiserlichen Akademie der Wissenschaften Wien, 79:391–428.

Thacker, B. H., Doebling, S. W., Hemez, F. M., Anderson, M. C., Pepin, J. E., and Rodriguez, E. A. (2004). *Concepts of Model Verification and Validation*. Technical report, Los Alamos National Lab.

The OpenFOAM Foundation (2017). *OpenFOAM v5 User Guide*. Technical report. URL: `https://github.com/fireFoam-dev`.

The Portland Group (2015). *OpenACC API 2.5 Reference Guide*. Technical report, NVIDIA Corporation.

Top500 (2017). *Top500 List*. Technical report, Top500. URL: `https://www.top500.org/lists/2017/11/`.

Tritsiklis, J. N. (1989). *A comparison of Jacobi and Gauss-Seidel parallel iterations*. Applied Mathematics Letters, 2(2):167–170.

van der Vorst, H. A. (1992). *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*. SIAM Journal on Scientific and Statistical Computing, 13(2):631–644.

Van Doormaal, J. P. and Raithby, G. D. (1984). *Enhancement of the SIMPLE method for predicting incompressible fluid flows*. Numerical Heat Transfer, 7(2):147–163.

van Nederveen, G. A. and Tolman, F. P. (1992). *Modelling multiple views on buildings*. Automation in Construction, 1(3):215–224.

VDI (2013). *VDI-Wärmeatlas*, volume 11., bearb. und erw. Aufl. of Springer Reference. Springer Vieweg, Berlin, Heidelberg.

Verma, S., Xuan, Y., and Blanquart, G. (2014). *An improved bounded semi-Lagrangian scheme for the turbulent transport of passive scalars*. Journal of Computational Physics, 272:1–22.

Viskanta, R. and Mengüç, M. P. (1987). *Radiation heat transfer in combustion systems*. Progress in Energy and Combustion Science, 13(2):97–160.

Vreman, B. (2004). *An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications*. Physics of Fluids, 16(10):3670–3681,.

Wang, Y., Qin, Q., See, S. C. W., and Lin, J. (2013). *Performance portability evaluation for OpenACC on Intel Knights Corner and Nvidia Kepler*.

White, F. M. (1991). *Viscous fluid flow*. McGraw-Hill series in mechanical engineering. McGraw-Hill, New York, 2nd edition.

Williams, S., Waterman, A., and Patterson, D. (2009). *Roofline: an insightful visual performance model for multicore architecture*. Commun. ACM, 52(4):65–76.

Würzburger, M. L. (2016). *Untersuchung des Rot-Schwarz Gauß-Seidel-Verfahrens für Diffusionsprobleme bezüglich Parallelisierung auf einer GPU*.

Yakhot, A., Liu, H., and Nikitin, N. (2006). *Turbulent flow around a wall-mounted cube: A direct numerical simulation*. International Journal of Heat and Fluid Flow, 27(6):994–1009.

Yeoh, G. H. and Yuen, K. K. (2009). *Computational Fluid Dynamics in Fire*. Elsevier Inc.

Zhou, L., Smith, A. C., and Yuan, L. (2016). *New improvements to MFIRE to enhance fire-modeling capabilities*. Mining Engineering, 68(6):45–50.

Band / Volume 27
Automatische Erfassung präziser Trajektorien in Personenströmen hoher Dichte
M. Boltes (2015), xii, 308 pp
ISBN: 978-3-95806-025-8
URN: urn:nbn:de:0001-2015011609

Band / Volume 28
Computational Trends in Solvation and Transport in Liquids
edited by G. Sutmann, J. Grotendorst, G. Gompper, D. Marx (2015)
ISBN: 978-3-95806-030-2
URN: urn:nbn:de:0001-2015020300

Band / Volume 29
Computer simulation of pedestrian dynamics at high densities
C. Eilhardt (2015), viii, 142 pp
ISBN: 978-3-95806-032-6
URN: urn:nbn:de:0001-2015020502

Band / Volume 30
Efficient Task-Local I/O Operations of Massively Parallel Applications
W. Frings (2016), xiv, 140 pp
ISBN: 978-3-95806-152-1
URN: urn:nbn:de:0001-2016062000

Band / Volume 31
A study on buoyancy-driven flows: Using particle image velocimetry for validating the Fire Dynamics Simulator
by A. Meunders (2016), xxi, 150 pp
ISBN: 978-3-95806-173-6
URN: urn:nbn:de:0001-2016091517

Band / Volume 32
Methoden für die Bemessung der Leistungsfähigkeit multidirektional genutzter Fußverkehrsanlagen
S. Holl (2016), xii, 170 pp
ISBN: 978-3-95806-191-0
URN: urn:nbn:de:0001-2016120103

Band / Volume 33
JSC Guest Student Programme Proceedings 2016
edited by I. Kabadshow (2017), iii, 191 pp
ISBN: 978-3-95806-225-2
URN: urn:nbn:de:0001-2017032106

Schriften des Forschungszentrums Jülich
IAS Series

Band / Volume 34
Multivariate Methods for Life Safety Analysis in Case of Fire
B. Schröder (2017), x, 222 pp
ISBN: 978-3-95806-254-2
URN: urn:nbn:de:0001-2017081810

Band / Volume 35
Understanding the formation of wait states in one-sided communication
M.-A. Hermanns (2018), xiv, 144 pp
ISBN: 978-3-95806-297-9
URN: urn:nbn:de:0001-2018012504

Band / Volume 36
A multigrid perspective on the parallel full approximation scheme in space and time
D. Moser (2018), vi, 131 pp
ISBN: 978-3-95806-315-0
URN: urn:nbn:de:0001-2018031401

Band / Volume 37
Analysis of I/O Requirements of Scientific Applications
S. El Sayed Mohamed (2018), XV, 199 pp
ISBN: 978-3-95806-344-0
URN: urn:nbn:de:0001-2018071801

Band / Volume 38
Wayfinding and Perception Abilities for Pedestrian Simulations
E. Andresen (2018), 4, x, 162 pp
ISBN: 978-3-95806-375-4
URN: urn:nbn:de:0001-2018121810

Band / Volume 39
Real-Time Simulation and Prognosis of Smoke Propagation in Compartments Using a GPU
A. Küsters (2018), xvii, 162, LIX pp
ISBN: 978-3-95806-379-2
URN: urn:nbn:de:0001-2018121902

Weitere ***Schriften des Verlags im Forschungszentrum Jülich*** unter
http://wwwzb1.fz-juelich.de/verlagextern1/index.asp

IAS Series
Band / Volume 39
ISBN 978-3-95806-379-2

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum