Discontinuous Galerkin methods and high performance computing approaches for ocean simulations

Von der Universität Bayreuth
zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Abhandlung
von

Sara Faghih-Naini

University of Bayreuth
Faculty of Mathematics, Physics and Computer Sciences
Chair for Scientific Computing

UNIVERSITÄT
BAYREUTH

Sara Faghih-Naini

Discontinuous Galerkin methods and high performance computing approaches for ocean simulations

Day of submission:	November 28, 2023
Day of oral examination:	May 6, 2024
Chairmen of doctoral committee:	Prof. Dr. Mario Bebendorf
Reviewers:	Prof. Vadym Aizinger
	Prof. Luca Bonaventura
Examiner:	Prof. Harald Köstler
Supervisor:	Prof. Vadym Aizinger

University of Bayreuth

Chair for Scientific Computing

Faculty of Mathematics, Physics and Computer Sciences

Universitätsstraße 30

95447 Bayreuth, Germany

Abstract

The shallow water equations (SWE) are a set of hyperbolic conservation laws frequently used to model oceanographic and atmospheric fluid flow. They are derived from the fundamental principles of mass and momentum conservation and are applicable when vertical dynamics can be considered negligible compared to horizontal effects. Therefore, they are a commonly used model to predict storm surges, tsunamis, and floods as well as to study tides and coastal ocean circulation.

Since finding analytical solutions for the SWE is limited to specific cases, numerical methods are the primary choice for solving these equations. However, the model poses challenges for the numerical approaches. These challenges originate from the nonlinear nature of the coupled equations, the different flow regimes ranging from smooth regions to shocks, and the complex computational grids arising from irregular domain boundaries and varying bottom topography. Hence, the SWE are also often employed to prototype numerical techniques for ocean circulation models. In this regard, discontinuous Galerkin (DG) methods stand out as a suitable approach, combining the respective strengths of continuous finite element and finite volume methods. They offer local conservation properties, robustness for handling shocks and discontinuities, applicability to complex geometries, and natural support for mesh and discretization order adaptivity.

Nonetheless, these benefits of DG discretizations come at the cost of high computational demands, which can only be partially mitigated through efficient parallel hardware utilization. As the increase in the number of transistors used in traditional central processing units levels off and the available hardware becomes more diverse, there is a need to explore new directions to enhance computational efficiency.

This thesis focuses on advancing the computational performance of DG discretizations applied to the two-dimensional SWE and explores several methodologies in pursuit of this goal.

It introduces a novel quadrature-free DG formulation for the nonlinear SWE, which relies solely on product-type nonlinearities. Traditional quadrature integrations are replaced with analytical evaluations. The method's stability is proven with a new analysis approach.

The discretization is implemented within the Python frontend GHODDESS, which generates domain specific language code for the automatic code generation framework ExaStencils, which, in turn, provides performance portability.

Discretization order (p-) adaptivity is incorporated into the numerical scheme, including a new adaptivity indicator independent of user-defined input parameters. Thereby, the number of degrees of freedom is reduced while maintaining the solution quality.
The thesis furthermore investigates an algorithmic redesign of the p-adaptive scheme that separates the adaptive and non-adaptive parts of the model code to improve the hardware utilization of a heterogeneous CPU–GPU system, resulting in accelerated computations.
Lastly, it presents simulations on masked block-structured grids for realistic ocean domains, which, on the one hand, are capable of accurately meshing fine-scale geometric features and, on the other, offer performance benefits associated with structured grid models.
All proposed algorithmic adaptions are evaluated with various numerical test cases encompassing discontinuous solutions and typical tidal flow problems. With these contributions, this thesis advances the state of the art in numerical methods for simulating shallow-water flows.

Zusammenfassung

Discontinuous Galerkin Methoden und Hochleistungsrechenansätze für Ozeansimulationen

Die Flachwassergleichungen (engl. *shallow water equations* – SWE) sind hyperbolische Erhaltungssätze, die häufig zur Modellierung ozeanografischer und atmosphärischer Strömungen verwendet werden. Sie leiten sich von den Grundprinzipien der Massen- und Impulserhaltung ab und sind anwendbar, wenn vertikale Dynamiken im Vergleich zu horizontalen Effekten als vernachlässigbar angesehen werden können. Daher werden sie häufig zur Vorhersage von Sturmfluten, Tsunamis und Überschwemmungen sowie zur Untersuchung von Gezeiten und der Ozeanzirkulation an der Küste verwendet.
Da analytische Lösungen für die SWE nur in speziellen Fällen gefunden werden können, sind numerische Methoden die bevorzugte Wahl zur Lösung dieser Gleichungen. Das Modell stellt jedoch Herausforderungen an die numerischen Ansätze. Diese ergeben sich aus der nicht-linearen Natur der gekoppelten Gleichungen, den unterschiedlichen Strömungsbereichen, die von glatten Regionen bis hin zu Schocks reichen, und den komplexen Rechengittern aufgrund unregelmäßiger Gebietsgrenzen und variabler Bodentopografie. Daher werden die SWE auch häufig als Prototyp für numerische Techniken für Ozeanzirkulationsmodelle verwendet. In diesem Zusammenhang zeichnen sich unstetige Galerkin (engl. *discontinuous Galerkin* – DG) Methoden als geeigneter Ansatz aus, da sie die jeweiligen Stärken von kontinuierlichen Finite-Elemente- und Finite-Volumen-Methoden kombinieren. Sie bieten lokale Erhaltungseigenschaften, Robustheit im Umgang mit Schocks und Unstetigkeiten, Anwendbarkeit auf komplexe Geometrien und natürliche Unterstützung für Gitter- und Diskretisierungsordnungs-Adaptivität.
Jedoch gehen diese Vorteile der DG-Diskretisierungen mit einem hohen Rechenaufwand einher, der nur teilweise durch effiziente parallele Hardwarenutzung gemildert werden kann. Mit dem abflachenden Anstieg der Anzahl von Transistoren in herkömmlichen Hauptprozessoren und der steigenden Vielfalt verfügbarer Hardware ist es notwendig, neue Ansätze zur Verbesserung der Recheneffizienz zu erforschen.

Diese Arbeit konzentriert sich auf die Verbesserung der Rechenleistung von DG-Diskretisierungen angewendet auf die zwei-dimensionalen SWE und untersucht mehrere Methoden, um dieses Ziel zu erreichen.

Sie stellt eine neue quadraturfreie DG-Formulierung für die nicht-linearen SWE vor, die ausschließlich auf Nichtlinearitäten in Produktform basiert. Traditionelle quadratur-basierte Integrationen werden durch analytische Auswertungen ersetzt. Die Stabilität der Methode wird mit einem neuen Ansatz analytisch bewiesen.
Die Diskretisierung ist in das Python-Frontend GHODDESS implementiert, das domänenspezifischen Sprachcode für das automatische Codegenerierungs-Framework ExaStencils generiert, welches wiederum die effiziente Rechenleistung auf verschiedener Hardware gewährleistet.
Die Diskretisierungsordnungs- (p-) Adaptivität ist in das numerische Schema integriert, einschließlich eines neuen Adaptivitätsindikators, der unabhängig von benutzerdefinierten Eingabeparametern ist. Dadurch wird die Anzahl der Freiheitsgrade reduziert, während die Lösungsqualität erhalten bleibt.
Die Arbeit untersucht außerdem eine algorithmische Neugestaltung des p-adaptiven Verfahrens, das die adaptiven und nicht-adaptiven Teile des Modellcodes trennt, um die Hardwarenutzung heterogener CPU–GPU-Systeme zu verbessern, was zu mehr Effizienz führt.
Abschließend werden Simulationen auf maskierten block-strukturierten Gittern für realistische Ozeangebiete präsentiert. Diese Gitter sind in der Lage, kleinskalige geometrische Merkmale genau zu erfassen, und bieten gleichzeitig Rechenleistungsvorteile in Verbindung mit strukturierten Gittermodellen.
Alle vorgeschlagenen algorithmischen Anpassungen werden anhand verschiedener numerischer Testfälle evaluiert, die unstetige Lösungen und typische Gezeitenströmungsprobleme umfassen.
Durch diese Beiträge treibt diese Arbeit den Stand der Technik bei numerischen Methoden zur Simulation von Flachwasserströmungen voran.

Acknowledgments

I extend my deepest gratitude to my supervisor, Prof. Vadym Aizinger, for his unwavering support and guidance during the course of this thesis and over the past eight years. His encouragement, trust, and mentorship have been invaluable in shaping the outcome of this work. Vadym Aizinger provided the necessary direction and the freedom to explore and develop diverse topics within this research. His expertise, constant willingness to engage in discussions, and ability to offer creative solutions have made our collaboration joyful and fruitful. Beyond the academic realm, Vadym Aizinger has been a constant source of advice and support, always available to lend a helping hand. I am profoundly grateful to him for all of this. Let me also thank him for proofreading this thesis, as always providing me with invaluable feedback.

I also wish to express my heartfelt appreciation to Prof. Harald Köstler, who served as my second supervisor. His innovative ideas, continuous support, and the scientific freedom he offered have continuously inspired and enriched my work.

Dr. Sebastian Kuckuk deserves special thanks for his inexhaustible support and guidance on implementation aspects. I thank him and Richard Angersbach for assistance with the ExaStencils code generation framework. Our collaborative discussions and software development endeavors have consistently been a rewarding experience.

This thesis was conducted as part of the DFG project 'Performance optimized software strategies for unstructured-mesh applications in ocean modeling'. Besides the people mentioned earlier, Prof. Daniel Zint and Dr. Roberto Grosso were also involved. Working together on this project was very enriching, and our collective work led to a highly satisfying outcome.

I extend my heartfelt gratitude to Prof. Andreas Rupp for his support and the numerous fruitful discussions during which he generously shared his impressive knowledge. Additionally, I would like to express my appreciation to him and Prof. Lassi Roininen for granting me the opportunity to visit AIMS Rwanda and for an inspiring and enriching experience there.

I would like to thank all my further collaborators, including Christoph Alt, Jennifer Faj, Jan Hönig, and Dr. Tobias Kenter for many insightful conversations.

Furthermore, I thank Dr. Hennes Hajduk for the fruitful exchange and friendship, and I greatly appreciate the valuable feedback from Prof. Dmitri Kuzmin.

I am grateful to Prof. Mario Bebendorf for letting me be part of the Chair of Scientific Computing at the University of Bayreuth, and I thank Prof. Kurt Chudej for lively conversations.

I am indebted to Prof. Ulrich Rüde and to Prof. Harald Köstler at the Chair of System Simulation at FAU Erlangen for making my employment possible.

I appreciate the efforts of those who proofread parts of this thesis, including Richard Angersbach, Hennes Hajduk, Dr. Franziska Henze, Samuel Kemmler, Sebastian Kuckuk, Andreas Rupp, Christoph Schwarzmeier, and Daniel Zint.
I would like to acknowledge support from Erlangen National High Performance Computing Center (NHR@FAU), especially from Dominik Ernst and Jan Laukemann.
I am thankful to Prof. Stefan Turek for granting access to the ICARUS cluster at the TU Dortmund and to Markus Geveler and Dominik Mütter for technical support on this cluster.
I thank Prof. Vadym Aizinger and Prof. Luca Bonaventura for serving as referees of this thesis, and Prof. Vadym Aizinger, Prof. Luca Bonaventura and Prof. Harald Köstler for serving as referees in my oral examination.
I am grateful to Dr. Balthasar Reuter for his support during my research assistant time and for forwarding the job advertisement for my next position.
I would like to thank Anke Bornmann in Bayreuth and Marina Lintl, Alexandra Lukas-Rother, and Iris Weiß in Erlangen for their support in handling various administrative matters. Furthermore, I thank Massimo Pinzer-Braese, the system administrator in Bayreuth, for his assistance. Additionally, I am deeply thankful to the system administration team in Erlangen, which comprises Christoph Alt, Richard Angersbach, Daniel Bauer, Martin Bauer, Frank Deserno, Dr. Sebastian Eibl, Jan Hönig, Christoph Schwarzmeier, and Dominik Thönnes, for their outstanding efforts in maintaining and ensuring the smooth operation of our IT infrastructure.
I thank my office mates in Bayreuth, Markus Büttner and Lisa Jagau, and Sebastian Kuckuk in Erlangen for lively exchanges. I am grateful to all my so far unmentioned current and former colleagues, including Sandra Aziz, Dr. Maximilian Bauer, Dr. Bernd Feist, Dr. Gaby Folger, Thomas Rau, Hannes Schmidt, and Christina Schwarz in Bayreuth and Fabian Böhm, Andrei Dumitrasc, Marco Heisig, Markus Holzer, Nils Kohl, Kajol Kulkarni, Girish Kumatagi, Benjamin Mann, Jonas Plewinski, Prof. Christoph Pflaum, Dr. Phillip Rall, Rafael Ravedutti Lucio Machado, Dr. Christoph Rettinger, Riccarda Scherner-Grießhammer, Jonas Schmitt, Helen Schottenhamml, Dr. Ramon Springer, Philipp Suffa, and Lukas Werner in Erlangen for creating an enjoyable working environment and introducing me to interesting research topics which have broadened my horizons. Moreover, I thank Sabine Aman and Dr. Mirjam Horn-Schott for engaging in lively conversations during our lunch breaks.
Supervising student projects has been a rewarding experience, and I am grateful for the opportunity to guide the work of Dinesh Parthasarathy, Samuel Kemmler, Kajol Kulkarni, Jonathan Schmalfuß, Adesh Shambhu, Julian Stahl, Markus Stüwe, and Yannik Wallerer.
I greatly appreciate Prof. Bo-Wen Shen's guidance and advice over the past seven years, and I thank him for giving me the first insights into research.
I extend my deepest appreciation to my parents, Sylvia Peitz-Faghih-Naini and Majid Faghih-Naini, and my sister, Samira Faghih-Naini. Their unconditional support in every aspect possible, their encouragement, and advice have been a constant source of strength for me, without which none of my achievements would have been possible. I furthermore express my wholehearted gratitude to my boyfriend, Christoph Schwarzmeier, for his love and unlimited support in many aspects. His presence has been a source of relaxation and rejuvenation throughout this endeavor.

Contents

Chapter 1 Introduction

Approximately 71 % of the Earth's surface is covered by the ocean, which plays a critical role in supporting life on our planet through its regulation of global climate and holding vast resources [Vis18]. The ocean hosts hundreds of thousands of species ranging from microscopic algae to the largest creature ever known to have lived on Earth – the blue whale. For millennia, people have depended on the ocean as a food source and a route for trade and exploration. Today, more than one-third of the global population lives within 100 km from the coast, and almost two-thirds of all megacities are located in the low-elevation coastal zone [RVH23]. Additionally, scientific interest in harnessing the ocean's renewable energy potential is growing. Several countries have leveraged the energy potential of ocean waves, temperature gradients, currents, or tides to drive turbines and generate electricity [NH18].
Nonetheless, the frequency of extreme events like hurricanes, cyclones, typhoons, and tsunamis has increased in recent decades. These events can potentially cause catastrophic flooding and widespread devastation in coastal areas, posing a growing threat to coastal populations [Sen+12]. To address this challenge, accurate simulations are crucial for predicting and mitigating damage and saving lives through early warning systems [GTS06]. These systems aim to forecast essential parameters, including tsunami arrival times, estimated wave heights, and inundation zones [Har+08].
Accurate ocean simulations significantly enhance weather forecasts. They are critical in tracking events like storm surges, influenced by ocean conditions such as sea surface temperatures and currents [Ben+07]. Additionally, ocean simulations are essential components of climate models. These models help forecast changes like sea level variations, ocean temperature shifts, and the influence of oceanic patterns on global climate phenomena like El Niño and La Niña [Gri+00; Chu+01].

Ocean simulations involve solving the complex differential equations that govern the fluid motion. Hence, the model needs to be discretized using numerical techniques such as finite difference, finite element, and finite volume methods. Achieving high resolution and accuracy results in large data volumes and long computation times. However, ensuring high numerical accuracy in low computational time is crucial for reliable predictions in extreme event simulations. Thus, employing advanced numerical techniques, coupled with parallelization and optimization strategies, and access to high-performance computing resources are fundamental.

The current computational strategy for numerical simulations of oceanic and atmospheric systems predominantly relies on massively parallel architectures and partly on hybrid platforms. An increasingly diverse spectrum of relevant computing architectures is emerging, and the need to use new technologies has become evident due to the flattening of Moore's Law [Moo98]. Therefore, new directions must be explored to enhance computational efficiency [Sha20].
The costs associated with running ocean simulations emphasize the importance of computational sophistication. Thus, substantial efforts are directed toward evaluating and optimizing the computational performance of operational ocean models [WKC97; KJ05; JCT17; Kol+19; RD19; Irr+22]. In addition, the community is increasingly focusing on automatic code generation to increase the productivity of model code development and to achieve a separation of concerns between model development and software engineering [Eng02; Tor+13; Afa+21; Sze+24].
The choice of computational grids which, are necessary for the numerical methods, is also a key factor influencing the trade-off between accuracy and computational performance. The main properties of grids are their spatial resolution, the number of elements, and the overall mesh quality. Therefore, they balance how accurately simulations approximate reality and how quickly they can provide valuable information to help mitigate the impact of extreme events.

This thesis focuses on applying discontinuous Galerkin (DG) methods to shallow-water-type flows. A key emphasis of this work lies in computational aspects and exploring methodologies to achieve performance improvements.
The shallow water model is a set of partial differential equations representing conservation laws that find application in situations where the vertical dimension is much smaller than typical horizontal scales. It is not limited solely to water bodies but is also relevant to other fluids, including air in atmospheric flows, where it serves as a convenient benchmark [Vre94].
In order to obtain computable expressions, the continuous equations need to be converted into discrete approximations. DG discretizations provide several advantages, as explained in the next section. However, they come at the cost of higher computational overhead compared to other methods. Hence, a performant implementation and the effective utilization of parallel computing resources are even more essential to achieve precise results within an appropriate time.

1.1 Objective of this work

As previously mentioned, the primary objective of this thesis is to enhance the computational performance of shallow water simulations that rely on DG methods. We investigate a range of numerical, algorithmic, and computational technologies that aim to improve the performance of our code.

Firstly, we present a new formulation of the SWE that exclusively contains product-type nonlinearities. This allows for an analytical evaluation of all integrals resulting from the

discretization, eliminating the need for quadrature rules. Using new analysis techniques, we theoretically prove the stability of the new formulation. We also evaluate the stability and the convergence behavior numerically.
To achieve performance portability across diverse hardware architectures, we employ automatic code generation. This involves extending an existing source-to-source compiler with a Python frontend, which translates the quadrature-free DG discretization of the SWE into domain specific language code. This code is subsequently transformed to a highly optimized parallel application tailored to the target hardware architecture.
Furthermore, we extend the DG scheme by incorporating discretization order (p-) adaptivity to reduce the number of degrees of freedom while maintaining the solution quality. We design a new adaptivity indicator that dynamically determines whether to increase or decrease the approximation order for each grid element. This indicator operates without user-defined input parameters and is compared to two further indicators for evaluation.
We also redesign the p-adaptive algorithm by taking advantage of the quadrature-free formulation of the SWE. Leveraging heterogeneous computing, the new algorithm aims to mitigate load-balancing issues and distributes computations between the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) using code generation techniques. The distribution is determined based on an evaluation of kernel-based performance metrics to minimize overall execution time.
We explore the usage of block-structured grids to further balance accuracy and computational performance. In particular we evaluate a new masking approach. This approach involves generating a grid covering a larger area than the actual computational domain and excluding elements outside of it. It aims to represent complex features while preserving some structure for performance optimizations.

1.2 Outline of this thesis

In the next chapter, we begin by introducing the shallow water equations and a new formulation that eliminates fraction-type nonlinearities, enabling quadrature-free integral evaluations. Following that, we present a p-adaptive discontinuous Galerkin discretization of the model, including a redesign of the algorithm for optimal hardware usage. Next, we introduce a new parameter-free adaptivity indicator. This is followed by stability proofs of both the continuous shallow water model and its discontinuous Galerkin discretization.
In Chapter 3, we explain a code generation framework and its Python frontend developed within the context of this work. Additionally, we present masked block-structured computational grids and a reference code used in this study.
We first outline the setups of all numerical examples used for evaluating the technologies presented throughout this work at the beginning of Chapter 4. Then, we assess the quadrature-free discretization, the adaptivity indicator, the algorithmic redesign, and the masked block-structured grids.
Finally, in Chapter 5, we conclude this thesis by summarizing our findings and outlining prospects for future research.

1.3 Previously published articles

Large parts of this thesis have already been published in the form of peer-reviewed journal articles, which are listed below.

[FNA22] S. Faghih-Naini and V. Aizinger. “p-adaptive discontinuous Galerkin method for the shallow water equations with a parameter-free error indicator”. In: *International Journal on Geomathematics* 13.18 (2022). DOI: `10.1007/s13137-022-00208-3`.

[FN+20] S. Faghih-Naini, S. Kuckuk, V. Aizinger, D. Zint, R. Grosso, and H. Köstler. “Quadrature-free discontinuous Galerkin method with code generation features for shallow water equations on automatically generated block-structured meshes”. In: *Advances in Water Resources* 138 (2020), p. 103552. DOI: `10.1016/j.advwatres.2020.103552`.

[FN+23b] S. Faghih-Naini, S. Kuckuk, D. Zint, S. Kemmler, H. Köstler, and V. Aizinger. “Discontinuous Galerkin method for the shallow water equations on complex domains using masked block-structured grids”. In: *Advances in Water Resources* 182 (2023), p. 104584. DOI: `10.1016/j.advwatres.2023.104584`.

Parts of this thesis are based on the following article, which is under review.

[FN+23a] S. Faghih-Naini, V. Aizinger, S. Kuckuk, R. Angersbach, and H. Köstler. “p-adaptive discontinuous Galerkin method for the shallow water equations on heterogeneous computing architectures”. In: *submitted to International Journal on Geomathematics, preprint available at https://doi.org/10.48550/arXiv.2311.11348* (2023).

Sara Faghih-Naini is the main author of the above mentioned articles.

In [FNA22], Sara Faghih-Naini conceptualized, implemented and evaluated the newly developed indicator, including verification and visualization of the results. Vadym Aizinger has contributed to this publication in the scope of supervision of Sara Faghih-Naini.
Sections 2.3.2, 2.3.4 and 4.3 are based on this publication.

In [FN+20], Sara Faghih-Naini implemented and evaluated the newly developed numerical model in collaboration with Sebastian Kuckuk, verified and validated it, generated the grids used for computations, conducted the simulations, and visualized the results. Daniel Zint, under the supervision of Roberto Grosso, developed the block-structured grid generator. Vadym Aizinger and Harald Köstler have contributed to this publication in the scope of supervision of Sara Faghih-Naini and Sebastian Kuckuk.
Sections 2.1, 2.2, 2.3.1, 3.1 and 4.2 are based on this publication.

In [FN+23b], Sara Faghih-Naini implemented, validated, verified and evaluated the numerical adaptations, generated the grids used for computations, conducted the simulations, and visualized the results. Samuel Kemmler did a preliminary implementation, including validation and all performance measurements. Daniel Zint, under the supervision of Roberto Grosso, developed the block-structured grid generator. Vadym Aizinger, Harald Köstler and Sebastian Kuckuk have contributed to this publication in the scope of supervision of Sara Faghih-Naini and Samuel Kemmler.
Sections 3.2 and 4.5 are based on this publication.

In [FN+23a], Sara Faghih-Naini implemented, verified and evaluated the newly developed numerical scheme, conducted all performance measurements and simulations, and visualized the results. Richard Angersbach implemented the necessary adaptations in the underlying code-generation framework. Vadym Aizinger, Harald Köstler and Sebastian Kuckuk have contributed to this publication in the scope of supervision of Sara Faghih-Naini and Richard Angersbach. Sections 2.3.3, 3.1 and 4.4 are based on this publication.

Furthermore, Sara Faghih-Naini is a coauthor in the following publications which are not used in this thesis.

[Alt+23] C. Alt, T. Kenter, S. Faghih-Naini, J. Faj, J.-O. Opdenhövel, C. Plessl, V. Aizinger, J. Hönig, and H. Köstler. “Shallow Water DG Simulations on FPGAs: Design and Comparison of a Novel Code Generation Pipeline”. In: *High Performance Computing.* Ed. by A. Bhatele, J. Hammond, M. Baboulin, and C. Kruse. Cham: Springer Nature Switzerland, 2023, pp. 86–105. DOI: 10.1007/978-3-031-32041-5_5.

[Faj+23] J. Faj, T. Kenter, S. Faghih-Naini, C. Plessl, and V. Aizinger. “Scalable Multi-FPGA Design of a Discontinuous Galerkin Shallow-Water Model on Unstructured Meshes”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference.* PASC ’23. Davos, Switzerland: Association for Computing Machinery, 2023. DOI: 10.1145/3592979.3593407.

[Ken+21] T. Kenter, A. Shambhu, S. Faghih-Naini, and V. Aizinger. “Algorithm-Hardware Co-Design of a Discontinuous Galerkin Shallow-Water Model for a Dataflow Architecture on FPGA”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference.* PASC ’21. Geneva, Switzerland: Association for Computing Machinery, 2021. DOI: 10.1145/3468267.3470617.

[Zin+22] D. Zint, R. Grosso, V. Aizinger, S. Faghih-Naini, S. Kuckuk, and H. Köstler. “Automatic Generation of Load-Balancing-Aware Block-Structured Grids for Complex Ocean Domains”. In: *30th International Meshing Roundtable, SIAM IMR 2022.* Zenodo, 2022. DOI: 10.5281/zenodo.6562440.

Chapter

Discontinuous Galerkin methods for the 2D shallow water equations 2

The *shallow water equations* (SWE) represent a set of hyperbolic equations describing fluid flows in domains where the horizontal length scale is significantly greater than the vertical length scale [Vre94]. This assumption is valid in various scenarios, prominently encompassing oceans, coastal regions, large lakes, and rivers. Following [Vre94; HB99; KC00; CRB11; Aiz19], we outline the derivation of the SWE starting from mass and momentum conservation.

Beginning with the *Reynolds averaged Navier–Stokes equations* formulated within a rotating frame of reference, incompressibility of the fluid is assumed, implying that density remains unaffected by pressure variations. Density differences due to salinity and temperature gradients can be modeled, and the *Boussinesq approximation* is applied when these differences are assumed to be small. The Boussinesq approximation replaces the actual water density with its reference value everywhere except in the gravity forcing term. Subsequently, non-dimensionalization of the system is achieved by introducing characteristic values for horizontal and vertical lengths and velocities, along with an aspect ratio of vertical to horizontal dimensions, typically at least of 1/10 for most realistic oceanic domains. Collecting the leading-order terms yields the *hydrostatic pressure condition*, which states the balance between the negative vertical derivative of the pressure and the gravitational force. Taking the hydrostatic balance as a diagnostic equation, the *hydrostatic equations of the ocean* are derived, constituting the most frequently utilized mathematical model for simulating *baroclinic* (i.e., variable density) circulation in global, regional, and coastal ocean. Compared to non-hydrostatic models, the hydrostatic system offers simpler numerical treatment. However, it does not account for vertical accelerations and fails to conserve vertical momentum.
Additional simplifications can be applied for problems not significantly affected by the density-driven dynamics but focusing instead on *fast moving surface waves* like tidal flows or tsunamis. Assuming constant density and vertically uniform horizontal velocity, vertical integration over the depth and applying kinematic and dynamic boundary conditions leads to the two-dimensional SWE system (2.1)–(2.2) on page 10.

There exist several techniques for discretizing the SWE. These encompass *finite difference (FD) methods*, representing one of the simplest and oldest approaches for solving differential equations. Differential operators are approximated by replacing the derivatives in the equation using differential quotients. One of the initial instances of employing FD methods to the SWE is [AL77]. Despite their ease of implementation, their popularity has declined owing to their lack of flexibility, primarily because they are commonly employed with structured meshes. *Finite element (FE) methods*, originating from structural mechanics, have found extensive application in various fluid mechanics problems. They subdivide the domain into cells, called elements, and seek a solution of the variational form of the partial differential equation (PDE). FE methods, first applied to the SWE in [Wan+72], excel at handling complex geometries and irregular boundaries, however, they result in higher implementation complexity and increased computational costs. A more recent development are *finite volume (FV) methods*, which are based on the principles of conservation laws. These methods directly discretize the balance equation and ensure flux conservation through control volumes. However, the formulation of high-order accurate FV schemes that maintain stability and avoid numerical oscillations can be intricate and computationally demanding. The initial utilization of such methods in the context of the SWE was in [AGN93].

Discontinuous Galerkin (DG) finite element methods combine favorable attributes of both FE and FV methods [CKS00]. These methods are based on the FE framework, employing a variational formulation. Furthermore, they utilize discontinuous test and trial spaces and numerical fluxes used in FV methods [CKS00]. In this thesis, we focus on the DG method because it has emerged as a powerful numerical technique for solving PDEs from a wide range of applications and since it possesses many favorable properties, as discussed below. The DG method was first proposed in 1973 in [RH73] for a time-independent linear hyperbolic equation. Subsequent advancements enabled its application to nonlinear hyperbolic conservation laws [CC89]. A significant breakthrough occurred with the combination of explicit, nonlinearly stable high-order *Runge–Kutta time discretizations* with a DG discretization in space with exact or approximate Riemann solvers for interface fluxes in [CLS89; CS89; CHS90; CS91; CS98b]. This advancement allowed the treatment of time-dependent nonlinear hyperbolic conservation laws involving first-order derivatives, such as the Euler equations [XS10].

The original DG method has three major generalization directions [Aiz19]. First, to solve second-order elliptic and parabolic problems, *interior penalty discontinuous Galerkin* methods were developed [Arn82], further classified based on the symmetry of the resulting bilinear form [Riv08]. Second, in [CS98a], the *local discontinuous Galerkin* method was introduced, motivated by the work in [BR97]. This method rewrites higher-order PDEs into first-order systems and applies the standard DG method to solve them. This mixed DG formulation has been the foundation for further developments, such as the *hybridized discontinuous Galerkin* method [CCS06]. The third direction involves more recent developments, considering *staggered discontinuous Galerkin* methods, wherein some discontinuous vertex or edge basis functions are employed in addition to element degrees of freedom [CE06].

We refer to [CKS00; XS10; DPE12; Rup19] for a broad overview of the history and more recent developments of DG methods.

DG methods offer several advantages compared to FD and FV methods, as outlined by [XS10] and [CKS00]:

- They can achieve an arbitrarily high formal order of accuracy by suitably choosing the degree of the approximating polynomials.
- They can easily handle complicated geometries and boundary conditions and are compatible with non-conforming meshes.
- They exhibit excellent parallelizability as the degrees of freedom of one element only need to be communicated to its face neighbors.
- They naturally support mesh and discretization order adaptivity because of the lack of continuity requirements on interfaces.
- They are proven to be at least $(p + \frac{1}{2})$-th order accurate in the L^2-norm for hyperbolic problems when piecewise polynomials of degree p are used, irrespective of the mesh structure [JP86]. However, the optimal rate of $p + 1$ is frequently observed in practice. For elliptic and parabolic problems convergence rates of $p + 1$ and p in the L^2-norm can be shown [Cas+01; Riv08].

Additionally, DG methods possess several desirable properties as summarized in [Aiz19]:

- They are locally conservative and exhibit strong stability properties.
- They are robust in handling problems involving shocks and discontinuities.
- They fully fit into the Galerkin/Petrov–Galerkin framework and allow to exploit the sophisticated analysis toolbox that leverages Sobolev space theory.

A serious drawback of DG methods is the significantly larger number of degrees of freedom required. However, this drawback can be mitigated by employing p-adaptivity and efficient parallel scaling.

After establishing the derivation of the SWE and various numerical approaches to solve them, our attention now shifts to established ocean circulation models. These models capture the intricate dynamics, aiding in understanding and forecasting oceanic behavior, and its broader impact on the environment. During the past decades, various models have been developed, each with unique characteristics and capabilities, and we highlight some of the most significant ones. These models can be broadly classified into two groups based on the computational mesh they employ [Reu20]. Among the well-established *structured grid models*, widely used examples of global ocean models are MITgcm[1] [Mar+97], based on the FV discretization, and NEMO[2] [Mad+91], POM[3] [BM87], and POP2[4] [SH94], based on FD methods. For regional and coastal ocean studies, popular models like Delft3D[5] [Ger+07] and ROMS[6] [Hai+00] employ FD approximations. In recent decades, *unstructured grid models* have gained popularity due to their ability to handle complex geometries and ocean topographies more effectively. Notable examples for regional and coastal ocean simulations include Delft3D FM[7] [Del18], FVCOM[8] [CLB03], and

[1] `https://mitgcm.org`
[2] `https://www.nemo-ocean.eu`
[3] `http://www.ccpo.odu.edu/POMWEB`
[4] `https://github.com/ESCOMP/POP2-CESM`
[5] `https://oss.deltares.nl/web/delft3d`
[6] `https://www.myroms.org`
[7] `https://oss.deltares.nl/web/delft3dfm`
[8] `http://www.smast.umassd.edu/Fisheries/modelerFV/aboutFVCOM.php`

ADCIRC[9] [Wes+92], based on FV schemes. A DG version of the latter has been developed in the past decade and is called DGSWEM[10] [Daw+11]. SCHISM[11] [Zha+16] is based on a hybrid FE–FV method and ICOM [For+04] is based on an FE discretization. SLIM[12] [WDL08] and Thetis[13] [Kär+18] rely on DG discretizations, and the latter employs code generation. For global circulations ICON-O [Kor+22], MPAS-Ocean[14] [Rin+13], and FESOM2[15] [Dan+17], based on FV discretizations, and its predecessor FESOM [DKS04], based on a FE method, are extensively used models. An early pioneer in using DG discretizations for three-dimensional hydrostatic equations is UTBEST3D [Aiz04]. For a comprehensive understanding of structured mesh models, [Kli+18] is a valuable resource, while [Dan13] provides deep insights into unstructured models.

In this chapter, we present the governing equations and a model reformulation. Subsequently, we describe our p-adaptive DG discretization, encompassing an algorithmic redesign and a novel adaptivity indicator. Finally, we conduct a stability analysis for the full two-dimensional SWE model.

2.1 Model equations and boundary conditions

We focus on the two-dimensional SWE which, following the notation in [FN+20], are given by

$$\partial_t \xi + \nabla \cdot \boldsymbol{q} = 0, \tag{2.1}$$

$$\partial_t \boldsymbol{q} + \nabla \cdot \left(\boldsymbol{q}\boldsymbol{q}^T / H \right) + \tau_{\mathrm{bf}} \boldsymbol{q} + \begin{pmatrix} 0 & -f_c \\ f_c & 0 \end{pmatrix} \boldsymbol{q} + gH\nabla\xi = \boldsymbol{F}. \tag{2.2}$$

They are defined on some two-dimensional domain $\Omega \subset \mathbb{R}^2$, and (2.1) represents the conservation of mass and (2.2) the conservation of momentum. By ξ (in m) we denote the surface elevation with respect to some datum, e.g., the mean sea level. The bathymetry with respect to the same datum is represented by h_b (in m) and $H = h_b + \xi$ is the total fluid depth as shown in Figure 2.1. The depth-integrated horizontal velocity field is expressed by $\boldsymbol{q} \equiv (U, V)^T$ (in $\frac{\mathrm{m}^2}{\mathrm{s}}$). We denote the Coriolis coefficient by f_c (in $\frac{1}{\mathrm{s}}$), the gravitational acceleration by g (in $\frac{\mathrm{m}}{\mathrm{s}^2}$) and the bottom friction coefficient by τ_{bf} (in $\frac{1}{\mathrm{s}}$). Some problems use a linear friction law, then $\tau_{\mathrm{bf}} = \mathrm{const.}$ In other cases we employ a standard quadratic friction law, that is $\tau_{\mathrm{bf}} = \frac{C_f\,|\boldsymbol{q}|}{H^2}$ with a constant C_f [Vre94]. Effects of variable atmospheric pressure and tidal potential are expressed through the body force $\mathbf{F}$ (in $\frac{\mathrm{m}^2}{\mathrm{s}^2}$).

[9]https://adcirc.org
[10]https://github.com/UT-CHG/dgswemv2
[11]http://ccrm.vims.edu/schismweb
[12]https://www.slim-ocean.be
[13]https://thetisproject.org
[14]https://mpas-dev.github.io
[15]https://fesom.de

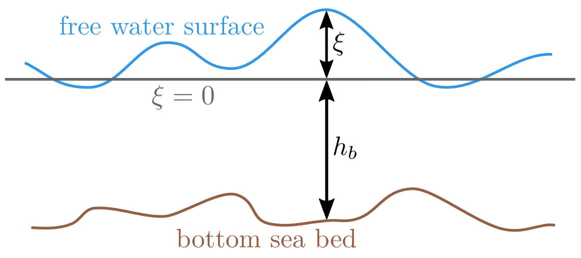


Figure 2.1: Definition of free surface elevation ξ and bathymetry h_b.

This work uses the following types of boundary conditions, where $\hat{\cdot}$ denotes a prescribed value for the corresponding unknown.
Dirichlet boundary: At a Dirichlet boundary, all unknowns are specified

$$\xi(t, \boldsymbol{x}) = \hat{\xi}(t, \boldsymbol{x}), \quad \boldsymbol{q}(t, \boldsymbol{x}) = \hat{\boldsymbol{q}}(t, \boldsymbol{x}).$$

Land boundary: At a land boundary, we assume no normal flow

$$\boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n} = 0.$$

Open-sea boundary: We prescribe the free surface elevation at open sea boundaries

$$\xi(t, \boldsymbol{x}) = \hat{\xi}(t, \boldsymbol{x}).$$

River boundary: For supercritical flow examples, we set the following river (inflow) boundary conditions

$$\xi(t, \boldsymbol{x}) = \hat{\xi}(t, \boldsymbol{x}), \quad q_{\boldsymbol{n}}(t, \boldsymbol{x}) = \hat{q}_{\boldsymbol{n}}(t, \boldsymbol{x}), \quad q_{\boldsymbol{\tau}}(t, \boldsymbol{x}) = \hat{q}_{\boldsymbol{\tau}}(t, \boldsymbol{x}),$$

with the normal and tangential integrated velocities $\hat{q}_{\boldsymbol{n}}(t, \boldsymbol{x})$ and $\hat{q}_{\boldsymbol{\tau}}(t, \boldsymbol{x})$.

Radiation boundary: At the outflow boundary of supercritical flow examples, no unknowns are prescribed.

Lastly, we provide *initial conditions* for the elevation and integrated velocity

$$\xi(0, \boldsymbol{x}) = \xi^0(\boldsymbol{x}), \quad \boldsymbol{q}(0, \boldsymbol{x}) = \boldsymbol{q}^0(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Omega. \tag{2.3}$$

Before we proceed to deriving the quadrature-free reformulation of the shallow water model, we present a compact representation of system (2.1)–(2.2) [AD02; FN+20].
We denote $\boldsymbol{c} := (\xi, U, V)^T$, apply some algebraic manipulations to the gravity term, and arrive at the following compact form

$$\partial_t \boldsymbol{c} + \nabla \cdot \tilde{\boldsymbol{A}}(\boldsymbol{c}) = \tilde{\boldsymbol{r}}(\boldsymbol{c}), \tag{2.4}$$

where

$$\tilde{\boldsymbol{A}}(\boldsymbol{c}) = \begin{pmatrix} U & V \\ \frac{U^2}{H} + \frac{g\xi(H+h_b)}{2} & \frac{UV}{H} \\ \frac{UV}{H} & \frac{V^2}{H} + \frac{g\xi(H+h_b)}{2} \end{pmatrix} \text{ and } \tilde{\boldsymbol{r}}(\boldsymbol{c}) = \begin{pmatrix} 0 \\ -\tau_{\mathrm{bf}}U + f_c V + g\xi\partial_x h_b + F_x \\ -\tau_{\mathrm{bf}}V - f_c U + g\xi\partial_y h_b + F_y \end{pmatrix}. \quad (2.5)$$

2.2 Quadrature-free model reformulation

In a *quadrature-free* DG scheme, all element and face integrals are computed analytically, without quadrature rules. In the upcoming paragraphs, we summarize existing works on quadrature-free discretizations and present our novel approach, building upon the findings from the published article [FN+20].

The primary advantage of employing a quadrature-free method lies in eliminating the innermost loop over quadrature points, providing better code optimization potential. This approach is not new and has already been investigated in [AS98; LA99]. Furthermore, [RM03] utilized a basis consisting of polynomial functions without numerical quadrature, reducing computational costs for boundary integrations. [MRC06] proposed a quadrature-free DG method for solving the level set equation using the efficient BLAS library in the context of interface capturing methods. However, until now, it has been exclusively applied to linear [DRRIA12] or product-type nonlinear operators, for example, to advection terms in Euler equations [Hil+06] or to the kinetic equations [HJ20]. Integrals involved in DG methods discretizing such problems only contain multidimensional polynomials and can be conveniently evaluated analytically.

However, when dealing with the SWE, the advective terms in the momentum equations formulated in the conservative unknowns contain fraction-type nonlinearities, making an analytical evaluation of integrals challenging or even unfeasible. To address this difficulty, [Nai15] adopted the velocity as the primary unknown instead of momentum for the atmospheric SWE, avoiding fraction-type terms. This approach was also used in [RDB18] for an ADER-DG implementation of oceanic SWE. Nevertheless, the conservative form of the SWE offers significant advantages, particularly in conserving essential physical properties of the system, such as momentum. Recently, in [LZ21], a quadrature-free discretization of the SWE, including wetting and drying, was proposed. In this approach, the underlying system remains unaltered, employing a nodal basis along with evaluations conducted at interpolation nodes, thereby circumventing the necessity for traditional quadrature computations. Therefore, analytical evaluations and thus precomputing all element and edge integrals are not possible, and furthermore, the system is not equivalent to its quadrature-based counterpart.

In our approach, we modify the PDE system such that it avoids fraction-type nonlinearities to enable the quadrature-free evaluation of element and edge integrals arising from the DG discretization of the SWE.

For this purpose, we introduce the depth-averaged velocity $\boldsymbol{u} = (u, v)^T$ (in $\frac{\mathrm{m}}{\mathrm{s}}$) related to $\boldsymbol{q}$ by $\boldsymbol{q} = \boldsymbol{u}H$ and replace system (2.4) by the *quadrature-free reformulation*

$$\partial_t \boldsymbol{c} + \nabla \cdot \boldsymbol{A}(\boldsymbol{c}, \boldsymbol{u}) = \boldsymbol{r}(\boldsymbol{c}, \boldsymbol{u}), \tag{2.6}$$

$$\boldsymbol{u}H = \boldsymbol{q}, \tag{2.7}$$

with

$$\boldsymbol{A}(\boldsymbol{c}, \boldsymbol{u}) = \begin{pmatrix} U & V \\ Uu + \frac{g\xi(H+h_b)}{2} & Uv \\ Vu & Vv + \frac{g\xi(H+h_b)}{2} \end{pmatrix} \text{ and } \boldsymbol{r}(\boldsymbol{c}, \boldsymbol{u}) = \begin{pmatrix} 0 \\ -\tau_{\mathrm{bf}} u + f_c V + g\xi \partial_x h_b + F_x \\ -\tau_{\mathrm{bf}} v - f_c U + g\xi \partial_y h_b + F_y \end{pmatrix}. \tag{2.8}$$

In case of the linear friction law we then use $\tau_{\mathrm{bf}} = \mathrm{const} \cdot H$ and in case of the quadratic one, we use $\tau_{\mathrm{bf}} = C_f\, |\boldsymbol{u}|$, both in $\frac{\mathrm{m}}{\mathrm{s}}$.

Note that $\boldsymbol{A}$ and $\boldsymbol{r}$ in (2.8) have no fraction-type nonlinearities as opposed to $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{r}}$ in (2.5). In Section 4.2, we evaluate the above reformulation together with adaptations in the numerical flux detailed in the following section.

The proposed approach exhibits the potential for generalization to specific classes of nonlinear operators, thereby enabling the utilization of the advantages offered by the quadrature-free methodology across a broader range of nonlinear problems.

2.3 Discretization of the model

In this section, we outline the DG discretization of system (2.6)–(2.8), building upon the previously published work [FN+20] and the preprint [FN+23a]. Subsequently, we elucidate the employed limiting and temporal discretization scheme, as detailed in [FNA22]. The p-adaptivity strategy, including a redesign of the DG discretization to optimize hardware utilization, is then presented based on [FN+23a]. Finally, we present our novel parameter-free adaptivity indicator, as proposed in [FNA22].

2.3.1 Spatial discretization by a quadrature-free DG method

The DG discretization of the quadrature-based two-dimensional SWE was originally proposed in [AD02]. We adhere to their methodology and introduce some modifications to the numerical flux to enable a quadrature-free evaluation of the integrals.

Let $\{\mathcal{T}_\Delta\}_{\Delta>0}$ represent a family of triangulations of the domain Ω, and let Ω_e, $e \in \{0, \dots, E\} = I_e$ be elements of $\mathcal{T}_\Delta$. To derive the local *variational formulation* of system (2.6)–(2.8) on an element Ω_e, we multiply the system with sufficiently smooth test functions $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, followed by integrating over Ω_e and integrating by parts. For the subsequent equations, we introduce the notation $(\cdot,\cdot)_{\Omega_e}$ and $\langle\cdot,\cdot\rangle_{\partial\Omega_e}$ to represent the L^2-scalar products on elements and edges, respectively. Furthermore, we denote by $\boldsymbol{n}_e$ an exterior unit normal to $\partial\Omega_e$ and get to

$$(\partial_t \boldsymbol{c}, \boldsymbol{\phi})_{\Omega_e} - (\boldsymbol{A}(\boldsymbol{c},\boldsymbol{u}), \nabla\boldsymbol{\phi})_{\Omega_e} + \langle \boldsymbol{A}(\boldsymbol{c},\boldsymbol{u}) \cdot \boldsymbol{n}_e, \boldsymbol{\phi}\rangle_{\partial\Omega_e} = (\boldsymbol{r}(\boldsymbol{c},\boldsymbol{u}), \boldsymbol{\phi})_{\Omega_e}\,, \tag{2.9}$$

$$(\boldsymbol{u}H, \boldsymbol{\psi})_{\Omega_e} = (\boldsymbol{q}, \boldsymbol{\psi})_{\Omega_e}\,. \tag{2.10}$$

Let $\mathbb{P}^p(\Omega_e)$ denote the polynomial space of order p defined on Ω_e. The initial conditions (2.3) are projected into the corresponding discrete space. To obtain the *semi-discrete formulation* from (2.9)–(2.10), we substitute $\boldsymbol{c}$ and $\boldsymbol{u}$ with the discrete solution $\boldsymbol{c}_\Delta$ and $\boldsymbol{u}_\Delta$, respectively, and use test functions $\boldsymbol{\phi}_\Delta \in \mathbb{P}^p(\Omega_e)^3$, and $\boldsymbol{\psi}_\Delta \in \mathbb{P}^p(\Omega_e)^2$. This leads to

$$(\partial_t \boldsymbol{c}_\Delta, \boldsymbol{\phi}_\Delta)_{\Omega_e} - (\boldsymbol{A}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta), \nabla\boldsymbol{\phi}_\Delta)_{\Omega_e} + \langle \widehat{\boldsymbol{A}}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta,\boldsymbol{c}_\Delta^+,\boldsymbol{u}_\Delta^+,\boldsymbol{n}_e), \boldsymbol{\phi}_\Delta\rangle_{\partial\Omega_e} = (\boldsymbol{r}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta), \boldsymbol{\phi}_\Delta)_{\Omega_e}\,, \tag{2.11}$$

$$(\boldsymbol{u}_\Delta H_\Delta, \boldsymbol{\psi}_\Delta)_{\Omega_e} = (\boldsymbol{q}_\Delta, \boldsymbol{\psi}_\Delta)_{\Omega_e}\,. \tag{2.12}$$

The element-local systems (2.12) are solved by an LU-factorization per substep of the time stepping scheme (cf. Section 2.3.2).
The edge flux $\boldsymbol{A}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta)\cdot\boldsymbol{n}_e$ is approximated on $\partial\Omega_e$ by a numerical flux $\widehat{\boldsymbol{A}}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta,\boldsymbol{c}_\Delta^+,\boldsymbol{u}_\Delta^+,\boldsymbol{n}_e)$. This numerical flux depends on the discontinuous values of the solution within the element Ω_e, namely $\boldsymbol{c}_\Delta, \boldsymbol{u}_\Delta$, as well as those of its edge-neighbor, $\boldsymbol{c}_\Delta^+, \boldsymbol{u}_\Delta^+$. When dealing with exterior domain boundaries, the prescribed boundary conditions for elevation or velocity are incorporated into the flux computation. The components of the flux $\widehat{\boldsymbol{A}}$ quantify the amount of each conserved unknown transported across the boundary $\partial\Omega_e$ in the direction of $\boldsymbol{n_e}$. For a locally conservative method, this flux must equal the negative flux of the neighboring element that shares the edge. The presence of discontinuities introduces a *Riemann problem* [LeV92; Tor09], for which various Riemann solvers are available, each offering distinct characteristics such as amount of numerical diffusion, computational cost, and stability properties. The simplest choice that guarantees the numerical stability of the method is the *Lax–Friedrichs flux* [LeV92], which is employed in all numerical simulations presented in Chapter 4 unless specified otherwise. Other choices of Riemann solvers include the Roe solver [Roe81], the FORCE solver [THD09], or the HLLC solver [TSS94].

The Lax–Friedrichs flux is defined as

$$\widehat{\boldsymbol{A}}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta,\boldsymbol{c}_\Delta^+,\boldsymbol{u}_\Delta^+,\boldsymbol{n}_e) = \frac{1}{2}\Big(\boldsymbol{A}(\boldsymbol{c}_\Delta,\boldsymbol{u}_\Delta) + \boldsymbol{A}(\boldsymbol{c}_\Delta^+,\boldsymbol{u}_\Delta^+)\Big)\cdot\boldsymbol{n}_e + \frac{\lambda}{2}(\boldsymbol{c}_\Delta - \boldsymbol{c}_\Delta^+), \tag{2.13}$$

where λ represents the maximum absolute eigenvalue of the matrix $\frac{\partial}{\partial \boldsymbol{c}}(\tilde{\boldsymbol{A}}(\boldsymbol{c},\boldsymbol{u})\cdot\boldsymbol{n})$. For the SWE, this eigenvalue is given by $\lambda(H,\boldsymbol{u}) = |\boldsymbol{u}\cdot\boldsymbol{n}| + \sqrt{gH}$. In practice, quadrature-based schemes typically use $\lambda(\hat{H}, \hat{\boldsymbol{u}})$, where $\hat{H}$ and $\hat{\boldsymbol{u}}$ are evaluated at each quadrature point using the Roe–Pike averaging method [RP85] as described in [Haj+18]. The nonlinearity of the expression for λ introduces challenges in its quadrature-free evaluation, making the computation rather intricate.

We propose an alternative approach motivated by the following consideration: in the Lax–Friedrichs solver presented in (2.13), the parameter λ is a coefficient for the penalty term. Typically, increasing the value of λ does not negatively affect the stability of the scheme, although it may introduce higher numerical diffusion. To enable a quadrature-free formulation, we use a single value of $\lambda_{|f}$ for each edge f of Ω_e, thus permitting an analytical evaluation of integrals involving $\widehat{\boldsymbol{A}}$ defined in (2.13). For this purpose, let $\boldsymbol{x}_f$ denote the midpoint of edge f, and we employ the approximation

$$\lambda_{|f} := \max_{\Omega_e : \boldsymbol{x}_f \in \partial\Omega_e} \left| \boldsymbol{u}_{\Delta|\Omega_e}(\boldsymbol{x}_f) \cdot \boldsymbol{n} \right| + \max_{\Omega_e : \boldsymbol{x}_f \in \partial\Omega_e} \sqrt{g H_{\Delta|\Omega_e}(\boldsymbol{x}_f)}. \tag{2.14}$$

Owing to the excessive numerical diffusion sometimes encountered with the Lax–Friedrichs flux, we opt for the FORCE flux [THD09] in certain cases while still using λ as described previously. The FORCE flux can be constructed as the arithmetic mean between the Lax–Friedrichs flux and the two-step version of the Lax–Wendroff flux

$$\widehat{\boldsymbol{A}}^{F} = \frac{1}{2}\left(\widehat{\boldsymbol{A}}^{LF} + \widehat{\boldsymbol{A}}^{LW}\right), \tag{2.15}$$

where the latter is defined as $\widehat{\boldsymbol{A}}^{LW} = \boldsymbol{A}\left(\boldsymbol{Q}^{LW}\right)$ with

$$\boldsymbol{Q}^{LW}(\boldsymbol{c}_\Delta, \boldsymbol{u}_\Delta, \boldsymbol{c}_\Delta^+, \boldsymbol{u}_\Delta^+, \boldsymbol{n}_e) = \frac{\lambda}{2}\left(\boldsymbol{A}(\boldsymbol{c}_\Delta, \boldsymbol{u}_\Delta) - \boldsymbol{A}(\boldsymbol{c}_\Delta^+, \boldsymbol{u}_\Delta^+)\right) \cdot \boldsymbol{n}_e + \frac{1}{2}(\boldsymbol{c}_\Delta + \boldsymbol{c}_\Delta^+).$$

Since the FORCE flux is more computationally expensive, we reserve its usage for scenarios where the Lax–Friedrichs flux introduces too much diffusion.

Notably, with either the Lax–Friedrichs or the FORCE flux and λ approximated as in (2.14), our semi-discrete formulation (2.11)–(2.12) only contains nonlinearities in product form, thus all edge integrals are well suited for an analytical integration using a quadrature-free approach. We present a comprehensive evaluation of the reformulated numerical scheme in Section 4.2.

In our implementation, we utilize a hierarchical modal basis, as this choice aligns optimally with the requirements of a p-adaptive scheme. Given $\varphi_{ei}(\boldsymbol{x})$, $i = 1, \ldots, P(p)$, a basis of $\mathbb{P}^p(\Omega_e)$, $\boldsymbol{c}_\Delta$ and $\boldsymbol{u}_\Delta$ can be represented as

$$\boldsymbol{c}_\Delta(\boldsymbol{x})_{|\Omega_e} = (\xi_\Delta, U_\Delta, V_\Delta)^T(\boldsymbol{x}) = \sum_{j=1}^{3} \sum_{i=1}^{P(p)} c_{ei}^j \varphi_{ei}(\boldsymbol{x})\, \boldsymbol{e}_j, \tag{2.16}$$

$$\boldsymbol{u}_\Delta(\boldsymbol{x})_{|\Omega_e} = (u_\Delta, v_\Delta)^T(\boldsymbol{x}) = \sum_{j=1}^{2} \sum_{i=1}^{P(p)} u_{ei}^j \varphi_{ei}(\boldsymbol{x})\, \boldsymbol{e}_j, \tag{2.17}$$

where c_{ei}^j and u_{ei}^j are the basis coefficients associated with the unknowns and $\boldsymbol{e}_j$ represents the j-th unit vector in $\mathbb{R}^3$ in (2.16) or $\mathbb{R}^2$ in (2.17). A basis of $\mathbb{P}^p(\Omega_e)$ is defined through a mapping from the corresponding reference basis on $\hat{\Omega}$ by

$$\varphi_{ei}(\boldsymbol{x}) = \begin{cases} \hat{\varphi}_i\left(\boldsymbol{F}_e^{-1}(\boldsymbol{x})\right), & \boldsymbol{x} \in \Omega_e, \\ 0, & \text{otherwise}, \end{cases} \qquad i \in \{1, \ldots, P(p)\}.$$

Our implementation employs triangles. The mapping $F_e : \hat{\Omega} \to \Omega_e,\ \hat{\boldsymbol{x}} \to \boldsymbol{x} := \boldsymbol{B}_e\hat{\boldsymbol{x}} + \boldsymbol{a}_{e1}$ is an affine-linear transformation (see Figure 2.2) from the reference triangle onto triangle Ω_e with

$$\boldsymbol{B}_e = \begin{bmatrix} B_{1,1}^e & B_{1,2}^e \\ B_{2,1}^e & B_{2,2}^e \end{bmatrix} := \begin{bmatrix} \boldsymbol{a}_{e2} - \boldsymbol{a}_{e1} & \boldsymbol{a}_{e3} - \boldsymbol{a}_{e1} \end{bmatrix}$$

and $\boldsymbol{a}_{e1}, \boldsymbol{a}_{e2}, \boldsymbol{a}_{e3}$ denoting the vertex coordinates of Ω_e.

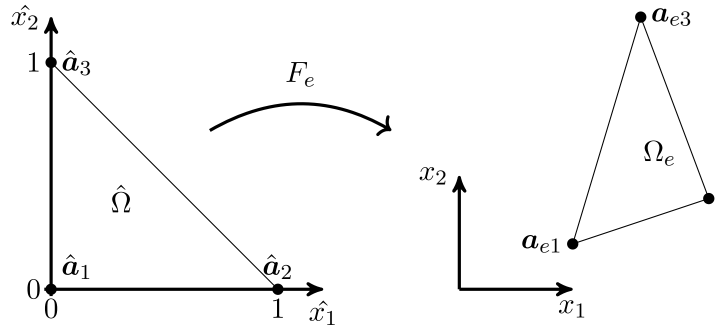


Figure 2.2: Affine-linear mapping F_e from the reference triangle $\hat{\Omega} = \{\hat{\boldsymbol{a}}_1, \hat{\boldsymbol{a}}_2, \hat{\boldsymbol{a}}_2\} = \left\{[0,0]^T, [1,0]^T, [0,1]^T\right\}$ to the physical triangle $\Omega_e = \{\boldsymbol{a}_{e,1}, \boldsymbol{a}_{e,2}, \boldsymbol{a}_{e,3}\}$ (from [FN+20]).

The number of basis functions $P(p)$ depends on the selected polynomial approximation space. In $\mathbb{R}^2$, it has the following values: $P(0)=1$, $P(1)=3$, $P(2)=6$, and $P(3)=10$. The reference basis functions utilized in our implementation are orthonormal with respect to the L^2-scalar product on $\hat{\Omega}$ and given by

$$\begin{aligned}
&\hat{\varphi}_1(\hat{\boldsymbol{x}}) = \sqrt{2}, \ \} \,\mathbb{P}^0(\hat{\Omega}) \\
&\hat{\varphi}_2(\hat{\boldsymbol{x}}) = 2 - 6\hat{x}, \\
&\hat{\varphi}_3(\hat{\boldsymbol{x}}) = \sqrt{12}(1 - \hat{x} - 2\hat{y}), \quad \Big\} \,\mathbb{P}^1(\hat{\Omega}) \\
&\hat{\varphi}_4(\hat{\boldsymbol{x}}) = \sqrt{6}\left(1 - 8\hat{x} + 10\hat{x}^2\right), \\
&\hat{\varphi}_5(\hat{\boldsymbol{x}}) = \sqrt{3}\left(-1 - 4\hat{x} + 5\hat{x}^2 + 12\hat{y} - 15\hat{y}^2\right), \\
&\hat{\varphi}_6(\hat{\boldsymbol{x}}) = \sqrt{45}\left(1 - 4\hat{x} + 3\hat{x}^2 - 4\hat{y} + 8\hat{x}\hat{y} + 3\hat{y}^2\right), \quad \Big\} \,\mathbb{P}^2(\hat{\Omega}) \\
&\hat{\varphi}_7(\hat{\boldsymbol{x}}) = \sqrt{8}\left(-1 + 15\hat{x} - 45\hat{x}^2 + 35\hat{x}^3\right), \\
&\hat{\varphi}_8(\hat{\boldsymbol{x}}) = \sqrt{24}\left(-1 + 13\hat{x} - 33\hat{x}^2 + 21\hat{x}^3 + 2\hat{y} - 24\hat{x}\hat{y} + 42\hat{x}^2\hat{y}\right), \\
&\hat{\varphi}_9(\hat{\boldsymbol{x}}) = \sqrt{40}\left(-1 + 9\hat{x} - 15\hat{x}^2 + 7\hat{x}^3 + 6\hat{y} - 48\hat{x}\hat{y} + 42\hat{x}^2\hat{y} - 6\hat{y}^2 + 42\hat{x}\hat{y}^2\right), \\
&\hat{\varphi}_{10}(\hat{\boldsymbol{x}}) = \sqrt{56}\left(-1 + 3\hat{x} - 3\hat{x}^2 + \hat{x}^3 + 12\hat{y} - 24\hat{x}\hat{y} + 12\hat{x}^2\hat{y} - 30\hat{y}^2 + 30\hat{x}\hat{y}^2 + 20\hat{y}^3\right). \quad \Big\} \,\mathbb{P}^3(\hat{\Omega})
\end{aligned} \tag{2.18}$$

Since the mapping from $\hat{\Omega}$ to Ω_e is affine-linear, the basis $\varphi_{ei}, i \in \{1, \ldots, P(p)\}$ retains this orthonormal property.

2.3.2 Slope limiting and temporal discretization

When employing FV and DG methods, the technique of *slope limiting* is widely utilized to enforce discrete maximum principles [BJ89; ZS11]. Slope limiters adjust the gradients and, if applicable, higher-order derivatives of polynomial approximations on mesh cells to preserve local bounds based on cell averages in neighboring elements [LeV92; KH23]. These limiters serve as post-processing tools without affecting local mass conservation. Within the context of DG approximations in space and explicit total variational diminishing time stepping schemes, they exploit the fact that the lowest-order (piecewise constant) part of a DG solution ensures the preservation of solution monotonicity and produces no spurious extrema for linear problems. As a result, this approach ensures admissible minimum and maximum values for the higher-order solution [Reu20]. *Vertex-based* slope limiters represent a specific class of limiters that utilize vertices of the computational mesh as control points [Kuz10; Aiz11]. They can be extended to operate on higher-order DG solutions in a hierarchical manner [Kuz13; Kuz14].

In our implementation, we apply a vertex-based slope limiter following [Kuz10; Aiz11] to piecewise linear and higher-order approximations where appropriate. For simplicity, we formulate our methodology in the remainder of this section for a generic scalar function $w(\boldsymbol{x})$ defined on Ω and its discretized counterpart $w_\Delta(\boldsymbol{x})$. For orthonormal basis functions used in our work, the local representation of the limiting operator $\Pi : \mathbb{P}^p(\Omega_e) \to \mathbb{P}^p(\Omega_e)$ is given by

$$\Pi\left(w_\Delta(\boldsymbol{x})_{|\Omega_e}\right) = w_{e1}\varphi_{e1} + \alpha_e \sum_{i=2}^{3} w_{ei}\,\varphi_{ei} + \delta_{\alpha_e,1} \sum_{i=4}^{P(p)} w_{ei}\,\varphi_{ei}, \tag{2.19}$$

where the limiting factor $\alpha_e \in [0,1]$ is computed as follows:

$$\alpha_e = \min_{i\in\{1,2,3\}} \begin{cases} \min\left(1, \frac{w_i^{\max} - w_e^0}{w_e(\boldsymbol{a}_{ei}) - w_e^0}\right), & \text{if } w_e(\boldsymbol{a}_{ei}) - w_e^0 > \varepsilon, \\ \min\left(1, \frac{w_i^{\min} - w_e^0}{w_e(\boldsymbol{a}_{ei}) - w_e^0}\right), & \text{if } w_e(\boldsymbol{a}_{ei}) - w_e^0 < -\varepsilon, \\ 1, & \text{otherwise}, \end{cases} \tag{2.20}$$

where we choose $\varepsilon = 10^{-5}$. Here, $w_i^{\max}$ and $w_i^{\min}$ represent the solution bounds given by the maximum and minimum of piecewise constant solutions w_e^0 on all elements sharing vertex $\boldsymbol{a}_{ei}$. Note that for an orthonormal basis, one has

$$w_e^0 = \frac{1}{|\Omega_e|} \int_{\Omega_e} w_\Delta(\boldsymbol{x})_{|\Omega_e} d\boldsymbol{x}.$$

In cases where (2.20) results in $\alpha_e < 1$, all degrees of freedom corresponding to superlinear (quadratic and higher-order) basis functions in (2.19) are set to zero, and, therefore we multiply the corresponding terms with the Kronecker delta $\delta_{\alpha_e,1}$, which is 1 if $\alpha_e = 1$ and 0 otherwise. In such instances, this limiting does not guarantee the exact preservation of bounds $w_i^{\max}$ and $w_i^{\min}$, but it effectively minimizes oscillations in most practical scenarios. In the test cases presented in Chapter 4, we calculate the limiting factor α_e from the free surface elevation and also apply it to the depth-integrated velocity.

In the context of solving time-dependent PDEs, *Runge–Kutta (RK) methods* present a diverse set of explicit and implicit schemes of varying order. However, traditional schemes may encounter spurious oscillations when applied to problems with discontinuities and steep gradients. To address this issue, *strong stability preserving* (SSP) RK shemes were introduced in [SO88; CLS89; CHS90] to preserve the *total variation diminishing* (TVD) property of the spatial semi-discretization [GS98; GST01; GKS11]. Recently, building upon SSP RK schemes, advanced time discretization schemes have been developed in [EG22; Kuz+22] to overcome the fourth-order barrier of the explicit SSP RK schemes and allow for larger time step sizes.

In our implementation, the temporal discretization of system (2.11)–(2.12) is accomplished using an SSP RK method. Following the presentation in [Reu+16], let $0 = t_1 < t_2 < \cdots < t_{\text{end}}$ be a possibly non-equidistant decomposition of the time interval and $\Delta_n t := t_{n+1} - t_n$ represent the size of the n-th time step. The update scheme of the s-stage SSP RK method with limiting operator Π is then given by

$$\begin{aligned}
\boldsymbol{c}_\triangle^{(0)} &:= \boldsymbol{c}_\triangle(t_n, \cdot), \\
\boldsymbol{c}_\triangle^{(i)} &:= \Pi\left(\omega_i \boldsymbol{c}_\triangle^{(0)} + (1-\omega_i)\left\{\boldsymbol{c}_\triangle^{(i-1)} + \Delta_n t\, \boldsymbol{L}(\boldsymbol{c}_\triangle^{(i-1)}, t_n + \delta_i \Delta_n t)\right\}\right), \ i = 1, \dots, s, \\
\boldsymbol{c}_\triangle(t_{n+1}, \cdot) &:= \boldsymbol{c}_\triangle^{(s)},
\end{aligned} \tag{2.21}$$

where $\boldsymbol{L}$ denotes the spatial discretization operator specified by (2.11)–(2.12). For the test cases presented in Chapter 4, we employ a two-stage SSP RK method, known as Heun's scheme, with coefficients $\omega_1 = 0$, $\omega_2 = 1/2$, $\delta_1 = 0$, and $\delta_2 = 1$ (cf. Equation (2.4) in [GS98]).

2.3.3 p-adaptivity and separation approach

Considering the complex flow structures observed in coastal regions and the time-dependent nature of solutions, achieving accurate resolution of flow features using a fixed computational grid and discretization order is often unpredictable or prohibitively expensive [Kub+09]. Therefore, adaptive mesh refinement (h-adaptivity) or local adjustment of the polynomial order of the discretization (p-adaptivity) are commonly employed to ensure sufficient spatial resolution for accurately capturing the evolving flow field. The previous work in [KWD06] demonstrated the benefits of using p-adaptivity instead of h-adaptivity, in particular for problems exhibiting smooth solutions in coastal regions when employing DG methods. Furthermore, for DG discretizations which rely on modal hierarchical bases, p-adaptive schemes are particularly attractive due to their straightforward implementation, in contrast to h- and hp-adaptive schemes. We focus on p-adaptivity, which involves adjusting the local approximation order depending on the evaluation of an adaptivity indicator, introduced in Section 2.3.4.

As our approach employs the quadrature-free formulation as introduced in Section 2.2, it is highly suitable for p-adaptive schemes. The analytic evaluation of all element and edge integrals completely avoids the main overhead of varying-order approximation spaces, i.e., the need for the most accurate and, thus, the most expensive quadrature rule or maintaining several quadrature rules of different orders.

Adaptive numerical schemes for time-dependent problems face a critical challenge when combined with massively parallel computing based on distributed memory parallelization. The primary concern revolves around achieving a well-balanced computational load throughout the simulation. Numerous load-balancing strategies have been proposed in the past, such as those in [Bis+00; HD00; TDF06; BB17]. However, these strategies introduce higher code complexity and additional computational overhead. Consequently, the popularity of adaptive numerical schemes has waned in some user communities like in numerical weather prediction or ocean modeling over the last decade. It is worth noting that recently some highly efficient frameworks which effectively manage scalability despite employing adaptivity, like presented in [OBB23], have been developed. To address this issue and rejuvenate the interest in adaptive schemes, we propose a novel approach focusing on the load-balancing challenge in the context of a p-adaptive DG method. Our strategy involves separating the numerical scheme and the solution algorithm into two parts: a lower-order non-adaptive component (fixed computation) and a higher-order adaptive component (correction computation). By offloading the correction computation to separate hardware, we can ensure that the load balance of the base computation remains unaffected. In addition, this encapsulation of the adaptive part of the numerical method in a separate kernel offers a meaningful way to map time-dependent adaptive FE schemes to task-based programming models particularly well-suited for heterogeneous hardware architectures, as illustrated in [Bos+13; GG+19]. In the following paragraphs, we delve into the algorithmic details of our novel *separation approach*.

When combined with hierarchical bases, the quadrature-free formulation utilized in our solver separates the discrete equations associated with the lower polynomial orders from the higher-order ones. The underlying concept involves the independent evaluation of updates for non-adaptive degrees of freedom in the DG approximation, such as the piecewise constant or piecewise linear components, separate from the adaptive higher-order part of the solution. This decoupling is facilitated by the product-type nonlinearities present in the quadrature-free formulation. Consequently, the p-adaptive scheme for such a discretization boils down to adding and removing terms without impacting the rest of the DG approximation. As depicted in Figure 2.3, the non-adaptive lower-order DG solution, here represented as piecewise constant (left) or piecewise linear (right), is computed for all elements. Subsequently, an adaptive correction involving piecewise linear and higher-order components (left) or piecewise quadratic and higher-order components (right) is selectively applied only where necessary. The setups evaluated in Section 4.4 include a constant non-adaptive part with a linear correction and a linear non-adaptive part with a quadratic correction. This approach is naturally extendable to any higher-order DG discretization.

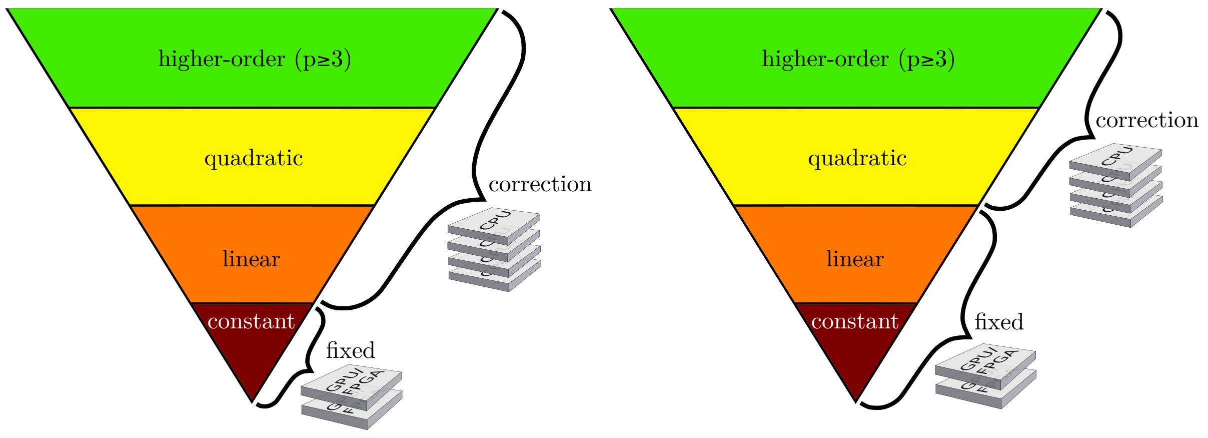


Figure 2.3: Schematic illustration of the separation approach: The solution for the non-adaptive components (left: piecewise constant, right: piecewise linear) is computed for all elements on dedicated hardware, e.g a GPU. An adaptive correction (left: $p \geq 1$, right: $p \geq 2$) is then selectively applied to specific elements. The latter computation can be offloaded to different hardware, e.g., a CPU (adapted from [FN+23a]).

We illustrate our separation approach for the mass conservation equation. All other integrals are separated in a similar fashion. First, we obtain the algebraic representation of element integrals by substituting the basis representations (2.16) and (2.17) into (2.11) and testing the first equation with $\boldsymbol{\phi}_\triangle = \varphi_{eq}\boldsymbol{e}_1$. For $q \in 1, \dots, P(p)$ the resulting expression is

$$(\boldsymbol{A}(\boldsymbol{c}_\triangle, \boldsymbol{u}_\triangle), \nabla(\varphi_{eq}\boldsymbol{e}_1))_{\Omega_e} = \sum_{i=1}^{P(p)} \left[c_{ei}^2 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial x} \varphi_{ei} \mathrm{d}\boldsymbol{x} + c_{ei}^3 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial y} \varphi_{ei} \mathrm{d}\boldsymbol{x} \right]. \tag{2.22}$$

We assume that the fixed non-adaptive calculation up to order b is performed for all elements, while the correction for order $b+1$ is only applied to selected elements. Thus, the fixed non-adaptive computation can be expressed as

$$(\boldsymbol{A}(\boldsymbol{c}_\triangle, \boldsymbol{u}_\triangle), \nabla(\varphi_{eq}\boldsymbol{e}_1))_{\Omega_e}^{\text{fixed}} = \sum_{i=1}^{P(b)} \left[c_{ei}^2 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial x} \varphi_{ei} \mathrm{d}\boldsymbol{x} + c_{ei}^3 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial y} \varphi_{ei} \mathrm{d}\boldsymbol{x} \right]$$

for $q \in 1, \dots, P(b)$.
The correction term is then divided into two cases. For $q \in 1, \dots, P(b)$, it is given by

$$(\boldsymbol{A}(\boldsymbol{c}_\triangle, \boldsymbol{u}_\triangle), \nabla(\varphi_{eq}\boldsymbol{e}_1))_{\Omega_e}^{\text{correction}} = \sum_{i=P(b)+1}^{P(p)} \left[c_{ei}^2 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial x} \varphi_{ei} \mathrm{d}\boldsymbol{x} + c_{ei}^3 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial y} \varphi_{ei} \mathrm{d}\boldsymbol{x} \right],$$

and for $q \in P(b)+1, \dots, P(p)$, the correction term is given by

$$(\boldsymbol{A}(\boldsymbol{c}_\triangle, \boldsymbol{u}_\triangle), \nabla(\varphi_{eq}\boldsymbol{e}_1))_{\Omega_e}^{\text{correction}} = \sum_{i=1}^{P(p)} \left[c_{ei}^2 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial x} \varphi_{ei} \mathrm{d}\boldsymbol{x} + c_{ei}^3 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial y} \varphi_{ei} \mathrm{d}\boldsymbol{x} \right].$$

The edge integrals are analogously separated while, of course, considering the local approximation order of the current element. Notably, when calculating the coefficient λ, i.e., (2.14), in front of the penalty term for the Lax–Friedrichs flux, the evaluation exclusively employs the constant part of the solution for the velocity and free surface elevation fields, irrespective of the approximation order. Similarly, the right-hand side computation of the nonlinear bottom friction relies on the piecewise constant solution component rather than the full-order approximation for determining the velocity magnitude. This particular approach has been seamlessly integrated into our implementation since it demonstrated no deterioration in the quality of the solution and allows us to circumvent the need for special treatment during the application of corrections.

The solution $\boldsymbol{u}_\triangle(t, \boldsymbol{x})_{|\Omega_e}$ of (2.12) involves solving an element-local linear system per RK substep. This system is different for different approximation orders, i.e., the higher-order degrees of freedom can actually affect the lower-order ones. However, in all benchmarks we investigated, excluding higher-order contributions from lower-order computations showed no negative effects, while maintaining an equal solution quality. Thus, for computing $\boldsymbol{u}_\triangle$, we employ an LU-factorization to now generally solve the system given by

$$(H_{i,j})_{i,j\in\{1,\dots,P(p)\}} \left(u_j^l\right)_{j\in\{1,\dots,P(p)\}} = \left(c_i^{l+1}\right)_{i\in\{1,\dots,P(p)\}}, \qquad l = 1, 2,$$

where the matrices and vectors are defined as follows:

$$\begin{aligned}
(H_{i,j})_{i,j\in\{1,\dots,P(p)\}} &:= \int_{\Omega_e} \left(\sum_{n=1}^{P(p,i,j)} c_{en}^1 \varphi_{en} + h_b \right) \varphi_{ej}\varphi_{ei}, \\
\left(u_j^l\right)_{j\in\{1,\dots,P(p)\}} &:= u_{ej}^l, \\
\left(c_i^{l+1}\right)_{i\in\{1,\dots,P(p)\}} &:= \int_{\Omega_e} \sum_{n=1}^{P(p)} c_{en}^{l+1} \varphi_{en}\varphi_{ei}, \qquad l = 1, 2,
\end{aligned}$$

and the function $P(p, i, j)$ denotes the number of terms in the summation:

$$P(p,i,j) = \begin{cases} 1, & \text{if } i = j = 1, \\ 3, & \text{if } p <= 1 \wedge !(i = j = 1), \\ 6, & \text{if } p <= 2 \wedge !(i <= 3 \wedge j <= 3), \\ 10, & \text{if } p <= 3 \wedge !(i <= 6 \wedge j <= 6). \end{cases}$$

The orthonormality property ensures that the lower-order terms in $\left(c_i^{l+1}\right)_{i\in\{1,\dots,P(p)\}}$ remain unaffected by the higher-order ones. Consequently, there is no need to re-assemble the lower-order system when applying the higher-order correction.

For a comprehensive evaluation of the performance of the separation approach, we present a detailed analysis in Section 4.4.

2.3.4 Adaptivity indicators

Adaptive discretizations need indicators to guide the adaptivity process. In this section, we provide an overview of existing approaches and introduce our novel indicator, which stands out by not requiring any problem-dependent parameters as user input. Additionally, we present two other indicators for comparison purposes.

The literature presents a diverse range of error and smoothness indicators for DG schemes, detailed extensively in a comprehensive survey in [Nad+19]. These indicators are broadly categorized based on the information they utilize to assess the local discretization error or solution regularity. Feature-based indicators are often derived from physical or theoretical problem attributes. One example is a method for vortex detection [Mit01]. Discretization-error- and residual-error-based indicators identify refinement regions as those with high errors of numerical solutions. One straightforward and computationally efficient scheme relies on measuring the local spectral decay of the DG solution [Esk11; TBR13]. It compares the absolute or relative magnitudes of degrees of freedom associated with different polynomial orders. Another popular indicator is the non-conformity error indicator, which considers the jump in the numerical solution as a measure of the error as illustrated in [KF03; RFS03]. Lastly, goal-oriented indicators like the one introduced in [HH02] evaluate the impact of numerical errors, utilizing weighted residuals from dual problems for effective error assessment. Numerous authors showcased the superiority of discretization-error-based methods for stationary problems, yet their high memory usage and computational cost limit their application.

A discretization-error based indicator, already applied successfully to the SWE, estimates local solution gradients [Kub+09; Mic+11]. The authors in [Mic+11] propose to combine a slope limiter with a p-adaptive DG method. However, the adaptivity and the slope-limiting procedures were not integrated in that study, i.e., they did not use information from each other. Furthermore, to the best of our knowledge, all previously employed indicators for the SWE depend on problem-specific parameter choices.

Our indicator employs techniques introduced in [Kri+04] for non-conformity estimators (KXRCF indicator), which quantify the absolute or relative size of discontinuities between the element-local solutions, primarily by integrating solution jumps across inter-element boundaries. However, specific requirements in our application necessitated the development of a new indicator for two main reasons. Firstly, computational efficiency is needed in handling highly dynamic simulation scenarios like tidal waves or tsunamis in the SWE. Additionally, the ability to detect large local errors using the lowest order approximation space and support for piecewise constant polynomial spaces is crucial, which cannot be achieved with methods like spectral decay.

The proposed *adaptivity indicator* satisfies the aforementioned requirements without relying on problem-specific parameters or sensitivity measures. It seamlessly integrates into the vertex-based slope limiter [Kuz10; Aiz11; Aiz+17; Haj+18; HKA19], effectively reducing the overall computational cost of the scheme.

To avoid oscillations in the approximation order within our numerical simulations, once the order is increased, we consistently require a minimum of ten time steps before allowing it to be decreased again for all adaptivity sensors. When reducing the approximation order of an element, all coefficients associated with higher-order degrees of freedom are reset to zero. Furthermore, an element's approximation order can only be in- and decreased by one at a time.

2.3.4.1 Novel parameter-free indicator

The primary objective of the novel indicator, denoted as JRL (Jump-Reconstruction-Limiting), is to identify resolved and under-resolved solution parts while distinguishing between smooth and non-smooth regions such as shocks. The indicator aims to approximate non-constant smooth solution regions through piecewise linear or quadratic polynomials, while automatically reverting to constant approximations in regions characterized by constant behavior. In the presence of shocks or discontinuities, the indicator attempts to prevent over- and undershoots by applying limiting techniques whenever possible or by appropriately lowering the approximation order when required.

The present approach draws upon relevant existing techniques in the field. Firstly, a gradient reconstruction method based on a local L^2-projection, as proposed in [KS13], where it serves as a parameter-free smoothness indicator for the unsteady linear advection equation. Secondly, [Aiz+17] introduced a flux-based gradient reconstruction technique, which was subsequently employed within the framework of anisotropic slope limiters for the DG method.

Figure 2.4 shows the indicator, combining the aforementioned concepts with a vertex-based slope limiter. This amalgamation enables the reuse of pre-computed data to enhance computational efficiency while seamlessly integrating p-adaptivity with slope limiting. The ensuing sections elaborate on the underlying principles governing the indicator's operation.

For simplicity, we again formulate our methodology in the remainder of this section for a generic scalar function $w(\boldsymbol{x})$ defined on Ω and its discretized counterpart $w_\Delta(\boldsymbol{x})$. Employing $\varphi_{ei}(\boldsymbol{x})$, with $i = 1, \ldots, P(p)$, as a basis of $\mathbb{P}^p(\Omega_e)$, w_Δ can be expressed as

$$w_\Delta(\boldsymbol{x})_{|\Omega_e} = \sum_{i=1}^{P(p)} w_{ei}\, \varphi_{ei}(\boldsymbol{x}) =: w_e^p.$$

In this representation, the superscript denotes the approximation order, and the absence of a superscript implies the default (full) approximation order. Additionally, we use the shorthand notation p0, p1, and p2 to refer to piecewise constant, linear, and quadratic approximation spaces, respectively.

In Chapter 4, the adaptivity indicator, is applied to the free surface elevation, that is, $w = \xi$, and the degrees of freedom of the depth-integrated velocity $\boldsymbol{q}$ are adapted correspondingly.

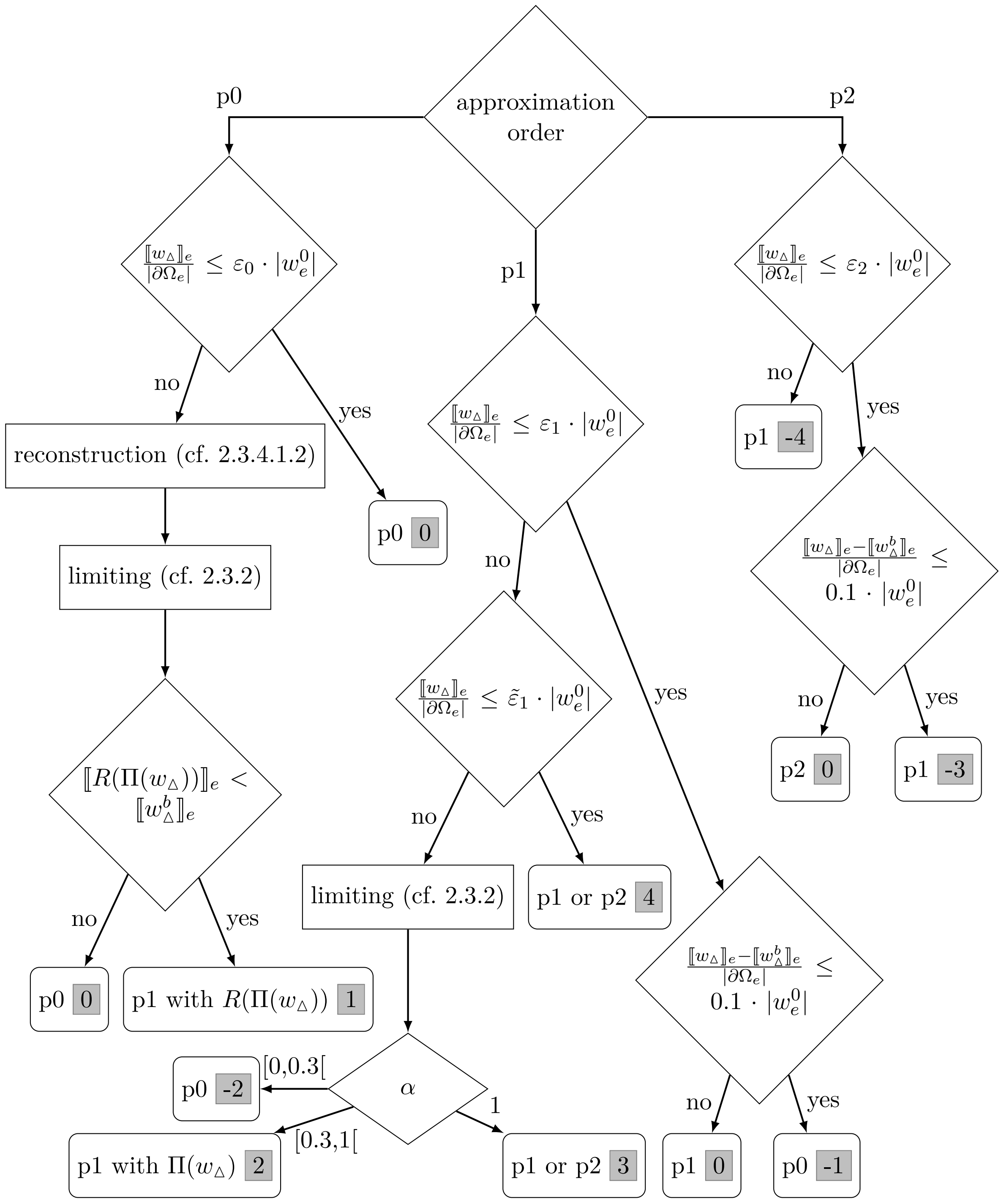

approximation order
p0
p1
p2
$\frac{[\![w_\triangle]\!]_e}{|\partial\Omega_e|} \leq \varepsilon_0 \cdot |w_e^0|$
$\frac{[\![w_\triangle]\!]_e}{|\partial\Omega_e|} \leq \varepsilon_1 \cdot |w_e^0|$
$\frac{[\![w_\triangle]\!]_e}{|\partial\Omega_e|} \leq \varepsilon_2 \cdot |w_e^0|$
no
yes
reconstruction (cf. 2.3.4.1.2)
p0 0
limiting (cf. 2.3.2)
$[\![R(\Pi(w_\triangle))]\!]_e < [\![w_\triangle^b]\!]_e$
p0 0
p1 with $R(\Pi(w_\triangle))$ 1
$\frac{[\![w_\triangle]\!]_e}{|\partial\Omega_e|} \leq \tilde{\varepsilon}_1 \cdot |w_e^0|$
limiting (cf. 2.3.2)
p1 or p2 4
α
[0,0.3[
p0 -2
[0.3,1[
p1 with $\Pi(w_\triangle)$ 2
1
p1 or p2 3
$\frac{[\![w_\triangle]\!]_e - [\![w_\triangle^b]\!]_e}{|\partial\Omega_e|} \leq 0.1 \cdot |w_e^0|$
p1 0
p0 -1
p1 -4
$\frac{[\![w_\triangle]\!]_e - [\![w_\triangle^b]\!]_e}{|\partial\Omega_e|} \leq 0.1 \cdot |w_e^0|$
p2 0
p1 -3

Figure 2.4: Flow chart depicting the JRL adaptivity indicator for orders 0, 1, and 2. Π is the limiting operator defined in (2.19), and R the reconstruction defined in (2.24). The gray boxes represent various adaption scenarios, with zero denoting no adaption and no limiting, negative values indicating a decrease, and positive values pointing to an increase of the approximation order or applying limiting (adapted from [FNA22]).

2.3.4.1.1 Error detection

Let us begin by establishing some notation. For the *base approximation order* $b := \max\{p-1, 0\}$, the *jump* associated with element Ω_e is defined as

$$[\![w_\triangle]\!]_e := \sum_{e^+\in I_e} \int_{\partial\Omega_e\cap\partial\Omega_{e^+}} \left|w_e^p - w_{e^+}^b\right| ds = \sum_{e^+\in I_e} \int_{\partial\Omega_e\cap\partial\Omega_{e^+}} \left|\sum_{i=1}^{P(p)} w_{ei}\varphi_{ei} - \sum_{i=1}^{P(b)} w_{e^+i}\varphi_{e^+i}\right| ds, \tag{2.23}$$

where I_e contains all element indices as stated in the beginning of Section 2.3.1. The *base jump* replaces w_e^p in formula (2.23) by the truncated approximation w_e^b on Ω_e, that is,

$$[\![w_\triangle^b]\!]_e := \sum_{e^+\in I_e} \int_{\partial\Omega_e\cap\partial\Omega_{e^+}} \left|w_e^b - w_{e^+}^b\right| ds.$$

It is primarily employed in combination with $[\![w_\triangle]\!]_e$ to estimate the solution regularity. The decision-making process is dependent upon the following thresholds: $\varepsilon_0 = 0.01$, $\varepsilon_1 = 0.005$, $\varepsilon_2 = 0.001$, and $\tilde{\varepsilon}_1 = 0.01$, all of which are suitable and remain invariant across diverse test cases and resolutions. However, users still have the option to change the parameters if desired or required by certain test cases.

The jump definition in (2.23) diverges slightly from the conventional jump definition (cf. (2.27) in Section 2.3.4.2.1) wherein a full order approximation is considered. The definition we offer is designed to mitigate the influence of over- and undershoots from neighboring elements, as well as the sequence in which increments and decrements involving neighboring elements occur.

2.3.4.1.2 Gradient reconstruction

Considering a p0 solution $w_\triangle^0(t, \boldsymbol{x})$ that satisfies $\frac{[\![w_\triangle^0]\!]_e}{|\partial\Omega_e|} \leq 0.01 \cdot |w_e^0|$, we construct a linear solution through the application of a rotationally invariant gradient reconstruction

$$R(w_\triangle(\boldsymbol{x})_{|\Omega_e}) = w_e^0 + \frac{\partial w_e^1}{\partial x}\psi_{e2}(\boldsymbol{x}) + \frac{\partial w_e^1}{\partial y}\psi_{e3}(\boldsymbol{x}). \tag{2.24}$$

This reconstruction utilizes the linear Taylor basis functions ψ_{ei} as outlined in Equation (6) of [Kuz10]. These basis functions are defined on the element Ω_e as follows:

$$\psi_{e1}(\boldsymbol{x}) = 1, \quad \psi_{e2}(\boldsymbol{x}) = x - x_e^c, \quad \psi_{e3}(\boldsymbol{x}) = y - y_e^c,$$

where $\boldsymbol{x}_e^c = (x_e^c, y_e^c)^T = \frac{1}{3}(\boldsymbol{a}_{e1} + \boldsymbol{a}_{e2} + \boldsymbol{a}_{e3})$ represents the centroid of Ω_e. The first coefficient is the solution's mean value, while the subsequent two coefficients are derived from the directional derivatives calculated along all edges. Let us consider Ω_{e^+} with centroid $\boldsymbol{x}_{e^+}^c = (x_{e^+}^c, y_{e^+}^c)$ sharing an edge with Ω_e. The directional derivative of w_e on $\partial\Omega_e \cap \partial\Omega_{e^+}$ in direction $\boldsymbol{d}_{e^+} = \boldsymbol{x}_{e^+}^c - \boldsymbol{x}_e^c$ can be approximated as follows:

$$\frac{\partial w_e}{\partial \boldsymbol{d}_{e^+}} \approx \frac{w_{e^+}^0 - w_e^0}{|\boldsymbol{d}_{e^+}|}. \tag{2.25}$$

After obtaining these approximations of the directional derivatives, a least squares problem is solved to determine the higher-order coefficients. Specifically, let $N \in \mathbb{R}^{3\times 2}$ represent the matrix containing the directions $\boldsymbol{d}_{e^+}$ in its rows, and let $\boldsymbol{\nu}_e \in \mathbb{R}^3$ denote the vector of directional derivatives defined in (2.25) for all neighboring elements Ω_{e^+}. The missing coefficients from (2.24) are then obtained as the solution of the least squares problem

$$\left(\frac{\partial w_e^1}{\partial x}, \frac{\partial w_e^1}{\partial y}\right)^T = (N^T N)^{-1} N^T \boldsymbol{\nu}_e. \tag{2.26}$$

Finally, the solution is transformed back into the orthonormal basis. For elements at the boundary of Ω, one can define the directional derivatives using a ghost layer. The aforementioned methodology naturally generalizes to arbitrary polygons.

2.3.4.2 Jump and gradient indicators

For the assessment of our novel indicator's performance, a comprehensive evaluation is presented in Section 4.3. This evaluation involves comparisons with two other adaptivity indicators. The first one, described in Section 2.3.4.2.1, represents a well-established scheme that relies on the analysis of inter-element jumps to assess the local solution regularity. The second one, outlined in Section 2.3.4.2.2, is our own version of the gradient indicator, specifically enhanced to accommodate p0 discretizations. It is essential to emphasize that these two indicators need either one (cf. Section 2.3.4.2.1) or three (cf. Section 2.3.4.2.2) user-defined thresholds as input parameters. These parameters must be calibrated manually and may differ for different scenarios, for limited and unlimited simulations as well as across different grid resolutions within the same scenario.

2.3.4.2.1 Jump indicator

The indicator introduced in [RFS03] computes the sum of the jumps across the edges of an element Ω_e. In our notation, this is expressed as

$$[\![w_\triangle]\!]_e^* := \sum_{e^+ \in I_e} \int_{\partial\Omega_e \cap \partial\Omega_{e^+}} \left| w_e^p - w_{e^+}^p \right| ds. \tag{2.27}$$

Subsequently, if the total jump observed over the element boundaries exceeds the threshold established by the user, the local approximation order is increased. Conversely, a decrease of the order is performed if the jump falls below the specified threshold. This indicator can be viewed as a simplified version of the JRL indicator since it also employs jump calculations for error detection. We refer to it as JE (Jump-Estimation) hereafter.

2.3.4.2.2 Gradient indicator

There exist gradient indicators in the literature, including the one introduced in [BS05] originally designed for the compressible Navier–Stokes equations and subsequently employed in the context of the SWE in [Kub+09]. This approach computes gradients utilizing the element-local solutions, making it unsuitable for applications with p0-discretizations. To address this deficiency, we designed a new scheme by using directional derivatives in the directions $\boldsymbol{d}_f = \boldsymbol{x}_f - \boldsymbol{x}_e^c$, where $\boldsymbol{x}_f$ represents the midpoint of edge $f \subset \partial\Omega_e$. The directional derivative is then approximated using the element centroid value and the edge midpoint values, extracted from both the current element and its neighboring edge, as illustrated by the following expressions:

$$\frac{\partial w_e}{\partial \boldsymbol{d}_f} \approx \frac{w_e(\boldsymbol{x}_f) - w_e(\boldsymbol{x}_e^c)}{|\boldsymbol{d}_f|} \quad \text{and} \quad \frac{\partial w_{e^+}}{\partial \boldsymbol{d}_f} \approx \frac{w_{e^+}(\boldsymbol{x}_f) - w_e(\boldsymbol{x}_e^c)}{|\boldsymbol{d}_f|}.$$

To decide, whether an increment of the approximation order is necessary, a comparison is made by evaluating

$$\left| \frac{\partial w_e}{\partial \boldsymbol{d}_f} - \frac{\partial w_{e^+}}{\partial \boldsymbol{d}_f} \right| < \varepsilon_w (\Delta_e)^p, \tag{2.28}$$

where $\Delta_e = \sqrt{2|\Omega_e|}$. In our implementation, this indicator is applied to the three primary unknowns, namely $w \in \{\xi, U, V\}$, in the following manner. If inequality (2.28) is not satisfied for all directions $\boldsymbol{d}_f$ and all three unknowns, the local approximation order of the corresponding element is increased. Conversely, if the inequality holds true for all directions and unknowns, the same comparison (2.28) is carried out for the base order solution w_e^b (see Sec. 2.3.4.1.1) on the current element, with $\varepsilon_w(\Delta_e)^b$ being employed as the upper bound of (2.28). Should the criterion also hold for w_e^b, it indicates that the lower-order approximation adequately captures the solution, thus justifying a decrease of the order. The threshold ε_w requires tailored selection for each unknown and scenario, although consistent usage across different mesh resolutions is possible. We refer to this indicator as GRE (Gradient-Reconstruction-Extended).

2.4 Stability analysis

The theoretical aspects of DG discretizations for modeling surface and subsurface flow, particularly concerning stability and *a priori* error estimates, have attracted significant attention over recent decades. A proof of discrete stability and an *a priori* error estimate for the DG method applied to the two-dimensional SWE, excluding nonlinear advection terms from the momentum equations, was presented in [Aiz04]. Building on this, [MDA15] further extended the DG stability proof to encompass a model involving shallow water and bed morphology dynamics. It is worth noting, however, that this approach employs an L^2-projection for handling the nonlinear terms resulting in an inconsistent model. A further contribution includes [AD07], which introduced a stability proof for the local DG discretization of the three-dimensional SWE model. Further stability proofs were accomplished in [Aiz+18] for the unsteady Darcy problem

without nonlinearities and in [Reu+19] for a coupled hydrostatic/Darcy system utilizing the local DG method. Additionally, an *a priori* error estimate is demonstrated in [Mar97] for FE discretization of the SWE and in [Pro02] for a simplified SWE model and various discretization types.
Remarkably, to the best of our knowledge, no stability proof exists for the two-dimensional SWE model that encompasses nonlinear advection terms, nor is there a discrete stability proof for the DG method applied to the full nonlinear system. In the subsequent sections, we introduce relevant mathematical tools, notations, and assumptions, preceding the presentation of a stability proof for the continuous SWE model (2.29)–(2.30), alongside a rigorous proof of stability for a DG discretization of the SWE building upon [Aiz04]. Our proof uses a new technique to handle the nonlinear advection terms and, additionally, the discrete result is independent of the mesh element size.

To accommodate inflow and outflow boundary conditions in the analysis, a slight adjustment is made to the quadrature-free reformulation (2.6)–(2.8) on page 13. This adaptation, shown in (2.29), involves interchanging $\boldsymbol{u}$ and $\boldsymbol{q}$ in the nonlinear advection terms. This modification facilitates the offsetting of outflow boundary terms against each other. The original choice presented in (2.8) is physically motivated by considering the advection's role in transporting the depth-integrated velocity. However, numerical experiments with the adapted formulation demonstrated analogous convergence and stability behavior to those detailed in Section 4.2. Additionally, since the bathymetry h_b does not vary in time and $H = h_b + \xi$, we take the time derivative $\partial_t H$ instead of $\partial_t \xi$ in the first equation. Furthermore, we do some algebraic manipulations to the gravity term again and arrive at the following system:

$$\partial_t \begin{pmatrix} H \\ U \\ V \end{pmatrix} + \nabla \cdot \begin{pmatrix} U & V \\ uU + \frac{gH^2}{2} & uV \\ vU & vV + \frac{gH^2}{2} \end{pmatrix} = \begin{pmatrix} 0 \\ -\tau_{\mathrm{bf}} u + f_c V + gH\partial_x h_b + F_x \\ -\tau_{\mathrm{bf}} v - f_c U + gH\partial_y h_b + F_y \end{pmatrix}, \tag{2.29}$$

$$\boldsymbol{u}H = \boldsymbol{q}. \tag{2.30}$$

In the remainder of this section, we assume the quadratic bottom friction law, that is, $\tau_{\mathrm{bf}} = C_f\,|\boldsymbol{u}|$, where C_f is constant. Furthermore, g and f_c are constant.

2.4.1 Mathematical tools and notation

The forthcoming subsections rely on certain results from Functional Analysis and the theory of PDEs. The most frequently used ones are cited below. They can be found in textbooks dedicated to the mathematical theory of FE methods, such as [GT01; BS07; Eva10].

First, we need to emphasize that scalar quantities are represented using regular characters, whereas bold font characters are reserved for vector-valued functions.

Definition 1 (Lebesgue spaces ([BS07] Section 1.1))
Let $\Omega \subseteq \mathbb{R}^d$ be open and consider $1 \leq p \leq \infty$. The Lebesgue spaces

$$L^p(\Omega) = \{f : \Omega \to \mathbb{R}^n \,|\, f \text{ is Lebesgue measurable}, \|f\|_{L^p} < \infty\}$$

are Banach spaces with norms

$$\|f\|_{L^p} := \left(\int_\Omega |f|^p \,\mathrm{d}x\right)^{\frac{1}{p}} \text{ for } 1 \leq p < \infty \text{ and } \|f\|_{L^\infty} := \operatorname*{ess\,sup}_\Omega |f| \leq \infty.$$

Definition 2 (Sobolev spaces ([BS07] Section 1.3))
Let $\Omega \subseteq \mathbb{R}^d$ be open, consider $k \in \mathbb{N}$ and $1 \leq p \leq \infty$.The Sobolev spaces

$$\begin{aligned} W^{k,p}(\Omega) = \{f \in L^p(\Omega) \,|\, & \text{ there exists a weak derivative } D^\alpha f \in L^p(\Omega) \\ & \text{for each multi-index } |\alpha| \leq k\} \end{aligned}$$

are Banach spaces. If $p = 2$, we usually write $H^k(\Omega) = W^{k,2}(\Omega)$.

Theorem 1 (Divergence (Gauss') theorem and integration by parts ([Neč12] Section 3.1))
Let $\Omega \subseteq \mathbb{R}^d$ be open and bounded with Lipschitz boundary, and let $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$. For $\boldsymbol{f} \in W^{1,1}(\Omega)^d$, $g \in W^{1,p}(\Omega)$, $\boldsymbol{h} \in W^{1,q}(\Omega)^d$, identified with their traces on $\partial\Omega$, the following relations hold:

$$\int_\Omega \nabla \cdot \boldsymbol{f} \,\mathrm{d}x = \int_{\partial\Omega} \boldsymbol{f} \cdot \boldsymbol{n} \,\mathrm{d}s$$

and

$$\int_\Omega \nabla g \cdot \boldsymbol{h} \,\mathrm{d}x = -\int_\Omega g \nabla \cdot \boldsymbol{h} \,\mathrm{d}x + \int_{\partial\Omega} g\boldsymbol{h} \cdot \boldsymbol{n} \,\mathrm{d}s.$$

Theorem 2 (Hölder's inequality ([Eva10] Section B.2))
Let $1 \leq p, q \leq \infty$ with $\frac{1}{p} + \frac{1}{q} = 1$. Then if $f \in L^p(\Omega)$ and $g \in L^q(\Omega)$, we have

$$\int_\Omega |fg| \;\leq\; ||f||_{L^p(\Omega)} ||g||_{L^q(\Omega)}. \tag{2.31}$$

Theorem 3 (Young's inequality with ε ([GT01] Section 7.1))
For any positive real numbers a, b, p, q with $\frac{1}{p} + \frac{1}{q} = 1$ the following inequality holds:

$$ab \;\leq\; \frac{\varepsilon a^p}{p} + \frac{b^q}{\varepsilon^{q/p} q}. \tag{2.32}$$

Theorem 4 (Grönwall's inequality ([Gro19]))
Let $\psi(t)$ be a continuous function satisfying

$$\psi(t) \le \alpha(t) + \int_{t_0}^{t} \beta(s)\psi(s)\,\mathrm{d}s, \quad t \in [t_0, T]$$

with continuous $\alpha(t)$ and $\beta(t) \ge 0$, then the following inequality holds:

$$\psi(t) \le \alpha(t) + \int_{t_0}^{t} \alpha(s)\beta(s)\exp\left(\int_{s}^{t}\beta(\sigma)\,\mathrm{d}\sigma\right)\mathrm{d}s \le \alpha(t) + C(\beta)\int_{t_0}^{t}\alpha(s)\beta(s)\,\mathrm{d}s,$$

with a constant

$$C(\beta) = \exp\left(\int_{t_0}^{t}\beta(s)\,\mathrm{d}s\right). \tag{2.33}$$

The mathematical notation follows standard conventions. Certain specifics utilized in this thesis are introduced and outlined below:

- The Euclidean scalar product of vectors $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^d$ is $\boldsymbol{a} \cdot \boldsymbol{b}$.
- L^2-inner products on domains $\Omega \subset \mathbb{R}^2$ and surfaces $\gamma \subset \mathbb{R}$ are denoted by $(\cdot\,,\,\cdot)_\Omega$ and $\langle\cdot\,,\,\cdot\rangle_\gamma$, respectively.
- For a non-overlapping partition $\mathcal{T}_\triangle$ of the domain $\Omega \subset \mathbb{R}^2$, the elements are referred to as Ω_e with indices $e \in I_e$.
- The faces are referred to as γ_i with indices i belonging to either the interior edge indices I_{int} or the indices of the exterior edges I_{ext}. The exterior edge indices are grouped into inflow and outflow edge indices, and denoted by I_{in} and I_{out}, respectively.
- For simplicity, we assume the triangulation $\mathcal{T}_\triangle$ to be geometrically conformal in the sense of Definition 1.55 in [EG04]. While all proofs and arguments hold true for geometrically non-conforming meshes, they would involve increased technical complexity.
- The unit normal vector on the boundary is denoted by $\boldsymbol{n}$.
- We fix the normals of all interior edges and define the jump $[\![\cdot]\!]$ and the average $\{\!\{\cdot\}\!\}$ as

  $$[\![w]\!] = (w - w^+) \quad \text{and} \quad \{\!\{w\}\!\} = \tfrac{1}{2}(w + w^+),$$

  where w denotes the solution on the current element and w^+ on its neighbor.
  The following well-known property holds: $[\![ab]\!] = \{\!\{a\}\!\}[\![b]\!] + [\![a]\!]\{\!\{b\}\!\}$.

2.4.2 Continuous stability

Starting from the quadrature-free reformulation (2.29)–(2.30) we multiply the equations by test functions, integrate them over Ω, and integrate by parts to deduce the *global weak formulation* explicitly detailing all integrals involved.

We distinguish between the inflow boundary $\partial\Omega_i := \{\partial\Omega : \boldsymbol{q}\cdot\boldsymbol{n} \le 0\}$ and the outflow boundary $\partial\Omega_o := \{\partial\Omega : \boldsymbol{q}\cdot\boldsymbol{n} > 0\}$. In the interest of simplicity, we assume Dirichlet boundary conditions on the inflow segment: $H = \hat{H} > 0$ and $\boldsymbol{u} = \hat{\boldsymbol{u}}$ should hold pointwise, where $\hat{}$ denotes a prescribed value for the corresponding unknown. Furthermore, $\boldsymbol{q} = \hat{\boldsymbol{u}}\hat{H} =: \hat{\boldsymbol{q}}$ is also specified at the inflow boundary. Such boundary conditions are typical in the context of supercritical flows. We assume that the boundary condition functions fulfill the required regularity conditions.

Problem 1 (Global weak formulation)
We seek a solution $(H, \boldsymbol{q}, \boldsymbol{u})$ with $H \in L^2(0,T;W^{1,4}(\Omega)) \cap C^1(0,T;L^4(\Omega))$, $\boldsymbol{q} \in \left(L^2(0,T;H^1(\Omega)) \cap C^1(0,T;L^2(\Omega))\right)^2$, and $\boldsymbol{u} \in \left(L^4(0,T;W^{1,4}(\Omega)) \cap C^1(0,T;L^4(\Omega))\right)^2$, such that for a.e. $t \in (0,T)$ and for all test functions $\vartheta \in H^1(\Omega)$, $\boldsymbol{\theta} \in \left(W^{1,4}(\Omega)\right)^2$, and $\boldsymbol{\psi} \in \left(L^2(\Omega)\right)^2$ the following holds:

$$(\partial_t H, \vartheta)_\Omega + \langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \vartheta\rangle_{\partial\Omega_i} + \langle \boldsymbol{q}\cdot\boldsymbol{n}, \vartheta\rangle_{\partial\Omega_o} - (\boldsymbol{q}\cdot\nabla, \vartheta)_\Omega = 0, \tag{2.34}$$

$$\begin{aligned}(\partial_t \boldsymbol{q}, \boldsymbol{\theta})_\Omega &+ \langle \hat{\boldsymbol{u}}(\hat{\boldsymbol{q}}\cdot\boldsymbol{n}), \boldsymbol{\theta}\rangle_{\partial\Omega_i} + \langle \boldsymbol{u}(\boldsymbol{q}\cdot\boldsymbol{n}), \boldsymbol{\theta}\rangle_{\partial\Omega_o} - (\boldsymbol{u}(\boldsymbol{q}\cdot\nabla), \boldsymbol{\theta})_\Omega + \langle \tfrac{g}{2}\hat{H}^2\boldsymbol{n}, \boldsymbol{\theta}\rangle_{\partial\Omega_i} + \langle \tfrac{g}{2}H^2\boldsymbol{n}, \boldsymbol{\theta}\rangle_{\partial\Omega_o}\\ &- \left(\tfrac{g}{2}H^2\nabla, \boldsymbol{\theta}\right)_\Omega + (C_f\,|\boldsymbol{u}|\boldsymbol{u}, \boldsymbol{\theta})_\Omega - \left(\left(\begin{smallmatrix}0 & -f_c\\ f_c & 0\end{smallmatrix}\right)\boldsymbol{q}, \boldsymbol{\theta}\right)_\Omega - (gH\nabla h_b, \boldsymbol{\theta})_\Omega = (\boldsymbol{F}, \boldsymbol{\theta})_\Omega\,,\end{aligned} \tag{2.35}$$

$$(\boldsymbol{u}H, \boldsymbol{\psi})_\Omega = (\boldsymbol{q}, \boldsymbol{\psi})_\Omega\,. \tag{2.36}$$

In Theorem 5, we present a stability result for Problem 1. Its proof, that is shown in the remainder of this section, employs the same approach as the one for the discrete model (Theorem 6). However, in the discrete one, additional integrals occur because of the presence of numerical flux terms. To keep the numbering of the integrals consistent between both proofs, some numbers are absent in the first proof.

Theorem 5
Assuming $H > 0$, the following stability result holds for the solution of Problem 1:

$$\begin{aligned}&||\sqrt{H(T)}\,\boldsymbol{u}(T)||^2_{L^2(\Omega)} + g||H(T)||^2_{L^2(\Omega)} + C_f\int_0^T ||\boldsymbol{u}(t)||^3_{L^3(\Omega)}\,\mathrm{d}t\\ &\quad+ \int_0^T \langle \boldsymbol{q}(t)\cdot\boldsymbol{n}, \boldsymbol{u}(t)\cdot\boldsymbol{u}(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t + 2g\int_0^T \langle \boldsymbol{q}(t)\cdot\boldsymbol{n}, H(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t\\ &\le K\left(||\sqrt{H(0)}\,\boldsymbol{u}(0)||^2_{L^2(\Omega)} + g||H(0)||^2_{L^2(\Omega)} + \int_0^T |\langle \hat{\boldsymbol{q}}(t)\cdot\boldsymbol{n}, \hat{\boldsymbol{u}}(t)\cdot\hat{\boldsymbol{u}}(t)\rangle_{\partial\Omega_i}|\,\mathrm{d}t\right.\\ &\quad\left.+2g\int_0^T |\langle \hat{\boldsymbol{q}}(t)\cdot\boldsymbol{n}, \hat{H}(t)\rangle_{\partial\Omega_i}|\,\mathrm{d}t + \tfrac{2}{9}\sqrt{\tfrac{6}{C_f}}\int_0^T ||\boldsymbol{F}(t)||^{\frac{3}{2}}_{L^{\frac{3}{2}}(\Omega)}\,\mathrm{d}t\right),\end{aligned} \tag{2.37}$$

where $K = K(|\Omega|, T)$, and its dependence on T is exponential as shown in (2.33).

Proof. We test (2.34) twice with $\vartheta = \frac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}$ and $\vartheta = gH$, respectively, (2.35) with $\boldsymbol{\theta} = \boldsymbol{u}$, and add the resulting three equations:

$$\begin{aligned}
&\underbrace{\left(\partial_t H, \tfrac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}\right)_\Omega}_{\Lambda_1} + \underbrace{\langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \tfrac{1}{2}\hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \boldsymbol{q}\cdot\boldsymbol{n}, \tfrac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}\rangle_{\partial\Omega_o}}_{\Lambda_3} \underbrace{- \left(\boldsymbol{q}\cdot\nabla, \tfrac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}\right)_\Omega}_{\Lambda_4} \\
&+ \underbrace{(\partial_t H, gH)_\Omega}_{\Lambda_5} + \underbrace{\langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, g\hat{H}\rangle_{\partial\Omega_i} + \langle \boldsymbol{q}\cdot\boldsymbol{n}, gH\rangle_{\partial\Omega_o}}_{\Lambda_7} \underbrace{- (\boldsymbol{q}\cdot\nabla, gH)_\Omega}_{\Lambda_8} + \underbrace{(\partial_t\boldsymbol{q}, \boldsymbol{u})_\Omega}_{\Lambda_9} \\
&+ \underbrace{\langle \hat{\boldsymbol{u}}(\hat{\boldsymbol{q}}\cdot\boldsymbol{n}), \hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \boldsymbol{u}(\boldsymbol{q}\cdot\boldsymbol{n}), \boldsymbol{u}\rangle_{\partial\Omega_o}}_{\Lambda_{11}} \underbrace{- (\boldsymbol{u}(\boldsymbol{q}\cdot\nabla), \boldsymbol{u})_\Omega}_{\Lambda_{12}} + \underbrace{\langle \tfrac{g}{2}\hat{H}^2\boldsymbol{n}, \hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \tfrac{g}{2}H^2\boldsymbol{n}, \boldsymbol{u}\rangle_{\partial\Omega_o}}_{\Lambda_{14}} \\
&\underbrace{- \left(\tfrac{g}{2}H^2\nabla, \boldsymbol{u}\right)_\Omega}_{\Lambda_{15}} + \underbrace{(C_f\,|\boldsymbol{u}|\boldsymbol{u}, \boldsymbol{u})_\Omega}_{\Lambda_{16}} \underbrace{- \left(\left(\begin{smallmatrix} 0 & -f_c \\ f_c & 0 \end{smallmatrix}\right)\boldsymbol{q}, \boldsymbol{u}\right)_\Omega}_{\Lambda_{17}} \underbrace{- (gH\nabla h_b, \boldsymbol{u})_\Omega}_{\Lambda_{18}} = \underbrace{(\boldsymbol{F}, \boldsymbol{u})_\Omega}_{\Lambda_{19}}.
\end{aligned}$$

We note that the Coriolis terms, namely Λ_{17}, for the x- and y-momentum equations cancel each other out:

$$-\left(\left(\begin{smallmatrix} 0 & -f_c \\ f_c & 0 \end{smallmatrix}\right)\boldsymbol{q}, \boldsymbol{u}\right)_\Omega \overset{(2.36)}{=} -\left(\left(\begin{smallmatrix} 0 & -f_c \\ f_c & 0 \end{smallmatrix}\right)\boldsymbol{u}H, \boldsymbol{u}\right)_\Omega = 0.$$

We start with Λ_9 and apply (2.36), the product rule, and (2.34):

$$\begin{aligned}
(\partial_t\boldsymbol{q}, \boldsymbol{u})_\Omega &= \lim_{\Delta t\to 0}\left(\tfrac{\boldsymbol{q}(t+\Delta t)-\boldsymbol{q}(t)}{\Delta t}, \boldsymbol{u}\right)_\Omega \overset{\substack{(2.36)\text{ since}\\ \boldsymbol{u}\in(L^2(\Omega))^2}}{=} \lim_{\Delta t\to 0}\left(\tfrac{\boldsymbol{u}(t+\Delta t)H(t+\Delta t)-\boldsymbol{u}(t)H(t)}{\Delta t}, \boldsymbol{u}\right)_\Omega \\
&= (\partial_t(\boldsymbol{u}H), \boldsymbol{u})_\Omega \overset{\substack{\text{prod.}\\ \text{rule}}}{=} (\partial_t H, \boldsymbol{u}\cdot\boldsymbol{u})_\Omega + (\partial_t\boldsymbol{u}, H\boldsymbol{u})_\Omega \\
&\overset{\substack{(2.34)\text{ since}\\ \boldsymbol{u}\cdot\boldsymbol{u}\in H^1(\Omega)}}{=} \underbrace{-\langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} - \langle \boldsymbol{q}\cdot\boldsymbol{n}, \boldsymbol{u}\cdot\boldsymbol{u}\rangle_{\partial\Omega_o}}_{\Lambda_{9b}} + \underbrace{(\boldsymbol{q}\cdot\nabla, \boldsymbol{u}\cdot\boldsymbol{u})_\Omega}_{\Lambda_{9c}} + \underbrace{\left(\tfrac{1}{2}\partial_t(\boldsymbol{u}\cdot\boldsymbol{u}), H\right)_\Omega}_{\Lambda_{9d}}.
\end{aligned}$$

Then, we add Λ_1 and Λ_{9d}:

$$\left(\partial_t H, \tfrac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}\right)_\Omega + \left(\tfrac{1}{2}\partial_t(\boldsymbol{u}\cdot\boldsymbol{u}), H\right)_\Omega = \tfrac{1}{2}\left(\partial_t(H\boldsymbol{u}\cdot\boldsymbol{u}), 1\right)_\Omega = \Upsilon_1'.$$

Now adding Λ_4, Λ_3, Λ_{9c}, Λ_{9b}, Λ_{12}, and Λ_{11} leads to

$$\begin{aligned}
&-\left(\boldsymbol{q}\cdot\nabla, \tfrac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}\right)_\Omega + \langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \tfrac{1}{2}\hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \boldsymbol{q}\cdot\boldsymbol{n}, \tfrac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}\rangle_{\partial\Omega_o} + (\boldsymbol{q}\cdot\nabla, \boldsymbol{u}\cdot\boldsymbol{u})_\Omega - \langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} \\
&- \langle \boldsymbol{q}\cdot\boldsymbol{n}, \boldsymbol{u}\cdot\boldsymbol{u}\rangle_{\partial\Omega_o} - (\boldsymbol{u}(\boldsymbol{q}\cdot\nabla), \boldsymbol{u})_\Omega + \langle \hat{\boldsymbol{u}}(\hat{\boldsymbol{q}}\cdot\boldsymbol{n}), \hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \boldsymbol{u}(\boldsymbol{q}\cdot\boldsymbol{n}), \boldsymbol{u}\rangle_{\partial\Omega_o} = (\star_1).
\end{aligned}$$

Adding the first to the fourth, the second to the fifth, and the third to the sixth term, and keeping the rest, we get

$$(\star_1) = \cancel{\left(\tfrac{1}{2}\boldsymbol{q}\cdot\nabla, \boldsymbol{u}\cdot\boldsymbol{u}\right)_\Omega} - \langle \tfrac{1}{2}\hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} - \langle \tfrac{1}{2}\boldsymbol{q}\cdot\boldsymbol{n}, \boldsymbol{u}\cdot\boldsymbol{u}\rangle_{\partial\Omega_o} \underbrace{- (\boldsymbol{u}(\boldsymbol{q}\cdot\nabla), \boldsymbol{u})_\Omega}_{=-\cancel{\left(\frac{1}{2}\boldsymbol{q}\cdot\nabla, \boldsymbol{u}\cdot\boldsymbol{u}\right)_\Omega}}$$
$$+ \langle \hat{\boldsymbol{u}}(\hat{\boldsymbol{q}}\cdot\boldsymbol{n}), \hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \boldsymbol{u}(\boldsymbol{q}\cdot\boldsymbol{n}), \boldsymbol{u}\rangle_{\partial\Omega_o} = \tfrac{1}{2}\langle \hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \underbrace{\tfrac{1}{2}\langle \boldsymbol{q}\cdot\boldsymbol{n}, \boldsymbol{u}\cdot\boldsymbol{u}\rangle_{\partial\Omega_o}}_{\geq 0} = \Upsilon_2'.$$

Then, we proceed by adding Λ_8, Λ_{15} and Λ_{14}, applying (2.36) and Gauss' theorem:

$$\underbrace{-(g\boldsymbol{q}\cdot\nabla, H)_\Omega}_{\substack{(2.36)\text{ since}\\ \nabla H\in\left(L^2(\Omega)\right)^2\\ = \quad -(g\boldsymbol{u}H\cdot\nabla, H)_\Omega = -\left(\frac{g}{2}\boldsymbol{u}\cdot\nabla, H^2\right)_\Omega}} - \left(\tfrac{g}{2}H^2\nabla, \boldsymbol{u}\right)_\Omega + \langle \tfrac{g}{2}\hat{H}^2\boldsymbol{n}, \hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \tfrac{g}{2}H^2\boldsymbol{n}, \boldsymbol{u}\rangle_{\partial\Omega_o}$$
$$\overset{\text{Gauss}}{=} -\langle \tfrac{g}{2}, \hat{H}^2\hat{\boldsymbol{u}}\cdot\boldsymbol{n}\rangle_{\partial\Omega_i} - \langle \tfrac{g}{2}, H^2\boldsymbol{u}\cdot\boldsymbol{n}\rangle_{\partial\Omega_o} + \langle \tfrac{g}{2}\hat{H}^2\boldsymbol{n}, \hat{\boldsymbol{u}}\rangle_{\partial\Omega_i} + \langle \tfrac{g}{2}H^2\boldsymbol{n}, \boldsymbol{u}\rangle_{\partial\Omega_o} = 0.$$

Next, we look at Λ_7:

$$\langle g\hat{\boldsymbol{q}}\cdot\boldsymbol{n}, \hat{H}\rangle_{\partial\Omega_i} + \underbrace{\langle g\boldsymbol{q}\cdot\boldsymbol{n}, H\rangle_{\partial\Omega_o}}_{\geq 0} = \Upsilon_3'.$$

We rewrite the bottom friction term Λ_{16}:

$$\left(C_f\,|\boldsymbol{u}|\boldsymbol{u}, \boldsymbol{u}\right)_\Omega = C_f\left(|\boldsymbol{u}|\boldsymbol{u}, \boldsymbol{u}\right)_\Omega = C_f||\boldsymbol{u}||^3_{L^3(\Omega)} = \Upsilon_4'.$$

We estimate Λ_{18} with Young's inequality (2.32) and Hölder's inequality (2.31) and put the terms to the right-hand side:

$$|-(gH\nabla h_b, \boldsymbol{u})_\Omega| \overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=1}}{\leq} \tfrac{1}{2}g^2||\nabla h_b||^2_{L^\infty(\Omega)}\,(1,H)_\Omega + \tfrac{1}{2}\left(\sqrt{H}\boldsymbol{u}, \sqrt{H}\boldsymbol{u}\right)_\Omega$$
$$\overset{\substack{(2.31)\text{ with}\\ p=q=2}}{\leq} \tfrac{1}{2}\left(g^2||\nabla h_b||^2_{L^\infty(\Omega)}||1||_{L^2(\Omega)}||H||_{L^2(\Omega)}\right) + \tfrac{1}{2}||\sqrt{H}\boldsymbol{u}||^2_{L^2(\Omega)}$$
$$\overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=\frac{1}{2}}}{\leq} \tfrac{1}{8}g^4||\nabla h_b||^4_{L^\infty(\Omega)}\underbrace{||1||^2_{L^2(\Omega)}}_{=|\Omega|} + \tfrac{1}{2}||H||^2_{L^2(\Omega)} + \tfrac{1}{2}||\sqrt{H}\boldsymbol{u}||^2_{L^2(\Omega)} = \Upsilon_5'.$$

Next, we estimate Λ_{19} with Young's inequality (2.32):

$$(\boldsymbol{F}, \boldsymbol{u})_\Omega \overset{\substack{(2.32)\text{ with}\\ p=\frac{3}{2},q=3,\varepsilon=\sqrt{\frac{2}{3C_f}}}}{\leq} \tfrac{2}{9}\sqrt{\tfrac{6}{C_f}}||\boldsymbol{F}||^{\frac{3}{2}}_{L^{\frac{3}{2}}(\Omega)} + \tfrac{C_f}{2}||\boldsymbol{u}||^3_{L^3(\Omega)} = \Upsilon_6'.$$

The only term left is Λ_5:

$$(\partial_t H, gH)_\Omega = \tfrac{g}{2}\left(\partial_t(H^2),1\right)_\Omega = \Upsilon_7'.$$

Now, we integrate everything from 0 to T and get the following from $\Upsilon_1', \dots, \Upsilon_7'$:

$$\begin{aligned}
\Upsilon_1' :&\quad \tfrac{1}{2}\int_0^T \left(\partial_t\left(H(t)\boldsymbol{u}(t)\cdot\boldsymbol{u}(t)\right),1\right)_\Omega \,\mathrm{d}t = \tfrac{1}{2}\left(||\sqrt{H(T)}\,\boldsymbol{u}(T)||^2_{L^2(\Omega)} - ||\sqrt{H(0)}\,\boldsymbol{u}(0)||^2_{L^2(\Omega)}\right) = \Upsilon_1,\\
\Upsilon_2' :&\quad \tfrac{1}{2}\int_0^T \langle\hat{\boldsymbol{q}}(t)\cdot\boldsymbol{n}, \hat{\boldsymbol{u}}(t)\cdot\hat{\boldsymbol{u}}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \underbrace{\tfrac{1}{2}\int_0^T \langle\boldsymbol{q}(t)\cdot\boldsymbol{n}, \boldsymbol{u}(t)\cdot\boldsymbol{u}(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t}_{\geq 0} = \Upsilon_2,\\
\Upsilon_3' :&\quad g\int_0^T \langle\hat{\boldsymbol{q}}(t)\cdot\boldsymbol{n}, \hat{H}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \underbrace{g\int_0^T \langle\boldsymbol{q}(t)\cdot\boldsymbol{n}, H(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t}_{\geq 0} = \Upsilon_3,\\
\Upsilon_4' :&\quad C_f\int_0^T ||\boldsymbol{u}(t)||^3_{L^3(\Omega)}\,\mathrm{d}t = \Upsilon_4,\\
\Upsilon_5' :&\quad \tfrac{1}{8}\int_0^T g^4|\Omega|||\nabla h_b||^4_{L^\infty(\Omega)}\,\mathrm{d}t + \tfrac{1}{2}\int_0^T ||H(t)||^2_{L^2(\Omega)}\,\mathrm{d}t + \tfrac{1}{2}\int_0^T ||\sqrt{H(t)}\boldsymbol{u}(t)||^2_{L^2(\Omega)}\,\mathrm{d}t = \Upsilon_5,\\
\Upsilon_6' :&\quad \tfrac{2}{9}\sqrt{\tfrac{6}{C_f}}\int_0^T ||\boldsymbol{F}(t)||^{\frac{3}{2}}_{L^{\frac{3}{2}}(\Omega)}\,\mathrm{d}t + \tfrac{C_f}{2}\int_0^T ||\boldsymbol{u}(t)||^3_{L^3(\Omega)}\,\mathrm{d}t = \Upsilon_6,\\
\Upsilon_7' :&\quad \tfrac{g}{2}\int_0^T \left(\partial_t\left(H(t)^2\right),1\right)_\Omega\,\mathrm{d}t = \tfrac{g}{2}\left(||H(T)||^2_{L^2(\Omega)} - ||H(0)||^2_{L^2(\Omega)}\right) = \Upsilon_7.
\end{aligned}$$

Then $||\sqrt{H(0)}\,\boldsymbol{u}(0)||^2_{L^2(\Omega)}$ and $||H(0)||^2_{L^2(\Omega)}$ are put to the right-hand side.

Finally, we apply Grönwall's inequality to $\frac{1}{2}\int_0^T ||H(t)||^2_{L^2(\Omega)}\,dt$ and to $\frac{1}{2}\int_0^T ||\sqrt{H(t)}\boldsymbol{u}(t)||^2_{L^2(\Omega)}\,dt$ from Υ_5 and arrive at the stated stability result after adding $\Upsilon_1, \dots, \Upsilon_7$, multiplying by 2, and reordering the terms. □

Remark 1

If the bathymetry is constant, as assumed in [Aiz04], Λ_{18} and consequently Υ_5 vanish and the proof does not need to make use of Grönwall's inequality. The constant in (2.37) *is then independent of T.*

2.4.3 Discrete stability

We partition our domain into elements denoted as Ω_e, $e \in I_e$, and state the *weak formulation on elements*. Let $V = \{u \in L^2(\Omega) : u_{|\Omega_e} \in H^1(\Omega_e), \forall\Omega_e\}$ and $W = \{u \in L^2(\Omega) : u_{|\Omega_e} \in W^{1,4}(\Omega_e), \forall\Omega_e\}$. Akin to the continuous scenario, we multiply the governing equations by the test functions, integrate over Ω_e, and integrate by parts. Assuming that the true solution is smooth and specifying the orientation of the normal vector $\boldsymbol{n}$ to ensure its outward direction with respect to the element possessing the lower index, we sum over all elements Ω_e.

Problem 2 (Weak formulation on elements)
We seek a solution $(H, \boldsymbol{q}, \boldsymbol{u})$ *with* $H \in C^1(0,T;W)$, $\boldsymbol{q} \in \left(C^1(0,T;V)\right)^2$, *and* $\boldsymbol{u} \in \left(C^1(0,T;W)\right)^2$, *such that for a.e.* $t \in (0,T)$ *and for all test functions* $\vartheta \in V$, $\boldsymbol{\theta} \in W^2$, *and* $\boldsymbol{\psi} \in V^2$ *the following holds:*

$$\sum_{e\in I_e} (\partial_t H, \vartheta)_{\Omega_e} + \sum_{i\in I_{\text{int}}} \langle \boldsymbol{q}\cdot\boldsymbol{n}, [\![\vartheta]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{ext}}} \langle \boldsymbol{q}\cdot\boldsymbol{n}, \vartheta\rangle_{\gamma_i} - \sum_{e\in I_e} (\boldsymbol{q}\cdot\nabla, \vartheta)_{\Omega_e} = 0, \tag{2.38}$$

$$\begin{aligned}\sum_{e\in I_e} (\partial_t \boldsymbol{q}, \boldsymbol{\theta})_{\Omega_e} &+ \sum_{i\in I_{\text{int}}} \langle \boldsymbol{u}(\boldsymbol{q}\cdot\boldsymbol{n}), [\![\boldsymbol{\theta}]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{ext}}} \langle \boldsymbol{u}(\boldsymbol{q}\cdot\boldsymbol{n}), \boldsymbol{\theta}\rangle_{\gamma_i} - \sum_{e\in I_e} (\boldsymbol{u}(\boldsymbol{q}\cdot\nabla), \boldsymbol{\theta})_{\Omega_e} \\ &+ \sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}H^2\boldsymbol{n}, [\![\boldsymbol{\theta}]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{ext}}} \langle \tfrac{g}{2}H^2\boldsymbol{n}, \boldsymbol{\theta}\rangle_{\gamma_i} - \sum_{e\in I_e} \left(\tfrac{g}{2}H^2\nabla, \boldsymbol{\theta}\right)_{\Omega_e} + \sum_{e\in I_e} (C_f|\boldsymbol{u}|\boldsymbol{u}, \boldsymbol{\theta})_{\Omega_e} \\ &- \sum_{e\in I_e} \left(\left(\begin{smallmatrix} 0 & -f_c \\ f_c & 0\end{smallmatrix}\right)\boldsymbol{q}, \boldsymbol{\theta}\right)_{\Omega_e} - \sum_{e\in I_e} (gH\nabla h_b, \boldsymbol{\theta})_{\Omega_e} = \sum_{e\in I_e} (\boldsymbol{F}, \boldsymbol{\theta})_{\Omega_e},\end{aligned} \tag{2.39}$$

$$\sum_{e\in I_e} (\boldsymbol{u}H, \boldsymbol{\psi})_{\Omega_e} = \sum_{e\in I_e} (\boldsymbol{q}, \boldsymbol{\psi})_{\Omega_e}. \tag{2.40}$$

As a next step, we seek to approximate the solution $(H, \boldsymbol{q}, \boldsymbol{u})$ to (2.6)–(2.8) using functions $H_\Delta \in C^1(0,T;W_\Delta)$, $\boldsymbol{q}_\Delta \in \left(C^1(0,T;V_\Delta)\right)^2$, and $\boldsymbol{u}_\Delta \in \left(C^1(0,T;W_\Delta)\right)^2$, where $V_\Delta \subset V$ and $W_\Delta \subset W$ are finite dimensional subspaces.
To ensure the method's consistency and stability, integrands in integrals over interior edges are replaced by numerical fluxes. They are specified as

$$\{\!\{\boldsymbol{u}_\Delta\}\!\}\{\!\{H_\Delta\}\!\} \text{ in (2.43) and} \tag{2.41}$$

$$\{\!\{\boldsymbol{u}_\Delta\}\!\}\{\!\{H_\Delta\}\!\}\{\!\{\boldsymbol{u}_\Delta\}\!\} \text{ and } \{\!\{H_\Delta^2\}\!\} \text{ in (2.44).} \tag{2.42}$$

We choose $H_\Delta, \vartheta \in \mathcal{P}^p$, $\boldsymbol{q}_\Delta, \boldsymbol{\theta} \in (\mathcal{P}^p)^2$, and $\boldsymbol{u}_\Delta, \boldsymbol{\psi} \in \left(\mathcal{P}^{\max\{0,p-1\}}\right)^2$, where $\mathcal{P}^p$ denotes the broken polynomial space, that is, $\mathcal{P}^p = \{u \in L^2(\Omega) : u_{|\Omega_e} \in \mathbb{P}^p(\Omega_e), \forall \Omega_e\}$ with $\mathbb{P}^p$ being the space of polynomials of degree $p \geq 0$. We project the initial conditions (2.3) onto the corresponding polynomial space via an L^2-projection.

By denoting $\boldsymbol{n}$ as the exterior unit normal vector on the boundary γ_i for $i \in I_{\text{ext}}$, where γ_i corresponds to exterior boundary edges depicting a normal flux, we once again differentiate between inflow edges $\{\gamma_i, i \in I_{\text{in}}\} := \{\gamma_i : \boldsymbol{q}\cdot\boldsymbol{n} \leq 0\}$ and outflow edges $\{\gamma_i, i \in I_{\text{out}}\} := \{\gamma_i : \boldsymbol{q}\cdot\boldsymbol{n} > 0\}$. In the interest of simplicity, we assume that the computational grid distinguishes exterior edges as either inflow or outflow edges when resolving the boundary. Furthermore, we assume the following conditions on inflow edges: $\boldsymbol{u} = \hat{\boldsymbol{u}}$ and $H = \hat{H}$ should hold pointwise. We get $\hat{q}_n := \hat{\boldsymbol{u}}\hat{H}\cdot\boldsymbol{n}$, where $\hat{\cdot}$ denotes a prescribed value for the corresponding unknown. On an outflow edge, if the discrete solution satisfies $\int_{\gamma_i} \boldsymbol{q}_\Delta \cdot \boldsymbol{n}\,\mathrm{d}s > 0$, this flux is retained and incorporated into the left-hand side of our estimate; otherwise, it is set to zero. No values are prescribed for the external boundary edges for the term $\sum_{i\in I_{\text{ext}}} \langle \frac{g}{2}H^2\boldsymbol{n}, \boldsymbol{\theta}\rangle_{\gamma_i}$ in (2.39). Again, we assume that the boundary condition functions fulfill the required regularity conditions.
We obtain the *semi-discrete finite element formulation* as in (2.11)–(2.12) on page 14.

Problem 3 (Semi-discrete finite element formulation)
We seek a solution $(H_\triangle, \boldsymbol{q}_\triangle, \boldsymbol{u}_\triangle)$ with $H_\triangle \in C^1(0,T;\mathcal{P}^p)$, $\boldsymbol{q}_\triangle \in \left(C^1(0,T;\mathcal{P}^p)\right)^2$ and $\boldsymbol{u}_\triangle \in \left(C^1\left(0,T;\mathcal{P}^{\max\{0,p-1\}}\right)\right)^2$ such that for a.e. $t \in (0,T)$ and for all test functions $\vartheta \in \mathcal{P}^p$, $\boldsymbol{\theta} \in (\mathcal{P}^p)^2$, and $\boldsymbol{\psi} \in \left(\mathcal{P}^{\max\{0,p-1\}}\right)^2$ the following holds:

$$\begin{aligned}&\sum_{e\in I_e}(\partial_t H_\triangle,\vartheta)_{\Omega_e} + \sum_{i\in I_{\text{int}}}\langle\{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}\cdot\boldsymbol{n},[\![\vartheta]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}}\langle\hat{q}_n,\vartheta\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}}\langle|\hat{q}_n|(H_\triangle-\hat{H}),\vartheta\rangle_{\gamma_i}\\&\quad+\sum_{i\in I_{\text{out}}}\langle\max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n},0\},\vartheta\rangle_{\gamma_i} - \sum_{e\in I_e}(\boldsymbol{q}_\triangle\cdot\nabla,\vartheta)_{\Omega_e} = 0, \end{aligned}\tag{2.43}$$

$$\begin{aligned}&\sum_{e\in I_e}(\partial_t\boldsymbol{q}_\triangle,\boldsymbol{\theta})_{\Omega_e} + \sum_{i\in I_{\text{int}}}\langle\{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}(\{\!\{\boldsymbol{u}_\triangle\}\!\}\cdot\boldsymbol{n}),[\![\boldsymbol{\theta}]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}}\langle\hat{\boldsymbol{u}}\hat{q}_n,\boldsymbol{\theta}\rangle_{\gamma_i}\\&\quad+3\sum_{i\in I_{\text{in}}}\langle|\hat{q}_n|\max\{|\hat{\boldsymbol{u}}|,|H_\triangle|\}(\boldsymbol{u}_\triangle-\hat{\boldsymbol{u}}),\boldsymbol{\theta}\rangle_{\gamma_i} + \sum_{i\in I_{\text{out}}}\langle\boldsymbol{u}_\triangle\max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n},0\},\boldsymbol{\theta}\rangle_{\gamma_i}\\&\quad-\sum_{e\in I_e}(\boldsymbol{u}_\triangle(\boldsymbol{q}_\triangle\cdot\nabla),\boldsymbol{\theta})_{\Omega_e} + \sum_{i\in I_{\text{int}}}\langle\tfrac{g}{2}\{\!\{H_\triangle^2\}\!\}\boldsymbol{n},[\![\boldsymbol{\theta}]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{ext}}}\langle\tfrac{g}{2}H_\triangle^2\boldsymbol{n},\boldsymbol{\theta}\rangle_{\gamma_i} - \sum_{e\in I_e}\left(\tfrac{g}{2}H_\triangle^2\nabla,\boldsymbol{\theta}\right)_{\Omega_e}\\&\quad+\sum_{e\in I_e}(C_f|\boldsymbol{u}_\triangle|\boldsymbol{u}_\triangle,\boldsymbol{\theta})_{\Omega_e} - \sum_{e\in I_e}\left(\left(\begin{smallmatrix}0 & -f_c\\ f_c & 0\end{smallmatrix}\right)\boldsymbol{q}_\triangle,\boldsymbol{\theta}\right)_{\Omega_e} - \sum_{e\in I_e}(gH_\triangle\nabla h_b,\boldsymbol{\theta})_{\Omega_e} = \sum_{e\in I_e}(\boldsymbol{F},\boldsymbol{\theta})_{\Omega_e},\end{aligned}\tag{2.44}$$

$$\sum_{e\in I_e}(\boldsymbol{u}_\triangle H_\triangle,\boldsymbol{\psi})_{\Omega_e} = \sum_{e\in I_e}(\boldsymbol{q}_\triangle,\boldsymbol{\psi})_{\Omega_e}.\tag{2.45}$$

To handle the boundary conditions without applying the inverse and trace inequalities, we add two penalty terms: $\sum_{i\in I_{\text{in}}}\langle|\hat{q}_n|(H_\triangle-\hat{H}),\vartheta\rangle_{\gamma_i}$ in (2.43) and $3\sum_{i\in I_{\text{in}}}\langle|\hat{q}_n|\max\{|\hat{\boldsymbol{u}}|,|H_\triangle|\}(\boldsymbol{u}_\triangle-\hat{\boldsymbol{u}}),\boldsymbol{\theta}\rangle_{\gamma_i}$ in (2.44).

In what follows, we present a stability theorem concerning Problem 3 that holds for $p \leq 2$. The remainder of this section is dedicated to its proof.

Theorem 6
Let $H_\triangle > 0$ and $p \leq 2$, then the following stability result holds for the solution of Problem 3:

$$\begin{aligned}&||\sqrt{H_\triangle(T)}\,\boldsymbol{u}_\triangle(T)||^2_{L^2(\Omega)} + g||H_\triangle(T)||^2_{L^2(\Omega)} + \int_0^T\langle|\hat{q}_n(t)|(1+\hat{H}(t)),\boldsymbol{u}_\triangle(t)\cdot\boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t\\&\quad+\int_0^T\langle\max\{\boldsymbol{q}_\triangle(t)\cdot\boldsymbol{n},0\},\boldsymbol{u}_\triangle(t)\cdot\boldsymbol{u}_\triangle(t)+2gH_\triangle(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t\\&\quad+\int_0^T\langle|\hat{q}_n(t)|\max\{|\hat{\boldsymbol{u}}(t)|,|H_\triangle(t)|\}\boldsymbol{u}_\triangle(t),\boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + g\int_0^T\langle|\hat{q}_n(t)|H_\triangle(t),H_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t\\&\quad+C_f\int_0^T||\boldsymbol{u}_\triangle(t)||^3_{L^3(\Omega)}\,\mathrm{d}t\\&\leq K\Bigg(||\sqrt{H_\triangle(0)}\,\boldsymbol{u}_\triangle(0)||^2_{L^2(\Omega)} + g||H_\triangle(0)||^2_{L^2(\Omega)} + \tfrac{1}{2}\int_0^T\langle|\hat{\boldsymbol{u}}(t)||\hat{q}_n(t)|,1\rangle_{\partial\Omega_i}\,\mathrm{d}t\\&\quad+\tfrac{9}{2}\int_0^T\langle|\hat{\boldsymbol{u}}(t)||\hat{q}_n(t)|\hat{\boldsymbol{u}}(t),\hat{\boldsymbol{u}}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \tfrac{81}{8g}\int_0^T\langle|\hat{q}_n(t)|\hat{\boldsymbol{u}}(t)\cdot\hat{\boldsymbol{u}}(t),\hat{\boldsymbol{u}}(t)\cdot\hat{\boldsymbol{u}}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t\\&\quad+4\int_0^T\langle g|\hat{q}_n(t)|,1\rangle_{\partial\Omega_i}\,\mathrm{d}t + 4\int_0^T\langle g|\hat{q}_n(t)|\hat{H}(t),\hat{H}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \tfrac{1}{4}\int_0^T g^4|\Omega|||\nabla h_b||^4_{L^\infty(\Omega)}\,\mathrm{d}t\\&\quad+\tfrac{4}{9}\sqrt{\tfrac{6}{C_f}}\int_0^T||\boldsymbol{F}(t)||^{\frac{3}{2}}_{L^{\frac{3}{2}}(\Omega)}\,\mathrm{d}t\Bigg),\end{aligned}\tag{2.46}$$

where $K = K(|\Omega|,T)$, and its dependence on T is exponential as shown in (2.33).

Proof. Recalling that $\boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle \in \mathcal{P}^{\max\{0,2p-2\}} \in \mathcal{P}^p$ for $p \leq 2$, we test (2.43) twice with $\vartheta = \frac{1}{2}\boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle$ and $\vartheta = gH_\triangle$, respectively, (2.44) with $\boldsymbol{\theta} = \boldsymbol{u}_\triangle$, and add the resulting three equations:

$$
\begin{aligned}
&\underbrace{\sum_{e\in I_e}\left(\partial_t H_\triangle, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_1} + \underbrace{\sum_{i\in I_{\text{int}}}\langle \{\!\!\{\boldsymbol{u}_\triangle\}\!\!\}\{\!\!\{H_\triangle\}\!\!\}\cdot\boldsymbol{n}, [\![\tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i}}_{\Lambda_2}\\
&\quad\underbrace{+\sum_{i\in I_{\text{in}}}\langle \hat{q}_n, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}}\langle |\hat{q}_n|(H_\triangle-\hat{H}), \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{out}}}\langle \max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n},0\}, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{\Lambda_3}\\
&\quad\underbrace{-\sum_{e\in I_e}\left(\boldsymbol{q}_\triangle\cdot\nabla, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_4} + \underbrace{\sum_{e\in I_e}\left(\partial_t H_\triangle, gH_\triangle\right)_{\Omega_e}}_{\Lambda_5} + \underbrace{\sum_{i\in I_{\text{int}}}\langle g\{\!\!\{\boldsymbol{u}_\triangle\}\!\!\}\{\!\!\{H_\triangle\}\!\!\}\cdot\boldsymbol{n}, [\![H_\triangle]\!]\rangle_{\gamma_i}}_{\Lambda_6}\\
&\quad\underbrace{+\sum_{i\in I_{\text{in}}}\langle g\hat{q}_n, H_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}}\langle g|\hat{q}_n|(H_\triangle-\hat{H}), H_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{out}}}\langle g\max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n},0\}, H_\triangle\rangle_{\gamma_i}}_{\Lambda_7}\\
&\quad\underbrace{-\sum_{e\in I_e}\left(g\boldsymbol{q}_\triangle\cdot\nabla, H_\triangle\right)_{\Omega_e}}_{\Lambda_8} + \underbrace{\sum_{e\in I_e}\left(\partial_t\boldsymbol{q}_\triangle, \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_9} + \underbrace{\sum_{i\in I_{\text{int}}}\langle \{\!\!\{\boldsymbol{u}_\triangle\}\!\!\}\{\!\!\{H_\triangle\}\!\!\}(\{\!\!\{\boldsymbol{u}_\triangle\}\!\!\}\cdot\boldsymbol{n}), [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i}}_{\Lambda_{10}}\\
&\quad\underbrace{+\sum_{i\in I_{\text{in}}}\langle \hat{\boldsymbol{u}}\hat{q}_n, \boldsymbol{u}_\triangle\rangle_{\gamma_i} + 3\sum_{i\in I_{\text{in}}}\langle |\hat{q}_n|\max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}(\boldsymbol{u}_\triangle-\hat{\boldsymbol{u}}), \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{out}}}\langle \boldsymbol{u}_\triangle\max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n},0\}, \boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{\Lambda_{11}}\\
&\quad\underbrace{-\sum_{e\in I_e}\left(\boldsymbol{u}_\triangle(\boldsymbol{q}_\triangle\cdot\nabla), \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_{12}} + \underbrace{\sum_{i\in I_{\text{int}}}\langle \tfrac{g}{2}\{\!\!\{H_\triangle^2\}\!\!\}\boldsymbol{n}, [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i}}_{\Lambda_{13}} + \underbrace{\sum_{i\in I_{\text{ext}}}\langle \tfrac{g}{2}H_\triangle^2\boldsymbol{n}, \boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{\Lambda_{14}}\\
&\quad\underbrace{-\sum_{e\in I_e}\left(\tfrac{g}{2}H_\triangle^2\nabla, \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_{15}} + \underbrace{\sum_{e\in I_e}\left(C_f|\boldsymbol{u}_\triangle|\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_{16}} \underbrace{-\sum_{e\in I_e}\left(\left(\begin{smallmatrix}0 & -f_c\\ f_c & 0\end{smallmatrix}\right)\boldsymbol{q}_\triangle, \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_{17}}\\
&\quad\underbrace{-\sum_{e\in I_e}\left(gH_\triangle\nabla h_b, \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_{18}} = \underbrace{\sum_{e\in I_e}\left(\boldsymbol{F}, \boldsymbol{u}_\triangle\right)_{\Omega_e}}_{\Lambda_{19}}.
\end{aligned}
$$

First, we note that the Coriolis terms, namely Λ_{17}, for the x- and y-momentum equations cancel each other out:

$$
-\sum_{e\in I_e}\left(\left(\begin{smallmatrix}0 & -f_c\\ f_c & 0\end{smallmatrix}\right)\boldsymbol{q}_\triangle, \boldsymbol{u}_\triangle\right)_{\Omega_e} \overset{(2.45)}{=} -\sum_{e\in I_e}\left(\left(\begin{smallmatrix}0 & -f_c\\ f_c & 0\end{smallmatrix}\right)\boldsymbol{u}_\triangle H_\triangle, \boldsymbol{u}_\triangle\right)_{\Omega_e} = 0.
$$

We start with Λ_9 and apply (2.45), the product rule, and (2.43):

$$\begin{aligned}
\sum_{e\in I_e} (\partial_t \boldsymbol{q}_\triangle, \boldsymbol{u}_\triangle)_{\Omega_e} &= \lim_{\Delta t\to 0} \sum_{e\in I_e} \left(\frac{\boldsymbol{q}_\triangle(t+\Delta t) - \boldsymbol{q}_\triangle(t)}{\Delta t}, \boldsymbol{u}_\triangle\right)_{\Omega_e} \\
&\overset{\substack{(2.45)\text{ since}\\ \boldsymbol{u}_\triangle\in\left(\mathcal{P}^{\max\{0,p-1\}}\right)^2}}{=} \lim_{\Delta t\to 0} \sum_{e\in I_e} \left(\frac{\boldsymbol{u}_\triangle(t+\Delta t)H_\triangle(t+\Delta t) - \boldsymbol{u}_\triangle(t)H_\triangle(t)}{\Delta t}, \boldsymbol{u}_\triangle\right)_{\Omega_e} = \sum_{e\in I_e} (\partial_t(\boldsymbol{u}_\triangle H_\triangle), \boldsymbol{u}_\triangle)_{\Omega_e} \\
&\overset{\substack{\text{prod.}\\ \text{rule}}}{=} \sum_{e\in I_e} (\partial_t H_\triangle, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle)_{\Omega_e} + \sum_{e\in I_e} (\partial_t \boldsymbol{u}_\triangle, H_\triangle \boldsymbol{u}_\triangle)_{\Omega_e} \\
&\overset{\substack{(2.43)\text{ since}\\ \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\in\mathcal{P}^p(\Omega)}}{=} \underbrace{- \sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}\cdot\boldsymbol{n}, [\![\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i}}_{\Lambda_{9a}} \\
&\quad \underbrace{- \sum_{i\in I_{\text{in}}} \langle \hat{q}_n, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} - \sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|(H_\triangle - \hat{H}), \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} - \sum_{i\in I_{\text{out}}} \langle \max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n}, 0\}, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{\Lambda_{9b}} \\
&\quad + \underbrace{\sum_{e\in I_e} (\boldsymbol{q}_\triangle\cdot\nabla, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle)_{\Omega_e}}_{\Lambda_{9c}} + \underbrace{\sum_{e\in I_e} \left(\tfrac{1}{2}\partial_t(\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle), H_\triangle\right)_{\Omega_e}}_{\Lambda_{9d}}.
\end{aligned}$$

Then, we add Λ_1 and Λ_{9d}:

$$\sum_{e\in I_e} \left(\partial_t H_\triangle, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\right)_{\Omega_e} + \sum_{e\in I_e} \left(\tfrac{1}{2}\partial_t(\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle), H_\triangle\right)_{\Omega_e} = \tfrac{1}{2}\sum_{e\in I_e} (\partial_t(H_\triangle \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle), 1)_{\Omega_e} = \Upsilon_1'.$$

Now adding Λ_4, Λ_2, Λ_3, Λ_{9c}, Λ_{9a}, Λ_{9b}, Λ_{12}, Λ_{10}, and Λ_{11} leads to:

$$\begin{aligned}
&- \sum_{e\in I_e} \left(\boldsymbol{q}_\triangle\cdot\nabla, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\right)_{\Omega_e} + \sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}\cdot\boldsymbol{n}, [\![\tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}} \langle \hat{q}_n, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
&+ \sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|(H_\triangle - \hat{H}), \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{out}}} \langle \max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n}, 0\}, \tfrac{1}{2}\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
&+ \sum_{e\in I_e} (\boldsymbol{q}_\triangle\cdot\nabla, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle)_{\Omega_e} - \sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}\cdot\boldsymbol{n}, [\![\boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} - \sum_{i\in I_{\text{in}}} \langle \hat{q}_n, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
&- \sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|(H_\triangle - \hat{H}), \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} - \sum_{i\in I_{\text{out}}} \langle \max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n}, 0\}, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
&- \sum_{e\in I_e} (\boldsymbol{u}_\triangle(\boldsymbol{q}_\triangle\cdot\nabla), \boldsymbol{u}_\triangle)_{\Omega_e} + \sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}(\{\!\{\boldsymbol{u}_\triangle\}\!\}\cdot\boldsymbol{n}), [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}} \langle \hat{\boldsymbol{u}}\hat{q}_n, \boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
&+ 3\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}(\boldsymbol{u}_\triangle - \hat{\boldsymbol{u}}), \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{out}}} \langle \boldsymbol{u}_\triangle \max\{\boldsymbol{q}_\triangle\cdot\boldsymbol{n}, 0\}, \boldsymbol{u}_\triangle\rangle_{\gamma_i} = (\star_1).
\end{aligned}$$

Reordering the terms, we get

$$\begin{aligned}
(\star_1) =& \cancel{\sum_{e\in I_e} \left(\tfrac{1}{2}\boldsymbol{q}_\triangle \cdot \nabla, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\right)_{\Omega_e}} - \tfrac{1}{2}\sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\} \cdot \boldsymbol{n}, [\![\boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} - \tfrac{1}{2}\sum_{i\in I_{\text{in}}} \langle \hat{q}_n, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
& - \tfrac{1}{2}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|(H_\triangle - \hat{H}), \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \tfrac{1}{2}\sum_{i\in I_{\text{out}}} \langle \max\{\boldsymbol{q}_\triangle \cdot \boldsymbol{n}, 0\}, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
& \underbrace{- \sum_{e\in I_e} (\boldsymbol{u}_\triangle(\boldsymbol{q}_\triangle \cdot \nabla), \boldsymbol{u}_\triangle)_{\Omega_e}}_{=-\cancel{\sum_{e\in I_e}\left(\frac{1}{2}\boldsymbol{q}_\triangle\cdot\nabla, \boldsymbol{u}_\triangle\cdot\boldsymbol{u}_\triangle\right)_{\Omega_e}}} + \sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}(\{\!\{\boldsymbol{u}_\triangle\}\!\} \cdot \boldsymbol{n}), [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}} \langle \hat{\boldsymbol{u}}\hat{q}_n, \boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
& + 3\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}(\boldsymbol{u}_\triangle - \hat{\boldsymbol{u}}), \boldsymbol{u}_\triangle\rangle_{\gamma_i}.
\end{aligned}$$

We note that $\sum_{i\in I_{\text{int}}} \langle \{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}(\{\!\{\boldsymbol{u}_\triangle\}\!\} \cdot \boldsymbol{n}), [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} = \sum_{i\in I_{\text{int}}} \langle \tfrac{1}{2}\{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\} \cdot \boldsymbol{n}, [\![\boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i}$ and see that the interior edge integrals also cancel. So do the element integrals and what is left are the inflow and outflow boundary terms. To begin with, the outflow boundary term and the first inflow boundary term are non-negative and thus incorporated into the left-hand side:

$$-\tfrac{1}{2}\sum_{i\in I_{\text{in}}} \langle \hat{q}_n, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \tfrac{1}{2}\sum_{i\in I_{\text{out}}} \langle \max\{\boldsymbol{q}_\triangle \cdot \boldsymbol{n}, 0\}, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} = \Upsilon_2'.$$

The remaining inflow terms are the following ones:

$$\begin{aligned}
& - \sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|(H_\triangle - \hat{H}), \tfrac{1}{2}\boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}} \langle \hat{\boldsymbol{u}}\hat{q}_n, \boldsymbol{u}_\triangle\rangle_{\gamma_i} + 3\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}(\boldsymbol{u}_\triangle - \hat{\boldsymbol{u}}), \boldsymbol{u}_\triangle\rangle_{\gamma_i} \\
& = \underbrace{\tfrac{1}{2}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|\hat{H}, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} + 3\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{\geq 0 = \Upsilon_3'} \\
& \quad \underbrace{-\tfrac{1}{2}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|H_\triangle, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{=(\star_2)\leq\frac{1}{2}\sum_{i\in I_{\text{in}}}\langle|\hat{q}_n|\max\{|\hat{\boldsymbol{u}}|,||H_\triangle|\}\boldsymbol{u}_\triangle,\boldsymbol{u}_\triangle\rangle_{\gamma_i}} + \underbrace{\sum_{i\in I_{\text{in}}} \langle \hat{\boldsymbol{u}}\hat{q}_n, \boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{=(\star_3)} \underbrace{-3\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}\hat{\boldsymbol{u}}, \boldsymbol{u}_\triangle\rangle_{\gamma_i}}_{=(\star_4)}.
\end{aligned}$$

Then, Υ_3' is non-negative and stays on the left-hand side and $(\star_3)$ is estimated using Young's inequality (2.32):

$$(\star_3) = \sum_{i\in I_{\text{in}}} \langle \hat{\boldsymbol{u}}\hat{q}_n, \boldsymbol{u}_\triangle\rangle_{\gamma_i} \overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=2}}{\leq} \sum_{i\in I_{\text{in}}} \langle |\hat{\boldsymbol{u}}||\hat{q}_n|, \boldsymbol{u}_\triangle \cdot \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \tfrac{1}{4}\sum_{i\in I_{\text{in}}} \langle |\hat{\boldsymbol{u}}||\hat{q}_n|, 1\rangle_{\gamma_i}.$$

We look at the two possible cases for $(\star_4)$ and apply Young's inequality (2.32) again:
Case 1: $\langle |\hat{\boldsymbol{u}}||\hat{q}_n|, 1\rangle_{\gamma_i} \geq \langle |\hat{q}_n||H_\triangle|, 1\rangle_{\gamma_i}$

$$-3\langle |\hat{\boldsymbol{u}}||\hat{q}_n|\hat{\boldsymbol{u}}, \boldsymbol{u}_\triangle\rangle_{\gamma_i} \overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=\frac{3}{2}}}{\leq} \tfrac{9}{4}\langle |\hat{\boldsymbol{u}}||\hat{q}_n|\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\rangle_{\gamma_i} + \langle |\hat{\boldsymbol{u}}||\hat{q}_n|\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\rangle_{\gamma_i},$$

Case 2: $\langle |\hat{\boldsymbol{u}}||\hat{q}_n|, 1\rangle_{\gamma_i} < \langle |\hat{q}_n||H_\triangle|, 1\rangle_{\gamma_i}$

$$\begin{aligned}
-3\langle |\hat{q}_n||H_\triangle|\hat{\boldsymbol{u}}, \boldsymbol{u}_\triangle\rangle_{\gamma_i} &\overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=\frac{3}{2}}}{\leq} \tfrac{9}{4}\langle |\hat{q}_n||H_\triangle|\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\rangle_{\gamma_i} + \langle |\hat{q}_n||H_\triangle|\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\rangle_{\gamma_i}\\
&\overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=\frac{2g}{9}}}{\leq} \tfrac{1}{4}\langle |\hat{q}_n||H_\triangle|^2, g\rangle_{\gamma_i} + \tfrac{81}{16g}\langle |\hat{q}_n|\hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\gamma_i} + \langle |\hat{q}_n||H_\triangle|\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\rangle_{\gamma_i},
\end{aligned}$$

and therefore, we get

$$\begin{aligned}
(\star_4) \leq &\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \tfrac{9}{4}\sum_{i\in I_{\text{in}}} \langle |\hat{\boldsymbol{u}}||\hat{q}_n|\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\rangle_{\gamma_i} + \tfrac{1}{4}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n||H_\triangle|^2, g\rangle_{\gamma_i}\\
&+ \tfrac{81}{16g}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|\hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\gamma_i}.
\end{aligned}$$

And we estimate

$$\begin{aligned}
(\star_2) + (\star_3) + (\star_4) \leq &\tfrac{5}{2}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n| \max\{|\hat{\boldsymbol{u}}|, |H_\triangle|\}\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle\rangle_{\gamma_i} + \tfrac{1}{4}\sum_{i\in I_{\text{in}}} \langle |\hat{\boldsymbol{u}}||\hat{q}_n|, 1\rangle_{\gamma_i}\\
&+ \tfrac{9}{4}\sum_{i\in I_{\text{in}}} \langle |\hat{\boldsymbol{u}}||\hat{q}_n|\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\rangle_{\gamma_i} + \tfrac{1}{4}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n||H_\triangle|^2, g\rangle_{\gamma_i} + \tfrac{81}{16g}\sum_{i\in I_{\text{in}}} \langle |\hat{q}_n|\hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}, \hat{\boldsymbol{u}}\cdot\hat{\boldsymbol{u}}\rangle_{\gamma_i} = \Upsilon_4'.
\end{aligned}$$

Then, we proceed by adding Λ_8, Λ_6, Λ_{15}, Λ_{13} and Λ_{14}. Since for $p \leq 2$ we have $\nabla H_\triangle \in \left(\mathcal{P}^{\max\{0,p-1\}}\right)^2$ and we apply (2.45) to

$$-\sum_{e\in I_e} (g\boldsymbol{q}_\triangle\cdot\nabla, H_\triangle)_{\Omega_e} \overset{(2.45)}{=} -\sum_{e\in I_e} (g\boldsymbol{u}_\triangle H_\triangle\cdot\nabla, H_\triangle)_{\Omega_e} = -\sum_{e\in I_e} \left(\tfrac{g}{2}\boldsymbol{u}_\triangle\cdot\nabla, H_\triangle^2\right)_{\Omega_e}$$

and obtain the following expression after applying Gauss' theorem:

$$\begin{aligned}
&\underbrace{-\sum_{e\in I_e} (g\boldsymbol{q}_\triangle\cdot\nabla, H_\triangle)_{\Omega_e}}_{=-\sum_{e\in I_e}\left(\frac{g}{2}\boldsymbol{u}_\triangle\cdot\nabla, H_\triangle^2\right)_{\Omega_e}} + \sum_{i\in I_{\text{int}}} \langle g\{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}\cdot\boldsymbol{n}, [\![H_\triangle]\!]\rangle_{\gamma_i} - \sum_{e\in I_e}\left(\tfrac{g}{2}H_\triangle^2\nabla, \boldsymbol{u}_\triangle\right)_{\Omega_e}\\
&\quad + \sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}\{\!\{H_\triangle^2\}\!\}\boldsymbol{n}, [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} + \sum_{i\in I_{\text{ext}}} \langle \tfrac{g}{2}H_\triangle^2\boldsymbol{n}, \boldsymbol{u}_\triangle\rangle_{\gamma_i}\\
&\overset{\text{Gauss}}{=} -\sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}, [\![H_\triangle^2\boldsymbol{u}_\triangle]\!]\cdot\boldsymbol{n}\rangle_{\gamma_i} - \cancel{\sum_{i\in I_{\text{ext}}} \langle \tfrac{g}{2}H_\triangle^2\boldsymbol{u}_\triangle\cdot\boldsymbol{n}, 1\rangle_{\gamma_i}} + \sum_{i\in I_{\text{int}}} \langle g\{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}\cdot\boldsymbol{n}, [\![H_\triangle]\!]\rangle_{\gamma_i}\\
&\quad + \sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}\{\!\{H_\triangle^2\}\!\}\boldsymbol{n}, [\![\boldsymbol{u}_\triangle]\!]\rangle_{\gamma_i} + \cancel{\sum_{i\in I_{\text{ext}}} \langle \tfrac{g}{2}H_\triangle^2\boldsymbol{n}, \boldsymbol{u}_\triangle\rangle_{\gamma_i}} = (\star_5).
\end{aligned}$$

We note that

$$\begin{aligned}
-\sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}, [\![H_\triangle^2\boldsymbol{u}_\triangle]\!]\cdot\boldsymbol{n}\rangle_{\gamma_i} &= -\sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}\{\!\{\boldsymbol{u}_\triangle\}\!\}, [\![H_\triangle^2]\!]\cdot\boldsymbol{n}\rangle_{\gamma_i} - \sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}\{\!\{H_\triangle^2\}\!\}, [\![\boldsymbol{u}_\triangle]\!]\cdot\boldsymbol{n}\rangle_{\gamma_i}\\
&= -\sum_{i\in I_{\text{int}}} \langle g\{\!\{\boldsymbol{u}_\triangle\}\!\}\{\!\{H_\triangle\}\!\}, [\![H_\triangle]\!]\cdot\boldsymbol{n}\rangle_{\gamma_i} - \sum_{i\in I_{\text{int}}} \langle \tfrac{g}{2}\{\!\{H_\triangle^2\}\!\}, [\![\boldsymbol{u}_\triangle]\!]\cdot\boldsymbol{n}\rangle_{\gamma_i},
\end{aligned}$$

and get $(\star_5) = 0$.

Next, we look at Λ_7:

$$\underbrace{\sum_{i\in I_{\text{in}}} \langle g\hat{q}_n, H_\triangle\rangle_{\gamma_i}}_{(\star_6)} + \underbrace{\sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|(H_\triangle - \hat{H}), H_\triangle\rangle_{\gamma_i}}_{(\star_7)} + \underbrace{\sum_{i\in I_{\text{out}}} \langle g \max\{\boldsymbol{q}_\triangle \cdot \boldsymbol{n}, 0\}, H_\triangle\rangle_{\gamma_i}}_{=\Upsilon_5'\geq 0}.$$

The outflow boundary term is non-negative and thus incorporated into the left-hand side. The penalty inflow term is split into the following parts:

$$\begin{aligned}(\star_6) + (\star_7) &= \sum_{i\in I_{\text{in}}} \langle g\hat{q}_n, H_\triangle\rangle_{\gamma_i} + \sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|(H_\triangle - \hat{H}), H_\triangle\rangle_{\gamma_i} \\ &= \sum_{i\in I_{\text{in}}} \langle g\hat{q}_n, H_\triangle\rangle_{\gamma_i} + \underbrace{\sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|H_\triangle, H_\triangle\rangle_{\gamma_i}}_{=\Upsilon_6'\geq 0} - \sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|\hat{H}, H_\triangle\rangle_{\gamma_i}.\end{aligned}$$

The non-negative term is incorporated into the left-hand side and the remaining two terms are estimated using Young's inequality (2.32):

$$\begin{aligned}&\sum_{i\in I_{\text{in}}} \langle g\hat{q}_n, H_\triangle\rangle_{\gamma_i} - \sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|\hat{H}, H_\triangle\rangle_{\gamma_i} \\ &\overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=4}}{\leq} 2\sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|, 1\rangle_{\gamma_i} + 2\sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|\hat{H}, \hat{H}\rangle_{\gamma_i} + \tfrac{1}{4}\sum_{i\in I_{\text{in}}} \langle g|\hat{q}_n|H_\triangle, H_\triangle\rangle_{\gamma_i} = \Upsilon_7'.\end{aligned}$$

We rewrite the bottom friction term Λ_{16}:

$$\sum_{e\in I_e} (C_f|\boldsymbol{u}_\triangle|\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle)_{\Omega_e} = C_f\sum_{e\in I_e} (|\boldsymbol{u}_\triangle|\boldsymbol{u}_\triangle, \boldsymbol{u}_\triangle)_{\Omega_e} = C_f||\boldsymbol{u}_\triangle||^3_{L^3(\Omega)} = \Upsilon_8'.$$

We estimate Λ_{18} with Young's inequality (2.32) and Hölder's inequality (2.31) and put the terms to the right-hand side:

$$\begin{aligned}&\Big|-\sum_{e\in I_e} (gH_\triangle\nabla h_b, \boldsymbol{u}_\triangle)_{\Omega_e}\Big| \overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=1}}{\leq} \tfrac{1}{2}g^2||\nabla h_b||^2_{L^\infty(\Omega)}\sum_{e\in I_e}(1, H_\triangle)_{\Omega_e} + \tfrac{1}{2}\sum_{e\in I_e}\left(\sqrt{H_\triangle}\boldsymbol{u}_\triangle, \sqrt{H_\triangle}\boldsymbol{u}_\triangle\right)_{\Omega_e} \\ &\overset{\substack{(2.31)\text{ with}\\ p=q=2}}{\leq} \tfrac{1}{2}\left(g^2||\nabla h_b||^2_{L^\infty(\Omega)}||1||_{L^2(\Omega)}||H_\triangle||_{L^2(\Omega)}\right) + \tfrac{1}{2}||\sqrt{H_\triangle}\boldsymbol{u}_\triangle||^2_{L^2(\Omega)} \\ &\overset{\substack{(2.32)\text{ with}\\ p=q=2,\varepsilon=\frac{1}{2}}}{\leq} \tfrac{1}{8}g^4||\nabla h_b||^4_{L^\infty(\Omega)}\underbrace{||1||^2_{L^2(\Omega)}}_{=|\Omega|} + \tfrac{1}{2}||H_\triangle||^2_{L^2(\Omega)} + \tfrac{1}{2}||\sqrt{H_\triangle}\boldsymbol{u}_\triangle||^2_{L^2(\Omega)} = \Upsilon_9'.\end{aligned}$$

Next, we estimate Λ_{19} with Young's inequality (2.32):

$$\sum_{e\in I_e} (\boldsymbol{F}, \boldsymbol{u}_\triangle)_{\Omega_e} \overset{\substack{(2.32)\text{ with}\\ p=\frac{3}{2},q=3,\varepsilon=\sqrt{\frac{2}{3C_f}}}}{\leq} \tfrac{2}{9}\sqrt{\tfrac{6}{C_f}}||\boldsymbol{F}||^{\frac{3}{2}}_{L^{\frac{3}{2}}(\Omega)} + \tfrac{C_f}{2}||\boldsymbol{u}_\triangle||^3_{L^3(\Omega)} = \Upsilon_{10}'.$$

The only term left is Λ_5:

$$\sum_{e\in I_e} (\partial_t H_\triangle, gH_\triangle)_{\Omega_e} = \tfrac{g}{2}\sum_{e\in I_e} \left(\partial_t(H_\triangle^2), 1\right)_{\Omega_e} = \Upsilon'_{11}.$$

Lastly, we integrate everything from 0 to T and obtain the following from $\Upsilon'_1, \dots, \Upsilon'_{11}$:

$$\begin{aligned}
\Upsilon'_1 :\quad & \tfrac{1}{2}\int_0^T \sum_{e\in I_e} (\partial_t(H_\triangle(t)\boldsymbol{u}_\triangle(t)\cdot\boldsymbol{u}_\triangle(t)), 1)_{\Omega_e}\,\mathrm{d}t \\
&= \tfrac{1}{2}\left(||\sqrt{H_\triangle(T)}\,\boldsymbol{u}_\triangle(T)||^2_{L^2(\Omega)} - ||\sqrt{H_\triangle(0)}\,\boldsymbol{u}_\triangle(0)||^2_{L^2(\Omega)}\right) = \Upsilon_1, \\
\Upsilon'_2 :\quad & \underbrace{-\tfrac{1}{2}\int_0^T \langle \hat{q}_n(t), \boldsymbol{u}_\triangle(t)\cdot\boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \tfrac{1}{2}\int_0^T \langle \max\{\boldsymbol{q}_\triangle(t)\cdot\boldsymbol{n}, 0\}, \boldsymbol{u}_\triangle(t)\cdot\boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t}_{\geq 0} = \Upsilon_2, \\
\Upsilon'_3 :\quad & \underbrace{\tfrac{1}{2}\int_0^T \langle |\hat{q}_n(t)|\hat{H}(t), \boldsymbol{u}_\triangle(t)\cdot\boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t}_{\geq 0} \\
&+ \underbrace{3\int_0^T \langle |\hat{q}_n(t)| \max\{|\hat{\boldsymbol{u}}(t)|, |H_\triangle(t)|\}\boldsymbol{u}_\triangle(t), \boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t}_{\geq 0} = \Upsilon_3, \\
\Upsilon'_4 :\quad & \tfrac{5}{2}\int_0^T \langle |\hat{q}_n(t)| \max\{|\hat{\boldsymbol{u}}(t)|, |H_\triangle(t)|\}\boldsymbol{u}_\triangle(t), \boldsymbol{u}_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \tfrac{1}{4}\int_0^T \langle |\hat{\boldsymbol{u}}(t)||\hat{q}_n(t)|, 1\rangle_{\partial\Omega_i}\,\mathrm{d}t \\
&+ \tfrac{9}{4}\int_0^T \langle |\hat{\boldsymbol{u}}(t)||\hat{q}_n(t)|\hat{\boldsymbol{u}}(t), \hat{\boldsymbol{u}}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t + \tfrac{1}{4}\int_0^T \langle |\hat{q}_n(t)||H_\triangle(t)|^2, g\rangle_{\partial\Omega_i}\,\mathrm{d}t \\
&+ \tfrac{81}{16g}\int_0^T \langle |\hat{q}_n(t)|\hat{\boldsymbol{u}}(t)\cdot\hat{\boldsymbol{u}}(t), \hat{\boldsymbol{u}}(t)\cdot\hat{\boldsymbol{u}}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t = \Upsilon_4, \\
\Upsilon'_5 :\quad & \underbrace{g\int_0^T \langle \max\{\boldsymbol{q}_\triangle(t)\cdot\boldsymbol{n}, 0\}, H_\triangle(t)\rangle_{\partial\Omega_o}\,\mathrm{d}t}_{\geq 0} = \Upsilon_5, \\
\Upsilon'_6 :\quad & \underbrace{g\int_0^T \langle |\hat{q}_n(t)|H_\triangle(t), H_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t}_{\geq 0} = \Upsilon_6, \\
\Upsilon'_7 :\quad & 2\int_0^T \langle g|\hat{q}_n(t)|, 1\rangle_{\partial\Omega_i}\,\mathrm{d}t + 2\int_0^T \langle g|\hat{q}_n(t)|\hat{H}(t), \hat{H}(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t \\
&+ \tfrac{1}{4}\int_0^T \langle g|\hat{q}_n(t)|H_\triangle(t), H_\triangle(t)\rangle_{\partial\Omega_i}\,\mathrm{d}t = \Upsilon_7, \\
\Upsilon'_8 :\quad & C_f\int_0^T ||\boldsymbol{u}_\triangle(t)||^3_{L^3(\Omega)}\,\mathrm{d}t = \Upsilon_8, \\
\Upsilon'_9 :\quad & \tfrac{1}{8}\int_0^T g^4|\Omega|||\nabla h_b||^4_{L^\infty(\Omega)}\,\mathrm{d}t + \tfrac{1}{2}\int_0^T ||H_\triangle(t)||^2_{L^2(\Omega)}\,\mathrm{d}t + \tfrac{1}{2}\int_0^T ||\sqrt{H_\triangle(t)}\boldsymbol{u}_\triangle(t)||^2_{L^2(\Omega)}\,\mathrm{d}t = \Upsilon_9, \\
\Upsilon'_{10} :\quad & \tfrac{2}{9}\sqrt{\tfrac{6}{C_f}}\int_0^T ||\boldsymbol{F}(t)||^{\frac{3}{2}}_{L^{\frac{3}{2}}(\Omega)}\,\mathrm{d}t + \tfrac{C_f}{2}\int_0^T ||\boldsymbol{u}_\triangle(t)||^3_{L^3(\Omega)}\,\mathrm{d}t = \Upsilon_{10}, \\
\Upsilon'_{11} :\quad & \tfrac{g}{2}\int_0^T \sum_{e\in I_e} \left(\partial_t(H_\triangle(t)^2), 1\right)_{\Omega_e}\,\mathrm{d}t = \tfrac{g}{2}\left(||H_\triangle(T)||^2_{L^2(\Omega)} - ||H_\triangle(0)||^2_{L^2(\Omega)}\right) = \Upsilon_{11}.
\end{aligned}$$

Then $||H_\triangle(0)||^2_{L^2(\Omega)}$ and $||\sqrt{H_\triangle(0)}\,\boldsymbol{u}_\triangle(0)||^2_{L^2(\Omega)}$ are also put to the right-hand side.

Finally, we apply Grönwall's inequality to $\frac{1}{2}\int_0^T ||H_\triangle(t)||^2_{L^2(\Omega)}\,\mathrm{d}t$ and $\frac{1}{2}\int_0^T ||\sqrt{H_\triangle(t)}\boldsymbol{u}_\triangle(t)||^2_{L^2(\Omega)}\,\mathrm{d}t$ from Υ_9 and arrive at the stated stability result after adding $\Upsilon_1,\dots,\Upsilon_{11}$, multiplying by 2, and reordering the terms. $\square$

This proof only holds for $p \leq 2$, however, numerical experiments confirm the stability also for higher orders.

Remark 2

- *In contrast to the stability estimate in [Aiz04], the above one does not make use of the trace and inverse inequalities, and therefore contains no terms with negative exponents of $\triangle$ and is still valid when approaching the limit $\triangle \to 0$.*
- *If the bathymetry is constant as assumed in [Aiz04], Λ_{18} and consequently Υ_9 vanish and the proof does not need to make use of Grönwall's inequality. The constant in* (2.46) *is then independent of T.*

Chapter

Software and computational grids 3

We implemented the discretization presented in Chapter 2 in a Python frontend to an automatic code generation framework. Further elaboration on this implementation is provided in Section 3.1 building upon the published work [FN+20] and the preprint [FN+23a]. For the simulation of realistic ocean domains, we employ block-structured grids, which are described in Section 3.2 based on the published articles [FN+20] and [FN+23b]. Our reference code is introduced in Section 3.3.

3.1 Code generation and implementation details

Code generation techniques in conjunction with *domain specific languages* (DSLs) are experiencing a boom in popularity within the field of computational science and engineering applications [Kos+10; Haw11; DS15]. These techniques empower application scientists to describe numerical models using abstract formulations, which are automatically translated into efficient code tailored to the specific characteristics of the target hardware. We give an overview of automatic code generation and DSLs, while also providing a detailed exposition of our own implementation.

A paramount advantage of automatic code generation is the attainment of performance portability across diverse hardware architectures. We highlight some particularly relevant approaches in our field. The earliest framework in this domain is ATMOL [Eng02], short for the atmospheric modeling language, which undergoes translation into FORTRAN with the aid of a code synthesis tool. Within the ICON climate model context, the ICON DSL [Tor+13] is embedded in FORTRAN, facilitating the generation of optimized FORTRAN code. PSyclone[16] [Ada+19] is a domain-specific compiler, also embedded in Fortran and used in the LFRic Project. GridTools[17] [Afa+21] is a DSL framework engineered for enhancing performance portability within the domain of weather and climate applications. It is embedded in C++ and successfully used to accelerate the dynamical core of the COSMO weather forecasting model. To enhance accessibility,

[16]`https://psyclone.readthedocs.io/en/stable`
[17]`https://github.com/GridTools/gridtools`

it also features a Python frontend, called GT4Py[18] [BN+22]. The precursor of GridTools, known as STELLA [Gys+15], an acronym for stencil loop language, is similarly integrated within C++. In the broader context of solving PDEs, the automated system Firedrake[19] [Rat+16] takes on a central role, using the FE method and the Unified Form Language DSL. A comprehensive overview of existing code generation and DSL frameworks is elucidated in [Kuc19].

This thesis focuses on utilizing and extending the open-source ExaStencils code generation framework[20] [Len+20].

3.1.1 State-of-the-art prior to this work

The ExaStencils framework introduces its own multi-layered external DSL, known as ExaSlang, tailored to the requirements of developing numerical models across diverse applications that need to solve PDEs [Kuc19]. It does not rely on other simulation software packages, opting instead to handle the entire composition of the simulation program through synthesizing input data, application parameters, and problem descriptions formulated in ExaSlang. Its output is a highly optimized and massively parallel code, amenable for execution on a range of hardware platforms encompassing CPU [KK16] and GPU clusters [KK18a; Len+20], as well as Field Programmable Gate Arrays (FPGAs) [Sch+18]. For automatic optimizations, ExaStencils supplies code transformations including but not limited to common subexpression elimination, polyhedral loop transformations, explicit single instruction multiple data (SIMD) vectorization, and address pre-calculation [Kro20].

ExaSlang itself offers different levels of abstraction (cf. Figure 3.1 (left), page 49): On layer 1 of ExaSlang, users may specify a continuous problem description, including functions, operators, governing equations, and boundary conditions, whereas layer 2 is concerned with their discretized counterparts and supports FD, FV, and FE discretizations. Specific solver algorithms can be expressed on layer 3 through mathematical formalism. Layer 4 completes the pipeline and is the most comprehensive layer of the ExaSlang DSL. It can accommodate the whole program specification, permitting data structure specialization, data communication strategies, input-output operations, and visualization mechanisms.

3.1.2 Necessary extensions for DG simulations of the SWE

The original ExaStencils toolchain did not support incorporating higher-order discretizations, such as the quadrature-free DG formulation. The strategy to integrate this new discretization was influenced by the necessity to fulfill several prerequisites. Primarily, the abstract representations

[18]`https://github.com/GridTools/gt4py`
[19]`https://www.firedrakeproject.org`
[20]`https://github.com/lssfau/ExaStencils`

employed for the components must maintain a balance: they ought to closely mirror the mathematical expressions while also easily translating into efficient code. Additionally, the structural framework or context of the abstract representation must gain acceptance within the user community.

To this end, within this thesis, we developed the open-source Python frontend GHODDESS[21] (Generation of Higher-Order Discretizations Deployed as ExaSlang Specifications) as an extension to the ExaStencils code generation framework. GHODDESS leverages the capabilities of the SymPy symbolic algebra package[22] [Meu+17] for the analytical evaluation of integrals and derivatives. The principal task of GHODDESS involves translating our DG scheme, implemented as a series of symbolic SymPy expressions, to ExaSlang layer 4.

Furthermore, one should note that ExaStencils is restricted to quadrilateral grids while our discretization in Section 2.3 aims for triangular ones. Within GHODDESS, we address this limitation by conceptually partitioning each element into two differently oriented triangles – lower and upper – in order to obtain a triangular grid.

In the subsequent sections, we outline the principal stages of our approach to map a mathematical model represented by a system of PDEs into efficient simulation code and describe each of them in detail. For comprehensive documentation and user-oriented tutorials, readers are directed to the ExaStencils homepage[23].

3.1.3 Mapping discretization to dedicated Python frontend

Working with GHODDESS is facilitated by already provided basic abstractions, including classes representing basis functions, triangles, and data fields.

The algebraic representations of the discrete scheme were implemented manually in our framework. For illustration, we consider a part of the element integral for ξ. Recalling Equation (2.22), we demonstrate its mapping to the reference basis (2.18), which results in

$$\begin{aligned}(\boldsymbol{A}(\boldsymbol{c}_\triangle, \boldsymbol{u}_\triangle), \nabla(\varphi_{eq}\boldsymbol{e}_1))_{\Omega_e} &= \sum_{i=1}^{P(p)} \left[c_{ei}^2 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial x} \varphi_{ei} \mathrm{d}\boldsymbol{x} + c_{ei}^3 \int_{\Omega_e} \frac{\partial \varphi_{eq}}{\partial y} \varphi_{ei} \mathrm{d}\boldsymbol{x} \right] \\ &= \sum_{i=1}^{P(p)} \left[\frac{c_{ei}^2}{|\det(\boldsymbol{B}_e)|} \left(B_{2,2}^e \int_{\hat\Omega} \frac{\partial \hat\varphi_q}{\partial \hat x} \hat\varphi_i \mathrm{d}\hat{\boldsymbol{x}} - B_{2,1}^e \int_{\hat\Omega} \frac{\partial \hat\varphi_q}{\partial \hat y} \hat\varphi_i \mathrm{d}\hat{\boldsymbol{x}} \right) \right. \\ &\quad \left. + \frac{c_{ei}^3}{|\det(\boldsymbol{B}_e)|} \left(-B_{1,2}^e \int_{\hat\Omega} \frac{\partial \hat\varphi_q}{\partial \hat x} \hat\varphi_i \mathrm{d}\hat{\boldsymbol{x}} + B_{1,1}^e \int_{\hat\Omega} \frac{\partial \hat\varphi_q}{\partial \hat y} \hat\varphi_i \mathrm{d}\hat{\boldsymbol{x}} \right) \right] \end{aligned} \tag{3.1}$$

for $q \in 1, \ldots, P(q)$.

This expression is further explained in an illustrative example in the context of GHODDESS. Specifically, the first component, that is, $\sum_{i=1}^{P(p)} \frac{c_{ei}^2}{|\det(\boldsymbol{B}_e)|} \left(B_{2,2}^e \int_{\hat\Omega} \frac{\partial \hat\varphi_q}{\partial \hat x} \hat\varphi_i \mathrm{d}\hat{\boldsymbol{x}} - B_{2,1}^e \int_{\hat\Omega} \frac{\partial \hat\varphi_q}{\partial \hat y} \hat\varphi_i \mathrm{d}\hat{\boldsymbol{x}} \right)$, can be implemented as shown in Listing 3.1 in GHODDESS.

[21]`https://i10git.cs.fau.de/ocean/ghoddess-release`
[22]`https://www.sympy.org`
[23]`https://www.exastencils.fau.de`

```
sum(cu(tri.orientation, i) * tri.det_b_inv * (
  tri.b[1, 1] * integrate_over_tri(basis.phi[i] * basis.phi_dx[q]) -
  tri.b[1, 0] * integrate_over_tri(basis.phi[i] * basis.phi_dy[q]))
 for i in basis.indices)
```

Listing 3.1: Implementation of the first part of (3.1) in GHODDESS. `cu` represents field c of the degrees of freedom for the U-component of the solution. Access to local degrees of freedom is parameterized with a triangle orientation (lower or upper) and an index for the respective basis function. `basis` provides an abstraction for the basis functions and their derivatives. `tri` allows accessing triangle data such as the transformation matrix B and the inverse of its determinant (`det_b_inv`). Finally, `basis.indices` $= P(p)$ is the number of basis functions (adapted from [FN+20]).

Information about the triangles, such as their orientation or the determinant of the mapping from the reference triangle and its reciprocal, are stored in `tri`. SymPy is then used for the analytical evaluation of integrals and derivatives. Theoretically, it can also be used to simplify the resulting expressions. This, however, increases execution times of the Python code tremendously and has only limited effect in practice as the ExaStencils code generator inherently performs many of the underlying optimization procedures.

3.1.4 Mapping Python to ExaSlang

To allow mapping from symbolic representations to ExaSlang, SymPy expressions are enriched with a few required abstractions, such as field symbols that correspond to accesses to ExaSlang fields storing quantities defined on the computational domain. GHODDESS also creates an auxiliary knowledge file holding parameters that guide the generation process as well as ExaSlang specifications. To be precise, we target layer 4 (as illustrated in Figure 3.1) since this requires only a single back end. The symbolic program description also contains the control flow for distributing individual kernels to designated architecture components.

Consider as an example the implementation of Listing 3.1 once again. The evaluation of this expression varies according to the selected discretization order. For instance, in the context of linear order and the second degree of freedom, denoted as $P(p) = 3$ and $q = 2$, the summation over integrals can be analytically computed by

$$\int_{\hat{\Omega}}(-6\sqrt{2})\mathrm{d}\hat{\boldsymbol{x}} + \int_{\hat{\Omega}}(-12+36\hat{x})\mathrm{d}\hat{\boldsymbol{x}} + \int_{\hat{\Omega}}(-6\sqrt{12}(1-\hat{x}-2\hat{y}))\mathrm{d}\hat{\boldsymbol{x}} = -3\sqrt{2}$$

using the basis formulation (2.18). Subsequently, this representation can be mapped to ExaSlang together with the remaining terms. Omitting other contributions, such as the latter half of the element integral, edge integrals, and the right-hand side would result in a code similar to the one depicted in Listing 3.2. This code segment is incorporated within the corresponding update kernel.

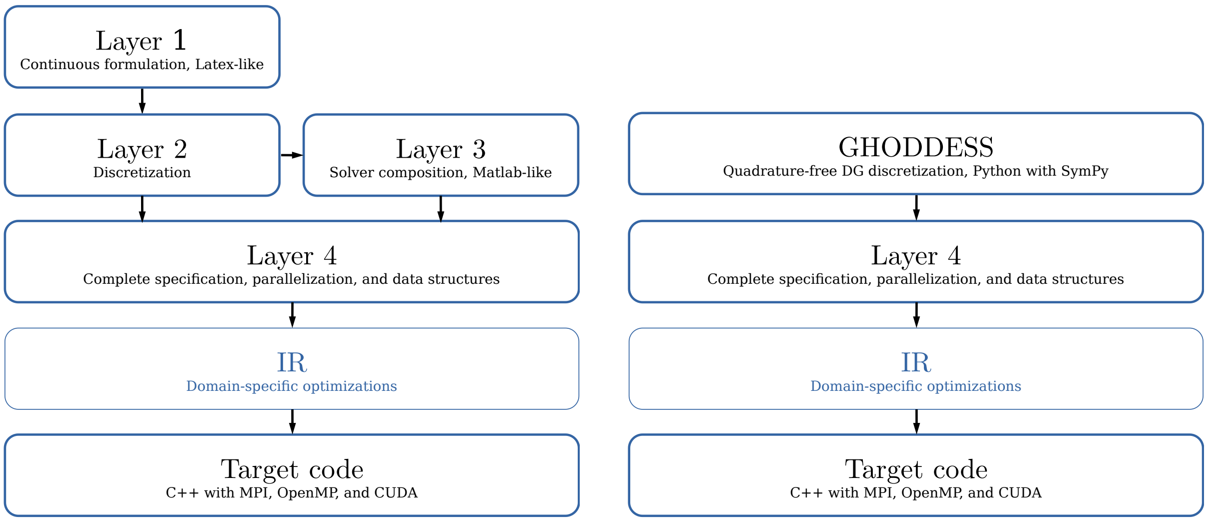

Layer 1
Continuous formulation, Latex-like
Layer 2
Discretization
Layer 3
Solver composition, Matlab-like
Layer 4
Complete specification, parallelization, and data structures
IR
Domain-specific optimizations
Target code
C++ with MPI, OpenMP, and CUDA
GHODDESS
Quadrature-free DG discretization, Python with SymPy
Layer 4
Complete specification, parallelization, and data structures
IR
Domain-specific optimizations
Target code
C++ with MPI, OpenMP, and CUDA

Figure 3.1: Schematics of the multi-layered approach of ExaSlang in the original (left) and adapted (right) workflow. In the latter case, the continuous and discrete problem are manually derived and implemented in GHODDESS which then directly maps to layer 4 (adapted from [FN+20]).

```
loop over cxiNewLower1 {
  cxiNewLower1 = (
   - 3.0 * 1.41421356237310 * bLower3 * cuLower0 * invDetBLower0 )
}
```

Listing 3.2: ExaSlang output for an update kernel based on the specification from Listing 3.1 for `basis.indices` $= 3$ and $q = 2$ (modified for readability). `cxiNewLower1` is the right-hand-side of the time stepping scheme for the 1st degree of freedom of ξ on lower triangles. `cuLower0` stores the 0th degree of freedom for U on triangles with lower orientation. `bLower3` and `invDetBLower0` are used to access the geometric information, that is, one entry of B and $\det(B)^{-1}$ which are pre-calculated to improve performance. The degrees of freedom are updated for every lower triangle (`loop over`) (adapted from [FN+20]).

Frequently used sub-expressions, such as parts of the transformation matrix B and the reciprocal of its determinant, can optionally be pre-computed and stored in designated data fields, e.g., `bLower3` and `invDetBLower0`, respectively, to improve performance. It is important to note that this kernel exclusively updates triangles with a lower orientation. An equivalent kernel updating the remaining upper triangles is also produced during the code generation.

In the context of higher-order DG approximations, the complexity and number of generated expressions grow significantly. Consequently, the resulting layer 4 file size can easily exceed several MB, and such configurations frequently have generation times exceeding an hour on modern consumer hardware. This substantial time consumption is primarily attributed to symbolic integral evaluations within the quadrature-free scheme, necessitating unrolling all associated loops, e.g., over basis functions. With increasing order of the DG method, the number of terms in the expressions increases cubically in the number of local basis functions. This results in several million nodes in the abstract syntax tree (AST), leading to a noticeable slow-down in all routines that need to traverse it. However, in practice the code generation step is only done once, and the same generated code can be used for many simulations, so this is not a major drawback.

3.1.5 Mapping ExaSlang to optimized code

The ExaStencils source-to-source compiler is capable of parsing the emitted ExaSlang code, applying code transformations and certain optimizations, and, finally, outputting a C++ and/or CUDA code parallelized with OpenMP and/or Message Passing Interface (MPI) for shared and/or distributed memory parallelism. For higher-order DG discretizations, the code generation process by ExaStencils can extend beyond one hour due to the considerable scale of expressions involved. However, it is worth noting that larger grids do not require more time for code generation.

Building upon the example from the previous section, we proceed to generate compilable C++ code similar to that depicted in Listing 3.3.

```
for (int i1 = 0; i1 < 16; i1 += 1) {
  for (int i0 = 0; i0 < 16; i0 += 1) {
    fieldData_cxiNewLower1[((18*i1)+i0+19)] = - ( 4.2426406871193 *
     fieldData_bLower3[((18*i1)+i0+19)] * fieldData_cuLower0[((18*i1)+i0+19)]
     * fieldData_invDetBLower0[((18*i1)+i0+19)] );
  }
}
```

Listing 3.3: An exemplary C++ output based on the specification from Listing 3.2 without optimizations (modified for readability). `fieldData_*` are data fields storing the degrees of freedom and other triangle-specific data. Loop bounds are specialized to the chosen problem configuration (adapted from [FN+20]).

As evident, no optimizations were carried out to produce clean code aimed at giving developers insight. Enabling such optimizations generates better-performing but less readable code, as showcased in Listing 3.4.

```
for (int i1 = 0; i1<16; i1 += 1) {
  double* const _fieldData_bLower3_p0 = &(fieldData_bLower3[(18*i1)]);
  /* similarly for _fieldData_cxiNewLower1_p0,
    _fieldData_cuLower0_p0, _fieldData_invDetBLower0_p0 */
  int _start = 0; int _end = 16;
  int _intermediate = std::max({_start,(_end-((_end-_start)%4))});
  if (_start<_intermediate) {
    __m256d _vec00 = _mm256_set1_pd(4.2426406871193);
    for (int i0 = _start; i0<_intermediate; i0 += 4) {
      __m256d _vec01 = _mm256_loadu_pd(&(_fieldData_bLower3_p0[(i0+19)]));
      __m256d _vec02 = _mm256_loadu_pd(&(_fieldData_cuLower0_p0[(i0+19)]));
      __m256d _vec03 = _mm256_loadu_pd(&(_fieldData_invDetBLower0_p0[(i0+19)]));
      __m256d _vec04;
```

```
        _vec04 = _mm256_xor_pd(_mm256_mul_pd(_mm256_mul_pd(_vec02, _vec03),
          _mm256_mul_pd(_vec00, _vec01)), _mm256_set1_pd(-0.0));
        _mm256_storeu_pd(&(_fieldData_cxiNewLower1_p0[(i0+19)]), _vec04);
      }
    }
    for (int i0 = _intermediate; i0<_end; i0 += 1) {
      fieldData_cxiNewLower1_p0[(i0+19)] = -( 4.2426406871193 *
       _fieldData_bLower3_p0[(i0+19)] * _fieldData_cuLower0_p0[(i0+19)]
       * _fieldData_invDetBLower0_p0[(i0+19)] );
    }
  }
```

Listing 3.4: An exemplary C++ output based on the specification from Listing 3.2 with enabled optimization (modified for readability). As before, `fieldData_*` are data fields storing the degrees of freedom and other triangle-specific data, and loop bounds are specialized to the chosen problem configuration. `_fieldData_*` hold addresses precomputed to enhance performance. `__m256*` and `_m256*` are AVX vector data types and intrinsics that allow for an explicit vectorization (adapted from [FN+20]).

3.1.6 Extensions for heterogeneous computing

Some approaches have been explored in the context of *heterogeneous simulations* of shallow water models. For instance, in [Ech+20], a coupled 1D–2D model for real flood cases is hybridized using a heterogeneous CPU–GPU architecture. A different approach was taken in [KK18b; CGD22] for a two-dimensional shallow water model, where the domain is partitioned into subdomains distributed across CPUs and GPUs. Furthermore, in [Fu+17], the global SWE are solved heterogeneously by decomposing subblocks of patches into CPU and accelerator components. However, to the best of our knowledge, prior to our contribution [FN+23a], no works attempt to adapt the original algorithm's numerics to parallelize it across diverse architectures.

The tight coupling between the base and correction computations in our separation approach, presented in Section 2.3.3, favors hardware architectures that minimize the performance impact of inter-component communication. Therefore, integrated CPU–GPU architectures represented by Systems-on-a-Chip (SoCs) like NVIDIA Jetson systems manifest as particularly promising candidates for our heterogeneous approach. Since the CPU and the GPU share the same die and the system memory, no distinct memory locations and transfer operations are needed. Consequently, such integrated setups enable low-latency communication between the CPU and GPU units.

The execution of kernels in a heterogeneous manner necessitates the application of data synchronization mechanisms and bookkeeping strategies. Automating these technically intricate procedures holds substantial potential for enhancing productivity, rendering them an ideal focus for code generation through the ExaStencils framework. By default, ExaStencils employs a standard data migration methodology suited for architectures featuring discrete memory locations for the host and device. It generates explicit memory transfer instructions that facilitate data exchange between these distinct memory locations, concurrently maintaining records of their versions. However, these transfer operations are time-consuming, incur notable latencies, and can significantly impact overall execution time, particularly when performed at a high frequency. As part of the separation approach, ExaStencils has been expanded to encompass additional memory management techniques of the CUDA platform, namely pinned memory, unified memory, and zero-copy memory, each of which are explained in the following enumeration.

- *Pageable memory* is referred to as memory that can be automatically swapped (paged) by the operating system between the primary storage, typically Random-Access Memory (RAM), and secondary storage, like external drives. GPUs, however, cannot directly access data residing in pageable host memory. Consequently, data transfers between the CPU and GPU often incur overhead from internal copy operations to page-locked or pinned host buffers issued by the CUDA runtime. To alleviate this challenge, CUDA offers the (de-)allocation of *pinned host memory* to avoid the additional copy and to increase the transfer bandwidth.
- *Unified memory* bundles the previously separate host and device memory allocations into a single allocation. This approach eliminates the necessity for explicit memory transfers, leveraging an automatic on-demand migration process determined by the CUDA runtime via a page-fault mechanism. While this model significantly simplifies the development of heterogeneous codes, it is often associated with performance overhead stemming from fault handling. Explicit prefetching of data can mitigate this performance penalty and allows for fine-grained overlapping with kernel executions at the cost of additional code complexity.
- *Zero-copy memory* allows GPU threads to access host memory directly. Users are provided with a shared virtual memory space for host and device data given by mapping the allocated host memory to the CUDA address space. This technique proves especially advantageous for systems equipped with integrated GPUs, such as the NVIDIA Jetson architectures. While this approach does not need explicit migration requests, synchronization between CPU and GPU execution is necessary for critical regions. The required bookkeeping for this purpose is automatically generated by the ExaStencils compiler.

Numerical results of the separation approach are discussed in Section 4.4, with a special emphasis on leveraging zero-copy memory within an NVIDIA Jetson architecture.

While beyond the scope of this work, we would like to highlight a recent noteworthy application of GHODDESS, which involves porting shallow water simulations to FPGAs. This is realized in combination with a template-based stencil processing library that provides FPGA-specific optimizations for a streaming execution model [Alt+23], rather than relying on the ExaStencils framework.

3.2 Masked block-structured grids

Real-world ocean domains have complex geometries and topographies. Thus, the accuracy and computational performance of numerical simulations on those domains is highly dependent on the computational grid quality. Unstructured meshes are often favored because of their geometrical flexibility [Fri+19]. Nonetheless, their irregular memory access patterns entail additional indexing and cache misses compared to their structured counterparts and make it also very challenging to achieve optimal computational efficiency on GPUs [Lac+14]. An innovative approach, as detailed in [Zin+19], addresses this by introducing automatic generation of *block-structured grids* (BSGs) for real-world ocean geometries, merging the geometric flexibility of unstructured meshes with structured grids' performance advantages, a technique employed in CFD applications for years [Kin93; SB96]. For details about the application of unstructured and structured grid models within the context of ocean simulations see [FN+23b].

To account for small-scale features like small islands and narrow channels, the methodology presented in [Zin+22] and [FN+23b] enhances the generated grids by allowing them to cover a larger area than the actual computational domain. It incorporates masking to exclude excessive grid elements beyond the computational domain's original boundaries. The use of land-sea masks to distinguish wet and dry cells is common in structured-grid global ocean models, as in NEMO [Irr+22] or ICON-O [Kor+22], facilitating accurate representation of complex geometries. The novelty of our approach is to utilize masked BSGs and, in particular, to tune the element count per block to optimize the trade-off between geometric flexibility and computational efficiency. The generated BSGs have a prescribed number of quadrilateral blocks, which are then refined using structured triangular grids. Afterward, masks are employed to accurately represent features that are too small to be adequately captured in the block structure.

The main steps of the implemented masked BSG generation procedure are illustrated in Figure 3.2 and are summarized below.

a) Begin with an initial unstructured mesh to extract bathymetry and mesh density within the target computational domain.
b) Reduce the initial mesh to double the target block number using the approach proposed in [Zin+19], relying on a modified error metric [Zin+22] for the quadratic mesh simplification scheme [GZ05]. Perform iterative edge collapses commencing with the edge causing the smallest error based on density information and proceed until the desired triangle count is achieved.
c) Transform triangles in the simplified grid into quad blocks using the Blossom-Quad procedure [Rem+12], utilizing Edmonds' Algorithm for dual graph perfect matching. When a perfect match does not exist, merge the remaining triangles via the method introduced in [ZG21].
d) Refine each quad block into a structured triangle grid to achieve the intended block-structured topology. Triangle orientation should prioritize higher-quality triangles based on the mean ratio metric [Zin+22].

e) Adjust element sizes by relocating interior vertices according to mesh density information. Outside the computational domain, mask triangles to restore the domain shape.
f) Align boundary vertices with exterior domain boundaries, and reposition interior vertices to improve the quality near boundaries.

This procedure is similar to the one for generating unmasked BSGs, except that in the unmasked case, a larger target number of blocks is necessary in step b) to capture the coastline within the simplified triangle grid. Furthermore, the masking is omitted in step e).

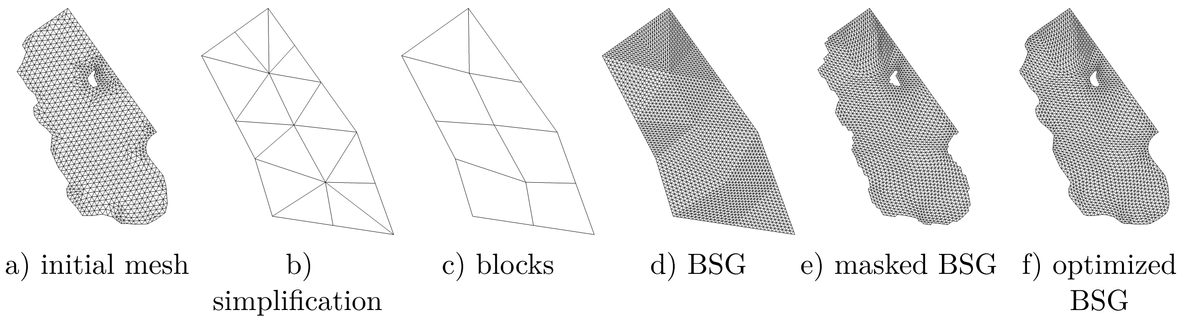


Figure 3.2: Masked BSG generation steps (from [FN+23b]).

A masked BSG generally encompasses three distinct element categories:

- active (regular grid elements),
- BC elements (adjoining active elements, where no solution is computed but the boundary condition is projected to edges shared with active elements),
- inactive elements.

To exploit the computational benefits of the regular grid structure within a block, the algorithmic treatment of all elements and edges in a block should follow the same pattern to the greatest extent possible. To achieve this, our implementation in GHODDESS incorporates a pre-processing stage before edge computations commence, reconstructing boundary conditions along edges between active and BC elements. Consequently, regular edge updates yield accurate boundary condition enforcement. Since one masked element can possess multiple unmasked neighbors, this process is executed iteratively on a local edge basis, interleaved with edge computations. Conducting three sweeps across all elements, each treating a single edge, demonstrates no negative performance impact compared to where a single sweep is executed, addressing all three computations simultaneously.

Two aspects of this masking approach have proven particularly advantageous. Firstly, BSGs are now able to correctly represent complex features such as small islands and narrow channels that were previously not meshable at all using conventional BSGs. Secondly, since the BSGs no longer have to follow all fine geometric details, there is more freedom in tuning the number of elements per grid block, enabling optimization for diverse hardware architectures. In fact, blocks with just a few elements allow to nearly perfectly capture the domain geometry, requiring minimal land-sea masking yet providing limited structure for performance optimization. Conversely, a BSG composed of a small number of blocks with numerous elements closely approximates a fully structured grid in terms of topology and computational efficiency. However, for complex domain geometries, a larger element fraction necessitates masking. We present numerical outcomes evaluating both masked and unmasked BSGs in Section 4.5.

3.3 Reference code: quadrature-based unstructured mesh DG SWE solver

To assess the accuracy of our quadrature-free discretization and the BSGs, in Sections 4.2 and 4.5, we compare simulation results to ones obtained with the quadrature-based unstructured mesh DG SWE solver UTBEST described in [AD02; Aiz04]. UTBEST adheres to the original SWE formulation (2.4), employing quadrature rules for element and edge integral computations, supporting piecewise constant, linear, and quadratic approximations on conforming triangular meshes. It utilizes the same temporal discretization schemes as GHODDESS and is a hybrid FORTRAN 77/C code, with core numerical components implemented in C. UTBEST performs all calculations in serial mode and employs the single precision floating-point format. Validation against the test case detailed in Section 4.1.5 has been conducted via comparisons with ADCIRC[24] and station measurement data [AD02]. Recently, UTBEST has been ported to OpenCL to operate on FPGAs, demonstrating substantial performance gains over the serial CPU version [Ken+21; Faj+23].

[24] `https://adcirc.org`

Chapter

Numerical Results 4

Within this chapter, a numerical assessment is conducted to evaluate the quadrature-free reformulation as elucidated in Section 2.2, along with the parameter-free adaptivity indicator (Section 2.3.4), the p-adaptivity separation approach (Section 2.3.3) and the utilization of masked block-structured grids presented in Section 3.2. Unless explicitly specified otherwise, all computations are carried out employing the GHODDESS code generation framework outlined in Section 3.1.

4.1 Setup of numerical examples

We commence by introducing a series of diverse example configurations that are used for numerical evaluations in the following sections. These configurations encompass an analytical sine wave, a radial dam break, a scenario of supercritical flow, a geostrophic adjustment test, and two distinct tidal flow scenarios. In the first three setups, the physical parameters are selected to allow computations on simple domains without the need for rescaling. Large parts of the following descriptions build upon the previously published articles [FN+20; FNA22; FN+23b] and the preprint [FN+23a].

4.1.1 Analytical sine wave on a quadrilateral domain

This initial test case serves mainly for verification and we consider dimensionless physical quantities. We chose a quadrilateral domain, which was first refined as a structured triangle grid. Following this, all nodal points of the grid were perturbed by a randomly generated displacement that was up to 20 % of the element edge length. This geometric configuration is depicted in Figure 4.1 on the left. The bathymetry was specified as

$$h_b = 1 + \tfrac{1}{1000}x + \tfrac{2}{1000}y.$$

The gravitational acceleration g was set to 0.16, and no influences of bottom friction or Coriolis force were considered.

We conducted the convergence studies detailed in Section 4.2 using an artificially manufactured analytical solution given by

$$\begin{aligned}\xi(x,y,t) &= 2 + a - 2\,C_a\,\sin\left(\tfrac{\pi(x+y+C_t\,t)}{600}\right),\\ U(x,y,t) &= 2\,a + C_a\,C_t \sin\left(\tfrac{\pi(x+y+C_t\,t)}{600}\right),\\ V(x,y,t) &= a + C_a\,C_t \sin\left(\tfrac{\pi(x+y+C_t\,t)}{600}\right),\end{aligned}$$

which also served to define suitable initial and Dirichlet boundary conditions. We executed all simulations for $t \in (0, 1500)$ seconds with the time step $\Delta t = 0.5\,\mathrm{s}$ for piecewise constant and linear approximations and $\Delta t = 0.01\,\mathrm{s}$ for piecewise quadratic and cubic ones. These time step sizes were chosen to ensure that time discretization errors remain negligible compared to those associated with spatial discretization. We set the remaining parameters as $C_a = 0.2$, $C_t = 0.2$, and $a = 0.3$. Figure 4.1 illustrates the mesh and the initial condition (left), the final solution (middle) on the coarsest mesh with 32 elements, as well as on a four times refined one with 8192 elements (right) using the piecewise linear (p1) DG discretization.

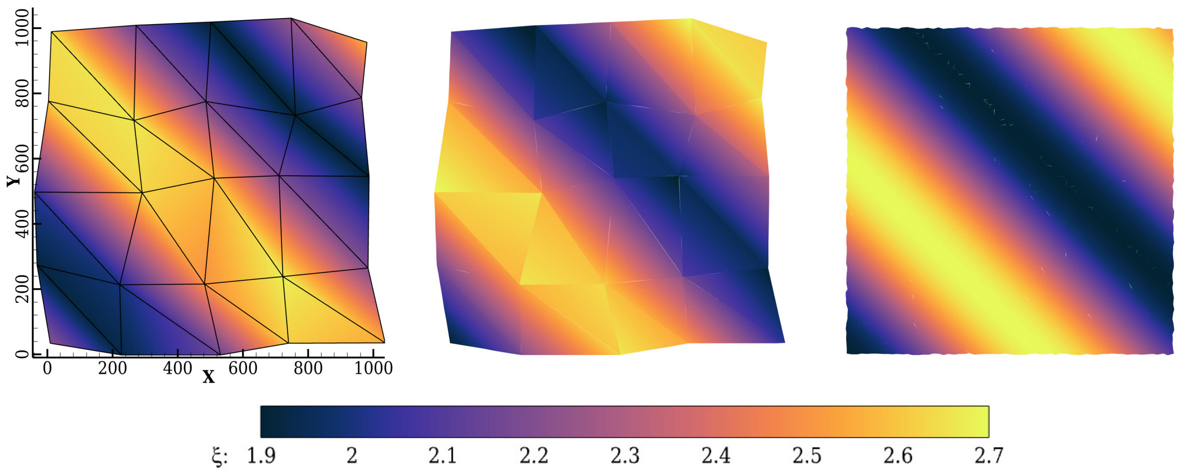


Figure 4.1: Analytical sine wave: quadrilateral domain with perturbed mesh and piecewise linear (p1) DG discretization: mesh with 32 elements and initial condition for the elevation (left), final solution on the same mesh (middle), and final solution on a four times refined mesh (right) (adapted from [FN+20]). All numbers are specified in meters.

4.1.2 Radial dam break

The radial dam break example is based on [LeV02; Haj21], again considering dimensionless physical quantities. We set $\Omega = [0, 5] \times [0, 5]$, $g = 1$, and fixed a constant bathymetry h_b, which is specified later in the respective sections. To incorporate the right-hand side terms into the performance evaluation, a slight linear bottom friction $\tau_{\mathrm{bf}} = 0.0001 \cdot H$ and a small Coriolis force with coefficient $f_c = 10^{-5}$ were imposed.

To make the test problem better suited for a p-adaptive approach, we set the initial condition as

$$\begin{aligned}
\xi(x,y,t) &= \begin{cases} 2 + 0.5\, e^{-15\left((x-2.5)^2+(y-2.5)^2\right)}, & \text{if } (x-2.5)^2+(y-2.5)^2 < 0.25, \\ 1, & \text{otherwise,} \end{cases} \\
U(x,y,t) &= 0, \\
V(x,y,t) &= 0.
\end{aligned}$$

The simulation results, which serve as a reference solution, are depicted in Figure 4.2. These results were obtained through computations on a uniform mesh comprising 524 288 elements. To mitigate numerical diffusion, this simulation utilized the FORCE scheme (see (2.15)) instead of the Lax–Friedrichs flux. Additionally, we employed a vertex-based slope limiter (see (2.19)) to avoid over- and undershoots. Since all external boundaries used land boundary conditions, the wave experienced reflection, as evident in Figure 4.2 (bottom right), which corresponds to $t = 3\,\text{s}$.

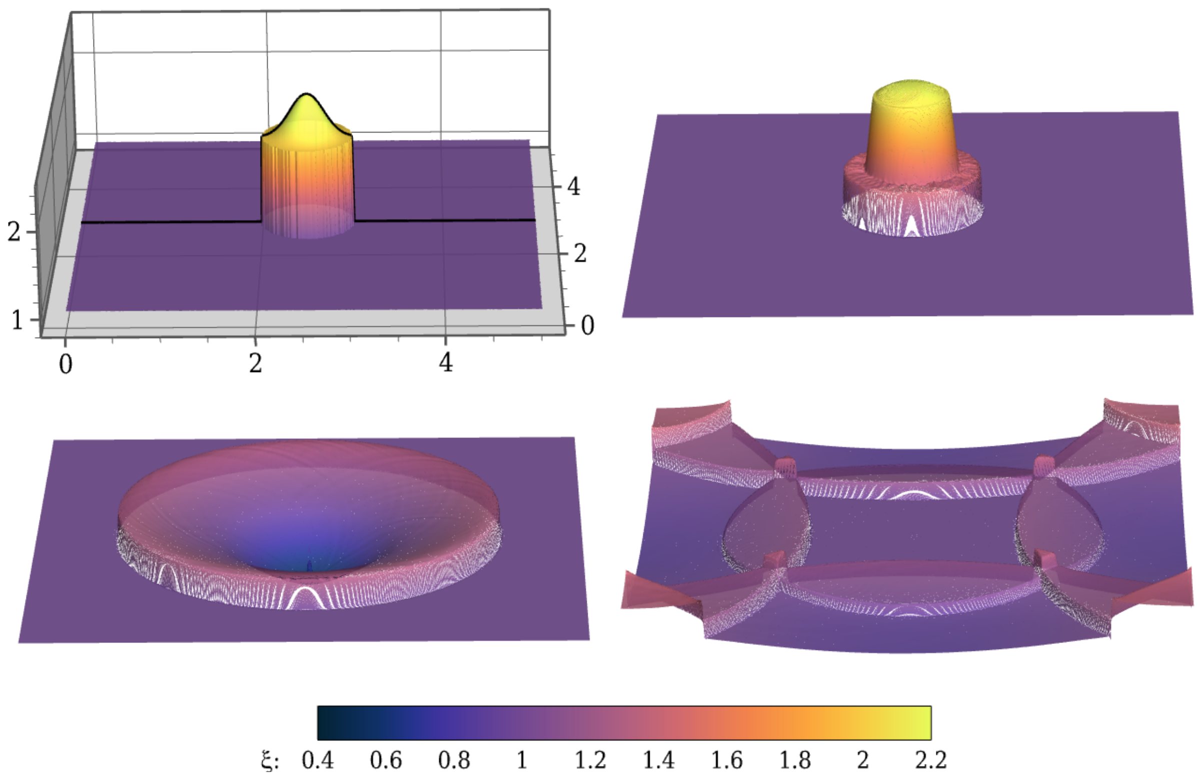


Figure 4.2: Radial dam break: projected initial free surface elevation with slice at $y = 2.5$ (top left) and limited linear approximation (p1) at $t = 0.1\,\text{s}$ (top right), $t = 1\,\text{s}$ (bottom left), and $t = 3\,\text{s}$ (bottom right) on a uniform refined mesh with 524 288 elements using the FORCE Riemann solver (adapted from [FNA22] and [FN+23a]). All numbers are specified in meters.

4.1.3 Supercritical flow

The robustness of our implementation is demonstrated using a problem featuring a discontinuous solution. We simulated a supercritical flow in a constricted channel with a constant bathymetry of 1 meter based on the configuration proposed in [ZO95]. The lateral channel boundary walls were confined on both sides with a constriction angle of 5° producing reflective cross-wave patterns. Flow was induced through the inflow (bottom) boundary, where free surface elevation and velocity were specified. There was no flow through the left and right boundaries, whereas there were radiation boundary conditions at the outflow (top) boundary. In this setup, we also consider dimensionless physical quantities. The inlet Froude number Fr is defined as $\mathrm{Fr} = u_i/\sqrt{g\,H_i}$, with the axial velocity u_i and the water depth H_i at the inlet. Here, we set Fr to 2.5, thus entering a supercritical regime. The gravitational acceleration g was assigned a value of 0.16, and neither bottom friction nor Coriolis force were imposed.

Figure 4.3 shows our unstructured mesh with 3155 elements, the BSG [Zin+19] with 3584 elements, and the exact steady-state solution projected on a mesh with approximately 230 000 elements. For an analytical solution to this problem, see, for example, [Ipp51].

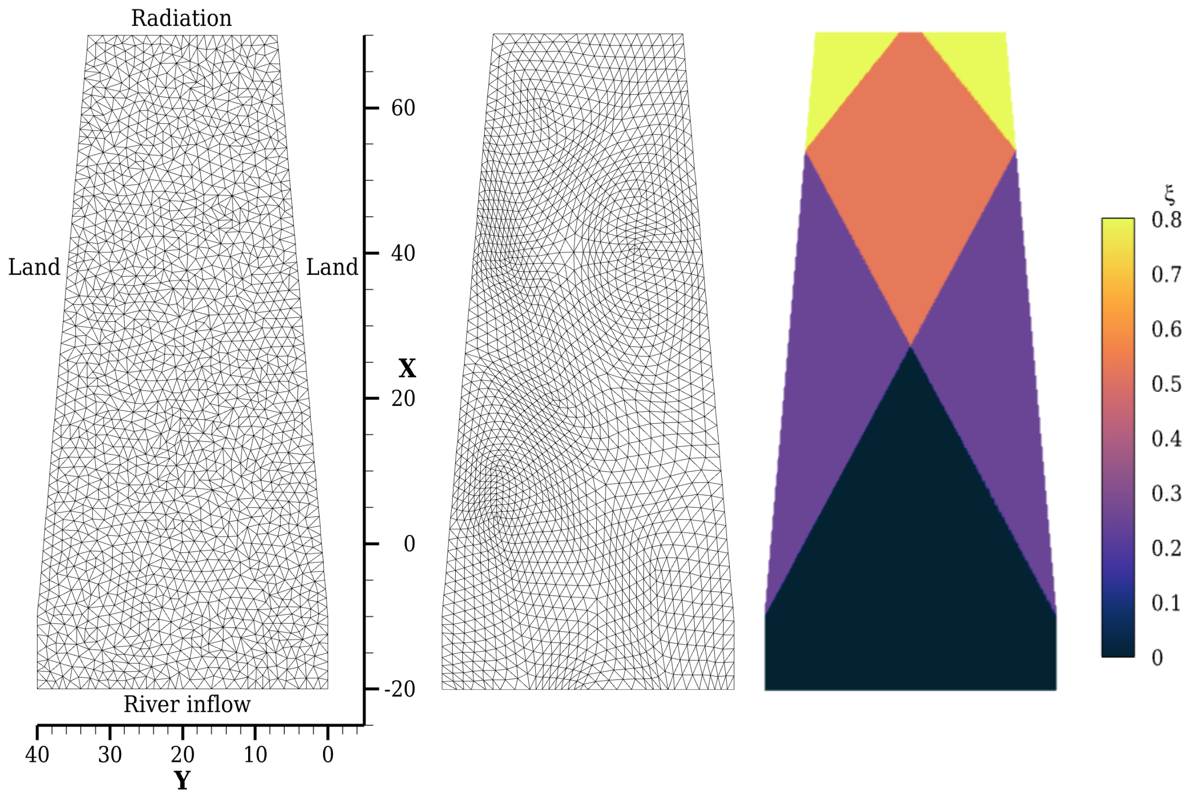


Figure 4.3: Supercritical flow in a constricted channel: unstructured mesh with 3155 elements and boundary types (left), BSG with 3584 elements (middle), and exact solution projected onto a mesh with approximately 230 000 elements (right) (adapted from [FN+20] and [FNA22]). All numbers are specified in meters.

4.1.4 Geostrophic adjustment

This test case assesses the capability to reproduce the geostrophic adjustment process, i.e., the interaction of gravitational and rotational forces leading to a complex stationary solution after an initial perturbation of the constant water height. We employed the setup outlined in [TBR13] and [Sze+24], with domain dimension $L = 10^7$ meters and a structured uniform mesh comprising 8192 triangles in $\Omega = [0, L] \times [0, L]$. We set $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$, fixed a constant bathymetry $h_b = 1000\,\mathrm{m}$ and imposed no bottom friction. Assuming an f-plane approximation, the constant Coriolis coefficient was set to $f_c = 0.0001\,\frac{1}{\mathrm{s}}$. Simulations, with a time step $\Delta t = 60\,\mathrm{s}$, were run for $t \in (0, 36\,000)$ seconds.
Land boundary conditions were applied to all external boundaries, and the initial condition was set as

$$\begin{aligned} \xi(x, y, t) &= 5 \exp\left(-\tfrac{(x-L/2)^2+(y-L/2)^2}{2(L/20)^2}\right), \\ U(x, y, t) &= 0, \\ V(x, y, t) &= 0. \end{aligned}$$

The initial free surface elevation is depicted in Figure 4.4. For verification, we compare our results in Section 4.2.3 with those presented in Figure 10 of [TBR13] and Figure 3 of [Sze+24].

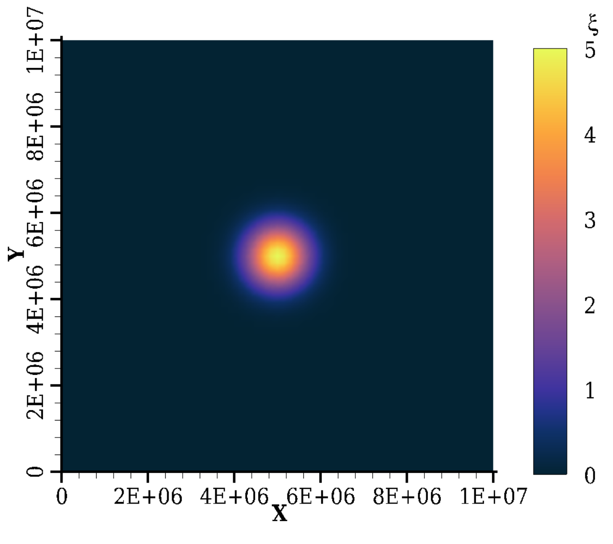


Figure 4.4: Geostrophic adjustment test: initial free surface elevation for piecewise quadratic (p2) DG approximation. All numbers are specified in meters.

4.1.5 Tidal flow at Bight of Abaco (Bahamas)

Our first real-world example considers a tide-driven flow scenario in the Bight of Abaco within the Bahamas archipelago, illustrated at the top in Figure 4.5. The domain geometry (rotated for historical reasons), the bathymetry, and the approximate positions of four recording stations for the free surface elevation and velocity with exact coordinates in meters (38 667, 49 333), (56 098, 9613), (41 263, 29 776), and (59 594, 41 149) are illustrated in Figure 4.5 (bottom left). The setup used $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$ and employed the quadratic friction law $\tau_{\mathrm{bf}} = C_f|\boldsymbol{u}|$ with coefficient $C_f = 0.009$. This approximation is justified because the area is small and the depths are shallow. Additionally, a constant Coriolis parameter of $f_c = 3.19 \cdot 10^{-5}\,\frac{1}{\mathrm{s}}$ was applied. The tidal forcing was composed of five harmonic constituents (O1, K1, N2, M2, S2) given analytically by

$$\begin{aligned}\hat{\xi}(t) = {}& 0.075\cos\left(\tfrac{t}{25.82} + 3.40\right) + 0.095\cos\left(\tfrac{t}{23.94} + 3.60\right) + 0.100\cos\left(\tfrac{t}{12.66} + 5.93\right)\\ &+ 0.395\cos\left(\tfrac{t}{12.42} + 0.00\right) + 0.060\cos\left(\tfrac{t}{12.00} + 0.75\right), \end{aligned} \tag{4.1}$$

where t in hours denotes the time elapsed from the beginning of the simulation, and $\hat{\xi}$ in meters represents the specified tidal elevation at the open sea boundary. Since in real-life ocean simulations, the initial conditions are often unknown or challenging to obtain, we performed a so-called cold start initialization, wherein the fluid domain is assumed to be initially at rest ($\xi_0 = 0$, $\boldsymbol{q}_0 = 0$), and the boundary condition – here, the tidal elevation – was gradually increased from zero over a span of two days. We imposed no normal flow boundary conditions at the land and island boundaries. The simulations were conducted over a ten-day period. The original unstructured mesh, comprising 1696 elements, is visualized in Figure 4.5 (bottom right).

4.1.6 Galveston Bay

Our concluding test case considers a tidal flow scenario within Galveston Bay, Texas. It is presented at the top in Figure 4.6. The domain geometry and the bathymetry are depicted in Figure 4.6 (bottom left), while the unstructured mesh of the bay consisting of 3397 elements is displayed at the bottom right. The physical domain is rather complicated: 17 islands are included in total, and the bathymetry varies sharply from the deep and narrow Houston Ship Channel crossing the bay to much shallower regions in the remainder of the bay. The imposed boundary conditions at the open sea and land boundaries were identical to those of the Bight of Abaco simulation. The simulations also utilized the quadratic bottom friction law $\tau_{\mathrm{bf}} = C_f|\boldsymbol{u}|$ with coefficient $C_f = 0.004$, a constant Coriolis forcing with coefficient $f_c = 7.07 \cdot 10^{-5}\,\frac{1}{\mathrm{s}}$, and a gravitational acceleration of $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$. We executed the simulations over a period of five days, starting from the lake-at-rest initial conditions and imposed tidal forcings via a linear ramp-up process spanning one day.

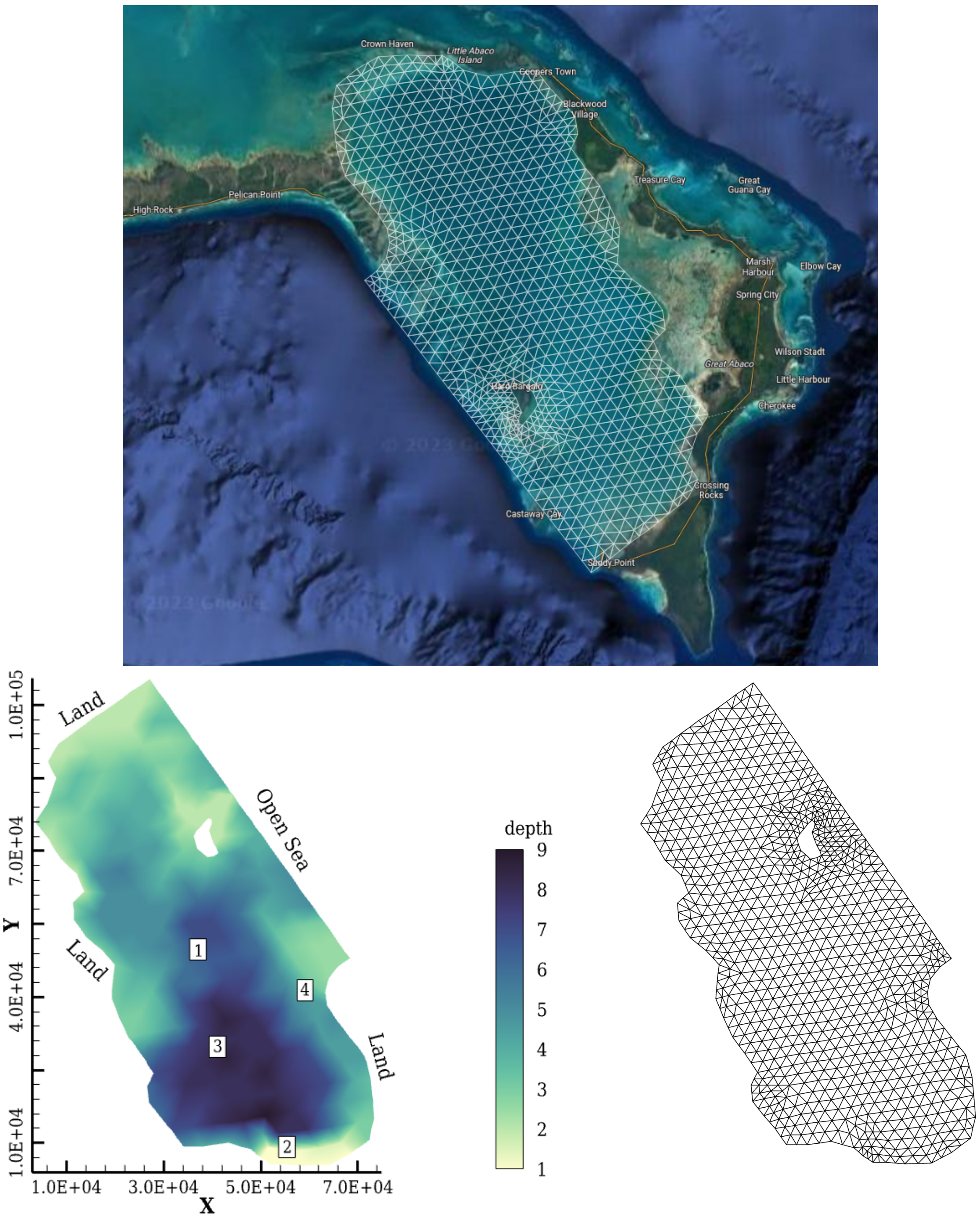


Figure 4.5: Tidal flow at Bight of Abaco: top: Google Maps excerpt overlaid with mesh, bottom: domain geometry (rotated for historical reasons), bathymetry, and approximate positions of four recording stations (left) and unstructured mesh with 1696 elements (right) (adapted from [FN+20] and [FN+23b]). All numbers are specified in meters.

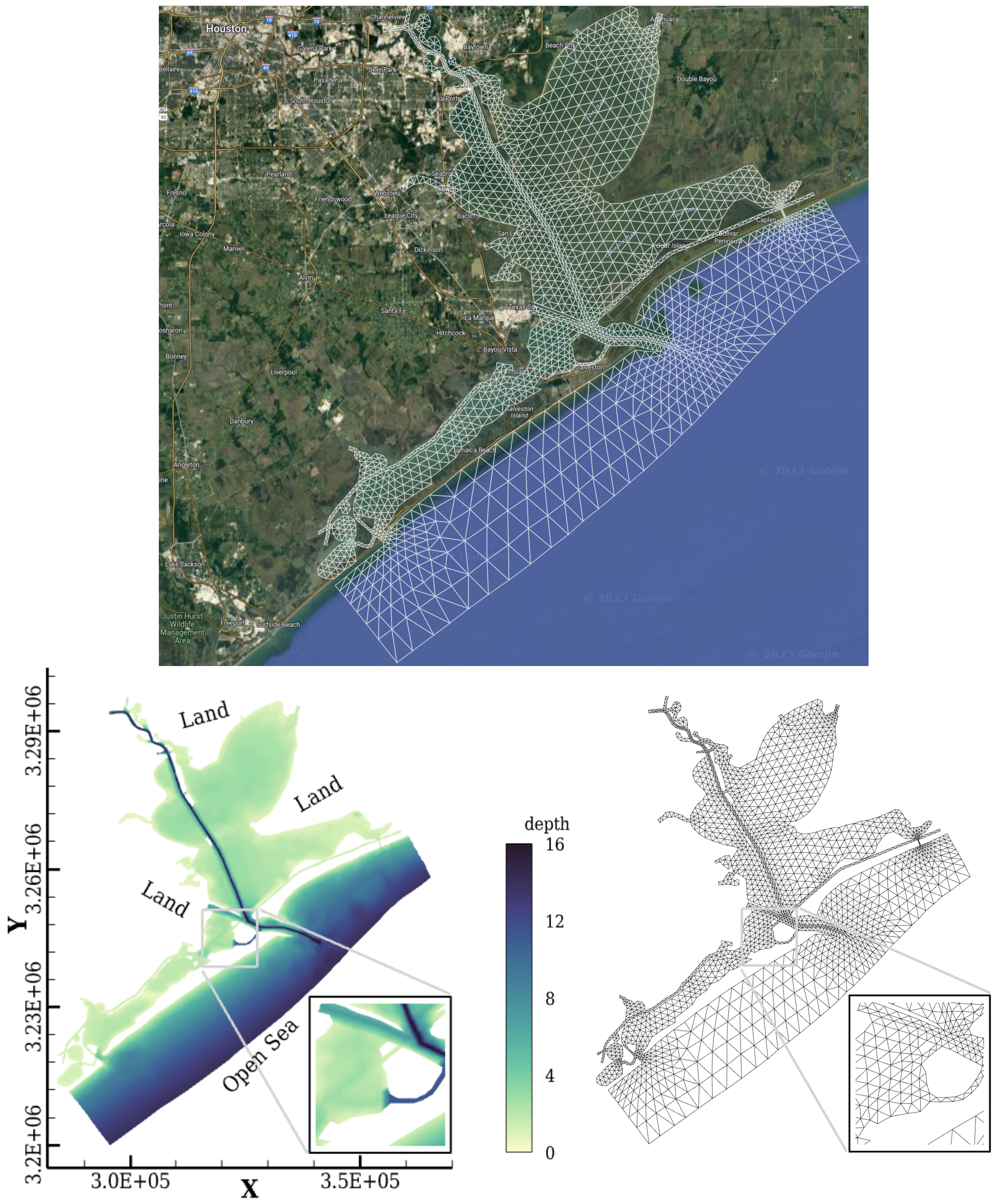


Figure 4.6: Galveston Bay: top: Google Maps excerpt overlaid with mesh, bottom: domain geometry and bathymetry (left) and unstructured mesh with 3397 elements (right) (adapted from [FN+23b]). All numbers are specified in meters.

4.2 Evaluation of the quadrature-free reformulation

Within this section, we present evidence showcasing that the accuracy and the stability of our quadrature-free formulation are closely comparable to that of the SWE solver that employs a standard, quadrature-based DG scheme. The results are partly based on our previously published work [FN+20].

4.2.1 Analytical sine wave setup

Initially, we conducted an investigation of the analytical sine wave on a randomly perturbed quadrilateral grid, as elaborated in Section 4.1.1. The errors corresponding to DG discretization spaces spanning from piecewise constants (p0) to piecewise cubics (p3), accompanied by their corresponding experimental convergence rates, are documented in Table 4.1 and graphically represented in Figure 4.7.

Table 4.1: Analytical sine wave: L^2-errors Err(·) and experimental orders of convergence EOC(·) for the square domain with perturbed grid and DG discretization orders $p = 0, 1, 2, 3$ (adapted from [FN+20]).

p	# elements	Err(ξ) in m	EOC(ξ)	Err(U) in m^2/s	EOC(U)	Err(V) in m^2/s	EOC(V)
0	32	1.36E+02	-	1.19E+02	-	1.37E+02	-
	128	7.46E+01	0.87	7.10E+01	0.75	7.26E+01	0.91
	512	3.79E+01	0.98	3.71E+01	0.94	3.78E+01	0.94
	2048	1.91E+01	1.00	1.91E+01	0.96	1.95E+01	0.96
	8192	9.55E+00	1.00	9.65E+00	0.98	9.91E+00	0.97
	32 768	4.78E+00	1.00	4.84E+00	1.00	5.02E+00	0.98
1	32	4.51E+01	-	1.22E+02	-	1.01E+02	-
	128	1.10E+01	2.04	2.04E+01	2.58	1.92E+01	2.40
	515	2.67E+00	2.04	6.28E+00	1.70	5.42E+00	1.83
	2048	6.79E−01	1.97	1.26E+00	2.32	1.21E+00	2.17
	8192	1.82E−01	1.90	2.65E−01	2.25	0.24E−01	2.32
2	32	6.43E+00	-	1.41E+02	-	2.15E+02	-
	128	9.97E−01	2.69	2.19E+00	2.69	2.54E+00	3.08
	512	1.24E−01	3.01	2.58E−01	3.09	2.42E−01	3.39
	2048	1.58E−02	2.97	2.63E−02	3.29	2.24E−02	3.43
3	32	9.67E−01	-	1.87E+00	-	2.62E+00	-
	128	6.08E−02	3.99	9.87E−02	4.25	1.16E−01	4.49
	512	3.59E−03	4.08	7.69E−03	3.68	6.92E−03	4.07
	2048	2.36E−04	3.93	4.32E−04	4.15	3.71E−04	4.22

The demonstrated convergence rates are as expected for all primary unknowns, namely, the free surface elevation and depth-integrated velocity components. Owing to increasing computational and code generation costs, runs for higher-order DG discretizations were stopped at coarser mesh resolutions than the lower-order runs. Nevertheless, the absolute L^2-errors for higher-order approximations are much smaller.

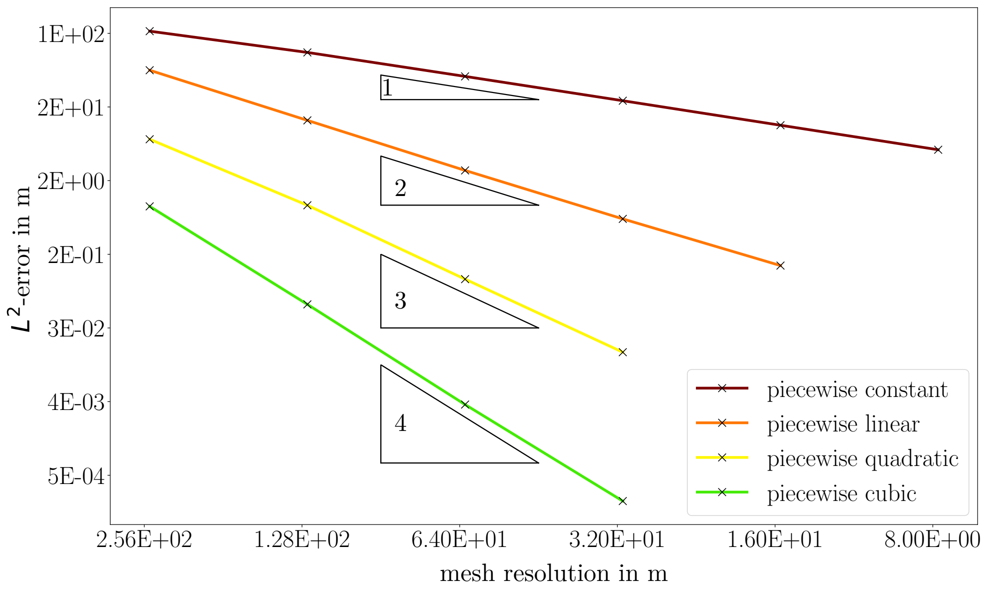


Figure 4.7: Analytical sine wave: L^2-errors for ξ vs. mesh resolution (cell width of the unperturbed mesh) (adapted from [FN+20]).

4.2.2 Supercritical flow

In the supercritical flow setup from Section 4.1.3, the results of our quadrature-free implementation display excellent alignment across all tested discretization orders (p0, p1 and p2) in comparison to results produced by the quadrature-based reference code UTBEST, described in Section 3.3. This concurrence is evident for both unstructured and block-structured grids. The steady state results were compared to those obtained utilizing the UTBEST model. In Figure 4.8 on the left, we present the piecewise linear (p1) DG approximation obtained by GHODDESS (quadrature-free) on the BSG (cf. Figure 4.3 middle). In the middle, we present the solution obtained by UTBEST (quadrature-based) on the same BSG, and on the right, the one of UTBEST (quadrature-based) on the unstructured mesh (cf. Figure 4.3 left). In particular, the solution's essential characteristics, namely the shock positions and the 'plateau' levels, agree very well among all runs and do not display any sensitivity to either the grid structure, DG discretization order, or quadrature-based/quadrature-free nature of the employed scheme.

Since no slope limiting was employed in our runs, the solutions for the piecewise linear and for any higher-order DG approximation exhibited over- and undershoots at the discontinuities. However, the shown results represent the converged steady-state solutions, and these over- and undershoots do not give rise to any stability difficulties for either the quadrature-based or quadrature-free schemes.

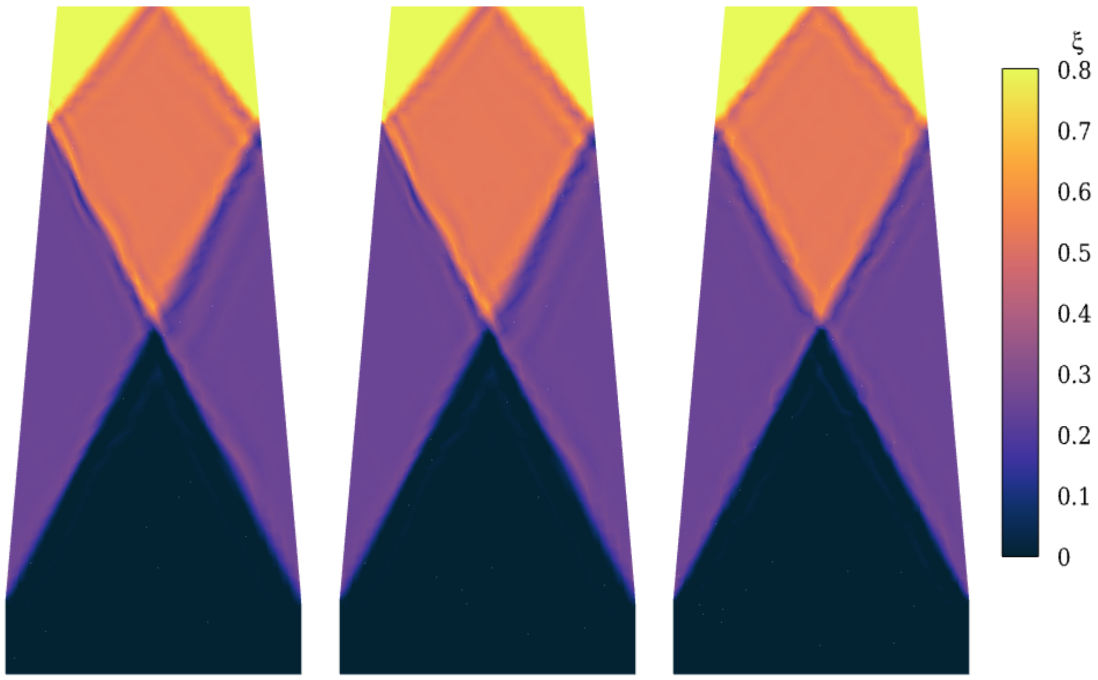


Figure 4.8: Supercritical flow in a constricted channel: piecewise linear (p1) quadrature-free steady-state solution by GHODDESS on the BSG (left), quadrature-based solution by UTBEST on the BSG (middle) and quadrature-based solution by UTBEST on the unstructured mesh (right). The solutions are in good agreement. The numbers are specified in meters (adapted from [FN+20]).

4.2.3 Geostrophic adjustment

We further investigate the ability of the proposed method to reproduce the geostrophic adjustment procedure described in Section 4.1.4. Comparing the quadrature-free results shown in Figure 4.9 with those in Figure 10 of [TBR13] and Figure 3 of [Sze+24], one observes excellent agreement.

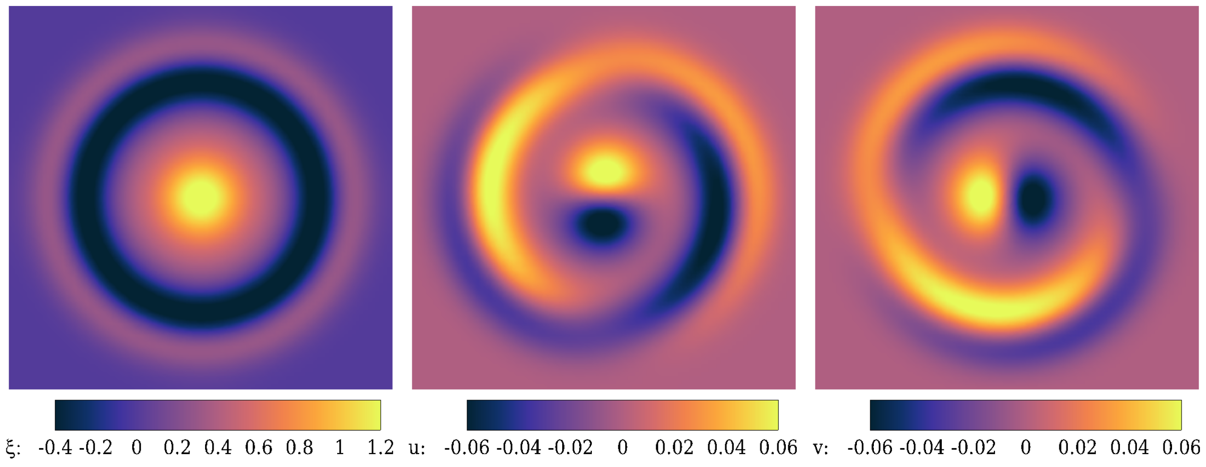


Figure 4.9: Geostrophic adjustment test: Quadrature-free piecewise quadratic (p2) solution by GHODDESS at $t = 36\,000\,\mathrm{s}$. Free surface elevation in meters (left), depth-averaged x-velocity (middle), and depth-averaged y-velocity (right) in $\frac{\mathrm{m}}{\mathrm{s}}$.

Additionally, we compare the quadrature-based results produced by UTBEST with the quadrature-free simulations obtained using GHODDESS in Figure 4.10. We illustrate the L^1-difference between these solutions for the elevation field at $t = 36\,000\,\mathrm{s}$ for different discretization orders. As we move from left to right, the order increases while the difference decreases. This trend is expected due to the very smooth nature of the solution in the current test case. Both the quadrature-based and the quadrature-free results converge towards the exact solution, thus showing a reduced discrepancy.

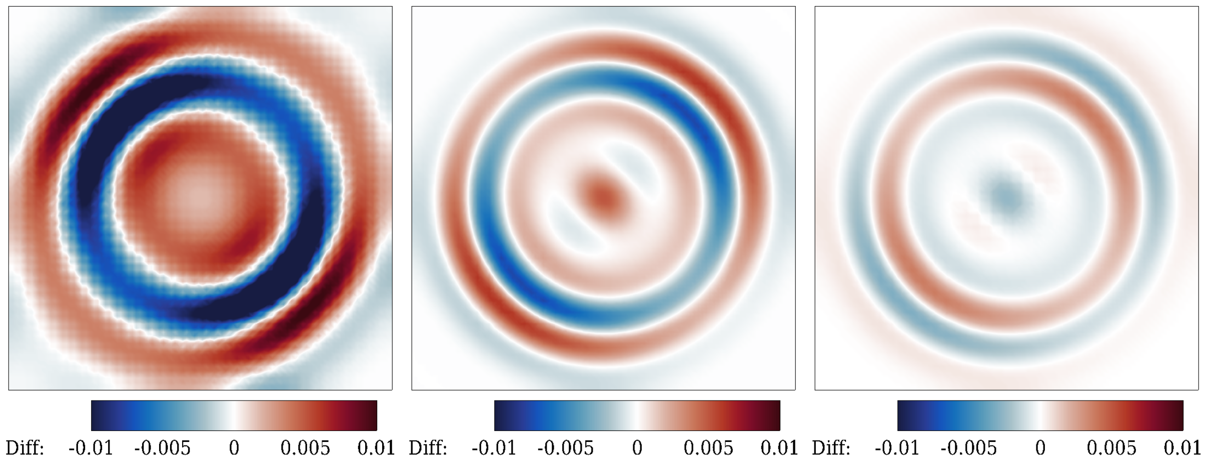


Figure 4.10: Geostrophic adjustment test at $t = 36\,000\,\mathrm{s}$. $\xi - \xi_{\mathrm{ref}}$ difference between GHODDESS (quadrature-free) and UTBEST (quadrature-based) solutions of the same DG discretization order: p0 (left), p1 (middle), p2 (right). All numbers are specified in meters.

4.2.4 Tidal flow at Bight of Abaco

In the tide-driven flow scenario as discussed in Section 4.1.5, the simulations were executed employing a constant time step of 15 seconds for the piecewise constant and piecewise linear DG discretizations (p0 and p1), while a time step of 10 seconds was utilized for the piecewise quadratic one (p2). The BSG containing 2624 elements used in this section is displayed in Figure 4.11. The simulation outcomes were compared to those obtained using the UTBEST model described in Section 3.3.

Figure 4.12 displays the chosen station comparisons, deliberately focusing on stations exhibiting the most notable discrepancies between the runs. Specifically, we considered the free surface elevation at Station 1 (top), the depth-averaged x-velocity at Station 2 (middle), and the depth-averaged y-velocity at Station 4 (bottom). Comparable or better agreement characterized the results for the remaining stations. Through these station comparisons, we aimed to address several open questions concerning our approach. Primarily, we quantified the effects of the modifications in the DG discretization which were necessitated by a quadrature-free scheme. Secondly, by additionally plotting UTBEST results for the unstructured mesh, we clarified the influence of BSGs on the simulation results.

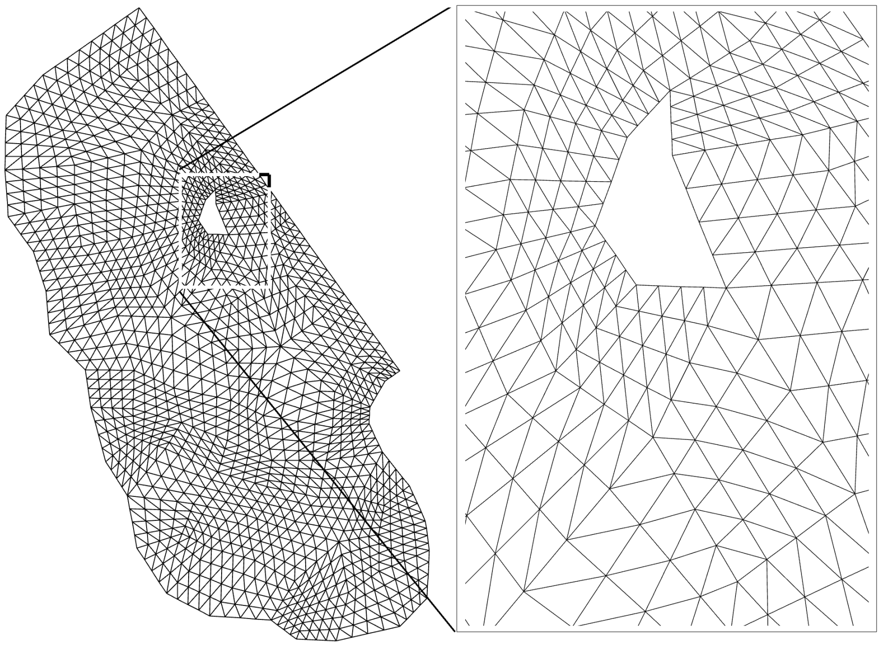


Figure 4.11: Tidal flow at Bight of Abaco: automatically generated BSG with 82 blocks and 32 elements each (adapted from [FN+20]).

The comparisons illustrated in Figure 4.12 demonstrate excellent agreement for the free surface elevation (top plot) across all different runs: UTBEST (quadrature-based scheme for p1) on the original unstructured mesh as depicted in Figure 4.5, UTBEST (quadrature-based scheme for p0, p1 and p2) on the BSG displayed in Figure 4.11, and GHODDESS (quadrature-free scheme for p0, p1 and p2) on the same BSG. Upon closely inspecting a zoomed-in view of the final elevation minimum, it becomes evident that the differences between the results for various discretization orders are much greater than those due to the quadrature integration or the mesh type. The depth-averaged velocities shown in the middle and bottom plots of Figure 4.12 display slightly more sensitivity. This observation is consistent with the findings reported in the station comparisons conducted in [AD02]. Nevertheless, it is worth noting that the absolute deviations in the x- and y-velocities are similar. However, the x-velocity plot for Station 2 presented in Figure 4.12 exaggerates those differences due to the much smaller x-velocity amplitude and the subsequently finer scaling of the vertical axis.

A more elaborate comparison between the quadrature-based results produced by UTBEST and quadrature-free simulations obtained using GHODDESS on the same BSG is shown in Figure 4.13. For the elevation field at the end of day 10 of the simulation displayed at the top left, we plot the L^1-difference between the UTBEST and the GHODDESS results for the piecewise constant (top right), piecewise linear (bottom left), and piecewise quadratic (bottom right) DG discretizations. Note that the effects of the quadrature-free scheme increase with the discretization order. In this particular test case, the solution exhibits quasi-periodicity, hence the spectral properties of the scheme are likely responsible for the error. As the discretization order increases, the spectral properties between the quadrature-based and quadrature-free approaches diverge more, which is attributed to the evaluation of edge fluxes that, in the piecewise constant case, leads to nearly identical discrete expressions. With rising discretization order, the number of edge quadrature points grows, and so do the differences in the edge flux computation between the quadrature-based and quadrature-free scheme (see (2.14) and the corresponding discussion in Section 2.3.1). Nevertheless, these differences are very small and, even in the worst case (p2), do not exceed one percent of the calculated value of the free surface elevation.

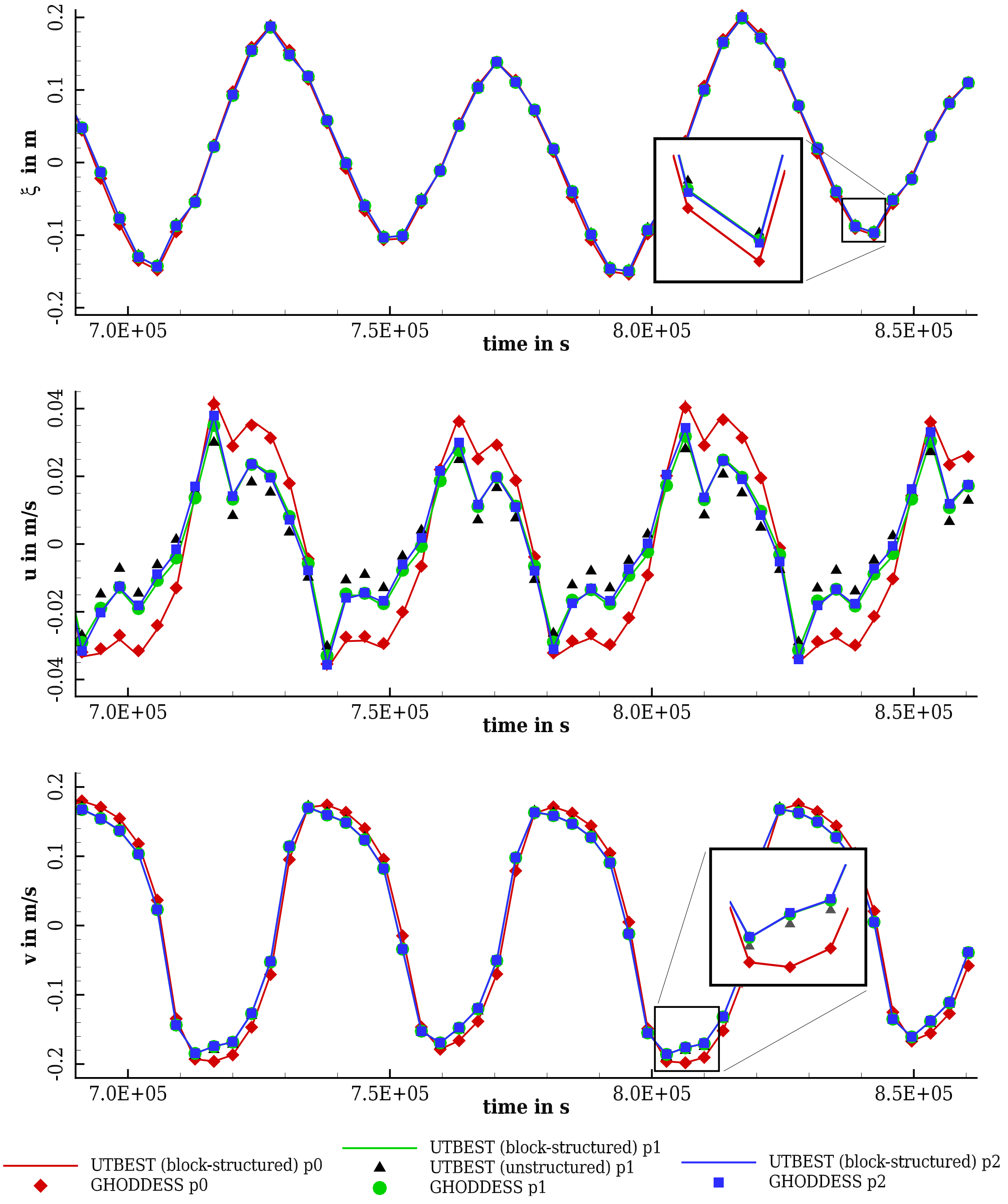


Figure 4.12: Tidal flow at Bight of Abaco: elevation at Station 1 (top), depth-averaged x-velocity at Station 2 (middle), depth-averaged y-velocity at Station 4 (bottom) for simulation days 9 and 10 (adapted from [FN+20]).

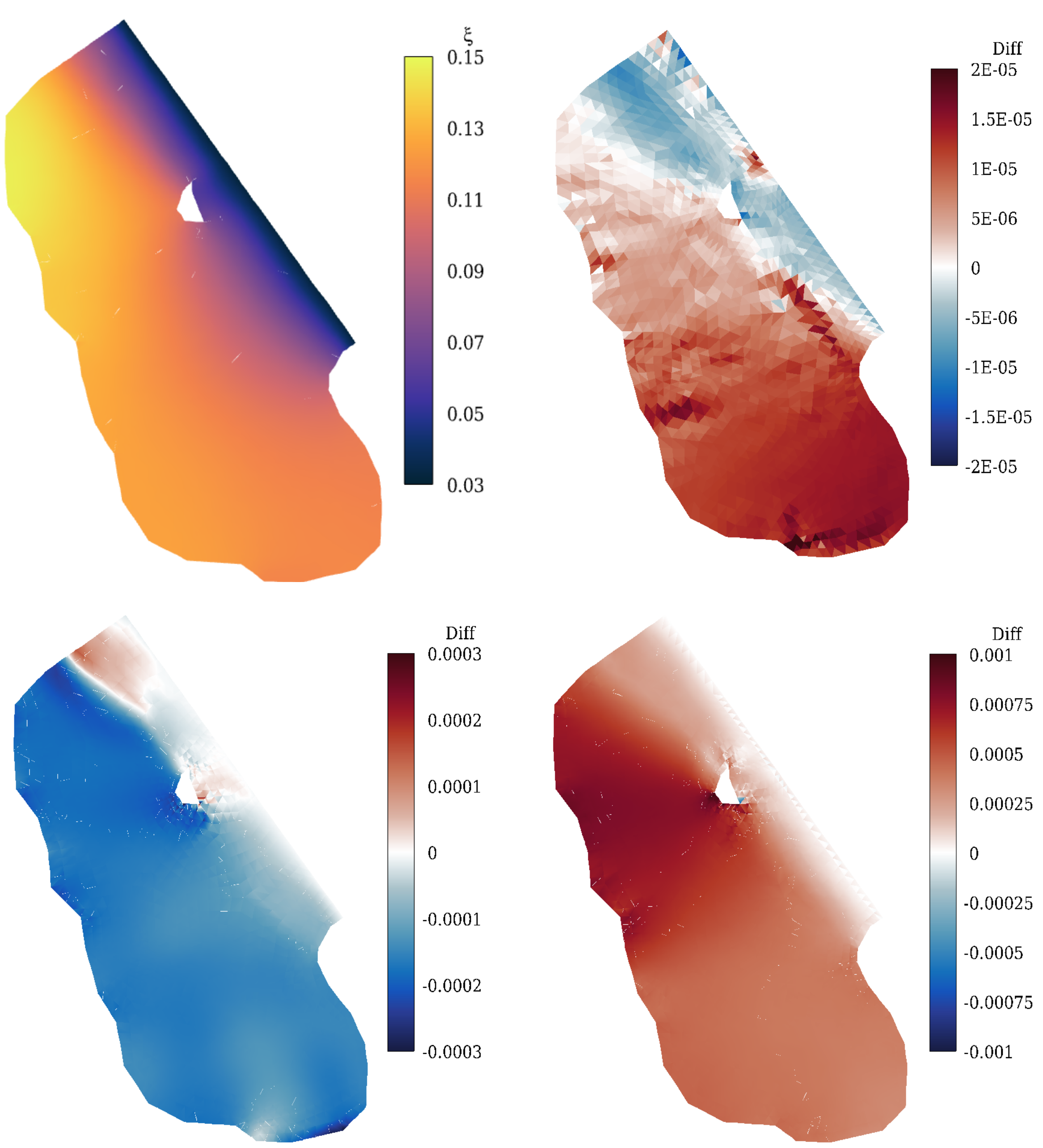


Figure 4.13: Tidal flow at Bight of Abaco: free surface elevation ξ at the end of day 10 obtained using UTBEST (quadrature-based) with p2 (top left). $\xi - \xi_{\text{ref}}$ difference between GHODDESS (quadrature-free) and UTBEST (quadrature-based) solutions of the same DG discretization order on the same BSG at the end of day 10: p0 (top right), p1 (bottom left), p2 (bottom right). All numbers are specified in meters (adapted from [FN+20]).

Finally, two properties of our quadrature-free discretization have to be mentioned here that play an important role for the SWE. Firstly, the local conservation of all primary variables is of paramount importance. Equally crucial is the well-balanced nature of the scheme, specifically with regard to zero-velocity conditions. Similar to the original quadrature-based DG scheme realized in UTBEST, these properties remain preserved in the new quadrature-free implementation since the modifications only concern the evaluation of element and edge integrals. No balance or conservation relationships have been affected, and no artificial terms have been introduced. The latter property was explicitly verified by running the Bight of Abaco test case in a lake-at-rest configuration (not shown here). The former property can be analytically demonstrated by choosing a discrete test function equal to one on a single element and zero otherwise.

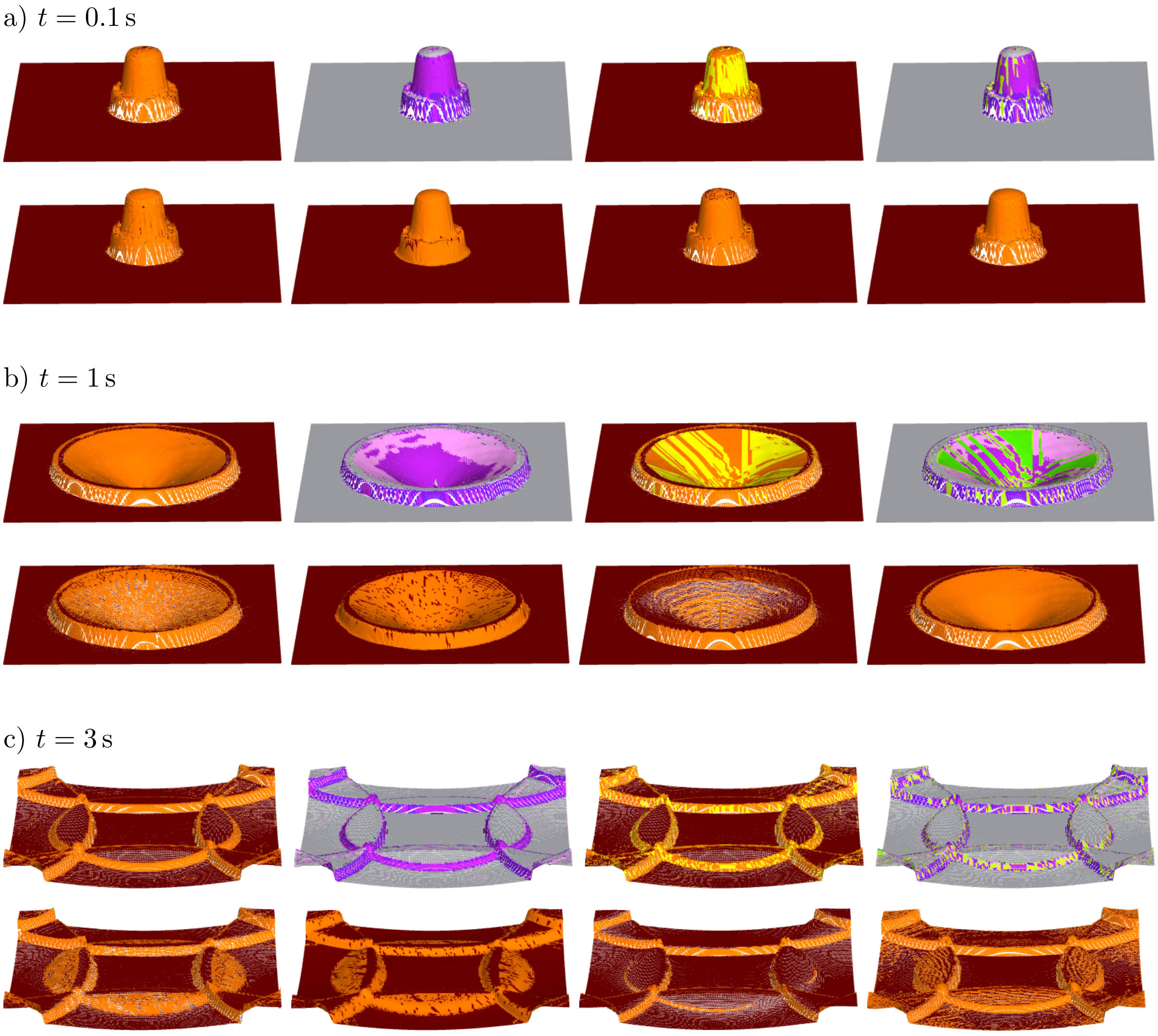
4.3 Evaluation of the parameter-free adaptivity indicator

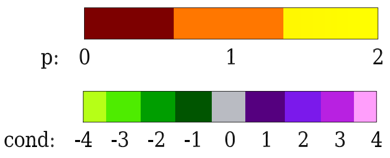
In this section, we evaluate the accuracy and robustness of the adaptivity indicator proposed in Section 2.3.4.1. We present two different benchmarks encompassing various flow regimes that differ in their spatial (constant, smoothly varying, shocks) and temporal (stationary, time-varying) attributes. In addition to our proposed indicator, we conducted simulations employing two alternative indicators as detailed in Section 2.3.4.2. Note that in this section, the simulations using the JRL indicator from Section 2.3.4.1 accept the reconstructed limited p1 solution in the left branch (p0) if its jump is smaller than 90 % of the base jump instead of 100 % as depicted in Figure 2.4. For better comparability, we present limited (see Section 2.3.2) and unlimited results for the adaptive test cases. The presented results are based on the published article [FNA22].

4.3.1 Radial dam break

First, we investigate the radial dam break example from Section 4.1.2, where the constant bathymetry was set to $h_b = 0$. A comparative examination of the p-adaptive solutions, employing different indicators, is presented in Figure 4.14 a) for $t = 0.1\,\mathrm{s}$, in Figure 4.14 b) for $t = 1\,\mathrm{s}$, and in Figure 4.14 c) for $t = 3\,\mathrm{s}$. For the JE indicator, optimal results, that is, the best balance between resolving the flow features well and using as few as possible degrees of freedom, were obtained by applying a threshold value of 0.0003. In case of the GRE indicator, in the unlimited case, this threshold for the free surface elevation was 0.8, while for velocity components, it was set to 1.7. In the case with limiting, a threshold for the free surface elevation of 0.2 and for the velocity components of 0.5 produced the best solution. Our new indicator effectively avoided over- and undershoots while not suffering from excessive numerical diffusion, as in the case of the JE indicator with and without limiting. Meanwhile, the GRE indicator yielded a satisfactory solution if the limiter was active. Otherwise, it was very diffusive and led to over- and undershoots.

These findings are further corroborated by the difference plots for the free surface elevation $\xi - \xi_{\mathrm{ref}}$ shown in Figure 4.15 and the L^1-errors listed in Table 4.2 alongside the fraction of elements with a specific order and the number of degrees of freedom. The simulation results shown in Figure 4.2 served as a reference.





top row:	p0-1, JRL p	p0-1, JRL adaption case	p0-2, JRL p	p0-2, JRL adaption case
bottom row:	p0-1, JE p	p0-1, JE lim p	p0-1, GRE p	p0-1, GRE lim p

Figure 4.14: Radial dam break: free surface elevation at different time levels. Adaption range: constant-linear (p0-1) or constant-quadratic (p0-2). Indicator: JRL, JE, JE limited, GRE, GRE limited. Local approximation order $p \in \{0, 1, 2\}$. Adaption case: see Figure 2.4 (adapted from [FNA22]).

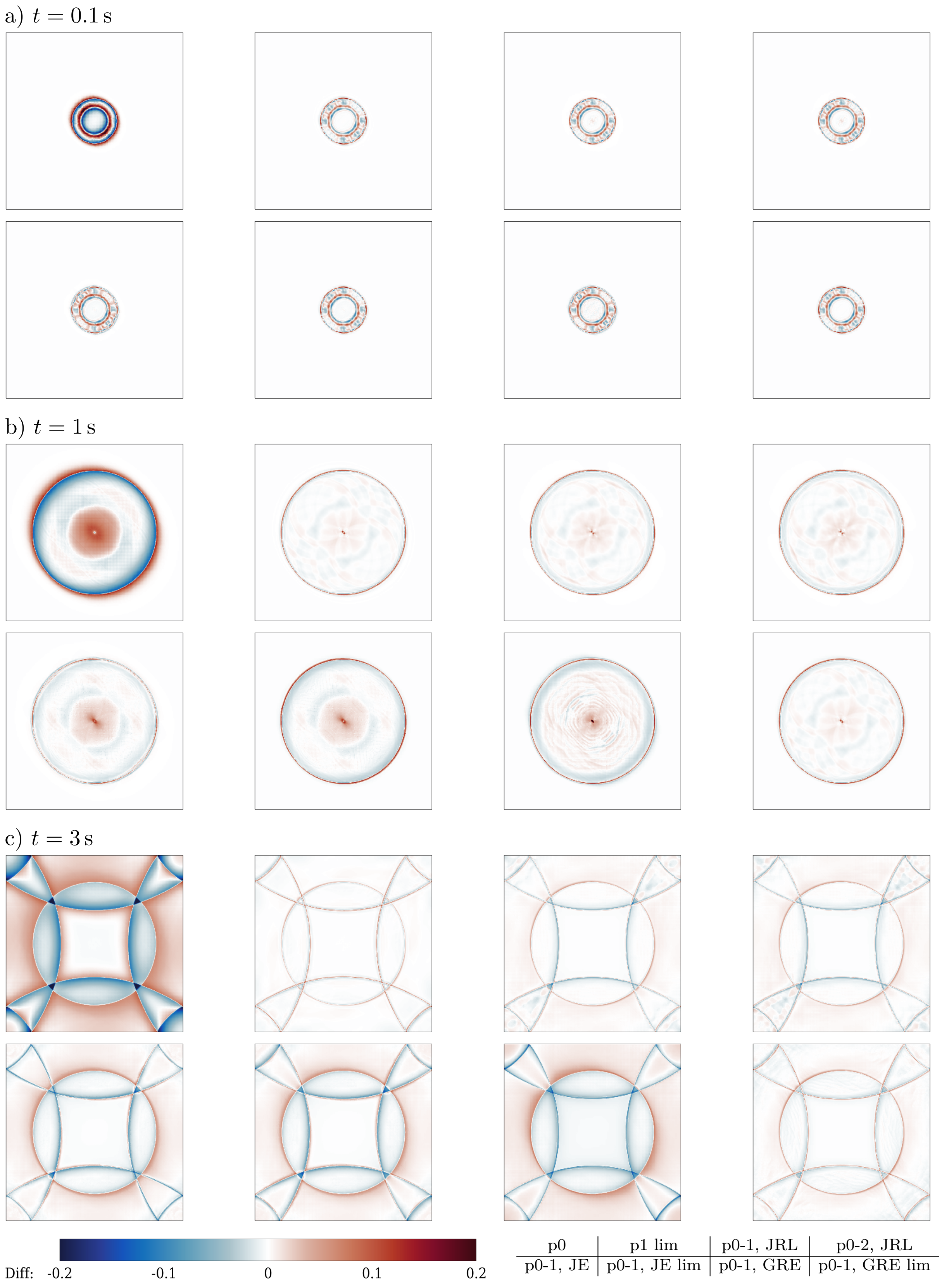


Figure 4.15: Radial dam break: $\xi - \xi_{\mathrm{ref}}$ difference plots at different time levels (from [FNA22]).

Table 4.2: Radial dam break: L^1-error for the free surface elevation, fraction of elements of each discretization order, number of degrees of freedom (DOF), and time step size. Color coding indicates the quality/efficiency (green=good, red=poor) (from [FNA22]).

time in s	# elements	approximation order	L^1-error	order fraction			# DOF	time step size
				0	1	2		
0.1	8192	0	1.79E-01	1.000	0.000	0.000	24 576	0.01
		1 lim	8.26E-02	0.000	1.000	0.000	73 728	0.005
		2 lim	9.64E-02	0.000	0.000	1.000	147 456	0.00125
		0-1 JRL	8.19E-02	0.938	0.062	0.000	27 624	0.005
		0-2 JRL	8.08E-02	0.938	0.042	0.020	29 124	0.005
	32 768	0	1.24E-01	1.000	0.000	0.000	98 304	0.005
		1 lim	4.02E-02	0.000	1.000	0.000	294 912	0.0025
		2 lim	4.64E-02	0.000	0.000	1.000	589 824	0.00625
		0-1 JRL	3.95E-02	0.945	0.055	0.000	109 104	0.0025
		0-1 JE	4.55E-02	0.939	0.061	0.000	110 292	0.0025
		0-1 JE_lim	4.19E-02	0.940	0.060	0.000	110 064	0.0025
		0-1 GRE	4.32E-02	0.948	0.052	0.000	108 492	0.0025
		0-1 GRE_lim	4.04E-02	0.936	0.064	0.000	110 820	0.0025
		0-2 JRL	3.90E-02	0.945	0.036	0.018	114 468	0.0025
	131 072	0	1.15E-01	1.000	0.000	0.000	393 216	0.0025
		1 lim_force	1.41E-02	0.000	1.000	0.000	1 179 648	0.00125
		2 lim	1.94E-02	0.000	0.000	1.000	2 359 296	0.0003125
		0-1 JRL	1.60E-02	0.964	0.036	0.000	421 848	0.001
		0-2 JRL	1.58E-02	0.962	0.027	0.012	437 016	0.001
1	8192	0	7.01E-01	1.000	0.000	0.000	24 576	
		1 lim	1.49E-01	0.000	1.000	0.000	73 728	
		2 lim	1.69E-01	0.000	0.000	1.000	147 456	
		0-1 JRL	1.34E-01	0.585	0.415	0.000	44 988	
		0-2 JRL	1.31E-01	0.580	0.213	0.207	60 510	
	32 768	0	4.39E-01	1.000	0.000	0.000	98 304	
		1 lim	7.19E-02	0.000	1.000	0.000	294 912	
		2 lim	7.68E-02	0.000	0.000	1.000	589 824	
		0-1 JRL	7.97E-02	0.643	0.357	0.000	168 456	
		0-1 JE	1.03E-01	0.652	0.348	0.000	166 776	
		0-1 JE_lim	1.36E-01	0.676	0.324	0.000	161 976	
		0-1 GRE	1.31E-01	0.871	0.129	0.000	123 732	
		0-1 GRE_lim	8.05E-02	0.617	0.383	0.000	173 568	
		0-2 JRL	7.55E-02	0.641	0.191	0.168	218 472	
	131 072	0	1.45E+00	1.000	0.000	0.000	393 216	
		1 lim_force	2.47E-02	0.000	1.000	0.000	1 179 648	
		2 lim	3.39E-02	0.000	0.000	1.000	2 359 296	
		0-1 JRL	6.43E-02	0.779	0.221	0.000	567 168	
		0-2 JRL	6.39E-02	0.782	0.133	0.085	664 134	
3	8192	0	1.29E+00	1.000	0.000	0.000	24 576	
		1 lim	2.16E-01	0.000	1.000	0.000	73 728	
		2 lim	3.18E-01	0.000	0.000	1.000	147 456	
		0-1 JRL	2.37E-01	0.292	0.708	0.000	59 400	
		0-2 JRL	2.26E-01	0.286	0.458	0.255	78 492	
	32 768	0	8.49E-01	1.000	0.000	0.000	98 304	
		1 lim	1.04E-01	0.000	1.000	0.000	294 912	
		2 lim	1.67E-01	0.000	0.000	1.000	589 824	
		0-1 JRL	1.43E-01	0.770	0.230	0.000	143 532	
		0-1 JE	2.86E-01	0.682	0.318	0.000	160 812	
		0-1 JE_lim	3.07E-01	0.678	0.322	0.000	161 565	
		0-1 GRE	3.63E-01	0.916	0.084	0.000	114 732	
		0-1 GRE_lim	1.51E-01	0.590	0.410	0.000	178 860	
		0-2 JRL	1.41E-01	0.759	0.177	0.065	164 760	
	131 072	0	2.03E+00	1.000	0.000	0.000	393 216	
		1 lim_force	3.68E-02	0.000	1.000	0.000	1 179 648	
		2 lim	1.46E-01	0.000	0.000	1.000	2 359 296	
		0-1 JRL	1.27E-01	0.907	0.093	0.000	466 668	
		0-2 JRL	1.30E-01	0.902	0.077	0.021	498 808	

At $t = 0.1\,\mathrm{s}$, in the adaptive cases, more than 93 % of the elements used order 0, and only between 35 % and 40 % of the degrees of freedom of the linear approximation were needed to achieve comparable L^1-errors. At $t = 1\,\mathrm{s}$, in order to obtain comparable L^1-errors, one needed up to approximately 61 % of the degrees of freedom of the uniformly linear approximation for p0-1 and for p0-2 up to about 82 %. Here, a substantial number of p1 elements were necessary to accurately capture the curvature. At $t = 3\,\mathrm{s}$, the distribution of the approximation order was somewhat more dependent on the resolution. Specifically, for higher resolutions, the fraction of lower-order elements increased, and vice versa. Nevertheless, even in the worst case only approximately 81 % of the degrees of freedom compared to the uniformly higher-order approximation were needed. The solutions with the highest resolution suffered from excessive numerical diffusion for orders 0 and 1 which could be remedied for order 1 by applying the FORCE flux.

To summarize the above investigations, our new parameter-free adaptivity indicator effectively identifies resolved and underresolved regions across different resolutions while adjusting local approximation orders accordingly. This was confirmed in a comparison with two further indicators that showed that our new scheme achieves a good balance between solution quality and number of degrees of freedom used.

4.3.2 Supercritical flow

The supercritical flow example described in Section 4.1.3 is characterized by shocks and constant plateaus. Figure 4.16 depicts the steady-state solutions obtained with various schemes on the BSG, as presented in Figure 4.3. It is apparent that the piecewise constant DG approximation is very diffusive, whereas the limited linear and quadratic solutions solutions do not exhibit this deficiency. As a result of using the adaption scheme, the constant-linear and the constant-quadratic solutions using indicator JRL do not suffer from over- or undershoots and accurately capture the jumps without introducing excessive levels of numerical diffusion.

The robustness and accuracy of our indicator can be inferred from Figure 4.17, which details the local approximation order and the adaption case used (cf. Figure 2.4). As desired, the constant plateaus were approximated using order 0, while higher orders were only activated in the vicinity of the discontinuities. When comparing the JE and GRE indicators, the simulations without limiting produced pronounced over- and undershoots in the vicinity of shocks whereas the limited JE and GRE indicators yielded reasonable results other than being slightly diffusive. For the JE indicator, optimal results were obtained with a threshold of 0.005. In the case of the GRE indicator, the best results were observed with a threshold of 0.02 for the free surface elevation and 0.05 for the velocity.

In Figure 4.18, we present difference plots of $\xi - \xi_{\text{exact}}$ to quantify the performance of different adaptivity indicators. This evaluation's reference was the exact solution projected into the

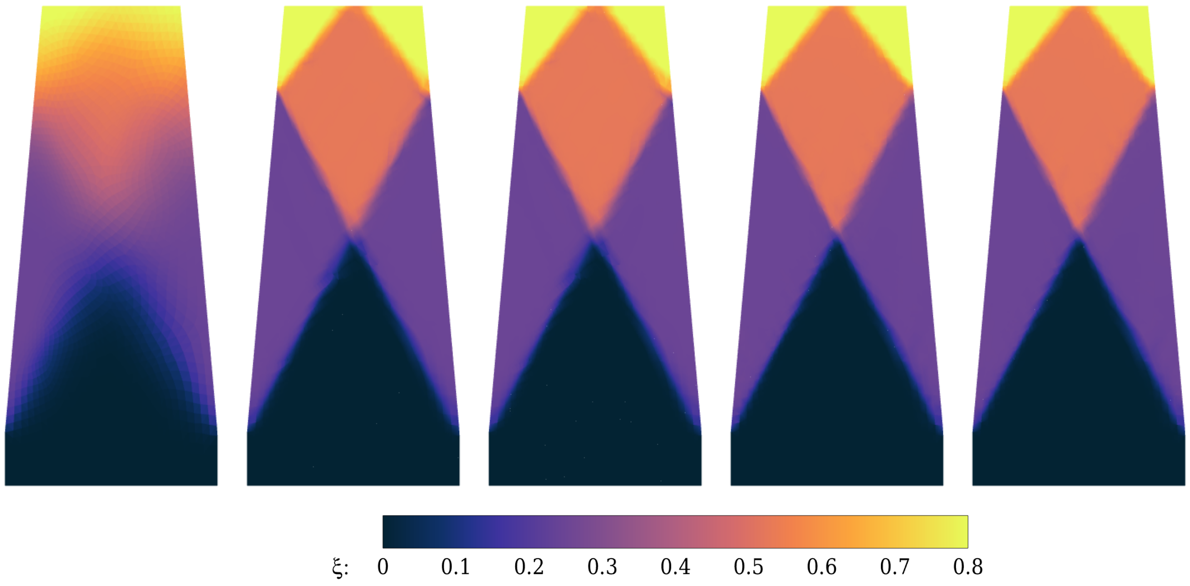
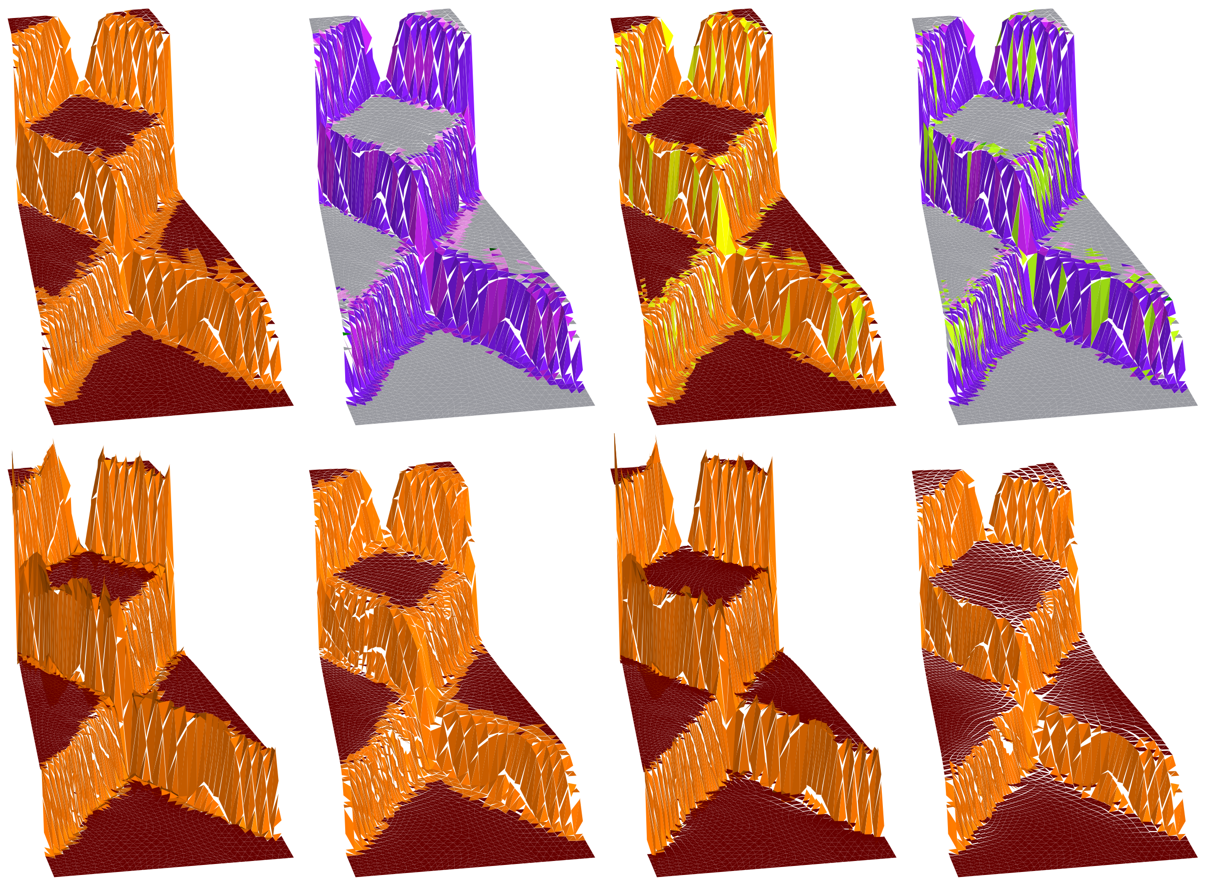


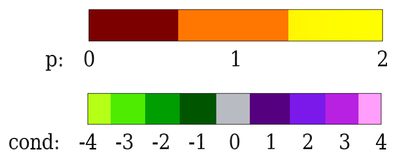
Figure 4.16: Supercritical flow in a constricted channel: free surface elevation. Left to right: constant (p0), limited linear (p1), limited quadratic (p2), constant-linear adaptive (p0-1), constant-quadratic adaptive (p0-2) solution with indicator JRL (adapted from [FNA22]).

constant DG space using a high resolution grid comprising 230 000 elements. The constant solution is clearly diffusive, while the limited linear and quadratic solutions look reasonable. The darker colors in the unlimited constant-linear solutions indicate over- and undershoots, which are suppressed in the fully limited adaptive solutions and also by our adaption scheme incorporating inherent limiting. In Table 4.3, the L^1-errors relative to the exact solution are listed next to the fraction of elements attributed to specific approximation orders and the total number of degrees of freedom. It is evident that between 64 % and 82 % of the elements employed a constant approximation. The p-adaptive solutions required only between 45 % and 58 % of the degrees of freedom of the uniformly linear approximation for p0-1 and approximately 63 % for p0-2.

Table 4.3: Supercritical flow in a constricted channel: L^1-error for the free surface elevation, fraction of elements of each discretization order, number of degrees of freedom, and time step used throughout the simulation. Color coding indicates the quality/efficiency (green=good, red=poor) (from [FNA22]).

approximation order	L^1-error	order fraction 0	1	2	# DOF	time step size
0	1.52e+02	1.000	0.000	0.000	10 752	0.2
1 lim	3.40E+01	0.000	1.000	0.000	32 256	0.1
2 lim	3.37E+01	0.000	0.000	1.000	64 512	0.05
0-1 JRL	3.08E+01	0.645	0.355	0.000	18 390	0.1
0-1 JE	2.98E+01	0.652	0.348	0.000	18 228	0.1
0-1 JE_lim	4.48E+01	0.630	0.370	0.000	18 702	0.1
0-1 GRE	2.96E+01	0.809	0.191	0.000	14 850	0.1
0-1 GRE_lim	4.97E+01	0.817	0.183	0.000	14 682	0.1
0-2 JRL	3.04E+01	0.654	0.283	0.063	20 211	0.1





top row:	p0-1, JRL p	p0-1, JRL adaption case	p0-2, JRL p	p0-2, JRL adaption case
bottom row:	p0-1, JE p	p0-1, JE lim p	p0-1, GRE p	p0-1, GRE lim p

Figure 4.17: Supercritical flow in a constricted channel: free surface elevation. Adaption range: constant-linear (p0-1) or constant-quadratic (p0-2). Indicator: JRL, JE, JE limited, GRE, GRE limited. Local approximation order $p \in \{0, 1, 2\}$. Adaption case: see Figure 2.4 (adapted from [FNA22]). The z-axis is scaled by a factor of 50.

This test case demonstrates the indicator's ability to capture shocks precisely, its efficient utilization of limiting to prevent over- and undershoots while maintaining controlled levels of diffusion, and its effective transition to p0 approximations in constant regions.

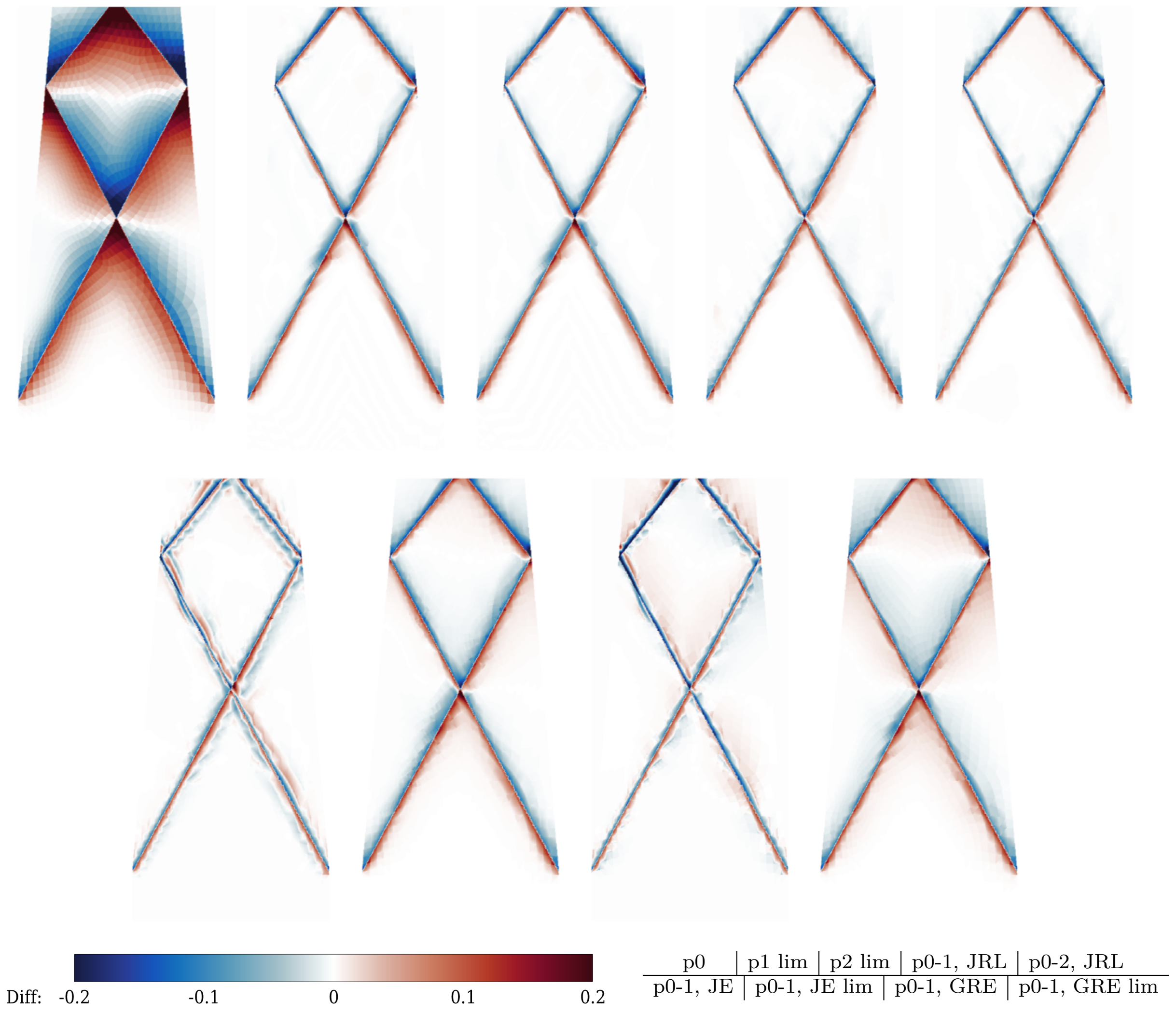


Figure 4.18: Supercritical flow in a constricted channel: $\xi - \xi_{\text{exact}}$ difference plots using the exact solution projected into the constant DG space on a grid with approximately 230 000 elements as the reference (from [FNA22]).

4.4 Evaluation of the separation approach

In this section, we focus on the computational performance of the separation approach proposed in Section 2.3.3 using two simulation scenarios. The results are based on the preprint [FN+23a]. These performance measurements were carried out on two testing platforms.

The first one was an NVIDIA Jetson AGX Xavier SoC (hereafter referred to as ARM–AGX), which is a part of the ICARUS[25] cluster at TU Dortmund, comprising an NVIDIA Carmel Armv8.2 CPU with eight cores alongside an NVIDIA Volta GPU. Throughout our test runs, the CPU's frequency remained fixed at 2100 MHz. Given the shared die and system memory between the CPU and GPU, no distinct memory locations and transfer operations are required, thereby

[25]http://www.mathematik.tu-dortmund.de/sites/icarus-green-hpc

enabling low-latency communication between the CPU and GPU. Therefore, integrated architectures such as the NVIDIA Jetson systems appear particularly promising for our heterogeneous approach and excel, in addition, in the energy-to-solution metric [Gev+16].

The second testing platform was a server with two AMD Epyc 7742 processors with 64 cores each and one NVIDIA Quadro RTX 6000 GPU (hereafter referred to as AMD–RTX). In this setup, the CPU frequency was consistently fixed at 2250 MHz.

For the parallelization of CPU code we used OpenMP and chose the number of threads to get similar execution times between our code's pure CPU and pure GPU versions. Specifically, three threads on the ARM–AGX and 64 threads on the AMD–RTX yielded optimal alignment. We used the most efficient memory management techniques for the measurements presented in the subsequent sections for each specific setup. On the ARM–AGX, the pure GPU implementation demonstrated a narrow speed advantage with pageable memory, whereas the heterogeneous approach exhibited clear superiority with zero-copy memory. On the AMD–RTX, pinned memory was the fastest across all code variants.

The primary computational kernels within our SWE code encompass the following numerical components (cf. Figures 4.21, 4.22 on pages 83, 84):

- edge computation (cf. (2.11) in Section 2.3.1),
- element and right-hand-side (RHS) computation (cf. (2.11) in Section 2.3.1),
- auxiliary computation $\boldsymbol{u}H = \boldsymbol{q}$ (cf. (2.12) in Section 2.3.1),
- minimum depth control to avoid negative depths,
- boundary condition (BC) evaluation,
- the RK step update (cf. (2.21)), and,
- in dynamically p-adaptive runs, the adaptivity indicator (cf. Figure 2.4).

To limit the number of different setups and to enhance understanding, we restrict the differences in approximation order to one within this section.

Measuring how individual kernels affect runtime is valuable for practical application tuning. A performance model could further guide the optimization process, but, due to the kernel complexity and the need for hardware mapping, this goes beyond the scope of our current work.

4.4.1 Radial dam break

The first example, the radial dam break presented in Section 4.1.2 with a constant bathymetry $h_b = 0.5$ on a randomly perturbed uniform grid was chosen because of the simplicity of the domain and the easy problem customizability. Its purpose is twofold: first, to assess the performance of the main computational kernels on the ARM–AGX architecture and, second, to quantify the effect of the separation approach on the total execution time. We designed a range of statically adaptive setups with varying fractions of higher-order elements. This was intended to facilitate a quantitative analysis of the overhead associated with employing a p-adaptive scheme versus adopting a higher-order scheme without adaptivity. The same test case, executed on the AMD–RTX platform, served to illustrate the latency effect of a discrete GPU on the execution time. Finally, a comprehensive evaluation of the computational performance of a dynamic p-adaptive simulation was conducted across diverse arrangements encompassing separated and unseparated (i.e., using the standard approach) as well as various hardware configurations, including CPU, GPU, and heterogeneous combinations.

A randomly perturbed uniform grid consisting of 2 097 152 triangles was employed for all performance measurements. For illustration purposes, Figure 4.19 (top) shows the free surface elevation at $t = 0.1$ s on a grid containing 131 072 triangles across different approximation orders. Meanwhile, Figure 4.19 (bottom) illustrates the local approximation order of the statically adaptive setup, wherein every 32nd element was constrained to employ a higher order. This specific test scenario was chosen because of the inherent challenge it poses to both CPU and GPU in terms of efficient vectorization and memory accesses. In the static setups, computations span 100 time steps utilizing $\Delta t = 0.00001$ s, each encompassing two substeps (RK stages). Subsequently, the execution time was averaged throughout 200 substeps.

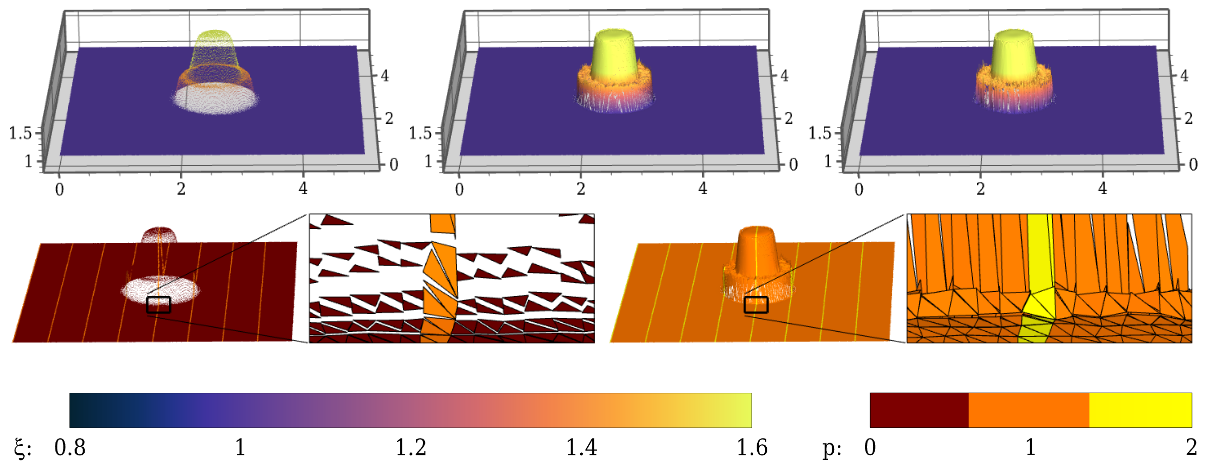


Figure 4.19: Radial dam break: free surface elevation at $t = 0.1$ s. Top row: constant (p0, left), linear (p1, middle), and quadratic solution (p2, right). Bottom row: statically adaptive solution with every 32nd element employing the higher approximation order: constant-linear (p0-1, left) and linear-quadratic (p1-2, right), color-coding shows the local approximation order (adapted from [FN+23a]).

The free surface elevation and the local approximation order for the dynamic p-adaptive cases are depicted in Figure 4.20 at $t = 0.1$ s, $t = 1.0$ s, and $t = 2.5$ s. These simulations were conducted for a total of 12 500 time steps with $\Delta t = 0.0002$ s. Kernel execution timings were averaged across all substeps to capture the variations in the adaptive part of the solution algorithm.

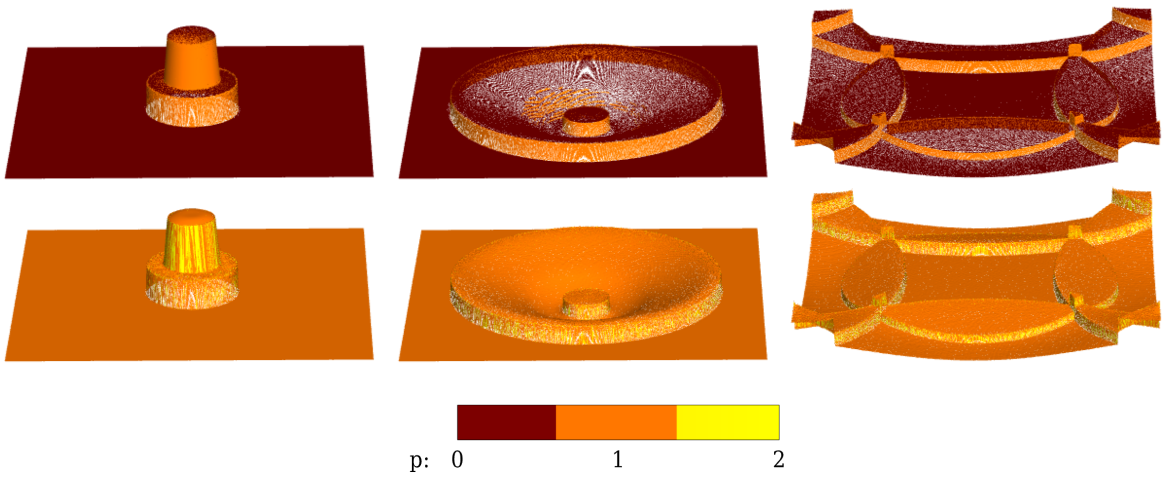
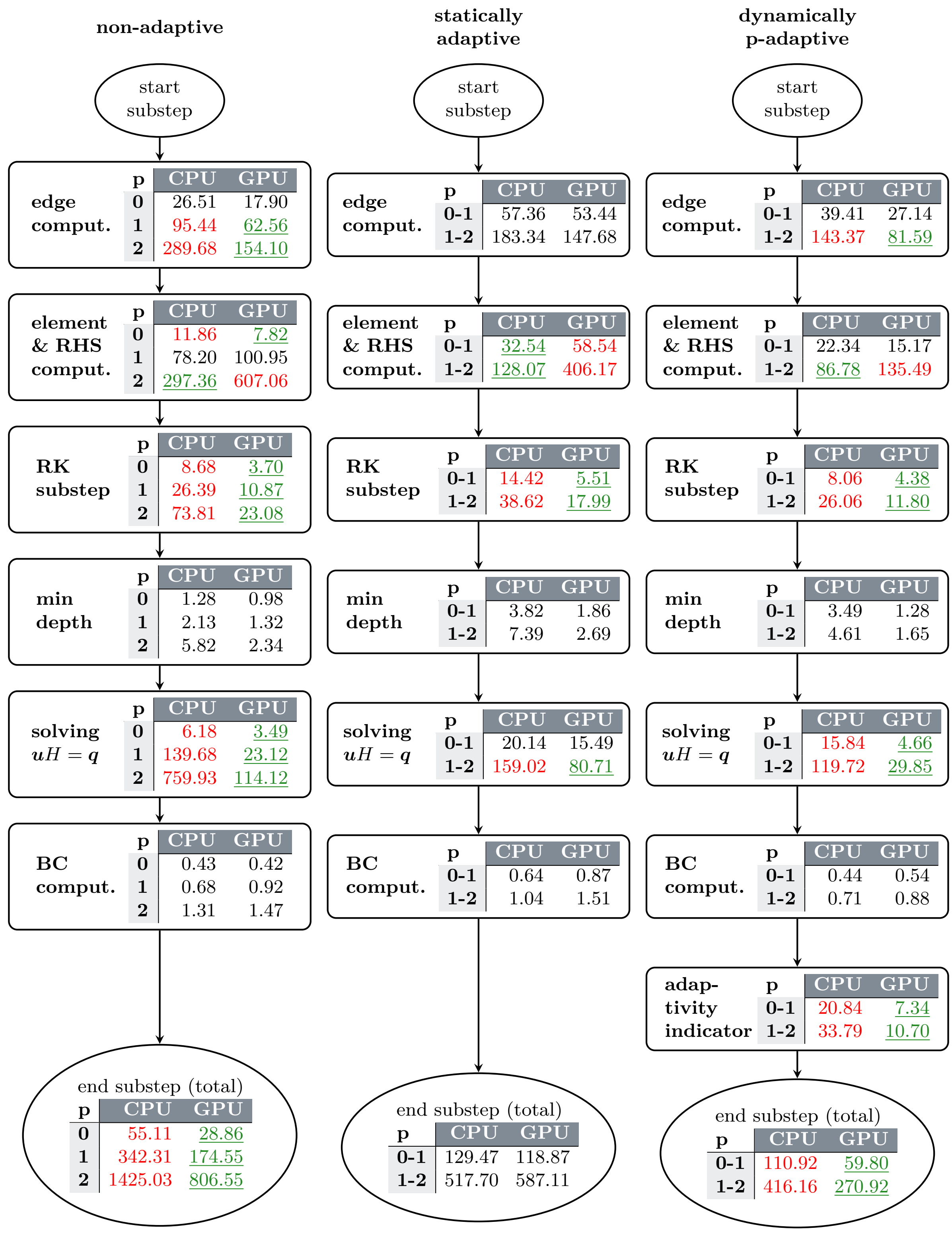


Figure 4.20: Radial dam break: dynamic p-adaptive test. Free surface elevation at $t = 0.1\,\mathrm{s}$ (left), $t = 1.0\,\mathrm{s}$ (middle), and $t = 2.5\,\mathrm{s}$ (right). Top row: p0-1, bottom row: p1-2. Color-coding shows the local approximation order (from [FN+23a]).

Figure 4.21 presents a comparative analysis between the unseparated configurations involving piecewise constant, linear, and quadratic solutions without adaptivity and the statically adaptive counterparts with 1/32 of the elements fixed at the higher order. Additionally, the dynamically p-adaptive outcomes are included in the comparison. It is evident that, for the adaptive setups, the total execution times were much lower than those of the non-adaptive higher-order version.

Subsequently, we activated our novel separation approach and considered the execution times of all kernels across both CPU and GPU architectures (refer to 'homog.' rows in Figure 4.22). To exploit further parallelism, these outcomes guided the allocation of kernels in the separated algorithm between the CPU and GPU (as denoted by 'heterog.' rows in Figure 4.22). The resulting optimal distribution assigned the correction computation to the CPU. In contrast, the fixed non-adaptive computation (denoted by 'fixed') and the remaining kernels, except for the BC computation, were offloaded to the GPU. This distribution led to approximately 22 % speedup compared to the fastest separated computation, either purely executed on the CPU or on the GPU. Contrasted with the fastest unseparated version, we noticed a speedup of 11 %.

In Figure 4.23, we present the substep execution times corresponding to statically adaptive scenarios with different ratios of higher-order elements, which were fixed during the run. Here, a comprehensive comparison was drawn across unseparated and separated schemes executed individually on the CPU, the GPU, or via an optimal heterogeneous distribution of kernels between these two architectures. This allows us to easily quantify the overhead incurred by separation, which mostly boils down to transferring the solution parts between the non-adaptive and adaptive kernels, in each specific configuration. Furthermore, we can infer that the CPU clearly outperformed the GPU when approximately 1/32 or more elements incorporate higher-order approximations. Conversely, the GPU was faster if the fraction of higher-order elements was small. Notably, all adaptive computations showcased improved efficiency relative to their non-adaptive counterparts employing higher-order approximations when executed on



non-adaptive

	p	CPU	GPU
edge comput.	0	26.51	17.90
	1	95.44	62.56
	2	289.68	154.10
element & RHS comput.	0	11.86	7.82
	1	78.20	100.95
	2	297.36	607.06
RK substep	0	8.68	3.70
	1	26.39	10.87
	2	73.81	23.08
min depth	0	1.28	0.98
	1	2.13	1.32
	2	5.82	2.34
solving $uH = q$	0	6.18	3.49
	1	139.68	23.12
	2	759.93	114.12
BC comput.	0	0.43	0.42
	1	0.68	0.92
	2	1.31	1.47
end substep (total)	0	55.11	28.86
	1	342.31	174.55
	2	1425.03	806.55

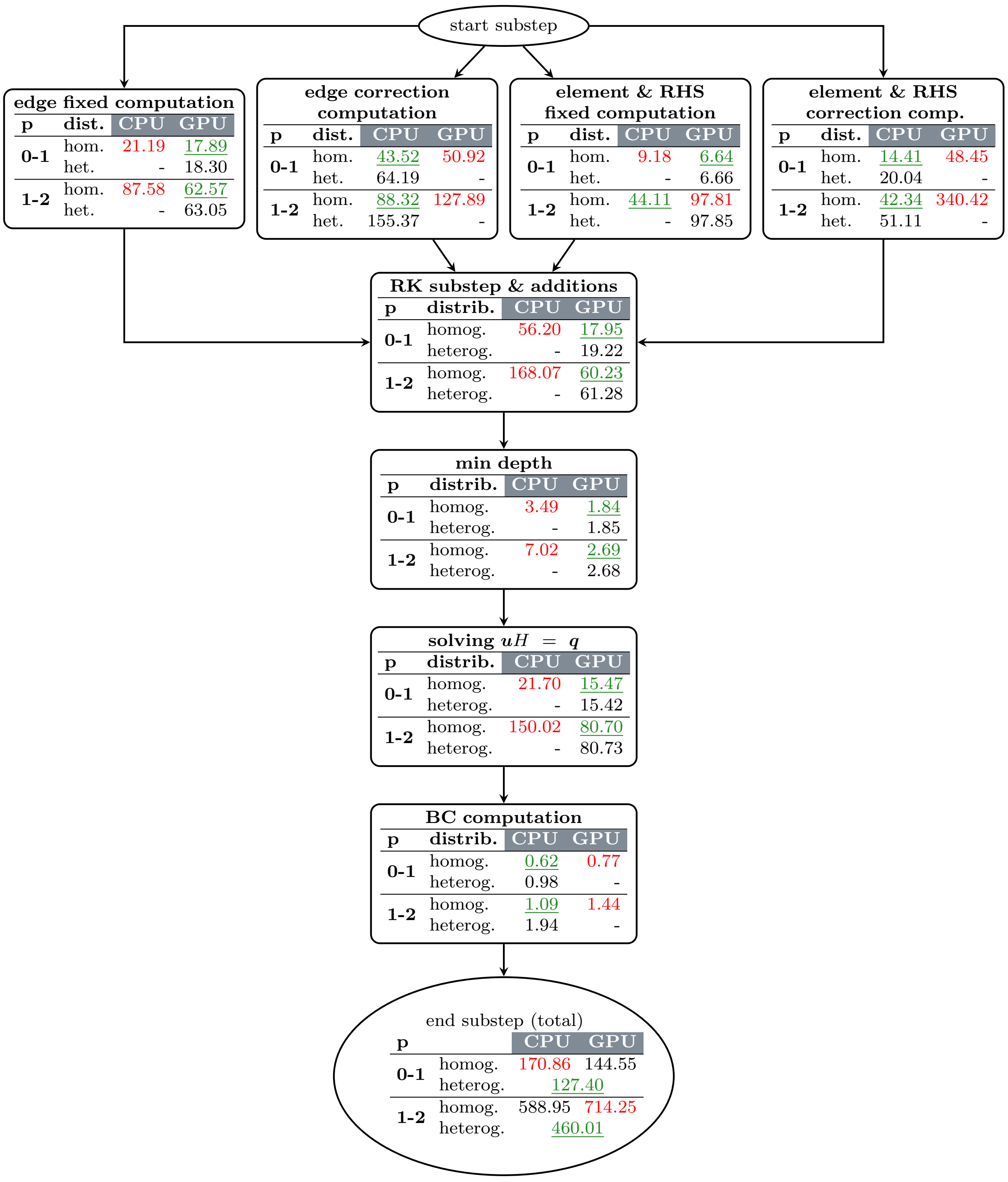
statically adaptive

	p	CPU	GPU
edge comput.	0-1	57.36	53.44
	1-2	183.34	147.68
element & RHS comput.	0-1	32.54	58.54
	1-2	128.07	406.17
RK substep	0-1	14.42	5.51
	1-2	38.62	17.99
min depth	0-1	3.82	1.86
	1-2	7.39	2.69
solving $uH = q$	0-1	20.14	15.49
	1-2	159.02	80.71
BC comput.	0-1	0.64	0.87
	1-2	1.04	1.51
end substep (total)	0-1	129.47	118.87
	1-2	517.70	587.11

dynamically p-adaptive

	p	CPU	GPU
edge comput.	0-1	39.41	27.14
	1-2	143.37	81.59
element & RHS comput.	0-1	22.34	15.17
	1-2	86.78	135.49
RK substep	0-1	8.06	4.38
	1-2	26.06	11.80
min depth	0-1	3.49	1.28
	1-2	4.61	1.65
solving $uH = q$	0-1	15.84	4.66
	1-2	119.72	29.85
BC comput.	0-1	0.44	0.54
	1-2	0.71	0.88
adap-tivity indicator	0-1	20.84	7.34
	1-2	33.79	10.70
end substep (total)	0-1	110.92	59.80
	1-2	416.16	270.92

Figure 4.21: Radial dam break: Data flow and kernel execution times (in ms) on the ARM–AGX platform for the unseparated setup. Piecewise constant, linear, and quadratic solutions (left), statically adaptive p0-1 and p1-2 solutions (middle), dynamically p-adaptive p0-1 and p1-2 solutions (right). We highlight significantly faster execution times (green, underlined) with a difference of more than 1/3 with respect to the slower ones (red) (from [FN+23a]).



edge fixed computation

p	dist.	CPU	GPU
0-1	hom.	21.19	17.89
	het.	-	18.30
1-2	hom.	87.58	62.57
	het.	-	63.05

edge correction computation

p	dist.	CPU	GPU
0-1	hom.	43.52	50.92
	het.	64.19	-
1-2	hom.	88.32	127.89
	het.	155.37	-

element & RHS fixed computation

p	dist.	CPU	GPU
0-1	hom.	9.18	6.64
	het.	-	6.66
1-2	hom.	44.11	97.81
	het.	-	97.85

element & RHS correction comp.

p	dist.	CPU	GPU
0-1	hom.	14.41	48.45
	het.	20.04	-
1-2	hom.	42.34	340.42
	het.	51.11	-

RK substep & additions

p	distrib.	CPU	GPU
0-1	homog.	56.20	17.95
	heterog.	-	19.22
1-2	homog.	168.07	60.23
	heterog.	-	61.28

min depth

p	distrib.	CPU	GPU
0-1	homog.	3.49	1.84
	heterog.	-	1.85
1-2	homog.	7.02	2.69
	heterog.	-	2.68

solving $uH = q$

p	distrib.	CPU	GPU
0-1	homog.	21.70	15.47
	heterog.	-	15.42
1-2	homog.	150.02	80.70
	heterog.	-	80.73

BC computation

p	distrib.	CPU	GPU
0-1	homog.	0.62	0.77
	heterog.	0.98	-
1-2	homog.	1.09	1.44
	heterog.	1.94	-

end substep (total)

p		CPU	GPU
0-1	homog.	170.86	144.55
	heterog.	127.40	
1-2	homog.	588.95	714.25
	heterog.	460.01	

Figure 4.22: Radial dam break: Data flow and kernel execution times (in ms) on the ARM–AGX platform for the separated statically adaptive p0-1 and p1-2 solutions. The faster and slower execution times are highlighted in green (underlined) and red, respectively, to substantiate the decision on the heterogeneous kernel distribution (adapted from [FN+23a]).

the CPU. This trend also held true for non-adaptive GPU execution times as long as the ratio of higher-order elements remained at or below 1/32. In the heterogeneous case, particularly for approximation order p1-2 and certain fractions of higher-order elements (specifically 1/32 and 1/64), we achieved a noteworthy speedup exceeding 10 % relative to the fastest homogeneous version.

The values left of the vertical dashed line displayed in Figure 4.23 illustrate the substep execution times for the dynamically p-adaptive scenario (refer to Figure 4.20) wherein solution accuracy is enforced to be similar to that of the full higher-order solution, cf. Section 4.3.1. For p0-1, an average of approximately 1/482 of the elements employed the higher-order approximation, while for p1-2, this fraction increased to 1/172. Given that dynamically p-adaptive executions outperformed their higher-order counterparts (p1 in Figure 4.23 (left) and p2 in Figure 4.23 (right)) by a factor of more than two, these measurements confirm the benefits of p-adaptivity in general. The heterogeneous version for p1-2 was faster than the separated homogeneous ones and than the unseparated CPU version but not faster than the unseparated GPU version.

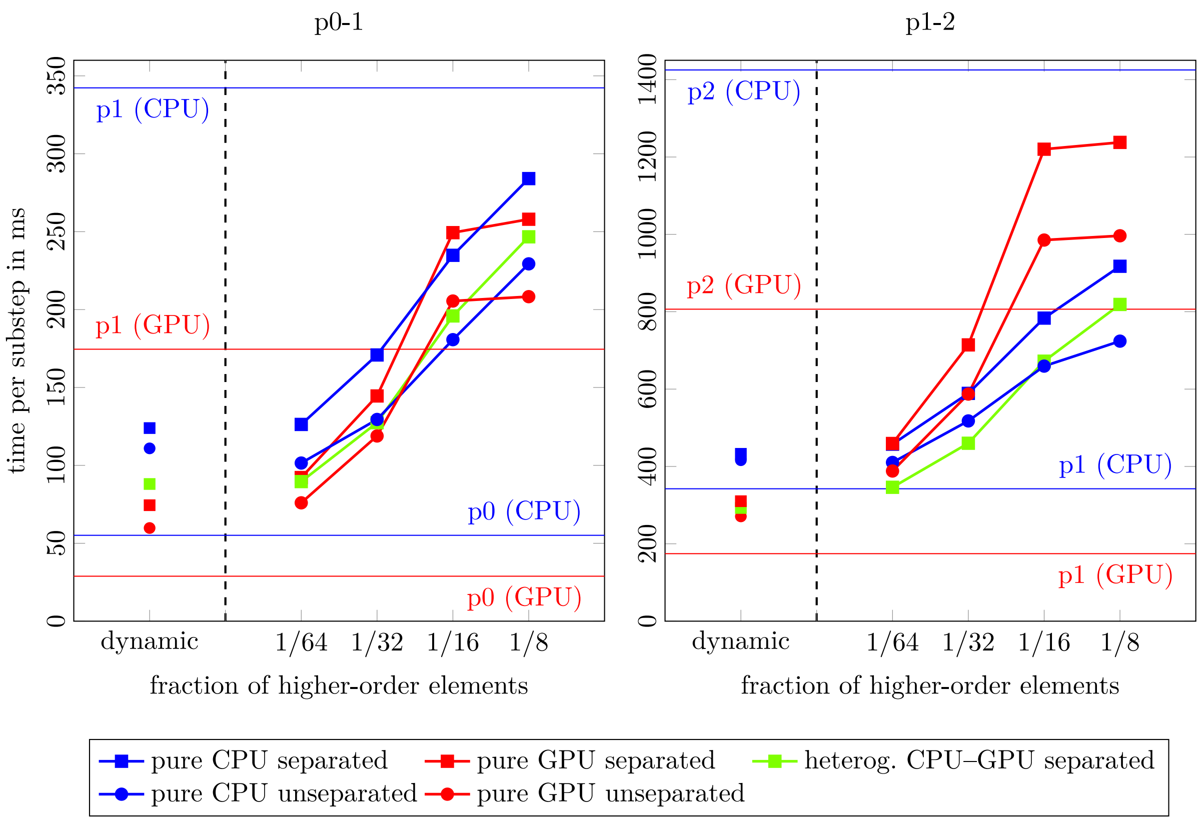


Figure 4.23: Radial dam break: ARM–AGX total execution times for non-adaptive, statically adaptive with different fractions of higher-order elements, and dynamically p-adaptive setups. For p0-1, in the latter case, on average (over the whole simulation), approximately 1/482 of the elements use the higher order and for p1-2, the average fraction of higher-order elements is 1/172. The horizontal lines mark the non-adaptive (p0, p1, and p2) execution times. Constant-linear (left) and linear-quadratic (right) approximation (adapted from [FN+23a]).

When comparing the p0-1 to the p1-2 versions, it is important to recognize that the performance difference between constant and linear computations was smaller than between linear and quadratic computations. Additionally, the overhead caused by separating the element and edge computations was, in some cases, so significant to the extent that it could not be compensated by distributing the kernels and concurrent computations. Here, flexible code generation can be used to the best advantage by easily generating configurations that provide the best performance depending on the problem setup and the hardware configuration.

Achieving efficient heterogeneous kernel distribution is made possible because of the CPU and the GPU sharing the memory on our ARM–AGX SoC architecture. However, in conventional hardware setups featuring discrete GPUs (as observed in the case of AMD–RTX), memory transfers between the CPU and GPU present a substantial bottleneck which is difficult to amortize by any performance benefits arising from heterogeneous kernel parallelism. For the separated versions and p0-1, we attained reasonable substep execution times of 24.9 ms on the CPU and 16.1 ms on the GPU. Notably, the corresponding times for the heterogeneous setup were significantly larger, measuring 101.0 ms. This pattern persisted within the context of the p1-2 approximation as well, where execution times of 68.1 ms, 72.4 ms, and 222.2 ms were recorded for the CPU, GPU, and the heterogeneous arrangement, respectively.

Table 4.4: Detailed kernel execution times (in ms) for different scenarios. The partial execution times were measured without overlap, i.e., with synchronization after the kernel calls, and therefore, their sums do not always match the total execution times (adapted from [FN+23a]).

test scenario			**dam break**		**dam break**		**dam break**		**dam break**		**dam break**		**Bahamas**		**dam break**	
adaptivity strategy			**static, 1/8**		**static, 1/16**		**static, 1/32**		**static, 1/64**		**dynamic**		**dynamic**		**static, 1/32**	
hardware platform			**ARM-AGX**		**ARM-AGX**		**ARM-AGX**		**ARM-AGX**		**ARM-AGX**		**ARM-AGX**		**AMD-RTX**	
kernel	**p**	**dist.**	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
edge fixed computation	**0-1**	hom.	21.6	17.6	21.6	17.9	21.2	17.9	21.4	17.6	20.7	17.9	90.4	30.6	4.1	1.7
		het.	—	18.3	—	18.3	—	18.3	—	18.3	—	18.3	—	29.8	—	8.4
	1-2	hom.	89.1	62.6	85.5	62.6	87.6	62.6	90.9	62.2	83.7	61.7	351.3	83.8	14.7	5.8
		het.	—	63.1	—	63.1	—	63.1	—	63.1	—	63.1	—	81.5	—	23.1
edge correction comp.	**0-1**	hom.	86.0	99.2	69.9	98.7	43.5	50.9	24.6	26.7	16.2	13.9	150.1	71.6	4.3	4.8
		het.	140.6	—	109.7	—	64.2	—	37.3	—	32.5	—	92.1	—	9.5	—
	1-2	hom.	196.0	254.0	161.4	251.9	88.3	127.9	48.7	65.5	20.5	23.5	279.9	189.5	12.3	12.0
		het.	411.0	—	281.7	—	155.4	—	83.4	—	38.3	—	225.4	—	22.9	—
elem & RHS fixed comp.	**0-1**	hom.	9.2	6.6	9.2	6.6	9.2	6.6	9.3	6.6	8.1	6.6	57.6	13.6	2.0	0.8
		het.	—	6.7	—	6.7	—	6.7	—	6.7	—	6.7	—	13.8	—	0.8
	1-2	hom.	43.7	97.8	43.9	97.8	44.1	97.8	44.0	97.8	42.1	97.8	185.5	117.5	5.0	9.0
		het.	—	97.9	—	97.9	—	97.9	—	97.9	—	98.0	—	117.1	—	9.1
elem & RHS correct. comp.	**0-1**	hom.	26.3	96.2	21.9	96.2	14.4	48.5	8.3	24.5	6.0	7.2	51.4	46.6	1.8	4.5
		het.	42.0	—	30.0	—	20.0	—	13.7	—	11.5	—	31.7	—	1.8	—
	1-2	hom.	105.3	680.5	75.7	679.7	42.3	340.4	22.5	170.7	8.8	36.9	155.8	251.8	3.7	31.5
		het.	127.5	—	92.8	—	51.1	—	29.3	—	12.5	—	131.8	—	3.9	—
RK substep & addition	**0-1**	hom.	97.7	32.1	76.6	25.3	56.2	18.0	42.3	14.1	31.7	16.3	270.4	32.2	9.9	4.3
		het.	—	32.8	—	26.2	—	19.2	—	15.5	—	17.7	—	34.1	—	59.4
	1-2	hom.	259.7	96.5	211.3	85.6	168.1	60.2	139.3	45.0	123.8	48.6	701.8	71.2	23.1	11.6
		het.	—	95.9	—	86.1	—	61.3	—	46.4	—	50.2	—	74.2	—	118.1
min depth	**0-1**	hom.	3.6	2.5	3.7	2.5	3.5	1.8	3.5	1.5	2.9	1.3	7.1	3.1	0.5	0.2
		het.	—	2.5	—	2.5	—	1.9	—	1.5	—	1.3	—	3.1	—	0.2
	1-2	hom.	8.9	3.8	8.1	3.8	7.0	2.7	5.6	2.1	4.7	1.7	11.0	4.0	0.9	0.3
		het.	—	3.8	—	3.8	—	2.7	—	2.1	—	1.7	—	4.0	—	0.3
solving $uH = q$	**0-1**	hom.	38.6	26.8	30.9	26.9	21.7	15.5	15.6	9.7	16.4	4.7	82.5	15.7	1.0	1.9
		het.	—	26.8	—	26.9	—	15.4	—	9.7	—	4.8	—	15.6	—	4.7
	1-2	hom.	213.8	143.6	196.4	138.1	150.0	80.7	105.0	52.3	112.5	29.9	311.4	71.3	5.8	11.7
		het.	—	143.2	—	138.3	—	80.7	—	52.5	—	30.0	—	71.3	—	14.5
BC computation	**0-1**	hom.	0.6	0.4	0.6	0.8	0.6	0.8	0.6	0.4	0.4	0.5	1.6	1.5	0.9	0.1
		het.	1.0	—	1.0	—	1.0	—	1.0	—	0.8	—	1.8	—	12.6	—
	1-2	hom.	1.2	1.5	1.2	1.5	1.1	1.4	1.1	0.8	0.7	0.9	2.3	1.8	1.6	0.2
		het.	2.2	—	2.1	—	1.9	—	1.9	—	1.2	—	2.5	—	25.6	—
indicator	**0-1**	hom.	—	—	—	—	—	—	—	—	21.1	7.4	93.9	61.2	—	—
		het.	—	—	—	—	—	—	—	—	—	8.1	—	60.5	—	—
	1-2	hom.	—	—	—	—	—	—	—	—	34.7	10.7	113.9	69.2	—	—
		het.	—	—	—	—	—	—	—	—	—	11.3	—	68.9	—	—
total (parallel)	**0-1**	hom.	284.1	258.0	234.8	249.4	170.9	144.6	126.3	92.2	123.9	74.4	826.2	379.7	24.9	16.1
		het.	246.7		195.9		127.4		89.6		88.1		286.5		101.0	
	1-2	homog.	917.5	1237.8	783.5	1220.2	589.0	714.3	457.5	459.3	432.6	310.2	2149.5	1061.0	68.1	72.4
		het.	818.8		672.2		460.1		346.1		292.7		716.1		222.2	

For a comprehensive breakdown of execution times on the AMD–RTX architecture, we direct the reader to the final column of Table 4.4 detailing kernel execution times for all adaptive measurements presented.

4.4.2 Tidal flow at Bight of Abaco with water hump

The second test setup demonstrates the applicability of our novel approach to more complex problems. We investigated a tide-driven flow in the Bight of Abaco as outlined in Section 4.1.5 with a BSG consisting of several blocks. The simulations were started from the lake-at-rest initial conditions with an added water column of 2 meters in height, essentially simulating a prototype tsunami simulation without wetting and drying. These simulations span 50 minutes and were driven by the tidal surface elevation at the open sea boundary. The tidal forcing consisting of five harmonic constituents was ramped up from zero over the period of 0.1 days. Figure 4.24 displays the BSG, which contains 256 blocks with only 32 elements each for better visualization. However, in the actual computations, a four times uniformly refined (achieved through bisecting each edge) BSG was utilized, yielding 8192 elements per block. The total number of elements employed was the same as in the uniform dam break examples. The simulations were executed for 12 000 time steps with $\Delta t = 0.25\,\mathrm{s}$.

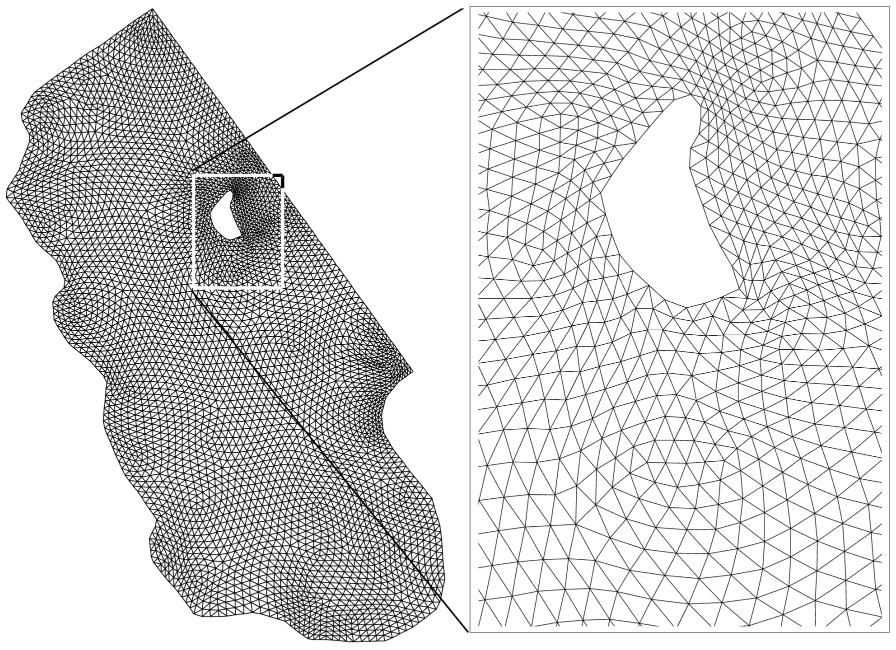


Figure 4.24: Tidal flow at Bight of Abaco: block-structured grid with 256 blocks of 32 elements each. The grid used for the computations was uniformly refined four times, i.e., containing 8192 elements per block (adapted from [FN+23a]).

Figure 4.25 illustrates the temporal evolution of the free surface elevation for various time instances, focusing on the constant-linear approximation (top) along with the corresponding local approximation orders for the constant-linear (middle) and the linear-quadratic (bottom) discretization. In the p0-1 case, about 24.5 % of the elements adopted order 1, and, in the p1-2 case, about 12.1 % of the elements employed order 2.

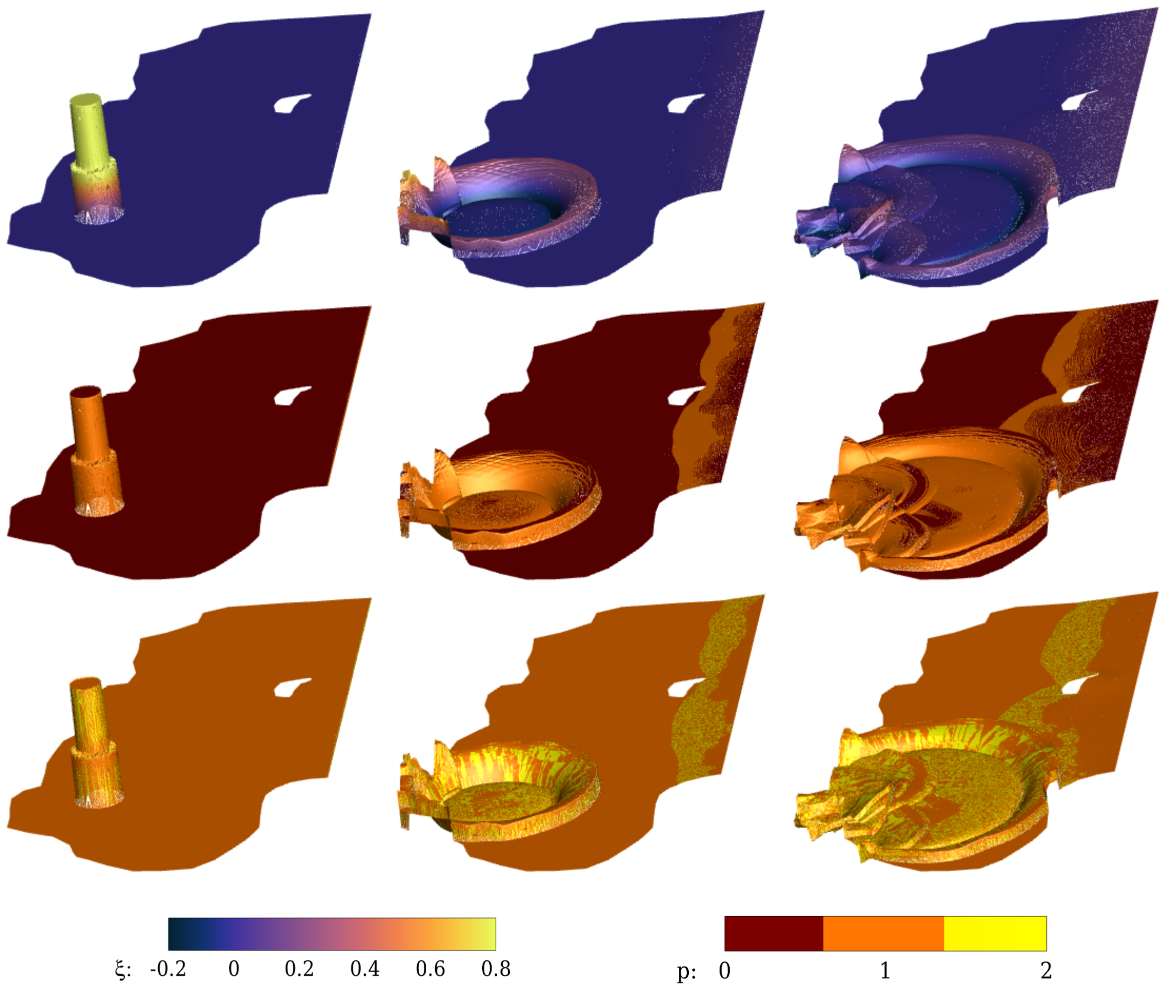


Figure 4.25: Tidal flow at Bight of Abaco: free surface elevation (top row) and local approximation orders for p0-1 (middle row) and p1-2 (bottom row) at $t = 1\,\mathrm{s}$ (left), $t = 25\,\mathrm{s}$ (middle) and $t = 50\,\mathrm{s}$ (right). The z-axis is scaled up by factor 10 000 (from [FN+23a]).

Within the heterogeneous case, we distributed the kernels among the CPU and the GPU, building upon the insights originating from the detailed performance evaluation in Section 4.4.1. Figure 4.26 provides a comprehensive breakdown of kernel execution times, encompassing unseparated and separated configurations on both the CPU and GPU, as well as the scenario involving a heterogeneous kernel distribution. In the heterogeneous case, the CPU kernels are depicted on the left, while GPU kernels are positioned on the right of the corresponding bars. The total execution time exceeded the individual kernel times because of data dependencies that hinder complete overlap. The heterogeneous kernel distribution, as derived in the preceding section, performed correction computations executed on the CPU. Meanwhile, fixed non-adaptive computations and the remaining kernels – excluding the BC computation – were handled on the GPU. Therefore, the heterogeneous bar (the rightmost bar of the corresponding subplots of Figure 4.26) mostly consists of the faster kernels (i.e., smaller blocks) out of separated CPU and GPU execution times plotted in the corresponding bars. When employing the CPU and the GPU in parallel, we obtained an approximate 13 % speedup for the constant-linear approximation and about 22 % speedup for the linear-quadratic one. For a detailed breakdown of kernel execution times, we direct attention to the 'Bahamas' column within Table 4.4. These results show that our approach also works well within the context of realistic simulation scenarios.

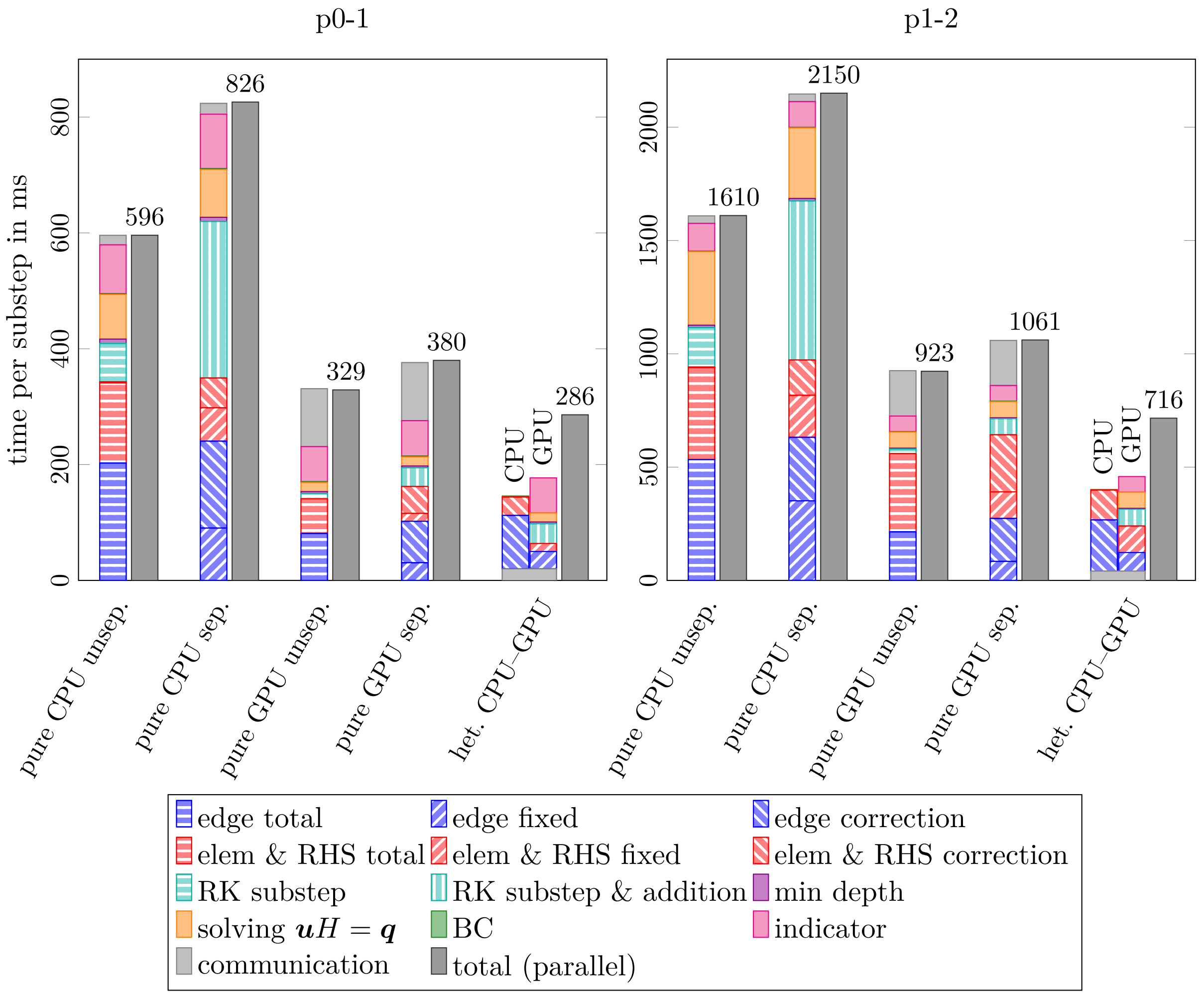


Figure 4.26: Tidal flow at Bight of Abaco: detailed kernel and total execution times for dynamically p-adaptive simulation with unseparated and separated setup on the CPU and the GPU as well as with the optimal heterogeneous distribution (from [FN+23a]).

4.5 Evaluation of block-structured grids with masking

Finally, we investigate masked block-structured grids, as presented in Section 3.2. We validate the accuracy of the numerical scheme and quantify the influence of element masking on the computational performance. This evaluation leverages two distinct real-world problem scenarios of increasing geometric complexity. The results are based on the published article [FN+23b].

The performance measurements in this section were conducted on a single node of the Meggie cluster[26] consisting of two Intel Xeon E5-2630 v4, each featuring ten physical cores. The clock frequency was fixed at 2.2 GHz with hyper-threading functionality disabled.

4.5.1 Tidal flow at Bight of Abaco

The initial test case revisits the tide-driven flow scenario within the Bight of Abaco, detailed in Section 4.1.5, focusing on a relatively small ocean domain of rather simple geometry. This domain is amenable to meshing with unmasked BSGs of satisfactory quality at a resolution comparable to that of the unstructured mesh. Therefore, this benchmark serves as the foundation for evaluating the performance discrepancies between masked and unmasked grids, encompassing variations in resolution, the number of elements per block, diverse DG discretization orders, and various parallelization approaches (MPI, OpenMP, hybrid MPI-OpenMP). Although this particular scenario can be accurately simulated using the coarsest mesh and the first-order DG discretization, the implications of the performance and scaling results are relevant for large, high-resolution simulations typically employed in tsunami and storm surge forecasts.

Figure 4.27 illustrates the original unstructured mesh consisting of 1696 elements (left), the unmasked BSG (middle), and the new masked BSG (right). The unmasked BSG encompasses 58 blocks, each comprising 32 elements; the masked BSG consists of 16 blocks, each containing 128 elements, and has 1860 unmasked (active) elements. The block count was strategically chosen to produce BSGs comparable to the unstructured mesh in terms of resolution and element count. For the piecewise constant discretization (p0), the time step for the unstructured mesh and the masked BSG runs was set equal to 40 s, while for the unmasked BSG, it was set to 30 s (imposed by the CFL condition). For DG orders p1 and p2, the time step sizes for each grid type equated to one-half and one-quarter of the p0 time step sizes, respectively.

First, we validate the simulation results by comparing them to those obtained using the UTBEST model, described in Section 3.3. The temporal evolution of the free surface elevation at Station 1, the depth-integrated x-velocity at Station 4, and the depth-integrated y-velocity at Station 2 is shown in Figure 4.28, showcasing the time series for days 9 and 10. These stations were intentionally chosen because of the most pronounced discrepancies observed across the simulations; it is noteworthy that other stations demonstrated either similar or more favorable agreement.

The findings in Figure 4.28 indicate excellent agreement for the free surface elevation (top plot) for all different runs: unstructured, unmasked block-structured, and masked block-structured for discretization orders p0, p1 and p2. Using zoom-ins, it becomes evident that the influence of discretization order surpassed that of masking. The depth-integrated velocities, shown in the middle and bottom subplots of Figure 4.28, display slightly more sensitivity. This trend aligns fully with the station comparisons previously conducted in [AD02] for this specific test case.

[26] Regional Computing Center Erlangen (RRZE)

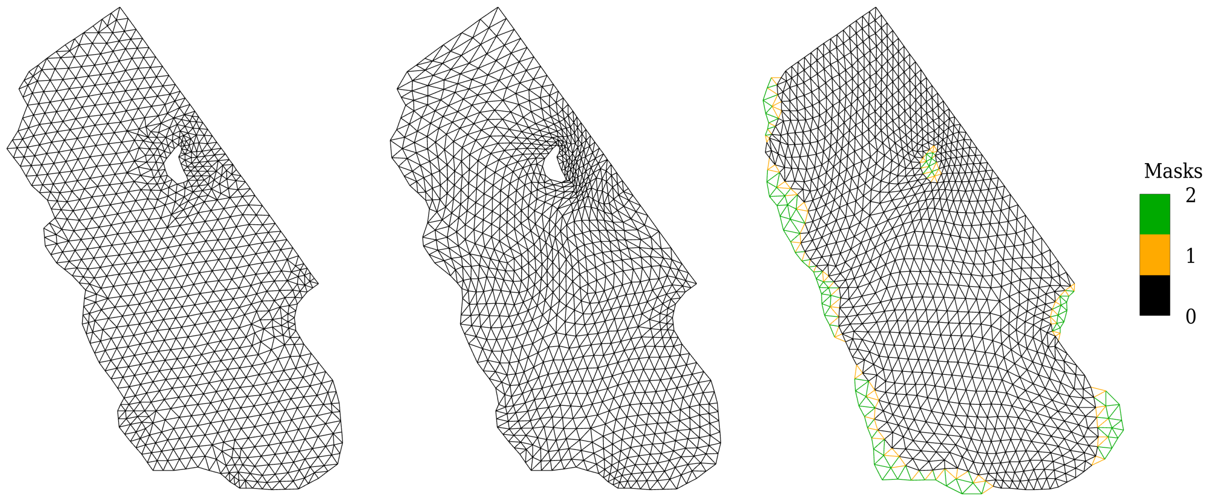


Figure 4.27: Tidal flow at Bight of Abaco: unstructured mesh (left), automatically generated unmasked (middle) and masked (right) BSGs. Black, orange, and green represent active, BC, and inactive elements, respectively (adapted from [FN+23b]).

We proceed to present a comprehensive assessment of computational performance across varying scenarios, encompassing unmasked and masked grids at different resolutions, different DG discretization orders, and a range of parallelization strategies. Only the solve phase of the numerical algorithm was included in the measurements, that is, the setup routines were omitted. To ensure better comparability, all simulations were conducted utilizing 20 cores, equivalent to a full computational node. For pure OpenMP and pure MPI parallelizations, each thread or each MPI rank was pinned to one physical core, respectively. In the context of hybrid parallelization, we chose a distribution of two MPI ranks with ten OpenMP threads each. They were pinned compactly, that is, each rank was pinned to a socket, and all corresponding threads were pinned to the physical cores of that socket.

Table 4.5 provides an overview of grids employed in our performance evaluations. Each type of grid, namely masked and unmasked, is represented by three different resolutions: coarse, medium, and fine. At each resolution, the unmasked grid contains four times as many blocks as the corresponding masked one, yet each individual block comprises only 1/4 of the elements. As a result, both the unmasked and masked grid variants have identical element counts and similar grid resolutions. Nonetheless, the number of active degrees of freedom (DOF) for solution fields can differ between grid types because of the effect of masking, as inactive and BC elements are excluded from the simulation. The fraction of active, inactive, and BC elements for each grid is shown in the corresponding column of Table 4.5. In our runs, masked grids of the same resolution contained fewer active elements than their unmasked counterparts; however, the difference in element count was limited to a maximum of 10 %.

Table 4.6 presents the performance metrics concerning the grids outlined in Table 4.5 for different DG discretization orders and parallelization types. We compared the time required to perform a complete time step normalized by the number of unknowns involved. This metric, called DOF

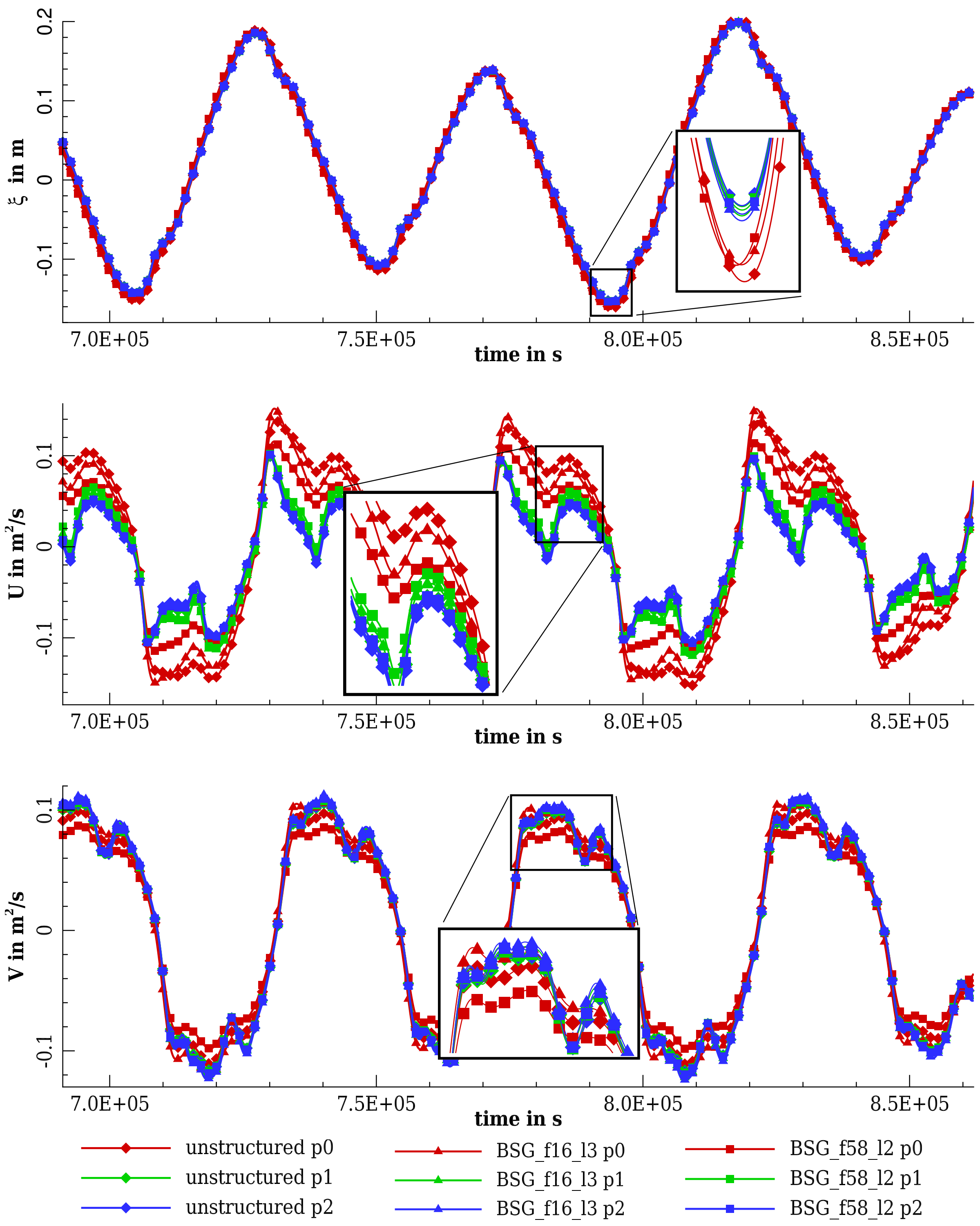


Figure 4.28: Tidal flow at Bight of Abaco: elevation at Station 1 (top), depth-integrated x-velocity at Station 4 (middle), depth-integrated y-velocity at Station 2 (bottom) for days 9 and 10. The triangles represent the masked BSG with 16 blocks and 128 elements each, and the squares represent the unmasked BSG with 58 blocks and 32 elements each (adapted from [FN+23b]).

Table 4.5: Tidal flow at Bight of Abaco: overview of unmasked and masked grids at different resolutions. All unmasked grids consist of 80 blocks, and all masked grids consist of 20 blocks (from [FN+23b]).

resolution	# blocks	# elements per block	element type fraction			total # elements	# DOF p0	# DOF p1	# DOF p2
			active	BC	inactive				
coarse	80	512	1.000	0.000	0.000	4.10e4	1.23e5	3.69e5	7.37e5
	20	2048	0.906	0.011	0.083	4.10e4	1.11e5	3.34e5	6.68e5
medium	80	2048	1.000	0.000	0.000	1.64e5	4.92e5	1.47e6	2.95e6
	20	8192	0.910	0.005	0.084	1.64e5	4.47e5	1.34e6	2.68e6
fine	80	8192	1.000	0.000	0.000	6.55e5	1.97e6	5.90e6	1.18e7
	20	32 768	0.913	0.003	0.085	6.55e5	1.78e6	5.38e6	1.07e7

per microsecond (*DOF per μs*), allows a fair performance comparison between different DG discretization orders, grid resolutions, and masked and unmasked grids. The outcomes reveal that the pure MPI parallelization outperformed the other two alternatives in most cases. One possible cause is the comparably high overhead introduced by the synchronization of OpenMP threads. Moreover, scaling issues could be posed by code parts serialized within each MPI rank in the hybrid case, such as within different MPI functionalities. An additional observation is a reduction in performance as the DG discretization order increased. This is in accordance with the expectations since the amount of work per DOF grows superlinearly with the discretization order, owing to the necessity for evaluating product terms across all element and edge integrals, cf. (2.11)–(2.12). This growth in computational work is ideally offset by (exponentially) higher accuracy of higher-order DG schemes. Also note that, across most scenarios, masked grids exhibited comparable or improved performance relative to their unmasked counterparts.

Table 4.6: Tidal flow at Bight of Abaco: performance of unmasked and masked grids for different resolutions (see Table 4.5), parallelization types, and DG discretization orders (from [FN+23b]).

resolution	parallelization	masking	# DOF per μs p0	# DOF per μs p1	# DOF per μs p2
coarse	OpenMP	no	174.79	154.24	80.53
	OpenMP	yes	158.59	137.95	76.80
	MPI	no	280.55	188.47	90.92
	MPI	yes	373.58	204.02	95.97
	Hybrid	no	180.97	148.95	80.17
	Hybrid	yes	145.34	146.23	82.93
medium	OpenMP	no	310.11	100.92	72.50
	OpenMP	yes	314.24	97.82	64.79
	MPI	no	438.86	115.17	80.02
	MPI	yes	497.20	106.51	71.26
	Hybrid	no	355.92	105.65	76.75
	Hybrid	yes	354.30	100.55	68.94
fine	OpenMP	no	201.07	103.03	76.00
	OpenMP	yes	164.91	104.78	75.98
	MPI	no	223.77	107.51	80.74
	MPI	yes	194.07	101.70	71.63
	Hybrid	no	160.17	92.31	71.65
	Hybrid	yes	175.45	99.19	71.06

While the metric *DOF per μs* provides a good measure of computational performance of a model code, the execution time specifies the total time to solution for a given problem at a prescribed level of accuracy – where accuracy is primarily influenced by factors such as grid resolution and discretization order. Given that masked grids of comparable resolution encompass a smaller number of DOF compared to their unmasked counterparts, a faster execution time of the masked grid is possible, even if its *DOF per μs* metric is comparatively lower. To facilitate a meaningful comparison, we normalized the acquired execution times by the total element count, which remains constant for both masked and unmasked grids sharing the same resolution. The results are summarized in Figure 4.29 and show performance in the *execution time per time step per element* metric.

The findings of Table 4.6 and Figure 4.29 are both based on the same configurations. Comparing them, we can see that they correspond very well to each other. Moreover, the performance benefits of using masked grids are even more clear in the *execution time per time step per element* metric. Evidently, masked grids are faster than their unmasked counterparts in many cases, and where the opposite is true, the differences are slight. On average, one sees an absolute performance improvement due to masking.

4.5.2 Galveston Bay

The last test scenario is the tidal flow in Galveston Bay, described in Section 4.1.6. Given the geometric complexity of the computational domain, employing an unmasked BSG for Galveston Bay would lead to a BSG with either very small blocks, thus limiting the potential for computational performance optimization, or a much higher resolution than that of the initial unstructured mesh. This illustrative example serves to demonstrate the accuracy and computational performance of our methodology for a complex bay geometry containing islands and narrow channels.

Figure 4.30 presents different masked BSGs employed in this study, where we zoomed into the region near the inlet to the ship channel. The BSG in Figure 4.30 (left) encompasses 1838 blocks, each containing 4 elements, resulting in a resolution that effectively captures the domain while masking only a few elements. In contrast, the grid with 465 blocks of 128 elements each in Figure 4.30 (middle) and the BSG with only 90 blocks of 2048 elements each in Figure 4.30 (right) involved the necessity to mask an increasingly larger fraction of elements to accommodate the intricate geometry of the domain.

As shown in Table 4.7, the two latter grids contain more elements in total but managed to resolve the ship channel better and also perform better in the *DOF per μs* metric when employing an MPI parallelization with 20 processes. We have to note here that, in this metric, the Galveston Bay simulations were somewhat slower than the corresponding setups in the Bight of Abaco test case (e.g., 74.79 vs. 106.51 *DOF per* μs for p1 on a masked BSG with 2048 elements per block parallelized with MPI). This difference can be attributed to a larger fraction of masked elements in the Galveston Bay BSGs as well as to load imbalances between MPI ranks that arose from

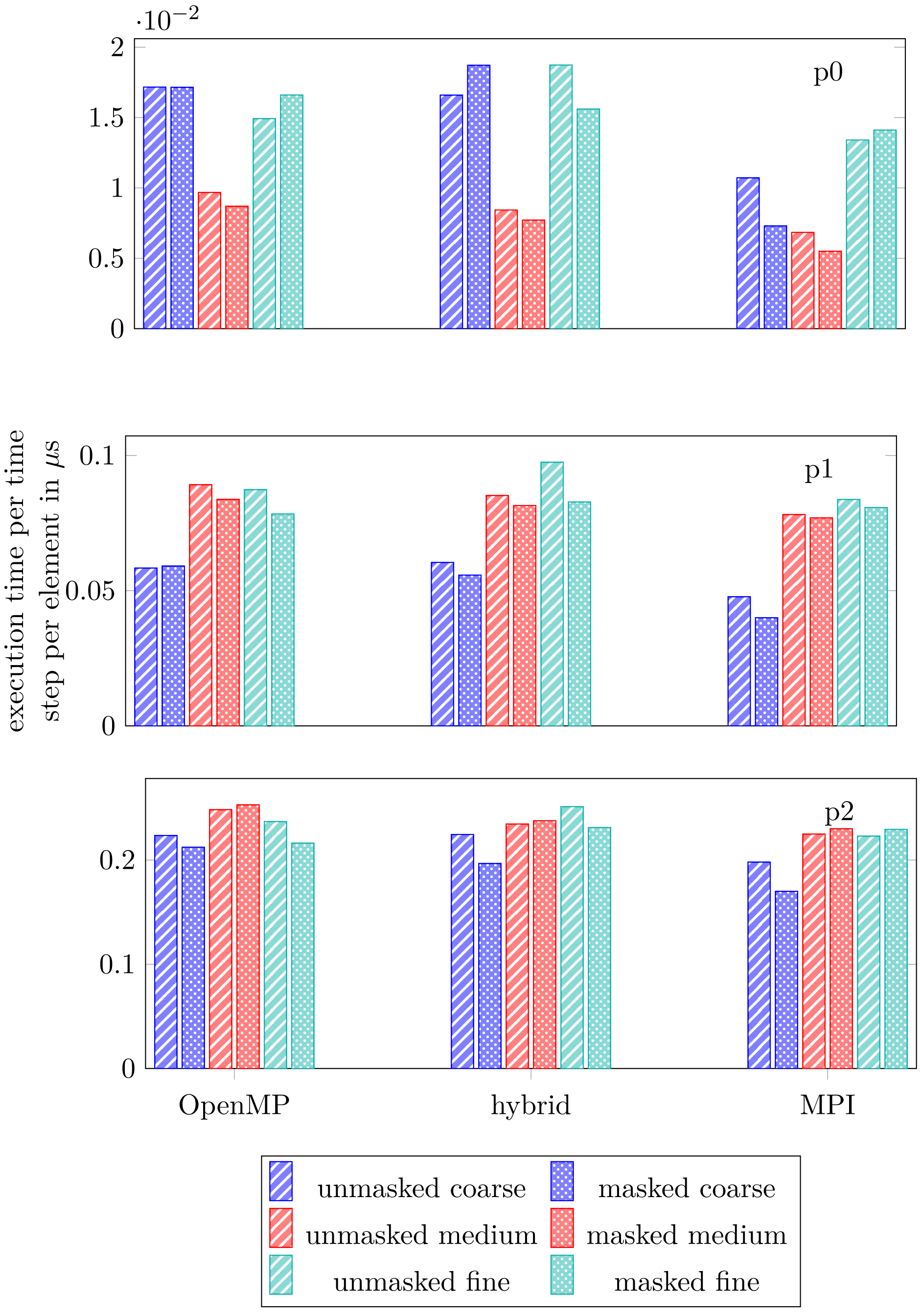


Figure 4.29: Tidal flow at Bight of Abaco: performance for different parallelization types (OpenMP, hybrid, MPI), discretization orders $p \in \{0, 1, 2\}$, and resolutions (coarse, medium, fine – Table 4.5). All times are scaled with the total number of elements in the corresponding grid (adapted from [FN+23b]).

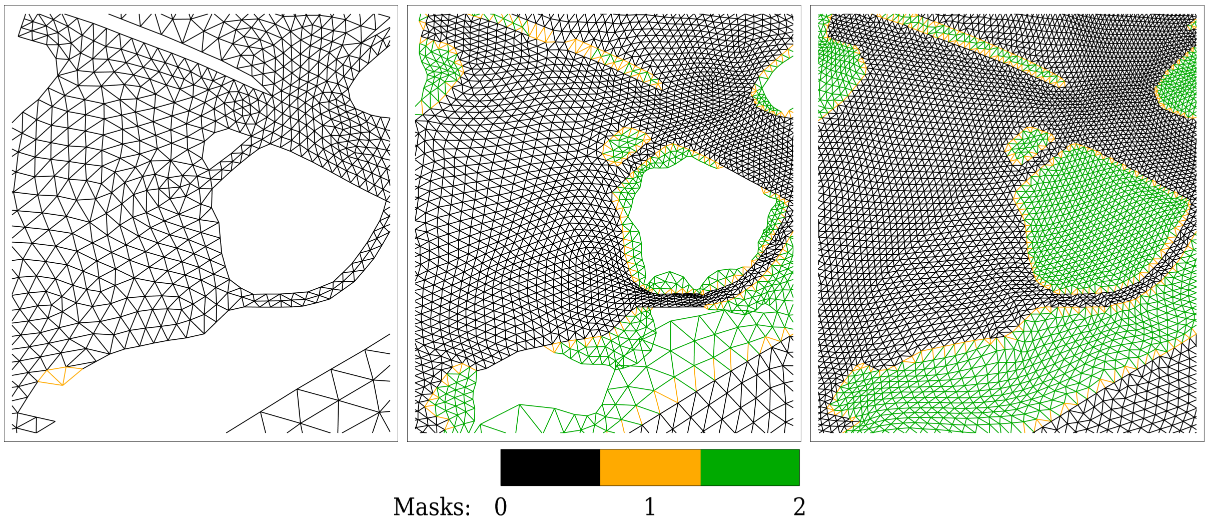


Figure 4.30: Galveston Bay: zoom-in of masked BSGs with 1838 (left), 465 (middle), and 90 (right) blocks. Black, orange, and green represent active, BC, and inactive elements, respectively (adapted from [FN+23b]).

uneven distribution of masked elements among blocks. Moreover, the fact that the number of blocks is not a multiple of 20 (number of MPI ranks) contributes to this difference. For this particular test case, the fine BSG resolutions and the resulting smaller time steps were certainly not necessary. Still, the results are indicative of the performance of numerous applications demanding a fine grid resolution, such as storm surge simulations.

Table 4.7: Galveston Bay: overview and performance of different grids (from [FN+23b]).

configuration	# blocks	# elements per block	element type fractions: active	BC	inactive	total # elements	time step	# DOF p1	# DOF per μs p1
unstructured mesh	1	3397	1.000	0.000	0.000	3.40e3	4	3.06e4	
BSG 1838×8	1838	8	0.997	0.002	0.001	1.47e4	1	1.32e5	26.11
BSG 465×128	465	128	0.854	0.043	0.103	5.95e4	0.5	4.57e5	76.69
BSG 90×2048	90	2048	0.724	0.028	0.248	1.84e5	0.25	1.20e6	74.79

The linear DG solution for the free surface elevation at the end of day 5 computed on the unstructured mesh is shown in Figure 4.31 (top left). The remaining three subfigures of Figure 4.31 contain the difference plots for the free surface elevation $\xi - \xi_{\text{ref}}$ for various masked BSGs using the solution on the unstructured mesh as the reference. All results indicate a good overall agreement, while the largest deviations are close to islands and small channels, as expected. Domain fragments with larger differences seen in the zoom-ins represent artifacts of visualizing the solution difference and correspond to locations where the BSGs and unstructured meshes do not exactly overlap.

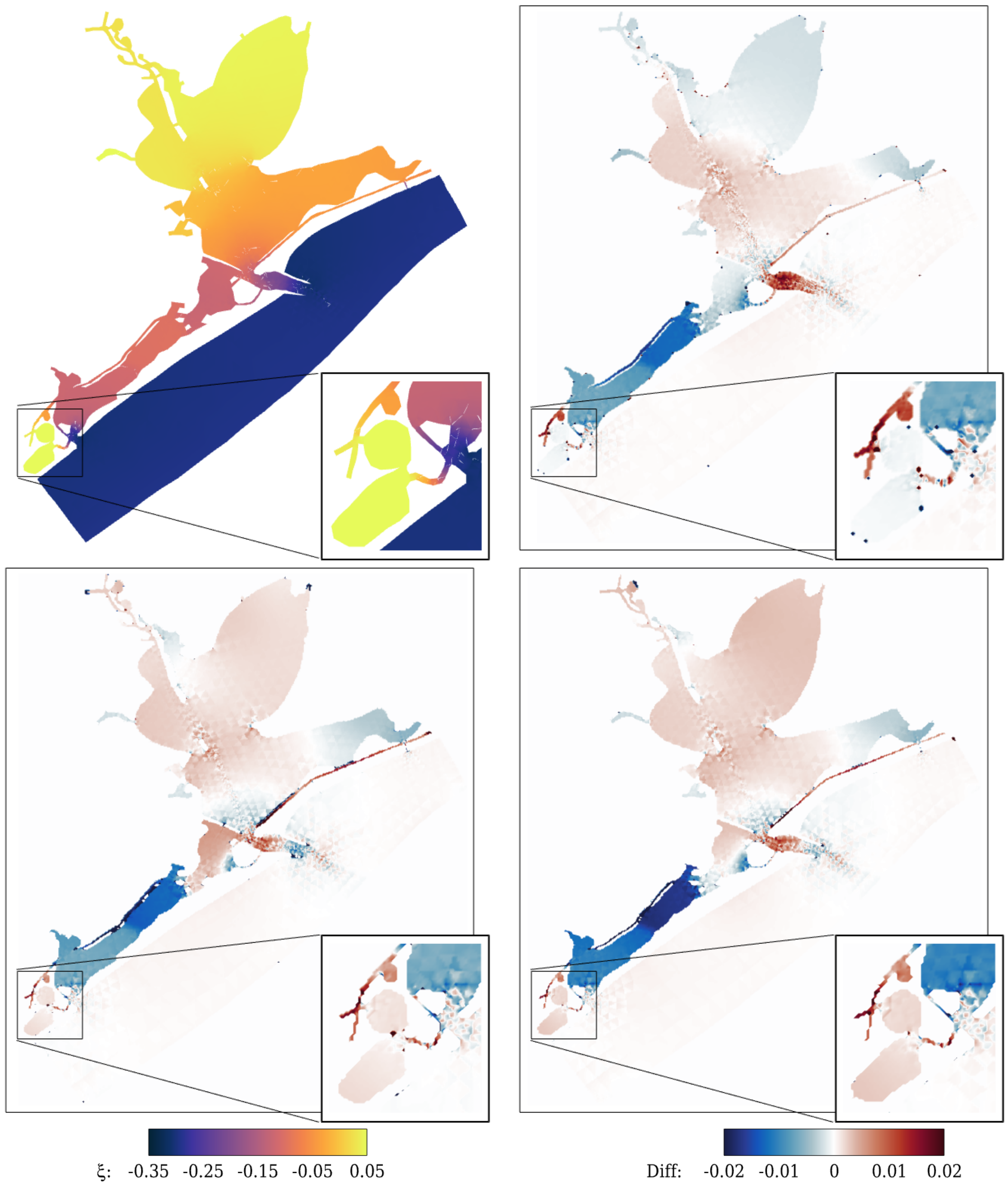

ξ: -0.35 -0.25 -0.15 -0.05 0.05
Diff: -0.02 -0.01 0 0.01 0.02

Figure 4.31: Galveston Bay: linear DG solution for the free surface elevation on unstructured mesh (top left); $\xi - \xi_{\text{ref}}$ difference plots for masked BSGs with 1838 (top right), 465 (bottom left), and 90 (bottom right) blocks using the unstructured mesh solution as the reference (adapted from [FN+23b]). All numbers are specified in meters.

Chapter 5 Conclusion and future prospects

The present thesis investigated a range of new numerical, algorithmic, and computational methodologies with the aim of enhancing the performance of ocean simulations. In particular, it focused on the application of a discontinuous Galerkin (DG) discretization to solve the two-dimensional shallow water equations (SWE) and evaluated the following aspects.

Quadrature-free model reformulation

A new formulation of the conservative SWE was developed, tailored for quadrature-free integration due to its exclusive reliance on product-type nonlinearities. This formulation forms the basis for the algorithmic adaptations in the context of heterogeneous computing. The stability of both the continuous model and the DG discretization for up to piecewise quadratic approximation spaces were proven. This led to the development of a new stability analysis approach to handle the nonlinear advection terms in the quadrature-free model formulation. An important benefit is that the discrete result remains independent of the mesh element size. Furthermore, the findings in numerical experiments reaching piecewise cubic approximations revealed that the new formulation achieves similar accuracy and stability as the quadrature-based one.

Python frontend and automatic code generation

The ExaStencils code generation framework was extended by integrating the open-source Python frontend GHODDESS. Leveraging the SymPy symbolic algebra package for analytical evaluations of integrals and derivatives, GHODDESS performs the translation of the quadrature-free DG formulation of the SWE into ExaSlang layer 4. This integration allowed the reuse of existing optimization and automatic parallelization strategies for various hardware architectures. Within GHODDESS, several optimizations were implemented, including data buffering, projections, and algorithmic variations, which can be selectively activated or deactivated as needed. All modifications were validated against reference results.

p-adaptivity and algorithmic adaptations for heterogeneous computing

A specially redesigned p-adaptive DG scheme for the SWE was introduced. Using a hierarchical modal basis, the approach separates computations associated with the lower-order degrees of freedom from the rest of the discretization. Additionally, automatic code generation techniques were exploited to distribute the computational kernels between the CPU and the GPU based on kernel performance evaluation for specific hardware. Performance measurements showcased the potential for significant performance enhancements in certain simulation scenarios. The new approach fully leverages the optimization potential of Systems-on-a-Chip type hardware platforms, where the CPU and the GPU share memory, and it has load-balancing advantages compared to standard adaptive schemes.

Parameter-free adaptivity indicator

A parameter-free adaptivity indicator was designed to identify resolved and under-resolved areas of the computational domain. It effectively distinguishes between smooth and non-smooth solution regions, adjusting the local approximation order accordingly. Slope limiting is integrated in the indicator and applied where necessary. This new indicator demonstrated similar or improved solution quality in two numerical examples compared to a limited uniformly linear approximation. Importantly, these results were achieved without calibration parameters and involved a significantly reduced number of degrees of freedom. A comparison against two further indicators revealed that the newly introduced scheme achieved a good balance between solution quality and degree of freedom utilization.

Masked block structured grids

The accuracy and computational performance of masked block-structured grids (BSGs) were assessed for realistic ocean domains of varying geometric complexity. In scenarios involving complex boundaries with numerous small-scale features, the masking methodology currently offers the only option to produce BSGs at resolutions comparable to unstructured meshes while maintaining computational efficiency. For simple domain geometries, the masking approach allows to generate BSGs with a higher element count per block compared to their unmasked counterparts of equivalent resolution, thus providing additional structure for leveraging performance enhancements. Simulation results obtained using masked BSGs demonstrated good agreement with those achieved using unstructured meshes. Moreover, performance evaluations showed the advantages of employing masked grids over unmasked ones, as the former allow for a reduction in the number of blocks while preserving grid quality.

Future prospects

Based on the scientific advancements provided by this thesis, a venue of novel and relevant research questions arises, offering prospects for future investigations:

- While numerical experiments suggest convergence for the DG discretization of the two-dimensional SWE system, including the nonlinear advection terms, proving an *a priori* error estimate is still an open question – a proof without requiring the local approximation order to be greater than one would be a significant advancement.
- Extending the parameter-free adaptivity indicator to higher approximation orders and adapting it for further applications such as the Euler equations or modifying it to be a suitable h-adaptivity indicator for the SWE is a potential area for future research.
- Developing a code generation framework capable of efficiently handling unstructured discretizations is an important milestone for realistic high-resolution tsunami or storm surge simulations in combination with domain specific languages.
- The methodology for generating BSGs and the code generation framework could be extended to hybrid meshes containing structured blocks in large parts of the domain and unstructured ones in regions with high geometric complexity or rapidly changing topography. This approach is currently addressed within a DFG project, where work on hybrid grids and numerical algorithms is already underway.
- Incorporating wetting and drying functionalities into GHODDESS is essential for storm surge and inundation forecasts. Nevertheless, this introduces challenges for the numerical simulation regarding robustness, efficiency, and the preservation of physical properties [OLK22].
- Conducting an in-depth performance analysis and comparison between the quadrature-based and the quadrature-free implementation is of interest. This requires an efficient implementation of both schemes ideally within the same framework, which is currently unavailable. Nevertheless, these implementations for the advection equation exist, but a comprehensive performance evaluation and tuning are ongoing efforts.
- An in-depth performance analysis and comparison between unstructured meshes and BSGs on CPUs and GPUs is of relevance. However, achieving this comparison necessitates the efficient implementation of support for both grid types ideally within the same framework, a task also currently in progress as part of the aforementioned DFG project.
- Also, porting the implementation to other types of hardware, such as integrated Intel GPUs or the NVIDIA Grace Hopper Superchip, and comparing the performance in terms of the energy-to-solution metric to traditional CPU and GPU realizations of the same numerical scheme could yield valuable insights.
- In the context of heterogeneous computing, developing an on-the-fly performance measurement system is desirable. Such a system could evaluate kernel execution times at specific intervals during simulation runs and automatically re-distribute kernels as needed.

Bibliography

Publications with own contributions

[Alt+23] C. Alt, T. Kenter, S. Faghih-Naini, J. Faj, J.-O. Opdenhövel, et al. “Shallow Water DG Simulations on FPGAs: Design and Comparison of a Novel Code Generation Pipeline”. In: *High Performance Computing*. Ed. by A. Bhatele, J. Hammond, M. Baboulin, and C. Kruse. Cham: Springer Nature Switzerland, 2023, pp. 86–105. DOI: `10.1007/978-3-031-32041-5_5`.

[FNA22] S. Faghih-Naini and V. Aizinger. “p-adaptive discontinuous Galerkin method for the shallow water equations with a parameter-free error indicator”. In: *International Journal on Geomathematics* 13.18 (2022). DOI: `10.1007/s13137-022-00208-3`.

[FN+23a] S. Faghih-Naini, V. Aizinger, S. Kuckuk, R. Angersbach, and H. Köstler. “p-adaptive discontinuous Galerkin method for the shallow water equations on heterogeneous computing architectures”. In: *submitted to International Journal on Geomathematics, preprint available at* ***https://doi.org/10.48550/arXiv.2311.11348*** (2023).

[FN+20] S. Faghih-Naini, S. Kuckuk, V. Aizinger, D. Zint, R. Grosso, et al. “Quadrature-free discontinuous Galerkin method with code generation features for shallow water equations on automatically generated block-structured meshes”. In: *Advances in Water Resources* 138 (2020), p. 103552. DOI: `10.1016/j.advwatres.2020.103552`.

[FN+23b] S. Faghih-Naini, S. Kuckuk, D. Zint, S. Kemmler, H. Köstler, et al. “Discontinuous Galerkin method for the shallow water equations on complex domains using masked block-structured grids”. In: *Advances in Water Resources* 182 (2023), p. 104584. DOI: `10.1016/j.advwatres.2023.104584`.

[Faj+23] J. Faj, T. Kenter, S. Faghih-Naini, C. Plessl, and V. Aizinger. “Scalable Multi-FPGA Design of a Discontinuous Galerkin Shallow-Water Model on Unstructured Meshes”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC ’23. Davos, Switzerland: Association for Computing Machinery, 2023. DOI: `10.1145/3592979.3593407`.

[Ken+21] T. Kenter, A. Shambhu, S. Faghih-Naini, and V. Aizinger. “Algorithm-Hardware Co-Design of a Discontinuous Galerkin Shallow-Water Model for a Dataflow Architecture on FPGA”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC ’21. Geneva, Switzerland: Association for Computing Machinery, 2021. DOI: `10.1145/3468267.3470617`.

[Zin+22] D. Zint, R. Grosso, V. Aizinger, S. Faghih-Naini, S. Kuckuk, et al. “Automatic Generation of Load-Balancing-Aware Block-Structured Grids for Complex Ocean Domains”. In: *30th International Meshing Roundtable, SIAM IMR 2022*. Zenodo, 2022. DOI: `10.5281/zenodo.6562440`.

Other publications

[Ada+19] S.V. Adams, R.W. Ford, M. Hambley, J.M. Hobson, I. Kavčič, et al. “LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models”. In: *Journal of Parallel and Distributed Computing* 132 (2019), pp. 383–396. DOI: 10.1016/j.jpdc.2019.02.007.

[Afa+21] A. Afanasyev, M. Bianco, L. Mosimann, C. Osuna, F. Thaler, et al. “GridTools: A framework for portable weather and climate applications”. In: *SoftwareX* 15 (2021), p. 100707. DOI: 10.1016/j.softx.2021.100707.

[Aiz04] V. Aizinger. “A discontinuous Galerkin method for two- and three-dimensional shallow-water equations”. PhD thesis. The University of Texas at Austin, 2004.

[Aiz11] V. Aizinger. “A geometry independent slope limiter for the discontinuous Galerkin method”. In: *Computational Science and High Performance Computing IV*. Ed. by E. Krause, Y. Shokin, M. Resch, D. Kröner, and N. Shokina. Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Springer, 2011, pp. 207–217. DOI: 10.1007/978-3-642-17770-5_16.

[Aiz19] V. Aizinger. “Unstructured finite element models for baroclinic ocean flows”. Habilitation thesis. Friedrich–Alexander–Universität Erlangen–Nürnberg, 2019.

[AD02] V. Aizinger and C. Dawson. “A discontinuous Galerkin method for two-dimensional flow and transport in shallow water”. In: *Advances in Water Resources* 25.1 (2002), pp. 67–84. DOI: 10.1016/S0309-1708(01)00019-7.

[AD07] V. Aizinger and C. Dawson. “The local discontinuous Galerkin method for three-dimensional shallow water flow”. In: *Computer Methods in Applied Mechanics and Engineering* 196.4 (2007), pp. 734–746. DOI: 10.1016/j.cma.2006.04.010.

[Aiz+17] V. Aizinger, A. Kosik, D. Kuzmin, and B. Reuter. “Anisotropic slope limiting for discontinuous Galerkin methods”. In: *International Journal for Numerical Methods in Fluids* 84.9 (2017), pp. 543–565. DOI: 10.1002/fld.4360.

[Aiz+18] V. Aizinger, A. Rupp, J. Schuetz, and P. Knabner. “Analysis of a mixed discontinuous Galerkin method for instationary Darcy flow”. In: *Computational Geosciences* 22 (2018). DOI: 10.1007/s10596-017-9682-8.

[AGN93] F. Alcrudo and P. Garcia-Navarro. “A high-resolution Godunov-type scheme in finite volumes for the 2D shallow-water equations”. In: *International Journal for Numerical Methods in Fluids* 16.6 (1993), pp. 489–505. DOI: 10.1002/fld.1650160604.

[AL77] A. Arakawa and V. R. Lamb. “Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model”. In: *General Circulation Models of the Atmosphere*. Ed. by J. Chang. Vol. 17. Methods in Computational Physics: Advances in Research and Applications. Elsevier, 1977, pp. 173–265. DOI: 10.1016/B978-0-12-460817-7.50009-4.

[Arn82] D. N. Arnold. “An Interior Penalty Finite Element Method with Discontinuous Elements”. In: *SIAM Journal on Numerical Analysis* 19 (1982). DOI: 10.1137/0719052.

[AS98] H. L. Atkins and C.-W. Shu. “Quadrature-free implementation of discontinuous Galerkin method for hyperbolic equations”. In: *AIAA Journal* 36.5 (1998), pp. 775–782.

[BB17] J. Baiges and C. Bayona. “Refficientlib: An Efficient Load-Rebalanced Adaptive Mesh Refinement Algorithm for High-Performance Computational Physics Meshes”. In: *SIAM Journal on Scientific Computing* 39.2 (2017), pp. C65–C95. DOI: 10.1137/15M105330X.

[BJ89] T. J. Barth and D. C. Jespersen. “The design and application of upwind schemes on unstructured meshes”. In: 1989. DOI: 10.2514/6.1989-366.

[BR97] F. Bassi and S. Rebay. “A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier–Stokes Equations”. In: *Journal of Computational Physics* 131.2 (1997), pp. 267–279. DOI: 10.1006/jcph.1996.5572.

[BN+22] T. Ben-Nun, L. Groner, F. Deconinck, T. Wicky, E. Davis, et al. “Productive Performance Engineering for Weather and Climate Modeling with Python”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2022. DOI: 10.1109/SC41404.2022.00078.

[Ben+07] M. A. Bender, I. Ginis, R. Tuleya, B. Thomas, and T. Marchok. “The Operational GFDL Coupled Hurricane–Ocean Prediction System and a Summary of Its Performance”. In: *Monthly Weather Review* 135.12 (2007), pp. 3965 –3989. DOI: 10.1175/2007MWR2032.1.

[Bis+00] R. Biswas, S. K. Das, D. Harvey, and L. Oliker. “Parallel dynamic load balancing strategies for adaptive irregular applications”. In: *Applied Mathematical Modelling* 25.2 (2000), pp. 109–122. DOI: 10.1016/S0307-904X(00)00040-8.

[BM87] A. F. Blumberg and G. L. Mellor. “A Description of a Three-Dimensional Coastal Ocean Circulation Model”. In: *Three-Dimensional Coastal Ocean Models*. American Geophysical Union (AGU), 1987, pp. 1–16. DOI: 10.1029/CO004p0001.

[Bos+13] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, et al. “PaRSEC: Exploiting Heterogeneity to Enhance Scalability”. In: *Computing in Science & Engineering* 15.6 (2013), pp. 36–45. DOI: 10.1109/MCSE.2013.98.

[BS07] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods*. Texts in Applied Mathematics. Springer New York, 2007. DOI: 10.1007/978-0-387-75934-0.

[BS05] A. Burbeau and P. Sagaut. “A dynamic p-adaptive Discontinuous Galerkin method for viscous flow with shocks”. In: *Computers & Fluids* 34 (2005), pp. 401–417. DOI: 10.1016/j.compfluid.2003.04.002.

[CCS06] J. Carrero, B. Cockburn, and D. Schötzau. “Hybridized Globally Divergence-Free LDG Methods. Part I: The Stokes Problem”. In: *Mathematics of Computation* 75.254 (2006), pp. 533–563. DOI: 10.1090/S0025-5718-05-01804-1.

[Cas+01] P. Castillo, B. Cockburn, I. Perugia, and D. Schötzau. “An a Priori Error Analysis of the Local Discontinuous Galerkin Method for Elliptic Problems”. In: *SIAM Journal on Numerical Analysis* 38.5 (2001), pp. 1676–1706. DOI: 10.1137/S0036142900371003.

[CGD22] A. Chaplygin, A. Gusev, and N. Diansky. “High-performance Shallow Water Model for Use on Massively Parallel and Heterogeneous Computing Systems”. In: *Supercomputing Frontiers and Innovations* 8 (2022). DOI: 10.14529/jsfi210407.

[CC89] G. Chavent and B. Cockburn. “The Local Projection P^0P^1-Discontinuous-Galerkin Finite Element Method for Scalar Conservation Laws”. In: *RAIRO. Modélisation Mathématique et Analyse Numérique* 23 (1989). DOI: 10.1051/m2an/1989230405651.

[CLB03] C. Chen, H. Liu, and R. C. Beardsley. “An unstructured grid, finite-volume, three-dimensional, primitive equations ocean model: application to coastal ocean and estuaries”. In: *Journal of atmospheric and oceanic technology* 20.1 (2003), pp. 159–186. DOI: 10.1175/1520-0426(2003)020<0159:AUGFVT>2.0.CO;2.

[CE06] E. T. Chung and B. Engquist. “Optimal Discontinuous Galerkin Methods for Wave Propagation”. In: *SIAM Journal on Numerical Analysis* 44.5 (2006), pp. 2131–2158. DOI: 10.1137/050641193.

[Chu+01] J. A. Church, J. M. Gregory, P. Huybrechts, M. Kuhn, K. Lambeck, et al. “Changes in Sea Level”. In: *Climate Change 2001: The Scientific Basis: Contribution of Working Group I to the Third Assessment Report of the Intergovernmental Panel.* Ed. by J. T. Houghton, Y. Ding, D. J. Griggs, M. Noguer, P. J. Van der Linden, et al. 2001, pp. 639–694.

[CHS90] B. Cockburn, S. Hou, and C.-W. Shu. “The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. IV. The multidimensional case”. In: *Mathematics of Computation* 54.190 (1990), pp. 545–581. DOI: 10.1090/S0025-5718-1990-1010597-0.

[CKS00] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *Discontinuous Galerkin Methods.* Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. DOI: 10.1007/978-3-642-59721-3.

[CLS89] B. Cockburn, S.-Y. Lin, and C.-W. Shu. “TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: one-dimensional systems”. In: *Journal of computational Physics* 84.1 (1989), pp. 90–113. DOI: 10.1016/0021-9991(89)90183-6.

[CS89] B. Cockburn and C.-W. Shu. “TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework”. In: *Mathematics of computation* 52.186 (1989), pp. 411–435. DOI: 10.1090/S0025-5718-1989-0983311-4.

[CS91] B. Cockburn and C-W. Shu. “The Runge-Kutta local projection P^1-discontinuous-Galerkin finite element method for scalar conservation laws”. In: *Mathematical Modelling and Numerical Analysis* 34.3 (1991), pp. 377–361. DOI: 10.1051/m2an/1991250303371.

[CS98a] B. Cockburn and C.-W. Shu. “The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems”. In: *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2440–2463. DOI: 10.1137/S0036142997316712.

[CS98b] B. Cockburn and C.-W. Shu. “The Runge–Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems”. In: *Journal of computational physics* 141.2 (1998), pp. 199–224. DOI: 10.1006/jcph.1998.5892.

[CRB11] B. Cushman-Roisin and J. M. Beckers. *Introduction to Geophysical Fluid Dynamics.* International Geophysics. Elsevier Science, 2011.

[DS15] I. Damyanov and M. Sukalinska. “Domain specific languages in practice”. In: *International Journal of Computer Applications* 115.2 (2015). DOI: 10.5120/20126-2205.

[Dan13] S. Danilov. “Ocean modeling on unstructured meshes”. In: *Ocean Modelling* 69 (2013), pp. 195–210. DOI: 10.1016/j.ocemod.2013.05.005.

[DKS04] S. Danilov, G. Kivman, and J. Schröter. “A finite-element ocean model: principles and evaluation”. In: *Ocean Modelling* 6.2 (2004), pp. 125–150. DOI: 10.1016/S1463-5003(02)00063-X.

[Dan+17] S. Danilov, D. Sidorenko, Q. Wang, and T. Jung. “The Finite-volumE Sea ice–Ocean Model (FESOM2)”. In: *Geoscientific Model Development* 10.2 (2017), pp. 765–789. DOI: 10.5194/gmd-10-765-2017.

[Daw+11] C. Dawson, E. Kubatko, J. Westerink, C. Trahan, C. Mirabito, et al. “Discontinuous Galerkin Methods for Modeling Hurricane Storm Surge”. In: *Advances in Water Resources* 34 (2011), pp. 1165–1176. DOI: 10.1016/j.advwatres.2010.11.004.

[DRRIA12] R. Della Ratta Rinaldi, A. Iob, and R. Arina. “An efficient discontinuous Galerkin method for aeroacoustic propagation”. In: *International Journal for Numerical Methods in Fluids* 69.9 (2012), pp. 1473–1495. DOI: 10.1002/fld.2647.

[Del18] D Deltares. “Delft3d flexible mesh suite”. In: *Delft3D-FLOW, User Manual; Deltares: Delft, The Netherlands* (2018).

[DPE12] D. Di Pietro and A. Ern. *Mathematical Aspects of Discontinuous Galerkin Methods*. Vol. 69. 2012. DOI: 10.1007/978-3-642-22980-0.

[Ech+20] I. Echeverribar, M. Morales-Hernández, P. Brufau, and P. García-Navarro. “Analysis of the performance of a hybrid CPU/GPU 1D2D coupled model for real flood cases”. In: *Journal of Hydroinformatics* 22.5 (2020), pp. 1198–1216. DOI: 10.2166/hydro.2020.032.

[Eng02] R. van Engelen. “ATMOL: A Domain-Specific Language for Atmospheric Modeling”. In: *Journal of Computing and Information Technology1; Vol.9 No.4* 9 (2002). DOI: 10.2498/cit.2001.04.02.

[EG04] A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*. Applied Mathematical Sciences. Springer New York, 2004. DOI: 10.1007/978-1-4757-4355-5.

[EG22] A. Ern and J.-L. Guermond. “Invariant-Domain-Preserving High-Order Time Stepping: I. Explicit Runge–Kutta Schemes”. In: *SIAM Journal on Scientific Computing* 44.5 (2022), A3366–A3392. DOI: 10.1137/21M145793X.

[Esk11] C. Eskilsson. “An hp-adaptive discontinuous Galerkin method for shallow water flows”. In: *International Journal for Numerical Methods in Fluids* 67 (2011), pp. 1605–1623. DOI: 10.1002/fld.2434.

[Eva10] L. C. Evans. *Partial Differential Equations*. Graduate studies in mathematics. American Mathematical Society, 2010.

[For+04] R. Ford, C. C. Pain, M. D. Piggott, A. J. H. Goddard, C. R. E. De Oliveira, et al. “A nonhydrostatic finite-element model for three-dimensional stratified oceanic flows. Part I: Model formulation”. In: *Monthly Weather Review* 132.12 (2004), pp. 2816–2831.

[Fri+19] O. B. Fringer, C. Dawson, R. He, D. K. Ralston, and Y. J. Zhang. “The future of coastal and estuarine modeling: Findings from a workshop”. In: *Ocean Modelling* 143 (2019), p. 101458. DOI: 10.1016/j.ocemod.2019.101458.

[Fu+17] H. Fu, L. Gan, C. Yang, W. Xue, L. Wang, et al. “Solving global shallow water equations on heterogeneous supercomputers”. In: *PLOS ONE* 12 (2017), e0172583. DOI: 10.1371/journal.pone.0172583.

[GG+19] M. Garcia-Gasulla, G. Houzeaux, R. Ferrer, A. Artigues, V. López, et al. “MPI+X: task-based parallelisation and dynamic load balance of finite element assembly”. In: *International Journal of Computational Fluid Dynamics* 33.3 (2019), pp. 115–136. DOI: 10.1080/10618562.2019.1617856.

[GZ05] M. Garland and Y. Zhou. “Quadric-based simplification in any dimension”. In: *ACM Transactions on Graphics* 24.2 (2005), pp. 209–239. DOI: 10.1145/1061347.1061350.

[GTS06] E. L. Geist, V. V. Titov, and C. E. Synolakis. “Tsunami: WAVE of CHANGE”. In: *Scientific American* 294.1 (2006), pp. 56–63. DOI: 10.1038/scientificamerican0106-56.

[Ger+07] H. Gerritsen, E. D. De Goede, F. W. Platzek, M. Genseberger, J. A. T. M. Van Kester, et al. “Validation Document Delft3D-FLOW; a software system for 3D flow simulations”. In: *The Netherlands: Delft Hydraulics, Report X* 356 (2007), p. M3470.

[Gev+16] M. Geveler, B. Reuter, V. Aizinger, D. Göddeke, and S. Turek. “Energy efficiency of the simulation of three-dimensional coastal ocean circulation on modern commodity and mobile processors”. In: *Computer Science : Research + Development* 31.4 (2016), pp. 225–234. DOI: 10.1007/s00450-016-0324-5.

[GT01] D. Gilbarg and N.S. Trudinger. *Elliptic Partial Differential Equations of Second Order*. Classics in Mathematics. Springer Berlin Heidelberg, 2001.

[GKS11] S. Gottlieb, D. Ketcheson, and C.-W. Shu. *Strong Stability Preserving Runge-Kutta and Multistep Time Discretizations*. WORLD SCIENTIFIC, 2011. DOI: 10.1142/7498.

[GS98] S. Gottlieb and C.-W. Shu. “Total variation diminishing Runge-Kutta schemes”. In: *Math. Comp.* 67.221 (1998), pp. 73–85. DOI: 10.1090/S0025-5718-98-00913-2.

[GST01] S. Gottlieb, C.-W. Shu, and E. Tadmor. “Strong Stability-Preserving High-Order Time Discretization Methods”. In: *SIAM Review* 43.1 (2001). DOI: 10.1137/S003614450036757X.

[Gri+00] S. M. Griffies, C. Böning, F. O. Bryan, E. P. Chassignet, R. Gerdes, et al. “Developments in ocean climate modelling”. In: *Ocean Modelling* 2.3 (2000), pp. 123–192. DOI: 10.1016/S1463-5003(00)00014-7.

[Gro19] T. H. Gronwall. “Note on the Derivatives with Respect to a Parameter of the Solutions of a System of Differential Equations”. In: *Annals of Mathematics* 20.4 (1919), pp. 292–296. DOI: 10.2307/1967124.

[Gys+15] T. Gysi, C. Osuna, O. Fuhrer, M. Bianco, and T. C. Schulthess. “STELLA: a domain-specific tool for structured grid methods in weather and climate models”. In: *SC ’15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* 2015, pp. 1–12. DOI: 10.1145/2807591.2807627.

[Hai+00] D. B. Haidvogel, H. G. Arango, K. Hedstrom, A. Beckmann, P. Malanotte-Rizzoli, et al. “Model evaluation experiments in the North Atlantic Basin: simulations in nonlinear terrain-following coordinates”. In: *Dynamics of Atmospheres and Oceans* 32.3 (2000), pp. 239–281. DOI: 10.1016/S0377-0265(00)00049-X.

[HB99] D. B. Haidvogel and A. Beckmann. *Numerical Ocean Circulation Modeling.* Vol. 2 Series on Environmelntal Science and Management. Imperical College Press, 1999. DOI: 10.1142/p097.

[Haj21] H. Hajduk. “Monolithic convex limiting in discontinuous Galerkin discretizations of hyperbolic conservation laws”. In: *Computers & Mathematics with Applications* 87 (2021), pp. 120–138. DOI: 10.1016/j.camwa.2021.02.012.

[Haj+18] H. Hajduk, B. R. Hodges, V. Aizinger, and B. Reuter. “Locally Filtered Transport for computational efficiency in multi-component advection-reaction models”. In: *Environmental Modelling & Software* 102 (2018), pp. 185–198. DOI: 10.1016/j.envsoft.2018.01.003.

[HKA19] H. Hajduk, D. Kuzmin, and V. Aizinger. “New directional vector limiters for discontinuous Galerkin methods”. In: *Journal of Computational Physics* 384 (2019), pp. 308–325. DOI: 10.1016/j.jcp.2019.01.032.

[HJ20] A. Hakim and J. Juno. “Alias-Free, Matrix-Free, and Quadrature-Free Discontinuous Galerkin Algorithms for (Plasma) Kinetic Equations”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* SC ’20. IEEE Press, 2020.

[Har+08] S. Harig, Chaeroni, W. S. Pranowo, and J. Behrens. “Tsunami simulations on several scales”. In: *Ocean Dynamics* 58 (2008), pp. 429–440. DOI: `10.1007/s10236-008-0162-5`.

[HH02] R. Hartmann and P. Houston. “Adaptive Discontinuous Galerkin Finite Element Methods for the Compressible Euler Equations”. In: *Journal of Computational Physics* 183.2 (2002), pp. 508–532. DOI: `10.1006/jcph.2002.7206`.

[Haw11] K. Hawick. “Engineering Domain-Specific Languages and automatic code generation for computational simulations of complex systems”. In: *Proceedings of the IASTED International Conference on Software Engineering and Applications, SEA 2011* (2011). DOI: `10.2316/P.2011.758-046`.

[HD00] B. Hendrickson and K. Devine. “Dynamic load balancing in computational mechanics”. In: *Computer Methods in Applied Mechanics and Engineering* 184.2 (2000), pp. 485–500. DOI: `10.1016/S0045-7825(99)00241-8`.

[Hil+06] K. Hillewaert, N. Chevaugeon, Ph. Geuzaine, and J.-F. Remacle. “Hierarchic multigrid iteration strategy for the discontinuous Galerkin solution of the steady Euler equations”. In: *International Journal for Numerical Methods in Fluids* 51.9-10 (2006), pp. 1157–1176. DOI: `10.1002/fld.1135`.

[Ipp51] A. Ippen. “High-Velocity Flow in Open Channels: A Symposium: Mechanics of Supercritical Flow”. In: *Transactions of the American society of Civil Engineers* 116.1 (1951), pp. 268–295. DOI: `10.1061/TACEAT.0006520`.

[Irr+22] G. Irrmann, S. Masson, É. Maisonnave, D. Guibert, and E. Raffin. “Improving ocean modeling software NEMO 4.0 benchmarking and communication efficiency”. In: *Geoscientific Model Development* 15.4 (2022), pp. 1567–1582. DOI: `10.5194/gmd-15-1567-2022`.

[JP86] C. Johnson and J. Pitkäranta. “An Analysis of the Discontinuous Galerkin Method for a Scalar Hyperbolic Equation”. In: *Mathematics of Computation* 46.173 (1986), pp. 1–26. DOI: `10.2307/2008211`.

[JCT17] K. Jung, Y.-K. Cho, and Y.-J. Tak. “Performance evaluation of ROMS v3.6 on a commercial cloud system”. In: *Geoscientific Model Development Discussions* 2017 (2017), pp. 1–43. DOI: `10.5194/gmd-2017-270`.

[KC00] L. Kantha and C. Clayson. *Numerical Models of Oceans and Oceanic Processes.* Vol. 66. International Geophysics. 2000.

[Kär+18] T. Kärnä, S. C. Kramer, L. Mitchell, D. A. Ham, M. D. Piggott, et al. “Thetis coastal ocean model: discontinuous Galerkin discretization for the three-dimensional hydrostatic equations”. In: *Geoscientific Model Development* 11.11 (2018), pp. 4359–4382. DOI: `10.5194/gmd-11-4359-2018`.

[KJ05] D. J. Kerbyson and P. W. Jones. “A Performance Model of the Parallel Ocean Program”. In: *The International Journal of High Performance Computing Applications* 19.3 (2005), pp. 261–276. DOI: `10.1177/1094342005056114`.

[Kin93] D. A. King. *Multiblock Grid Generation Results of the EC/BRITE-EURAM Project EUROMESH.* Vol. 44. Notes on Numerical Fluid Mechanics. 1993.

[Kli+18] K. Klingbeil, F. Lemarié, L. Debreu, and H. Burchard. “The numerics of hydrostatic structured-grid coastal ocean models: State of the art and future perspectives”. In: *Ocean Modelling* 125 (2018), pp. 80–105. DOI: 10.1016/j.ocemod.2018.01.007.

[Kol+19] N. V. Koldunov, V. Aizinger, N. Rakowsky, P. Scholz, D. Sidorenko, et al. “Scalability and some optimization of the Finite-volumE Sea ice–Ocean Model, Version 2.0 (FESOM2)”. In: *Geoscientific Model Development* 12.9 (2019), pp. 3991–4012. DOI: 10.5194/gmd-12-3991-2019.

[Kor+22] P. Korn, N. Brüggemann, J. H. Jungclaus, S. J. Lorenz, O. Gutjahr, et al. “ICON-O: The Ocean Component of the ICON Earth System Model—Global Simulation Characteristics and Local Telescoping Capability”. In: *Journal of Advances in Modeling Earth Systems* 14.10 (2022), e2021MS002952. DOI: 10.1029/2021MS002952.

[Kos+10] T. Kosar, N. Oliveira, M. Mernik, M. João, M. Pereira, et al. “Comparing General-Purpose and Domain-Specific Languages: An Empirical Study”. In: *Computer Science and Information Systems* 438 (2010). DOI: 10.2298/CSIS1002247K.

[KF03] L. Krivodonova and J. E. Flaherty. “Error Estimation for Discontinuous Galerkin Solutions of Two-Dimensional Hyperbolic Problems”. In: *Adv. Comput. Math.* 19 (2003), pp. 57–71. DOI: 10.1023/A:1022894504834.

[Kri+04] L. Krivodonova, J. Xin, J.-F. Remacle, N. Chevaugeon, and J. E. Flaherty. “Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws”. In: *Applied Numerical Mathematics* 48.3 (2004). Workshop on Innovative Time Integrators for PDEs, pp. 323–338. DOI: 10.1016/j.apnum.2003.11.002.

[Kro20] S. Kronawitter. “Automatic Performance Optimization of Stencil Codes”. PhD thesis. Universität Passau, 2020, xiii, 130 Seiten.

[Kub+09] E. J. Kubatko, S. Bunya, C. Dawson, and J. J. Westerink. “Dynamic p-adaptive Runge–Kutta discontinuous Galerkin methods for the shallow water equations”. In: *Computer Methods in Applied Mechanics and Engineering* 198.21 (2009). Advances in Simulation-Based Engineering Sciences – Honoring J. T. Oden, pp. 1766–1774. DOI: 10.1016/j.cma.2009.01.007.

[KWD06] E. J. Kubatko, J. J. Westerink, and C. Dawson. “hp Discontinuous Galerkin methods for advection dominated problems in shallow water flow”. In: *Computer Methods in Applied Mechanics and Engineering* 196.1 (2006), pp. 437–451. DOI: 10.1016/j.cma.2006.05.002.

[Kuc19] S. Kuckuk. “Automatic Code Generation for Massively Parallel Applications in Computational Fluid Dynamics”. PhD thesis. Friedrich–Alexander–Universität Erlangen–Nürnberg, 2019. DOI: 10.25593/978-3-96147-274-1.

[KK16] S. Kuckuk and H. Köstler. “Automatic Generation of Massively Parallel Codes from ExaSlang”. In: *Computation* 4.3 (2016). Special Issue on High Performance Computing (HPC) Software Design, 27:1–27:20. DOI: 10.3390/computation4030027.

[KK18a] S. Kuckuk and H. Köstler. “Whole Program Generation of Massively Parallel Shallow Water Equation Solvers”. In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 2018, pp. 78–87. DOI: 10.1109/CLUSTER.2018.00020.

[KK18b] S. Kuckuk and H. Köstler. “Whole Program Generation of Massively Parallel Shallow Water Equation Solvers”. In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 2018, pp. 78–87. DOI: 10.1109/CLUSTER.2018.00020.

[Kuz10] D. Kuzmin. “A vertex-based hierarchical slope limiter for p-adaptive discontinuous Galerkin methods”. In: *Journal of Computational and Applied Mathematics* 233.12 (2010). Finite Element Methods in Engineering and Science (FEMTEC 2009), pp. 3077–3085. DOI: 10.1016/j.cam.2009.05.028.

[Kuz13] D. Kuzmin. “Slope limiting for discontinuous Galerkin approximations with a possibly non-orthogonal Taylor basis”. In: *International Journal for Numerical Methods in Fluids* 71.9 (2013), pp. 1178–1190. DOI: 10.1002/fld.3707.

[Kuz14] D. Kuzmin. “Hierarchical slope limiting in explicit and implicit discontinuous Galerkin methods”. In: *Journal of Computational Physics* 257 (2014), pp. 1140–1162. DOI: 10.1016/j.jcp.2013.04.032.

[KH23] D. Kuzmin and H. Hajduk. *Property-Preserving Numerical Schemes for Conservation Laws.* WORLD SCIENTIFIC, 2023. DOI: 10.1142/13466.

[Kuz+22] D. Kuzmin, M. Quezada de Luna, D. Ketcheson, and J. Grüll. “Bound-preserving Flux Limiting for High-Order Explicit Runge–Kutta Time Discretizations of Hyperbolic Conservation Laws”. In: *Journal of Scientific Computing* 91 (2022). DOI: 10.1007/s10915-022-01784-0.

[KS13] D. Kuzmin and F. Schieweck. “A parameter-free smoothness indicator for high-resolution finite element schemes”. In: *Open Mathematics* 11.8 (2013), pp. 1478–1488. DOI: 10.2478/s11533-013-0254-4.

[Lac+14] A. Lacasta, M. Morales-Hernández, J. Murillo, and P. García-Navarro. “An optimized GPU implementation of a 2D free surface simulation model on unstructured meshes”. In: *Advances in Engineering Software* 78 (2014), pp. 1–15. DOI: 10.1016/j.advengsoft.2014.08.007.

[Len+20] C. Lengauer, S. Apel, M. Bolten, S. Chiba, U. Rüde, et al. “ExaStencils: Advanced Multigrid Solver Generation”. In: *Software for Exascale Computing - SPPEXA 2016-2019.* Ed. by H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel. Cham: Springer International Publishing, 2020, pp. 405–452.

[LeV92] R. J. LeVeque. *Numerical Methods for Conservation Laws.* Lectures in Mathematics ETH Zürich, Department of Mathematics Research Institute of Mathematics. Springer Basel AG, 1992.

[LeV02] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253.

[LZ21] L. Li and Q. Zhang. “Development of an efficient wetting and drying treatment for shallow-water modeling using the quadrature-free Runge-Kutta discontinuous Galerkin method”. In: *International Journal for Numerical Methods in Fluids* 93.2 (2021), pp. 314–338. DOI: 10.1002/fld.4884.

[LA99] D. Lockard and H. Atkins. “Efficient implementations of the quadrature-free discontinuous Galerkin method”. In: *Proceeding of 14th AIAA CFD conference. AIAA*. 1999, pp. 526–536. DOI: 10.2514/6.1999-3309.

[Mad+91] G. Madec, P. Delecluse, M. Crepon, and M. Chartier. “A three-dimensional numerical study of deep-water formation in the northwestern Mediterranean Sea”. In: *Journal of Physical Oceanography* 21.9 (1991), pp. 1349–1371. DOI: 10.1175/1520-0485(1991)021<1349:ATDNSO>2.0.CO;2.

[MRC06] E. Marchandise, J.-F. Remacle, and N. Chevaugeon. “A quadrature-free discontinuous Galerkin method for the level set equation”. In: *Journal of Computational Physics* 212.1 (2006), pp. 338–357. DOI: 10.1016/j.jcp.2005.07.006.

[Mar+97] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey. “A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers”. In: *Journal of Geophysical Research: Oceans* 102.C3 (1997), pp. 5753–5766. DOI: 10.1029/96JC02775.

[Mar97] M. L. Martinéz. “A Priori Error Estimates of Finite Element Models of Systems of Shallow Water Equations”. PhD thesis. Rice University, 1997.

[Meu+17] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, et al. “SymPy: symbolic computing in Python”. In: *PeerJ Computer Science* 3 (2017), e103. DOI: 10.7717/peerj-cs.103.

[Mic+11] C. Michoski, C. Mirabito, C. Dawson, D. Wirasaet, E. J. Kubatko, et al. “Adaptive hierarchic transformations for dynamically p-enriched slope-limiting over discontinuous Galerkin systems of generalized equations”. In: *Journal of Computational Physics* 230.22 (2011), pp. 8028–8056. DOI: 10.1016/j.jcp.2011.07.009.

[MDA15] C. Mirabito, C. Dawson, and V. Aizinger. “An a priori error estimate for the local discontinuous Galerkin method applied to two-dimensional shallow water and morphodynamic flow”. In: *Numerical Methods for Partial Differential Equations* 31.2 (2015), pp. 397–421. DOI: 10.1002/num.21914.

[Mit01] S. M. Mitran. “A Comparison of Adaptive Mesh Refinement Approaches for Large Eddy Simulation”. In: *Technical Report.* 2001, p. 12.

[Moo98] G. E. Moore. “Cramming More Components Onto Integrated Circuits”. In: *Proceedings of the IEEE* 86.1 (1998), pp. 82–85. DOI: 10.1109/JPROC.1998.658762.

[Nad+19] F. Naddei, M. de la Llave Plata, V. Couaillier, and F. Coquel. “A comparison of refinement indicators for p-adaptive simulations of steady and unsteady flows using discontinuous Galerkin methods”. In: *Journal of Computational Physics* 376 (2019), pp. 508–533. DOI: 10.1016/j.jcp.2018.09.045.

[Nai15] R. Nair. “Quadrature-Free Implementation of a Discontinuous Galerkin Global Shallow-Water Model via Flux Correction Procedure”. In: *Monthly Weather Review* 143.4 (2015), pp. 1335–1346. DOI: 10.1175/MWR-D-14-00174.1.

[Neč12] J. Nečas. *Direct Methods in the Theory of Elliptic Equations.* Springer, 2012. DOI: 10.1007/978-3-642-10455-8.

[NH18] S. P. Neill and M. R. Hashemi. *Fundamentals of ocean renewable energy: generating electricity from the sea.* Academic Press, 2018.

[OBB23] G. Orlando, T. Benacchio, and L. Bonaventura. “An IMEX-DG solver for atmospheric dynamics simulations with adaptive mesh refinement”. In: *Journal of Computational and Applied Mathematics* 427 (2023), p. 115124. DOI: 10.1016/j.cam.2023.115124.

[OLK22] S. Ortleb, J. Lambrechts, and T. Kärnä. “Wetting and Drying Procedures for Shallow Water Simulations”. In: *The Mathematics of Marine Modelling: Water, Solute and Particle Dynamics in Estuaries and Shallow Seas.* Ed. by H. Schuttelaars, A. Heemink, and E. Deleersnijder. Cham: Springer International Publishing, 2022, pp. 287–314. DOI: 10.1007/978-3-031-09559-7_11.

[Pro02] J. K. Proft. “Multi-algorithmic numerical strategies for the solution of shallow water models”. PhD thesis. The University of Texas at Austin, 2002.

[RDB18] L. Rannabauer, M. Dumbser, and M. Bader. “ADER-DG with a-posteriori Finite-Volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework”. In: *Computers & Fluids* 173 (2018), pp. 299–306. DOI: 10.1016/j.compfluid.2018.01.031.

[RM03] P. Rao and P. J. Morris. “A generalized quadrature free discontinuous Galerkin method”. In: *Computational Fluid and Solid Mechanics 2003*. Elsevier Inc., 2003, pp. 2105–2109. DOI: 10.1016/B978-008044046-0.50517-0.

[Rat+16] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, et al. “Firedrake: Automating the Finite Element Method by Composing Abstractions”. In: *ACM Transactions on Mathematical Software* 43.3 (2016), 24:1–24:27. DOI: 10.1145/2998441.

[RH73] W. H. Reed and T. R. Hill. “Triangular mesh methods for the neutron transport equation”. In: *Los Alamos Scientific Laboratory* (1973).

[RVH23] L. Reimann, A. T. Vafeidis, and L. E. Honsel. “Population development as a driver of coastal risk: Current trends and future pathways”. In: *Cambridge Prisms: Coastal Futures* 1 (2023), e14. DOI: 10.1017/cft.2023.3.

[RFS03] J.-F. Remacle, J. E. Flaherty, and M. S. Shephard. “An Adaptive Discontinuous Galerkin Technique with an Orthogonal Basis Applied to Compressible Flow Problems”. In: *SIAM Review* 45.1 (2003), pp. 53–72. DOI: 10.1137/S00361445023830.

[Rem+12] J.-F. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen, et al. “Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm”. In: *International Journal for Numerical Methods in Engineering* 89.9 (2012), pp. 1102–1119. DOI: 10.1002/nme.3279.

[Reu20] B. Reuter. “The discontinuous Galerkin method for free surface and subsurface flows in geophysical applications”. PhD thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2020, p. 262.

[Reu+16] B. Reuter, V. Aizinger, M. Wieland, F. Frank, and P. Knabner. “FESTUNG: A MATLAB/GNU Octave toolbox for the discontinuous Galerkin method, Part II: Advection operator and slope limiting”. In: *Computers and Mathematics with Applications* 72.7 (2016), pp. 1896–1925. DOI: 10.1016/j.camwa.2016.08.006.

[Reu+19] B. Reuter, A. Rupp, V. Aizinger, and P. Knabner. “Discontinuous Galerkin method for coupling hydrostatic free surface flows to saturated subsurface systems”. In: *Computers & Mathematics with Applications* 77.9 (2019), pp. 2291–2309. DOI: 10.1016/j.camwa.2018.12.020.

[Rin+13] T. Ringler, M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, et al. “A multi-resolution approach to global ocean modeling”. In: *Ocean Modelling* 69 (2013), pp. 211–232. DOI: 10.1016/j.ocemod.2013.04.010.

[Riv08] B. Rivière. *Discontinuous Galerkin methods for solving elliptic and parabolic equations: theory and implementation*. SIAM, 2008.

[RD19] R. Robertson and C. Dong. “An evaluation of the performance of vertical mixing parameterizations for tidal mixing in the Regional Ocean Modeling System (ROMS)”. In: *Geoscience Letters* 6 (2019). DOI: 10.1186/s40562-019-0146-y.

[Roe81] P. L. Roe. “Approximate Riemann solvers, parameter vectors, and difference schemes”. In: *Journal of Computational Physics* 43.2 (1981), pp. 357–372. DOI: 10.1016/0021-9991(81)90128-5.

[RP85] P. L. Roe and J. Pike. “Efficient Construction and Utilisation of Approximate Riemann Solutions”. In: *Proc. of the sixth int’l. symposium on Computing methods in applied sciences and engineering, VI.* 1985, pp. 499–518.

[Rup19] A. Rupp. “Simulating Structure Formation in Soils across Scales using Discontinuous Galerkin Methods”. PhD thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2019, p. 187.

[Sch+18] C. Schmitt, M. Schmid, S. Kuckuk, H. Köstler, J. Teich, et al. “Reconfigurable Hardware Generation of Multigrid Solvers with Conjugate Gradient Coarse-Grid Solution”. In: *Parallel Processing Letters* 28.04 (2018), p. 1850016. DOI: `10.1142/S0129626418500160`.

[Sen+12] S. I. Seneviratne, N. Nicholls, D. Easterling, C. M. Goodess, S. Kanae, et al. “Changes in Climate Extremes and their Impacts on the Natural Physical Environment”. In: *Managing the Risks of Extreme Events and Disasters to Advance Climate Change Adaptation: Special Report of the Intergovernmental Panel on Climate Change.* Ed. by C. B. Field, V. Barros, T. F. Stocker, and Q. Dahe. Cambridge University Press, 2012, 109–230. DOI: `10.1017/CBO9781139177245.006`.

[Sha20] J. Shalf. “The future of computing beyond Moore’s Law”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378 (2020), p. 20190061. DOI: `10.1098/rsta.2019.0061`.

[SO88] C.-W. Shu and S. Osher. “Efficient implementation of essentially non-oscillatory shock-capturing schemes”. In: *Journal of Computational Physics* 77.2 (1988), pp. 439–471. DOI: `10.1016/0021-9991(88)90177-5`.

[SH94] Y. Song and D. Haidvogel. “A Semi-implicit Ocean Circulation Model Using a Generalized Topography-Following Coordinate System”. In: *Journal of Computational Physics* 115.1 (1994), pp. 228–244. DOI: `10.1006/jcph.1994.1189`.

[SB96] S. P. Spekreijse and J. W. Boerstoel. “Multiblock grid generation. Part II: Multiblock aspects”. In: *Computational Fluid Dynamics* (1996).

[Sze+24] K. Szenes, N. Discacciati, L. Bonaventura, and W. Sawyer. “Domain-specific implementation of high-order Discontinuous Galerkin methods in spherical geometry”. In: *Computer Physics Communications* 295 (2024), p. 108993. DOI: `10.1016/j.cpc.2023.108993`.

[TDF06] J. D. Teresco, K. D. Devine, and J. E. Flaherty. “Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations”. In: *Numerical Solution of Partial Differential Equations on Parallel Computers.* Ed. by A. M. Bruaset and A. Tveito. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 55–88.

[Tor09] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction.* Springer, 2009. DOI: `10.1007/b79761`.

[THD09] E. F. Toro, A. Hidalgo, and M. Dumbser. “FORCE Schemes on Unstructured Meshes I: Conservative Hyperbolic Systems”. In: *J. Comput. Phys.* 228.9 (2009), 3368–3389. DOI: `10.1016/j.jcp.2009.01.025`.

[TSS94] E. F. Toro, M. Spruce, and W. Speares. “Restoration of the contact surface in the HLL-Riemann solver”. In: *Shock Waves* 4 (1994), pp. 25–34. DOI: `10.1007/BF01414629`.

[Tor+13] R. Torres, J. Kunkel, L. Linardakis, and T. Ludwig. “ICON DSL: A Domain-Specific Language for climate modeling”. In: *International Conference for High Performance Computing, Networking,Storage and Analysis, Denver.* 2013.

[TBR13] G. Tumolo, L. Bonaventura, and M. Restelli. “A semi-implicit, semi-Lagrangian, p-adaptive discontinuous Galerkin method for the shallow water equations”. In: *Journal of Computational Physics* 232.1 (2013), pp. 46–67. DOI: 10.1016/j.jcp.2012.06.006.

[Vis18] M. Visbeck. “Ocean science research is key for a sustainable future”. In: *Nature Communications* 9 (2018). DOI: 10.1038/s41467-018-03158-3.

[Vre94] C. B. Vreugdenhil. *Numerical Methods for Shallow-Water Flow.* Springer Science & Business Media, 1994.

[Wan+72] H.-H. Wang, P. Halpen, J. Douglas, and T. Dupont. “Numerical Solutions of the One-Dimensional Primitive Equations Using Galerkin Approximations With localized Basis Functions”. In: *Monthly Weather Review* 100.10 (1972), pp. 738 –746. DOI: 10.1175/1520-0493(1972)100<0738:NSOTOP>2.3.CO;2.

[WKC97] P. Wang, D. S. Katz, and Y. Chao. “Optimization of a Parallel Ocean General Circulation Model”. In: *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing.* SC ’97. New York, NY, USA: Association for Computing Machinery, 1997, 1–11. DOI: 10.1145/509593.509618.

[Wes+92] J. J. Westerink, R. A. Luettich, C. A. Blain, and N. W. Scheffner. “ADCIRC: An Advanced Three-Dimensional Circulation Model for Shelves, Coasts, and Estuaries. Report 2. User’s Manual for ADCIRC-2DDI”. In: *Contractors Report to the US Army Corps of Engineers* (1992).

[WDL08] L. White, E. Deleersnijder, and V. Legat. “A three-dimensional unstructured mesh finite element shallow-water model, with application to the flows around an island and in a wind-driven, elongated basin”. In: *Ocean Modelling* 22.1-2 (2008), pp. 26–47. DOI: 10.1016/j.ocemod.2008.01.001.

[XS10] Y. Xu and C.-W. Shu. “Local Discontinuous Galerkin Methods for High-Order Time-Dependent Partial Differential Equations”. In: *Communications in Computational Physics* 7.1 (2010), pp. 1–46. DOI: 10.4208/cicp.2009.09.023.

[ZS11] X. Zhang and C.-W. Shu. “Maximum-principle-satisfying and positivity-preserving high-order schemes for conservation laws: Survey and new developments”. In: *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences* 467 (2011). DOI: 10.1098/rspa.2011.0153.

[Zha+16] Y. J. Zhang, F. Ye, E. V. Stanev, and S. Grashorn. “Seamless cross-scale modeling with SCHISM”. In: *Ocean Modelling* 102 (2016), pp. 64–81. DOI: 10.1016/j.ocemod.2016.05.002.

[ZO95] O. C. Zienkiewicz and P. Ortiz. “A split-characteristic based finite element model for the shallow water equations”. In: *International Journal for Numerical Methods in Fluids* 20.8-9 (1995), pp. 1061–1080. DOI: 10.1002/fld.1650200823.

[ZG21] D. Zint and R. Grosso. “A Hybrid Approach to Fast Indirect Quadrilateral Mesh Generation”. In: *Numerical Geometry, Grid Generation and Scientific Computing.* Cham: Springer International Publishing, 2021, pp. 281–294. DOI: 10.1007/978-3-030-76798-3_18.

[Zin+19] D. Zint, R. Grosso, V. Aizinger, and H. Köstler. “Generation of Block Structured Grids on Complex Domains for High Performance Simulation”. In: *Numerical Geometry, Grid Generation and Scientific Computing.* Ed. by Vladimir A. Garanzha, Lennard Kamenski, and Hang Si. Cham: Springer International Publishing, 2019, pp. 87–99. DOI: 10.1007/978-3-030-23436-2_6.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass ich die Hilfe von gewerblichen Promotionsberatern bzw. -vermittlern oder ähnlichen Dienstleistern weder bisher in Anspruch genommen habe, noch künftig in Anspruch nehmen werde.

Zusätzlich erkläre ich hiermit, dass ich keinerlei frühere Promotionsversuche unternommen habe.

Bayreuth, November 28, 2023

Sara Faghih-Naini