Technische Universität München
Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Computation in Engineering

Efficient Parallel Simulations of Flood Propagation Including Wet-Dry Problems

Bobby Minola Ginting

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:

Prof. Dr.-Ing. Michael Manhart

Prüfer der Dissertation:

1. Prof. Dr. rer. nat. Ernst Rank
2. Prof. Dr.-Ing. Reinhard Hinkelmann
3. Priv.-Doz. Dr. rer. nat. habil. Ralf-Peter Mundani

Die Dissertation wurde am 30.07.2019 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 07.10.2019 angenommen.

Acknowledgment

This work would not have been possible without the financial support of the German Academic Exchange Service (DAAD) in the scope of Research Grants – Doctoral Programmes in Germany 2015/16 (57129429).

I am especially indebted to Prof. Rank who has provided me the opportunity to pursue my doctoral study at his chair. As my supervisor, he guided me to become an independent researcher in my field. Without his help, patience, and encouragement, this work would never be finished. I would like also to express my gratitude to Dr. Mundani who has been supportive of my study. As my teacher and mentor, he always provided me a professional guidance and helped me understand the field of informatics.

I am very grateful to Prof. Hinkelmann (Technical University of Berlin) for his comprehensive review for my work. I met him first in June 2017 at a hydroinformatics conference in Nice (France). His works have shown me the importance of informatics in hydrosystems modeling. I would like to thank Prof. Manhart as the chairman of my doctoral defense. Also, fruitful discussions with him in turbulence modeling are highly appreciated.

My special thanks go to Nevena Perović, Christoph Ertl, Dr. Özgen-Xian (Lawrence Berkeley National Laboratory), and Franziska Tügel (Technical University of Berlin) for their helps in my study. I would like also to thank my colleagues from Chair for Computation in Engineering as well as Chair of Computational Modeling and Simulation who have helped me greatly, in particular, Dr. Kollmannsberger, Davide D'Angella, Mohamed Elhaddad, John Jomo, Nina Korshunova, Philipp Kopp, László Kudela, Alexander Paolini, Alexander Braun, Felix Eickeler, Katrin Jahr, and Jimmy Abualdenien. I especially thank Hanne Cornils for her support in all administrative processes during my study. I acknowledge the contribution of Mengjie Zhao and Uğurcan Sarı for the great teamwork.

Most importantly, I would like to thank Kim's family, Park's family, Qifan Yang, and Corina Schaller as my family in Germany. Finally, I wish to thank Herli Ginting (my father), Agustina Sembiring (my mother), Agi Hagata Ginting (my brother), and Vanny Gracya Ginting (my sister) for all of their supports.

Contents

List of Figures

List of Tables

Abstract

Over the past decades, shallow water models have extensively been used to simulate flood propagations in many applications ranging from small to large domains such as drainage flows, urban areas, river flows, the rainfall-runoff process in a catchment area, dam-break, tsunamis, etc. The main focus of the development of these models was on accuracy so that the results can be used comprehensively to understand the complex flow properties, e.g. depth and velocity propagation, of the cases studied. Although these shallow water models were based on quite simple schemes/formulas and lots of simplifications were made, the simulation processes still took days or weeks due to the limitation of computing resources—even for a relatively small domain.

Nowadays, one can speak of a new era of shallow water models. This is indicated by rapid technological developments in the field of computing hardware, such as clusters and supercomputers, all the way to petascale supercomputing systems, enabling researchers to carry out numerical simulations even for very large domains in quite a short time. In tsunami modeling, for example, such modern computing resources allow one to significantly reduce the computational time from weeks to hours by utilizing parallel computations. Considering today's situation and even conditions in the future, it is obvious that in addition to accuracy, computational efficiency has to be addressed within the development of numerical models.

The aim of this work is to develop a shallow water model that utilizes advanced strategies specifically designed for a parallel computing framework on modern multi- and many-core architectures. The framework allows one to efficiently exploit the potential of the modern computing hardware in order to significantly reduce the computational time, which is not only beneficial for researchers and engineers when solving complex flow problems, but also for decision makers aiming to develop a well-integrated early warning system that is able to provide results more or less in real time. The numerical model was developed with modern finite volume schemes that employ both Riemann and non-Riemann solvers, which have been proven to be quite robust in simulating the most complex phenomena in the scope of shallow flow modeling, such as transcritical flows, shock waves, and moving wet-dry fronts. Specifically, a cell-centered finite volume method was employed in which the computations were performed in an edge-based manner instead of a cell-based one.

This research was started by investigating the efficiency of the employed solvers with regard to their ability in utilizing the vectorization support of modern hardware. Uniform rectangular grids were used to ease the implementation, and a novel cell-edge reordering strategy was thus proposed to enable vectorization as supported by modern chipsets. Furthermore, load imbalances due to wet-dry phenomena — which exist in most real-world applications of flood propagation and typically hinder efficient parallelization — were taken into account. Under the circumstances — despite the fact that methods have become more advanced in the course

of time — efficient parallel codes have not been widely investigated yet. Essentially, wet cells lead to more computational effort than the dry ones; due to the unpredictable emergence or disappearance of these cells during runtime, load imbalances concerning an initial mapping of cells to computational cores/nodes is to be expected, thus a suitable load balancing technique is inevitable. To this end, a novel strategy *weighted-dynamic load balancing* was proposed. Further, within the same parallelization framework, a novel approach — namely a combination of the hydrostatic and topography reconstruction with scalable wall functions — was also developed to include a turbulence model for wet-dry simulations, in particular, where cases of flow-past-structure are considered. This manner gives flexibility to users when generating meshes in order to increase the accuracy without deteriorating the computational results due to stability limitations of the standard wall functions.

As all the aforementioned steps were conducted with structured meshes, applications using unstructured meshes — due to the flexibility of such meshes in dealing with the shape of complex domains that may not be achieved by (high-resolution) structured meshes — are taken into consideration in this work. Also, this step is intended to allow for a wider range of implementations. Therefore, another data structure type has to be considered in the code, where cells and edges were indexed according to an ordering given by space-filling curves. To this regard, two options are available; the first one is to reorder the cells with space-filling curves, followed by the consecutive determination of the corresponding edges according to the renumbered cells, or vice versa, as the second option.

Five main findings are pointed out in this work. Firstly, the cell-edge reordering strategy enables with respect to the node-level performance an efficient vectorization of the computation kernel of the Riemann and non-Riemann solvers, thus achieving a better performance on the aforementioned architectures. Secondly, the results indicate that the proposed load balancing strategy is suitable to efficiently tackle load imbalances due to the wet-dry problems, thus allowing for better parallel efficiency. Thirdly, it is shown that high-resolution meshes can be used to increase the accuracy in capturing turbulent flow properties when dealing with wet-dry phenomena, without having to consider the classical formulation of the standard wall functions. Fourthly, one can see that the proposed model serves to accurately simulate recirculating turbulent flows for flow-past-structure cases. Finally, the second data structure — that is devised for unstructured meshes in conjunction with space filling curves reordering — is shown to be highly flexible with regard to parallelization, and it turns out to be promisingly efficient. All the strategies proposed are conceptually simple and can be extended into a framework for massive parallel computations on supercomputers in the future.

Zusammenfassung

Während der letzten Jahrzehnte wurden Flachwassermodelle für vielseitige Problemstellungen auf verschiedenen räumlichen Skalen verwendet, z.B. zur Untersuchung von Entwässerungssystemen, Niederschlagsabfluss in urbanen und natürlichen Einzugsgebieten, Dammbruchereignissen, Tsunamis, etc. Der Schwerpunkt der Entwicklung dieser Modelle lag auf Genauigkeit, damit die Ergebnisse umfassend verwendet werden konnten, um die komplexen Fließeigenschaften der untersuchten Fälle, z.B. durch Angabe der Wassertiefen und Geschwindigkeiten, verstehen zu können. Obwohl diese Flachwassermodelle anhand einfacher Schemata/Formeln entwickelt und viele Vereinfachungen vorgenommen wurden, dauerten die Simulationsprozesse aufgrund begrenzter Rechenkapazitäten auch für relativ kleine Modellgebiete Tage oder Wochen.

Heutzutage kann man von einer neuen Ära der Flachwassermodelle sprechen, bedingt durch die schnelle technologische Entwicklung von Clustersytemen und Supercomputern bis hin zu Petascale-Architekturen, die es ermöglichen, numerische Simulationen in kurzer Zeit auch für sehr große Rechengebiete durchzuführen. Für die Tsunamimodellierung sind beispielweise solche modernen Maschinen in der Lage, die Rechenzeit durch Parallelisierung von Wochen auf Stunden zu verkürzen. Angesichts der heutigen und zukünftigen Situation ist es offensichtlich, dass bei der Entwicklung numerischer Modelle neben der Genauigkeit auch die Recheneffizienz berücksichtigt werden muss.

Im Rahmen dieser Arbeit wurde ein Flachwassermodell mit fortschrittlichen Strategien entwickelt, die gezielt für moderne Mehr- und Multikernarchitekturen entworfen wurden. Dieses Framework ermöglicht, die Rechenzeit erheblich zu verkürzen, was nicht nur für Forscher und Ingenieure von erheblichem Vorteil ist, um komplexe Strömungsprobleme zu lösen, sondern auch für Entscheidungsträger, um ein gut integriertes Frühwarnsystem zu entwickeln, welches in der Lage ist, Ergebnisse mehr oder weniger in Echtzeit zu liefern. Das numerische Modell wurde mit modernen Finite-Volume-Schemata entwickelt, die sowohl Riemann- als auch Nicht-Riemann-Löser verwenden und die sich im Rahmen der Flachwassermodellierung für die komplexesten Phänomene, wie transkritische Strömungen, Stoßwellen und Nass-Trocken-Fronten, als sehr robust erwiesen haben. Im Speziellen wurde eine zellzentrierte Finite-Volume-Methode verwendet, bei der die Berechnungen kanten- statt zellbasiert durchgeführt wurden.

Diese Arbeit beginnt mit der Untersuchung der Effizienz der verwendeten Löser in Bezug auf ihre Fähigkeit, die Vektorisierung moderner Hardware auszunutzen. Um die Implementierung zu vereinfachen, wurden uniforme, rechteckige Zellen verwendet. Weiterhin wurde eine neue Strategie zur Zell-Kanten-Umordnung angewendet, um die Vektorisierung, die von modernen Chipsätzen unterstützt wird, zu ermöglichen. Es wurde spezielles Augenmerk auf die entstehenden Lastungleichgewichte aufgrund der Nass-Trocken-Phänomene gelegt, die

in den meisten realen Anwendungen der Überflutungssimulation existieren und die typischerweise eine effiziente Parallelisierung verhindern. Trotz der Tatsache, dass sich die Methoden im Laufe der Zeit weiterentwickelt haben, wurden effiziente, parallele Codes noch nicht umfassend unter diesem Gesichtspunkt untersucht. Grundsätzlich führen nasse Zellen zu mehr Rechenaufwand als trockene Zellen. Aufgrund der unvorhersehbaren Entstehung oder des Verschwindens dieser Zellen während der Laufzeit, sind Lastungleichgewichte hinsichtlich einer anfänglichen Zuordnung von Zellen zu Rechenkernen/-knoten zu erwarten. Folglich ist eine geeignete Strategie zur Lastbalanzierung unabdingbar. Hierzu wurde eine neue Strategie entwickelt: Die gewichtete-dynamische Lastbalanzierung. Weiterhin wurde ein neuartiger Ansatz entwickelt, bei dem eine Kombination von hydrostatischer und topographischer Rekonstruktion mit skalierbaren Wandfunktionen implementiert wurde. Dies erlaubt die Integration eines Turbulenzmodells in die Nass-Trocken-Simulationen, insbesondere wenn Fälle mit Strukturumströmung berücksichtigt werden. Auf diese Weise wird eine hohe Flexibilität bei der Netzgenerierung erreicht, um die Genauigkeit zu erhöhen, ohne die Ergebnisse aufgrund der Stabilitätsbeschränkung der Standardwandfunktionen zu verschlechtern.

Alle oben genannten Schritte wurden anhand strukturierter Netze durchgeführt. Aufgrund ihrer Flexibilität für komplexe Rechengebiete, für die hochaufgelöste, strukturierte Gitter nicht möglich wären, werden in dieser Arbeit auch Anwendungen mit unstrukturierten Gittern berücksichtigt. Damit wird ein breiteres Spektrum von Implementierungen möglich. Dafür muss im Code eine andere Datenstruktur berücksichtigt werden, wobei die Zellen und Kanten mit Hilfe von raumfüllenden Kurven indiziert wurden. Dafür stehen zwei Optionen zur Verfügung. Entweder werden die Zellen anhand einer raumfüllenden Kurve umgeordnet bevor die entsprechenden Kanten nacheinander nach den neunummerierten Zellen geordnet werden oder umgekehrt.

In dieser Arbeit werden fünf Hauptergebnisse aufgezeigt. Erstens ermöglicht die Zell-Kanten-Umordnung im Hinblick auf die Leistung auf Knotenebene eine effiziente Vektorisierung des Rechenkernels, d.h. der Riemann- und Nicht-Riemann-Löser und somit eine bessere Leistung für die oben genannten Architekturen. Zweitens ist die vorgeschlagene Lastbalanzierung für eine effiziente Lösung der Lastungleichgewichte bei Nass-Trocken-Problemen geeignet und ermöglicht somit eine bessere, parallele Effizienz. Drittens wird gezeigt, dass hochauflösende Netze verwendet werden können, um die Genauigkeit der Eigenschaften von turbulenten Strömungen bei Simulationen mit Nass-Trocken-Phänomenen zu erhöhen, ohne die klassische Formulierung der Standardwandfunktionen berücksichtigen zu müssen. Viertens ist das Modell in der Lage, turbulente Umwälzströmungen bei Fällen mit Strukturumströmung akkurat zu simulieren. Schließlich erweist sich die zweite Datenstruktur, die für unstrukturierte Netze entwickelt wurde, in Verbindung mit einer Umordnung durch raumfüllende Kurven als sehr geeignet für die Parallelisierung und stellt sich als effizient heraus. Alle vorgeschlagenen Strategien sind konzeptuell einfach und können zur Nutzung in Frameworks für massiv parallele Berechnungen auf Supercomputern weiterentwickelt werden.

Publications

This dissertation was written based on the publications in peer-reviewed scientific journals and chapter books, for which Bobby Minola Ginting conducted the majority of the works independently. The following papers are included in the appendix of this thesis.

Journals

Paper 1

B.M. Ginting and R.-P. Mundani. Parallel Flood Simulations for Wet-Dry Problems Using Dynamic Load Balancing Concept. *Journal of Computing in Civil Engineering (ASCE)*, 33(3): 04019013, 2019. https://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000823.

Paper 2

B.M. Ginting. Central-Upwind Scheme for 2D Turbulent Shallow Flows using High-Resolution Meshes with Scalable Wall Functions. *Computers & Fluids*, 179:394–421, 2019. https://dx.doi.org/10.1016/j.compfluid.2018.11.014.

Paper 3

B.M. Ginting and R.-P. Mundani. Comparison of Shallow Water Solvers: Applications for Dam-Break and Tsunami Cases with Reordering Strategy for Efficient Vectorization on Modern Hardware. *Water*, 11(4):639, 2019. https://dx.doi.org/10.3390/w11040639.

Paper 4

B.M. Ginting and H. Ginting. Hybrid Artificial Viscosity–Central-Upwind Scheme for Recirculating Turbulent Shallow Water Flows. *Journal of Hydraulic Engineering (ASCE)*, 145(12): 04019041, 2019. https://dx.doi.org/10.1061/(ASCE)HY.1943-7900.0001639.

Chapter books

Paper 5

B.M. Ginting and R.-P. Mundani. Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography. In P. Gourbesville, J. Cunge, and G. Caignaert, editors, *Advances in Hydroinformatics*, chapter 4, pages 51–74. Springer Water, Springer, Singapore, 2018. https://dx.doi.org/10.1007/978-981-10-7218-5_4.

Paper 6

B.M. Ginting, P.K. Bhola, C. Ertl, R.-P. Mundani, M. Disse, and E. Rank. Hybrid-Parallel Simulations and Visualisations of Real Flood and Tsunami Events using Unstructured Meshes on High-Performance Cluster Systems. In *Advances in Hydroinformatics*, Springer, *accepted*, 2019.

Supplementary Contributions

Conference paper

B.M. Ginting, R.-P. Mundani, and E. Rank. Parallel Simulations of Shallow Water Solvers for Modelling Overland Flows. In G. La Loggia, G. Freni, V. Puleo, and M. De Marchis, editors, *13th International Conference on Hydroinformatics*, volume 3, pages 788–799. EPiC Series in Engineering, 2018. https://dx.doi.org/10.29007/wdn8.

Posters

Poster 1

B.M. Ginting. High Performance Computing of Coupling 2D and 3D Numerical Modelling of Flood Propagation and its High Performance Interface and Visualisation. In *Doctoral Candidate's Day 2016 of Department Graduate Center Civil, Geo and Environmental Engineering at Technical University of Munich*, 21 January 2016, Munich, Germany. https://dx.doi.org/10.13140/RG.2.2.29721.42081/1.

Poster 2

B.M. Ginting. High Performance Computing of Coupling 2D and 3D Numerical Modelling of Flood Propagation and its High Performance Interface and Visualisation. In *Doctoral Candidate's Day 2018 of Department Graduate Center Civil, Geo and Environmental Engineering at Technical University of Munich*, 1 February 2018, Munich, Germany. https://dx.doi.org/10.13140/RG.2.2.13885.31205.

Poster 3

P.K. Bhola, B.M. Ginting, J. Leandro, K. Broich, R.-P. Mundani, and M. Disse. Model Parameter Uncertainty of a 2D Hydrodynamic Model for the Assessment of Disaster Resilience. In *EnviroInfo 2018*, 5–7 September 2018, Garching, Germany. doi: https://dx.doi.org/10.13140/RG.2.2.14641.63849.

Presentations

Presentation 1

B.M. Ginting. Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography. In *SimHydro 2017: Choosing The Right Model in Applied Hydraulics*, 14–16 June 2017, Nice, France.

Presentation 2

B.M. Ginting. Parallel Simulations of Shallow Water Solvers for Modelling Overland Flows. In *13th International Conference on Hydroinformatics*, 1–6 July 2018, Palermo, Italy.

Presentation 3

B.M. Ginting. Towards HPC-based Shallow Flow Simulations for Various Problems. In *EuroHPC Summit Week*, 13–17 May 2019, Poznan, Poland. `https://events.prace-ri.eu/event/850/contributions/725/`.

Presentation 4

B.M. Ginting. Hybrid-Parallel Simulations and Visualisations of Real Flood and Tsunami Events using Unstructured Meshes on High-Performance Cluster Systems. In *SimHydro 2019: Which Models for Extreme Situations and Crisis Management?*, 12–14 June 2019, Nice, France.

Abbreviations

AoS	arrays of structs
ASM	algebraic stress model
AV	artificial viscosity
AVX	Advanced Vector Extension
CCFV	cell-centered finite volume
CFL	Courant-Friedrichs-Lewy
CPU	central processing unit
CU	central-upwind
HLL	Harten-Lax-van Leer
HLLC	Harten-Lax-van Leer Contact
HPC	high-performance computing
MPI	message passing interface
MUSCL	Monotonic Upwind Scheme for Conservation Laws
NSEs	Navier-Stokes equations
OpenMP	open multiprocessing
Pe	Peclet
RANS	Reynolds-averaged Navier-Stokes
RKFO	Runge-Kutta fourth-order
RKSO	Runge-Kutta second-order
SFC	space-filling curve
SIMD	Single Instruction Multiple Data
SoA	structs of arrays
ScWF	scalable wall functions
StWF	standard wall functions
SWEs	shallow water equations
WDLB	weighted-dynamic load balancing

1. Introduction

Flood events are not only dangerous to human life, but can also cause great losses of property. According to [12], a flood is defined as a situation in which water temporarily covers a stretch of land that is usually not covered by water. Generally, there are several possible causes of flood events, for example, heavy rainfall in urban areas, dam-breaks, tsunamis in coastal areas, etc. These phenomena are strongly related to natural hazards, the characteristics of which are very complex—not only regarding spatial aspects, but also temporal ones. Due to their complexity, it is not possible to predict such hazards precisely, even with the most advanced pieces of technological equipment. However, some preventative measures may be taken, one of which is to study flow characteristics with the help of numerical tools to determine the negative impacts of a possible flood within the scope of disaster management.

Solving the shallow water equations (SWEs) is probably the most common approach conducted by many researchers and engineers when developing a numerical model for flood simulations. The SWEs can be obtained by depth-integrating the Navier-Stokes equations (NSEs), assuming that the horizontal length scale of the flood waves is much larger than the vertical one. It is generally impossible to obtain an analytical solution of the SWEs, so it is necessary to solve the SWEs numerically. Over the past 20 years, several different approaches to the solution of the SWEs have been proposed—involving finite difference, finite element, and finite volume methods, see [15; 73; 47; 39; 10; 37; 41; 11; 29; 57; 14; 52; 17; 71; 67; 72], among others. Over the past decades, finite volume methods have been widely used for free-surface water simulations. These methods rely on the application of so-called “solvers”, which can generally be categorized into two groups: Riemann solvers, such as Roe [59], Harten-Lax-van Leer (HLL) [27], and Harten-Lax-van Leer Contact (HLLC) [62]; and non-Riemann solvers, such as central-upwind (CU) [39] and artificial viscosity (AV) [17]. Note that the terminology of solver here is different from that in most of the fields in computational mechanics, e.g. a “p-FEM-solver” that refers to a solution algorithm for a complete problem—or from a solver that is used for a linear equation system, e.g. direct solver, iterative solver, etc. Here, the solver refers to a small part of the overall program, e.g. the one that computes the convective fluxes of the SWEs.

Both Riemann and non-Riemann solvers have become very popular due to their simplicity, robustness, and built-in conservation properties. Nevertheless, among the Riemann solvers themselves, one can see differences in the computational complexity of each scheme. The HLLC solver, for instance, was decomposed for a Riemann problem by dividing its final solution into four regions: left, left-contact, right-contact, and right states. This translates to an implementation that makes vast use of branch statements (`if-then-else`), which are computationally expensive. In contrast, the CU method, as a Riemann-solver-free scheme, is free of any branch statement, which leads to less computational overhead.

According to [1], in terms of high-performance computing (HPC), parallel computations can be considered as a way to enable the simultaneous use of thousands of processors for the problems under consideration. Currently, parallel computations are becoming more and more common in free-surface water modeling. The reason is obvious, as this allows one to carry out simulations that were hardly possible due to their huge computational cost. Some examples shall be explained. In [55], a parallel shallow flow model called LISFLOOD-FP was developed with open multiprocessing (OpenMP). A robust and accurate model called ParBrezo was developed in [60] for free-surface water simulations with a message passing interface (MPI). A parallel code of coupled shallow flow and transport equations was developed in [46] employing OpenMP on a multi-core architecture. A real dam-break case was comprehensively investigated in [69] by means of an OpenMP parallel shallow water model. Simulating the SWEs without any source term, [43] utilized an OpenMP-AVX based model to benefit not only from parallelization but also from vectorization. In [40], simulations with OpenMP were carried out to model rainfall-runoff events. A highly-efficient model parallelized with hybrid OpenMP-MPI called sam(oa)^2 was presented in [49] to solve the SWEs for tsunami flow applications. An efficient MPI model called FullSWOF2D was developed in [65] and used for overland flow type simulations.

The first investigation in this work deals with a thorough study, see [21] (**Paper 5**), to profile the computational complexity with respect to a solver's node-level performance on a desktop computer (single-core) with a Haswell architecture (Intel Xeon E3-1200 v3). The in-house code used in this work will be referred to as NUFSAW2D, which stands for Numerical simUlation of Free-surface ShAllow Water 2D. During this study, the main focus was primarily put on vectorization in order to exploit instruction level parallelism, which is supported by most compilers' optimization techniques [5; 56]. Also, vectorization is regarded the "easiest" way to increase computational efficiency, as the code requires less (or even no) modification. Nevertheless, the choice of proper data structures is inevitable, as not all structures support vectorization. It was shown in [21] that the HLLC method failed to utilize vectorization due to its branching formula, thus being more expensive than the AV and CU methods. The AV technique was found to be able to utilize vectorization almost as efficiently as the CU method, but in some particular cases turned out to be more accurate than the CU and HLLC schemes.

Encouraged by these node-level results, the efficiency of three solvers (Roe, HLLC, and CU) was further investigated on the level of shared-memory systems in [22] (**Paper 3**). The main focus was on utilizing vectorization for the solvers more efficiently, in particular, to enable vectorization for the HLLC method, which was not possible in the previous attempt. Three different systems with a modern hardware architecture — Intel Xeon E5-2690/Sandy-Bridge-E (Advanced Vector Extension/AVX), Intel Xeon E5-2697 v3/Haswell (AVX2), and Intel Xeon Phi/Knights Landing (AVX-512) — were employed to perform several dam-break and tsunami simulations. Both single-core and multi-core computations were carried out. The cell-edge reordering strategy proposed in [23] was employed in this paper in order to maintain contiguous memory access patterns, even in case of parallel (OpenMP) computations. It was found

in [22] that the cell-edge reordering strategy enabled vectorization for all solvers investigated, where the CU and HLLC schemes became the most efficient and the most inefficient one, respectively. This strongly indicates that the branching condition as the natural formulation of the HLLC solver will still be the main obstacle to achieve an efficient performance, even for simulations using the next generation of modern hardware.

The next investigation in this work deals with wet-dry simulations that belong to the most difficult phenomena in shallow flow modeling. When encountering wet-dry problems, a parallel shallow water code must not only be accurate but also efficient. Here, in order to produce accurate results, two important criteria should be fulfilled: the *well-balance property* and the *positivity-preserving property*. During the past decades, these two criteria have gotten a lot of attention in connection with both Riemann and non-Riemann solvers, see the non-exhaustive list of references [3; 34; 16; 6; 30; 31; 9; 51; 8], and the methods are still becoming more and more advanced. However, efficient parallel codes for wet-dry phenomena were often of minor interest only, although such phenomena can cause load imbalances among cores/nodes, thus decreasing the parallel efficiency. This is due to a special treatment of cells in case of wet-dry problems, where wet cells cause more computational effort than the dry ones. Note that due to the unpredictable emergence or disappearance of wet or dry cells during runtime, a load imbalance problem is very likely to occur without a proper load balancing strategy, even when considering wet-dry cells during an initial load distribution.

Some works that investigated the effect of wet-dry problems on the efficiency of a parallel shallow water code are described here. ParBrezo in [60] was introduced with a justification to distinguish wet cells from the dry ones, so that a relatively balanced load proportion could be assigned to each node. The idea was to find a proper ratio value of wet and dry cells by running a preliminary simulation with a specific domain decomposition. Once it finished, it was possible to roughly estimate the ratio of such wet and dry cells (e.g. 1:1, 2:1, 3:2, 3:1, or 5:1) before the output was used to correct the previous domain decomposition. Because such a ratio value is only applied once at the beginning of the simulations, this strategy is similar to the concept of a static domain decomposition. Recently, FullSWOF2D in [65] tackled the inefficient vectorization due to the branching condition of wet-dry problems by changing the code structure into a branch-free vectorized version. This technique, however, requires profound knowledge about intrinsic programming — by which a compiler directly implements a function given in code rather than linking to a library-based application of the function — and remains challenging for branching solvers such as the HLL and HLLC methods. In [24], NUFSAW2D employed dynamic load balancing (with OpenMP directive) to tackle load imbalances among cores for wet-dry simulations. Nevertheless, a significant performance decrease was still shown and it is therefore important to develop a new strategy that can tackle load imbalances due to wet-dry problems more efficiently.

To this end, a novel *weighted-dynamic load balancing* (WDLB) technique was proposed for NUFSAW2D in [23] (**Paper 1**), which requires no preliminary run as done in [60]. This tech-

nique was integrated into an edge-based cell-centered finite volume (CCFV) method, where two main computation levels were considered: edge-driven and cell-driven calculations. At the edge-driven level, computations were carried out separately between internal and boundary edges, as the latter ones require less computational time. The CU method was chosen in this paper due to its efficiency according to the two previous investigations. All these implementations were eased by means of the cell-edge reordering strategy.

Having successfully balanced load distributions due to wet-dry problems in [23] for up to 6.4 million cells (12.8 million edges), two turbulence models were included into NUFSAW2D: κ-ϵ model and algebraic stress model (ASM). This was motivated by a fact that the SWEs neglect the diffusive terms, thus taking only the frictional effects from the bed into account—but not the shear stresses, which originate from the fluid itself. In flow-past-structure cases (especially in turbulent regimes) all the mass, momentum, and heat transfers in water exist and fluctuate with respect to time. Such a momentum transfer will physically affect the shear stresses, and these stresses must therefore be accounted for. Although a number of (two-equation) turbulence models for the SWEs is available, see [58; 66; 7; 68], the integration of such models into the shallow water models still remains challenging when wet-dry problems exist. This is (partly) due to the implementation of the standard wall functions (StWF) for moving boundary geometries, which exist in the presence of wet-dry phenomena.

With the StWF, one needs to keep the near-wall mesh spacing in the range of $y_p^+ \geq 11.067$, where y_p^+ is a dimensionless wall distance, namely by placing the center of the first boundary cells in the logarithmic region. To determine a proper near-wall mesh size that satisfies this condition, it is in most cases necessary to perform a grid-independent study first. In [35], another method, namely the enhanced wall treatment, was compared with the StWF. The former requires y_p^+ to be as low as possible, hence the meshes near the walls need to be adaptively refined. If this adaptive technique is employed for inviscid shallow flow simulations, however, problems with respect to simultaneously preserving the C-property and the mass conservation may occur. Although a solution was presented in [42], the extension for turbulence simulations may become very complex. For the use of high-resolution meshes to increase accuracy, the suggestions of the grid study in [35] are difficult to be realized for practical purposes in wet-dry simulations, see Section 4.2.4 for detail. Therefore, a new approach was proposed in [18] (**Paper 2**), combining the hydrostatic and topography reconstructions and the scalable wall functions (ScWF). Here, the wet-dry phenomena near the interfaces are captured accurately, and the turbulence values near such wet-dry interfaces and the other wall boundaries are calculated properly. The ASM was employed, which is a non-linear extension of the κ-ϵ model. The proposed approach resolved the limitations of the StWF, thus giving users flexibility in generating meshes.

In some particular cases, it can be noticed that the CU scheme, despite its efficiency, is not able to capture recirculating turbulent shallow flows properly, unless very fine meshes are used. A similar phenomenon was investigated previously in [53], where the CU method cap-

tured the inviscid recirculating shallow flows less accurately than the Roe and HLLC solvers. This led to the idea of combining the AV technique and the CU scheme in order to simulate recirculating turbulent flows, see [19] (**Paper 4**). The hybrid AV-CU scheme was able to resolve the inability/inaccuracy of the CU scheme in simulating recirculating turbulent flows, and turned out to be more accurate and cheaper than the HLLC solver.

All the aforementioned results were obtained using rectangular meshes. Although the shape of complex domains can be approximated using high-resolution rectangular cells, unstructured meshes are in many applications preferable due to their flexibility. Also, with a data structure that not only supports applications for structured meshes but also the unstructured ones, a parallel shallow water code can allow for a wider range of implementations. To this regard, a second data structure type was introduced in NUFSAW2D in [25] (**Paper 6**) for unstructured meshes (hybrid quadrilateral and triangular cells) supporting hybrid OpenMP-MPI computations. Three main objectives were considered: (1) contiguous edge and cell numbers, thus easing parallelization, (2) consecutive communication patterns among nodes, and (3) contiguous memory access patterns inside each node. The first objective was addressed by means of a block-distribution, and the other objectives were achieved by employing space-filling curves (SFCs) to reorder the meshes. The data structure devised served to implement the WDLB technique for the unstructured meshes of up to 3.4 million cells (5.1 million edges) when simulating real river flood and tsunami cases.

In summary, having modern hardware and small to moderate compute clusters available in academia and industry, shallow water simulations even for huge/complex problems become possible due to advanced parallelization strategies that could hardly be performed in earlier times. In addition to being accurate, a parallel code should also be efficient and able to comprehensively exploit the capability of such computing resources on all scales from the instruction level to the block and process level of parallelization. In particular, it is therefore of uttermost importance to carefully investigate wet-dry phenomena as the main factor regarding a performance decrease within the parallel code.

2. Mathematical and Computational Framework

2.1. Mathematical Formulation

2.1.1. Shallow Water Equations (with Turbulence Model)

The SWEs with turbulence model, which are also known as the (depth-averaged) Reynolds-averaged Navier-Stokes (RANS) equations, can be written, closely following the notation of [58; 66; 7; 68], as

$$\frac{\partial Q}{\partial t} + \frac{\partial C_x}{\partial x} + \frac{\partial C_y}{\partial y} = \frac{\partial D_x}{\partial x} + \frac{\partial D_y}{\partial y} + S_{bx} + S_{by} + S_f + S_o \,, \tag{2.1}$$

where Q denotes the conservative variables, C_x and C_y are the convective fluxes (or the convective terms), D_x and D_y denote the diffusive fluxes (or the viscous terms), which are obtained based on the Boussinesq's assumption, S_{bx} and S_{by} denote the bed-slope terms, S_f defines the bed friction term, and S_o denotes the rainfall/infiltration term. These variables are expressed in matrix notation as

$$Q = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} \,, \quad C_x = \begin{bmatrix} hu \\ huu + \dfrac{gh^2}{2} \\ hvu \end{bmatrix} \,, \quad C_y = \begin{bmatrix} hv \\ huv \\ hvv + \dfrac{gh^2}{2} \end{bmatrix} \,,$$

$$D_x = \begin{bmatrix} 0 \\ 2h(\nu_e + \nu_t)\dfrac{\partial u}{\partial x} - \dfrac{2}{3}h\kappa \\ h(\nu_e + \nu_t)\left(\dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x}\right) \end{bmatrix} \,, \quad D_y = \begin{bmatrix} 0 \\ h(\nu_e + \nu_t)\left(\dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x}\right) \\ 2h(\nu_e + \nu_t)\dfrac{\partial v}{\partial y} - \dfrac{2}{3}h\kappa \end{bmatrix} \,,$$

$$S_{bx} = \begin{bmatrix} 0 \\ -gh\dfrac{\partial z_b}{\partial x} \\ 0 \end{bmatrix} \,, \quad S_{by} = \begin{bmatrix} 0 \\ 0 \\ -gh\dfrac{\partial z_b}{\partial y} \end{bmatrix} \,, \quad S_f = \begin{bmatrix} 0 \\ -c_f u\sqrt{u^2+v^2} \\ -c_f v\sqrt{u^2+v^2} \end{bmatrix} \,, \quad S_o = \begin{bmatrix} R - I \\ 0 \\ 0 \end{bmatrix} \,. \tag{2.2}$$

The variables h (m), u (m/s), and v (m/s) are the depth and velocities in x and y directions, respectively, g (m/s^2) is the gravity acceleration, ν_e (m^2/s) and ν_t (m^2/s) are the kinematic viscosity and the eddy viscosity, respectively, κ (m^2/s^2) is the turbulence kinetic energy, z_b (m) is the bed contour, n_m (s/m$^{1/3}$) is the Manning coefficient, where $c_f = g n_m^2 h^{-\frac{1}{3}}$, and R (m/s) and I (m/s) are the rainfall and infiltration, respectively. Note that for the inviscid SWEs, the values of D_x and D_y are zero, thus requiring no turbulence model.

2.1.2. Algebraic Stress Model

A turbulence model is required to compute κ and ν_t in Eq. (2.2). According to [7], the algebraic stress model (ASM) can be interpreted as a non-linear extension of eddy viscosity models and is, here, correlated with the κ-ϵ model. Therefore, the κ-ϵ model is first presented and expressed as

$$\frac{\partial \Phi}{\partial t} + \frac{\partial C_{\Phi,x}}{\partial x} + \frac{\partial C_{\Phi,y}}{\partial y} = \frac{\partial D_{\Phi,x}}{\partial x} + \frac{\partial D_{\Phi,y}}{\partial y} + S_{\kappa\text{-}\epsilon} \, . \tag{2.3}$$

In matrix notation, these variables are denoted as

$$\Phi = \begin{bmatrix} h\kappa \\ h\epsilon \end{bmatrix} , \quad C_{\Phi,x} = \begin{bmatrix} h\kappa u \\ h\epsilon u \end{bmatrix} , \quad C_{\Phi,y} = \begin{bmatrix} h\kappa v \\ h\epsilon v \end{bmatrix} ,$$

$$D_{\Phi,x} = \begin{bmatrix} \sigma_\kappa^{-1} h\nu_t \dfrac{\partial \kappa}{\partial x} \\ \sigma_\epsilon^{-1} h\nu_t \dfrac{\partial \epsilon}{\partial x} \end{bmatrix} , \quad D_{\Phi,y} = \begin{bmatrix} \sigma_\kappa^{-1} h\nu_t \dfrac{\partial \kappa}{\partial y} \\ \sigma_\epsilon^{-1} h\nu_t \dfrac{\partial \epsilon}{\partial y} \end{bmatrix} , \quad S_{\kappa\text{-}\epsilon} = \begin{bmatrix} P_h + P_{\kappa b} - h\epsilon \\ c_{\epsilon 1} \dfrac{\epsilon}{\kappa} P_h + P_{\epsilon b} - c_{\epsilon 2} h \dfrac{\epsilon^2}{\kappa} \end{bmatrix} , \tag{2.4}$$

where ϵ (m^2/s^3) denotes the dissipation rate of the turbulence kinetic energy. The terms P_h, $P_{\kappa b}$, and $P_{\epsilon b}$ are defined as

$$\begin{aligned} P_h &= h\nu_t \left(2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2 \right) , \\ P_{\kappa b} &= c_f^{-0.5} U_*^3 , \quad P_{\epsilon b} = \frac{c_{\epsilon\Gamma}\, c_{\epsilon 2}\, c_\mu^{0.5}\, c_f^{-0.75}}{h} , \end{aligned} \tag{2.5}$$

where $U_* = \sqrt{c_f\left(u^2 + v^2\right)}$ denotes the bed friction velocity. The eddy viscosity ν_t is calculated as

$$\nu_t = c_\mu \left(\frac{\kappa^2}{\epsilon} \right) . \tag{2.6}$$

All coefficients in Eqs. (2.4)–(2.6) are given in [58; 64] by numerous iterations of data-fitting as

$$c_\mu = 0.09 \, , \;\; c_{\epsilon 1} = 1.44 \, , \;\; c_{\epsilon 2} = 1.92 \, , \;\; \sigma_\kappa = 1.0 \, , \;\; \sigma_\epsilon = 1.3 \, , \;\; c_{\epsilon\Gamma} = \left[1.8, 3.6\right] . \tag{2.7}$$

Closely following [7], the ASM is now employed to recalculate P_h (later denoted by P_h^*) by considering three components of the Reynolds stresses that appear in shallow water flows: $\overline{u'^2}$, $\overline{u'v'}$, and $\overline{v'^2}$. To account for the production of turbulent energy due to the horizontal velocity gradient, a formula P_h^* is employed, namely

$$P_h^* = h \left[-\overline{u'^2}\left(\frac{\partial u}{\partial x}\right) - \overline{v'^2}\left(\frac{\partial v}{\partial y}\right) - \overline{u'v'}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \right] + \left[\frac{P_{uu,b} + P_{vv,b}}{2} \right] , \tag{2.8}$$

where $P_{uu,b}$ and $P_{vv,b}$ are taken into account for the vertical production of turbulent energy

due to bed shear. First, the stresses $\overline{u'^2}$, $\overline{u'v'}$, and $\overline{v'^2}$ are calculated. Assuming a constant value of $c_1 = 1.8$, in tensorial form these stresses are computed as

$$m_{ij} r_j = c_{11} b_i \ , \tag{2.9}$$

where $i, j \in \big[1, 3\big]$, $r_j = \big[\overline{u'^2}, \overline{v'^2}, \overline{u'v'}\big]^\mathsf{T}$, and $c_{11} = c_1 + \dfrac{P_h}{\epsilon} - 1$. The matrix m_{ij} is expressed as

$$m_{ij} = c_{11}\delta_{ij} - (1 - c_2)\frac{\kappa}{\epsilon} a_{ij} \ , \tag{2.10}$$

where $c_2 = 0.6$ and δ_{ij} is Kronecker delta. The matrix a_{ij} and the vector b_i are defined by

$$\begin{bmatrix} -\dfrac{4}{3}\dfrac{\partial u}{\partial x} & \dfrac{2}{3}\dfrac{\partial v}{\partial y} & -\dfrac{4}{3}\dfrac{\partial u}{\partial y} + \dfrac{2}{3}\dfrac{\partial v}{\partial x} \\ -\dfrac{2}{3}\dfrac{\partial u}{\partial x} & -\dfrac{4}{3}\dfrac{\partial v}{\partial y} & -\dfrac{4}{3}\dfrac{\partial v}{\partial x} + \dfrac{2}{3}\dfrac{\partial u}{\partial y} \\ -\dfrac{\partial v}{\partial x} & -\dfrac{\partial u}{\partial y} & -\left(\dfrac{\partial u}{\partial x} + \dfrac{\partial v}{\partial y}\right) \end{bmatrix} \ , \quad \begin{bmatrix} \dfrac{2}{3}\kappa + \dfrac{\kappa}{\epsilon}\dfrac{(1-c_2)}{c_{11}}\left(\dfrac{2}{3}P_{uu,b} - \dfrac{1}{3}P_{vv,b}\right) \\ \dfrac{2}{3}\kappa + \dfrac{\kappa}{\epsilon}\dfrac{(1-c_2)}{c_{11}}\left(\dfrac{2}{3}P_{vv,b} - \dfrac{1}{3}P_{uu,b}\right) \\ \dfrac{\kappa}{\epsilon}\dfrac{(1-c_2)}{c_{11}}P_{uv,b} \end{bmatrix} \ , \tag{2.11}$$

respectively. The values of $P_{uu,b}$, $P_{vv,b}$, and $P_{uv,b}$ are calculated by

$$P_{uu,b} = 2\frac{c_f u^2 |\mathbf{U}|}{h} \ , \quad P_{vv,b} = 2\frac{c_f v^2 |\mathbf{U}|}{h} \ , \quad P_{uv,b} = 2\frac{c_f uv |\mathbf{U}|}{h} \ , \tag{2.12}$$

where $|\mathbf{U}|$ can be computed by Keulegan's law. For the sake of simplicity, $|\mathbf{U}|$ is assumed to be similar to the depth-averaged velocity.

2.2. Numerical Model

2.2.1. Spatial Discretization

The first step in the derivation of the CCFV method is the generation of a set of sub-domains. Therefore, as an example, the computational domain Ω — which is bounded by the closed polygon ABCD — will be divided into four non-overlapping sub-domains Ω_1, Ω_2, Ω_3, and Ω_4, see Fig. 1. Both Eqs. (2.1) and (2.3) can be integrated, for example over the sub-domain Ω_1, by applying the Gauss divergence theorem, thus

$$\iint_{\Omega_1} \frac{\partial Q}{\partial t} d\Omega_1 + \oint_{\Gamma_{\mathsf{AEGH}}} \Big(C_x + C_y - D_x - D_y - S_{bx} - S_{by}\Big) \cdot \vec{n} \, d\Gamma_{\mathsf{AEGH}} = \iint_{\Omega_1} \big(S_f + S_o\big) \, d\Omega_1 \ , \tag{2.13}$$

$$\iint_{\Omega_1} \frac{\partial \Phi}{\partial t} d\Omega_1 + \oint_{\Gamma_{\mathsf{AEGH}}} \Big(C_{\Phi,x} + C_{\Phi,y} - D_{\Phi,x} - D_{\Phi,y}\Big) \cdot \vec{n} \, d\Gamma_{\mathsf{AEGH}} = \iint_{\Omega_1} S_{\kappa\text{-}\epsilon} \, d\Omega_1 \ , \tag{2.14}$$

where Γ denotes the line boundary of a sub-domain and $\vec{n} = \big[n_x, n_y\big]^\mathsf{T}$ is the unit normal vector pointing outward of the boundary. Letting N be the total number of edges surrounding a sub-domain, the sub-domain Ω_1 in Fig. 1 has four edges (AE, EG, GH, and HA) denoted by

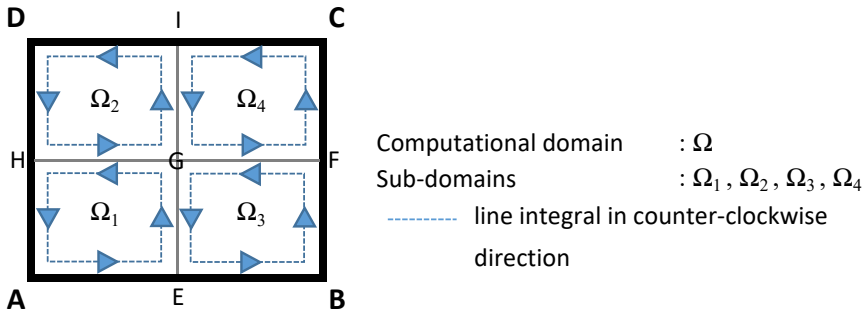


Figure 1 : Generation of sub-domains from a computational domain Ω

$i \in [1, 4]$, and thus $N = 4$.

The line integrals in Eqs. (2.13) and (2.14) are approximated as

$$\oint_{\Gamma_{\text{AEGH}}} \Big(C_x + C_y - D_x - D_y - S_{bx} - S_{by} \Big) \cdot \vec{n} \, d\Gamma_{\text{AEGH}} = \sum_{i=1}^{N} \Big(\big(C_x - D_x - S_{bx}\big) \, n_x + \big(C_y - D_y - S_{by}\big) \, n_y \Big)_i \, \Delta L_i \, , \tag{2.15}$$

$$\oint_{\Gamma_{\text{AEGH}}} \Big(C_{\Phi,x} + C_{\Phi,y} - D_{\Phi,x} - D_{\Phi,y} \Big) \cdot \vec{n} \, d\Gamma_{\text{AEGH}} = \sum_{i=1}^{N} \Big(\big(C_{\Phi,x} - D_{\Phi,x}\big) \, n_x + \big(C_{\Phi,y} - D_{\Phi,y}\big) \, n_y \Big)_i \, \Delta L_i \, , \tag{2.16}$$

where ΔL is the edge length. With the CCFV method, the integral of vector Q for the sub-domain Ω_1 is defined as

$$Q_{\Omega_1} = \frac{1}{A_{\Omega_1}} \iint_{\Omega_1} Q \, d\Omega_1 \, , \tag{2.17}$$

where A_{Ω_1} is the area of sub-domain Ω_1. Note that the integrals of vectors S_f, S_o, Φ, and $S_{\kappa\text{-}\epsilon}$ can be defined in a similar way. From now on, a sub-domain refers to a cell denoted by Ω_e with a line boundary Γ_e, for which N may vary depending on the shape of the cell used. By definition of Eqs. (2.15)–(2.17), both Eqs. (2.13) and (2.14) for a cell can be rewritten as

$$A_{\Omega_e} \frac{\partial Q_{\Omega_e}}{\partial t} + \sum_{i=1}^{N} \Big(\big(C_x - D_x - S_{bx}\big) \, n_x + \big(C_y - D_y - S_{by}\big) \, n_y \Big)_i \, \Delta L_i = A_{\Omega_e} \big(S_{f_{\Omega_e}} + S_{o\Omega_e} \big) \, , \tag{2.18a}$$

$$A_{\Omega_e} \frac{\partial \Phi_{\Omega_e}}{\partial t} + \sum_{i=1}^{N} \Big(\big(C_{\Phi,x} - D_{\Phi,x}\big) \, n_x + \big(C_{\Phi,y} - D_{\Phi,y}\big) \, n_y \Big)_i \, \Delta L_i = A_{\Omega_e} S_{\kappa\text{-}\epsilon\Omega_e} \, . \tag{2.18b}$$

Note that the values of h, u, v, z_b, n_m, R, I, κ, and ϵ are defined at the center of each cell. For the sake of completeness, the water elevation η (m) is written here as $\eta = h + z_b$.

A solver such as Roe, HLLC, CU, or AV scheme is required to calculate the convective fluxes in Eqs. (2.18a) and (2.18b). Prior to applying such a solver, the Monotonic Upwind Scheme for Conservation Laws (MUSCL) method is employed to reconstruct the left (L) and right (R) states of an edge so that a second-order spatial accuracy can be achieved. Note that both conservative and primitive variable reconstructions are employed in this work. The former is used for inviscid simulations and the latter is employed for turbulence simulations, where the gradient values required for the computation of the MUSCL method can be used directly to calculate the diffusive fluxes. Such fluxes are computed with the centered method simply by averaging the values of the two corresponding cells for the value stored at an edge. Note that as one of the objectives of this work is to compare the efficiency of several solvers, the computation of the bed-slope term must be performed independently from the convective fluxes so that a fair comparison can be obtained. Therefore, a non-Riemann technique is applied to discretize the bed-slope term, which is applicable to all implemented solvers. A semi-implicit treatment is utilized for the bed friction and the turbulence source terms.

2.2.2. Temporal Discretization

In this work, two temporal discretization schemes were employed: the Runge-Kutta second-order (RKSO) and the Runge-Kutta fourth-order (RKFO) method. The former was used in [23; 18; 19; 25] and the latter in [21; 22]. For the sake of brevity, only the RKSO method is explained here. The procedures of the RKSO method can be expressed as

$$\mathbf{K}^t_{\Omega_e} = -\frac{\Delta t}{A_{\Omega_e}}\left[\sum_{i=1}^{N}\Big(\big(C_x - D_x - S_{bx}\big)\,n_x + \big(C_y - D_y - S_{by}\big)\,n_y\Big)^t_i \Delta L_i\right] + \Delta t\, S^t_{o\Omega_e}\,,$$
$$Q^{t*}_{\Omega_e} = Q^t_{\Omega_e} + \mathbf{K}^t_{\Omega_e}\,, \quad Q^{t*}_{\Omega_e} \leftarrow \Pi^{-1}_{\Omega_e} Q^{t*}_{\Omega_e}\,, \quad Q^{t+1}_{\Omega_e} = \frac{1}{2}\left(Q^t_{\Omega_e} + Q^{t*}_{\Omega_e} + \mathbf{K}^{t*}_{\Omega_e}\right), \tag{2.19a}$$

$$\mathbf{K}_{\Phi}{}^t_{\Omega_e} = -\frac{\Delta t}{A_{\Omega_e}}\left[\sum_{i=1}^{N}\Big(C_{\Phi,x}\,n_x + C_{\Phi,y}\,n_y\Big)^t_i \Delta L_i\right],$$
$$\Phi^{t*}_{\Omega_e} = \Phi^t_{\Omega_e} + \mathbf{K}_{\Phi}{}^t_{\Omega_e}\,, \quad \Phi^{t*}_{\Omega_e} \leftarrow \Phi^{t*}_{\Omega_e} + \Delta t\,\mathbf{H}^t_{\Omega_e}\,, \quad \Phi^{t+1}_{\Omega_e} = \frac{1}{2}\left(\Phi^t_{\Omega_e} + \Phi^{t*}_{\Omega_e} + \mathbf{K}_{\Phi}{}^{t*}_{\Omega_e}\right), \tag{2.19b}$$

where t (s) and $t*$ (s) denote the discrete time steps and Δt (s) is the time step size. The variable Π relates to the friction term $S_{f_{\Omega_e}}$ and is calculated in a semi-implicit manner as

$$\Pi_{\Omega_e} = 1 \;+\; g\,\Delta t\left[(1-\theta)\left(\frac{n_m^2\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}\right)^{t*}_{\Omega_e} + \theta\left(\frac{n_m^2\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}\right)^{(t*-1)}_{\Omega_e}\right], \tag{2.20}$$

where θ is an implicitness coefficient ($0 < \theta < 1$). The variable $\mathbf{H}$ denotes the turbulence source term, which is computed semi-implicitly as

$$\mathbf{H}_{\Omega_e}^{(t*-1)} = \left[\max\left(\frac{\mathbf{H}_1^{(t*-1)}}{A_{\Omega_e}}, 0 \right) + \mathbf{H}_2^{(t*-1)} + \mathbf{H}_3^{(t*-1)} \right] + \Phi_{\Omega_e}^{t*} \left[\min\left(\frac{\mathbf{H}_1^{(t*-1)}}{\left(A_{\Omega_e} \Phi_{\Omega_e}^{(t*-1)}\right)}, 0 \right) + \mathbf{H}_4^{(t*-1)} \right] . \tag{2.21}$$

The variable $\mathbf{H}_1$ relates to the convective fluxes of the turbulence model, while $\mathbf{H}_2$, $\mathbf{H}_3$, and $\mathbf{H}_4$ correspond to the turbulence source term. These variables are expressed as

$$\mathbf{H}_1 = \begin{bmatrix} \oint_{\Gamma_e} \left[\sigma_\kappa^{-1} h \nu_t \left(\frac{\partial \kappa}{\partial x} + \frac{\partial \kappa}{\partial y} \right) \right] \cdot \vec{n} \, d\Gamma_e \\ \oint_{\Gamma_e} \left[\sigma_\epsilon^{-1} h \nu_t \left(\frac{\partial \epsilon}{\partial x} + \frac{\partial \epsilon}{\partial y} \right) \right] \cdot \vec{n} \, d\Gamma_e \end{bmatrix} , \quad \mathbf{H}_2 = \begin{bmatrix} P_{h\Omega_e} \\ \left(c_{\epsilon 1} \frac{\epsilon}{\kappa} P_h \right)_{\Omega_e} \end{bmatrix} ,$$
$$\mathbf{H}_3 = \begin{bmatrix} P_{\kappa b \Omega_e} \\ P_{\epsilon b \Omega_e} \end{bmatrix} , \quad \mathbf{H}_4 = \begin{bmatrix} -\left(\frac{\epsilon}{\kappa} \right)_{\Omega_e} \\ -\left(c_{\epsilon 2} \frac{\epsilon}{\kappa} \right)_{\Omega_e} \end{bmatrix} . \tag{2.22}$$

Without the friction term, one can start the computations by explicitly calculating $h_{\Omega_e}^{t*}$, $hu_{\Omega_e}^{t*}$, and $hv_{\Omega_e}^{t*}$ in vector $Q_{\Omega_e}^{t*}$. Afterwards, both $u_{\Omega_e}^{t*}$ and $v_{\Omega_e}^{t*}$ are calculated by dividing $hu_{\Omega_e}^{t*}$ and $hv_{\Omega_e}^{t*}$ with $h_{\Omega_e}^{t*}$—up to here, no friction term was taken into account. From now on, the friction term must be included in order to update the values of $hu_{\Omega_e}^{t*}$ and $hv_{\Omega_e}^{t*}$. In wet-dry cases, since $h_{\Omega_e}^{t*}$ can be advanced without any source term, its value is used to determine whether $hu_{\Omega_e}^{t*}$, $hv_{\Omega_e}^{t*}$, $h\kappa_{\Omega_e}^{t*}$, and $h\epsilon_{\Omega_e}^{t*}$ should be computed or not. For instance, if $10^{-6} \leq h_{\Omega_e}^{t*} \leq 10^{-10}$ m, cell Ω_e is categorized as dry cell, thus the values of $hu_{\Omega_e}^{t*}$, $hv_{\Omega_e}^{t*}$, $u_{\Omega_e}^{t*}$, $v_{\Omega_e}^{t*}$, $\kappa_{\Omega_e}^{t*}$, and $\epsilon_{\Omega_e}^{t*}$ are simply set to a very small value (e.g. 10^{-15}); otherwise, all corresponding values are computed normally. Note that the mass conservation with the threshold parameter were monitored in all simulations. For both inviscid and turbulence simulations in this work, the variable Δt is limited by the Courant-Friedrichs-Lewy (CFL) number, while the Peclet (Pe) number that satisfies Pe $\leq$ 2/CFL is considered for the latter, see [28]. All details for both spatial and temporal discretization can be found in [17; 18; 19].

2.3. Integration of Numerical Scheme into Parallelization Framework

With respect to Eqs. (2.18a) and (2.18b), the variables Q, S_f, S_o, Φ, and $S_{\kappa\text{-}\epsilon}$ are located at cell-centers, whereas the variables C_x, C_y, D_x, D_y, S_{bx}, S_{by}, $C_{\Phi,x}$, $C_{\Phi,y}$, $D_{\Phi,x}$, and $D_{\Phi,y}$ belong to edges. As an initial condition, the values in vector Q (i.e. h, u, and v), the values in vector Φ (i.e. κ and ϵ), and the values of z_b and n_m are given and saved at all cell-centers, whereas no initial values at edges are given. The values of R and I are known (or set to zero) for each simulation in this work. It is therefore not required to save the values of these

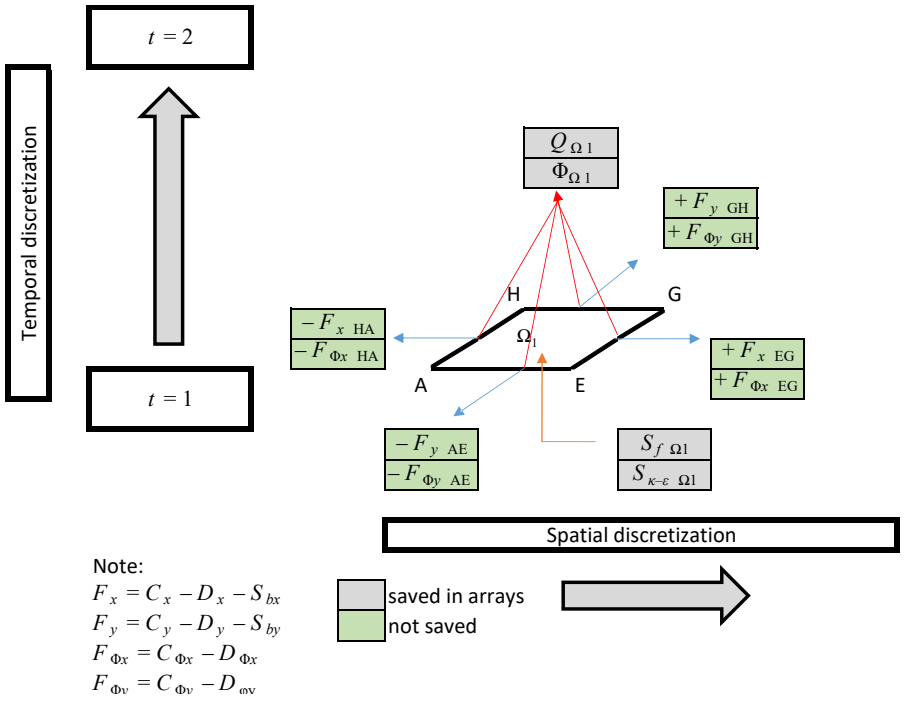


Figure 2 : Concept of computation in cell-based CCFV. In this approach, the values located at edges are not saved in arrays and computed with the unit normal vectors pointing outward of the boundaries. Therefore, computations of an edge that corresponds to two adjacent cells occur twice with the same value but different algebraic signs.

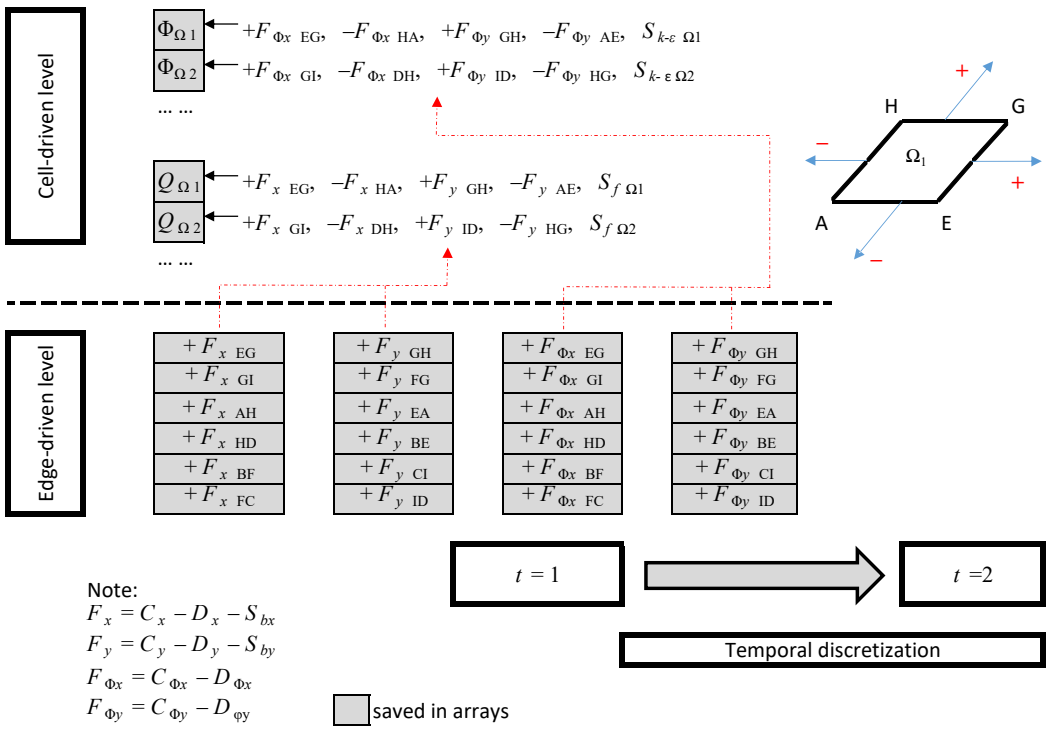


Figure 3 : Concept of computation in edge-based CCFV. In this approach, the values located at edges are saved in arrays. Computations for all edges (edge-driven level) must be finished prior to the ones for cells (cell-driven level). For this, all edge values are first assumed to have a positive sign. Once the computations for the edge-driven level are finished, the computations of the cell-driven level can be performed by accessing the edge values but with the corresponding (actual) signs of the unit normal vectors.

variables at cell-centers. In subsequent time steps, vectors Q and Φ will be advanced using all the aforementioned values from the previous time step, whereas the values of z_b and n_m remain constant during simulation time as a fixed-bed model is employed in this work. Note that in subsequent time steps, values located at edges will always be computed but do not have to be saved, depending on the chosen approach. This has the following reasons explained in the next paragraph.

Using a CCFV model, the solution advancement can be performed in two ways: cell-based and edge-based, see Figs. 2 and 3 for an illustration according to the domain discretization introduced in Fig. 1. In the cell-based approach (Fig. 2), all values are given at cell-centers, hence, one does not redundantly have to save the values at edges. In other words, the values of Q and Φ for any subsequent time step can be advanced using variables at cell-centers and edges in one single loop. Because computations for all edges can be performed in a straight-forward manner by accessing the values from their corresponding cell-centers, there is no need to save such edge values in an array. This approach thus requires less memory. However, the computation of an edge corresponding to two adjacent cells — see, for example, edge GH in Fig. 1 that corresponds to cells Ω_1 and Ω_2 — occurs twice with the same value but different algebraic sign, hence, the values of the convective and diffusive fluxes become redundant. Note that as calculations of the convective fluxes deal with computations of a solver (the most expensive part of the SWEs component, see [22]), the central-processing unit (CPU) time will significantly increase.

In contrast, for the edge-based approach (Fig. 3), one first needs to compute and then save the values at all edges entirely (edge-driven level) prior to advancing the solution at cells (cell-driven level). This means that the advancement process at cells is conducted later by accessing the computed edge values. For this, the value of each edge is only computed once (in one direction), see Fig. 3, where all edge values for the computations of the edge-driven level are assumed to have a positive sign and saved in an array. Later, these edge values are accessed for the computations of the cell-driven level but use the corresponding (actual) signs of the unit normal vectors. On the one hand, this approach requires more memory but, on the other hand, it avoids redundant computations of an edge for its two corresponding cells.

For parallelization, the cell-based approach becomes more advantageous. It is easy to be parallelized because no cell dependency occurs. Although the edge-based approach may also not be difficult to be parallelized, an advanced strategy must be considered to provide an index relationship between edges and cells, unless a poor memory access pattern is obtained, leading to significantly worse performance. In order to give a brief overview of these two approaches, a 3×3 computational domain is given.

Following the cell-based approach, the parallelization can easily be carried out by directly mapping cells to cores (OpenMP) or distributing cells to different nodes (MPI), see Fig. 4. With

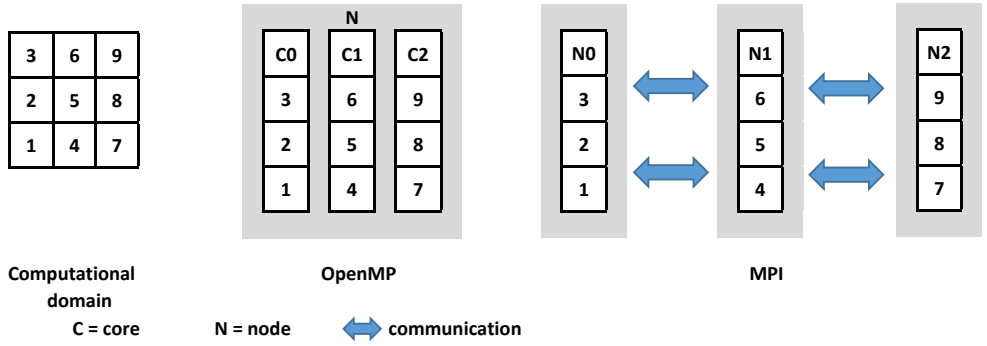


Figure 4 : Concept of parallelization in cell-based CCFV. With OpenMP, an automatic cell distribution is shown, where cells are equally and consecutively allocated to cores. An example of a manual cell distribution within MPI is presented; note that several different decompositions may also be obtained depending on how the user decides. With MPI, an equally- and consecutively-distributed amount of cells is achieved among nodes as well. However, communication between nodes is required.

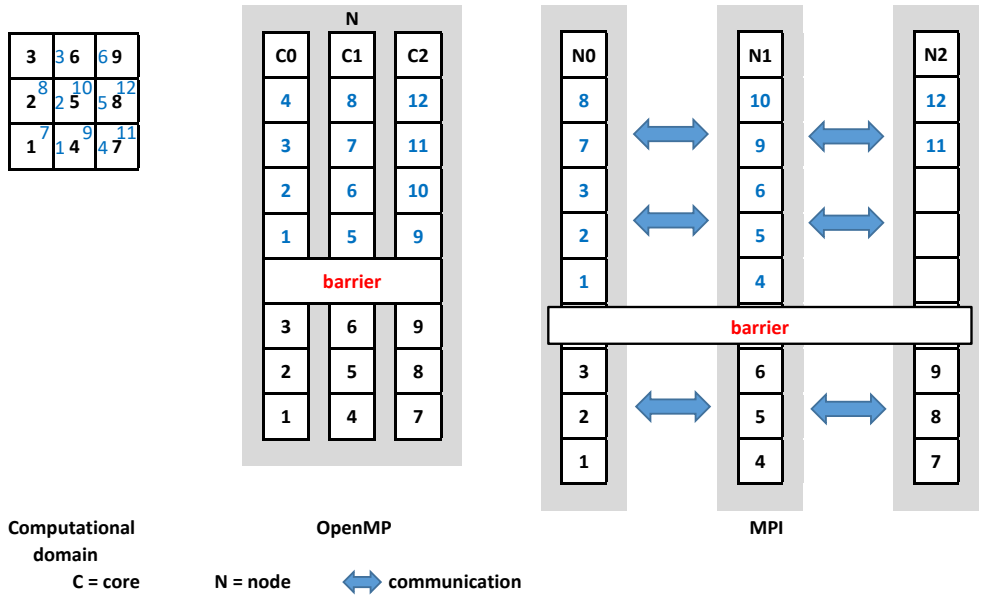


Figure 5 : Concept of parallelization in edge-based CCFV. For the sake of simplicity, instead of 24 edges, only 12 edges of the computational domain are included here as an example. Within MPI, after a manual cell distribution is chosen by the user, cells are equally and consecutively distributed among nodes. However, it is not possible to equally and consecutively distribute edges to nodes, because the (manual) distribution has been targeted previously on the cell distribution. In the edge-based approach, more communication is required than in the cell-based approach. With OpenMP, automatic cell and edge allocations are shown, where both cells and edges are equally and consecutively allocated into cores. Note that a barrier is required to ensure that the computations at the edge-driven level are finished before the computations at the cell-driven levels are started.

OpenMP, the cell allocation is conducted automatically, e.g. using the directive `!$omp do`, while the cell distribution within MPI is performed based on the domain decomposition as intended by the user. It is shown in Fig. 4 that cells are equally and consecutively allocated/distributed to cores/nodes with OpenMP/MPI. However, communication among nodes is required for MPI.

In the edge-based approach, one has to consider not only the cell allocation/distribution but also the edge one. Again, cells and edges can be mapped to cores automatically using

OpenMP, see Fig. 5. However, with MPI, the edge distribution to nodes may turn out to be imbalanced because the (manual) decomposition has been targeted previously (by the user) on the cell distribution. From Fig. 5 (MPI), it follows that cells can be consecutively distributed among nodes, where each node receives the same amount of cells. However, each node does not obtain the same amount of edges, thus causing load imbalance. Note that more communication is required in the edge-based approach than in the cell-based one. With OpenMP, both edges and cells can be allocated consecutively to cores in a simple fashion. This approach also allows each core to have the same amount of edges and cells.

Even though the edge-based approach entails a larger memory footprint, it was chosen for this work, because the higher computational cost of the cell-based approach (with all its advantages) deteriorates the computational efficiency/performance as the primary optimization objective. The challenging task is to design a parallelization strategy that can satisfy the three aforementioned main objectives within the framework of hybrid OpenMP-MPI. Note that even for a perfect allocation/distribution strategy, load imbalances can still occur due to the emergence or disappearance of wet-dry edges/cells dynamically during runtime. This issue will be investigated thoroughly in Chapter 4.

3. Problem Statements and Objectives

As high-performance hardware becomes increasingly available both in academia and industry, parallelization can be considered as a way to obtain the solution of shallow flow simulations faster than performing sequentially. There are numerous references regarding parallel programming, allowing researchers and engineers to develop their own parallel shallow water codes. However, there is still a lack of deep understanding of what is possible in order to make codes more efficient. Here, the term “efficiency” is to be understood with respect to the number of operations or the time necessary for the execution of a program on a single-core architecture in comparison to a parallel approach, which allows for the distribution of tasks among the cores/nodes of a compute cluster for parallel execution including advanced strategies, e.g. to tackle load imbalances.

The main objective of this work is to develop a parallel shallow water code as well as computational strategies, thus enabling simulations with a large domain or with a large amount of high-resolution meshes, which cannot be treated with a sequential approach due to high computational cost. This objective is achieved through the following steps:

- **Profiling of different solvers with regard to their node-level performance**
  This point is the first important step towards the development of a parallel code due to the fact that the computations of the convective fluxes of the SWEs are the ones with the largest computational cost or time. Therefore, two common shallow water solvers (HLLC and CU) were investigated, in particular, to study their behavior with regard to vectorization supported by modern hardware, which can be considered the finest (and non-trivial) level of parallelism. Additionally, an alternative scheme (AV technique) was proposed.

- **Design of a flexible data structure to facilitate parallelization**
  At this step, a novel cell-edge reordering strategy was devised in order to facilitate the parallelization by computing internal edges (edges that belong to two cells) separately from the boundary ones (edges that belong only to one cell) because the former are generally more expensive than the latter ones. This separate-way-parallelization was intended to support a balanced load distribution among cores. In addition to parallelization, the cell-edge reordering strategy was also devised to enable vectorization for all solvers more efficiently, including the HLLC method, for which vectorization was not possible in the previous step.

- **Development of a load balancing strategy for wet-dry phenomena**
  In the existence of wet-dry problems, despite the proposed separate-way-parallelization, load imbalances among cores can still occur. This phenomenon is thus taken into careful consideration and a novel weighted-dynamic load balancing strategy is proposed. This technique was applied by reallocating the total number of edges and cells dynamically to cores during runtime so that the task distributions among cores exist periodically despite using static meshes. Since some procedures of this technique can only be performed

sequentially, it may not be efficient to apply the WDLB technique in every single time step.

- **Integration of a turbulence model for wet-dry problems**
  This step is considered because in reality flow-past-structure cases may appear together with wet-dry problems. For such cases — when the flow is turbulent — the mass, momentum, and heat transfers in water exist and fluctuate, which is the reason that the effects of the shear stresses become significant. An inviscid shallow water model is not able to capture the shear stresses, and a turbulence model has to be included. To this end, a novel approach was developed that combines the hydrostatic and topography reconstructions with scalable wall functions, enabling users to employ high-resolution meshes for wet-dry phenomena without having to estimate the wall friction velocity prior to a simulation.

- **Investigation of a solver's behavior for simulating recirculating turbulent flows**
  This issue is further addressed because not all solvers can model recirculating turbulent flows properly, while such flows often exist in flow-past-structure cases. A novel approach was developed for this reason, combining the AV technique with the CU scheme in order to simulate recirculating turbulent flows. The former was employed to solve the convective fluxes of the SWEs, while the latter one served to calculate the convective fluxes of the turbulence model. A comparison between the hybrid AV-CU, HLLC, and CU schemes highlights the advantages of the proposed approach.

- **Integration of space-filling curves for reordering unstructured meshes**
  While all implementations in the aforementioned steps utilize structured meshes, another data structure type was designed for the use of unstructured meshes with hybrid quadrilateral and triangular cells so that NUFSAW2D can allow for a wider range of implementations for shallow water flows. In order to support a flexible decomposition for unstructured meshes within the framework of hybrid OpenMP-MPI, edges and cells were always indexed sequentially, thus enabling a partition with a block-distribution. To provide the consecutive communication patterns among nodes and the contiguous memory access patterns inside each node, cells or edges were renumbered by means of SFCs. The block-distribution served to improve the implementation of the WDLB technique in the existence of wet-dry problems.

4. Summary of Results

4.1. Research Questions and Hypotheses

The research questions to be addressed in this work are formulated as:

1. How important is an investigation of the computational complexity of a solver's algorithm to support efficient vectorization on single-core architectures as a basic consideration prior to the development of a parallel code?
2. How can a data structure in an edge-based CCFV model be devised in a simple manner to enable efficient vectorization without any intrinsic function?
3. How necessary is a separate-way parallelization — which categorizes the task distribution based on the computational complexity of each component of a shallow water model — to ensure that a balanced load proportion is assigned to each core?
4. How significant are wet-dry phenomena in causing load imbalances among cores/nodes in a framework of a parallel shallow water model, and how would a load balancing strategy have to be designed to tackle such imbalances?
5. How can a two-equation turbulence model be robustly incorporated into a shallow water model to simulate flow-past-structure cases with wet-dry problems so that users can flexibly employ high-resolution meshes to increase the accuracy without having to estimate the wall friction velocity prior to running a simulation?
6. How would (common) shallow water solvers behave in terms of accuracy and efficiency in simulations of recirculating turbulent flows as one of the most common phenomena in flow-past-structure cases? Would, in addition to such common solvers, a more accurate but cheaper scheme be possible to model such flows?
7. What tactics should be used to automatically parallelize a domain with unstructured meshes?

Henceforth, it is important to point out that an efficient optimized executable of a shallow water code may strongly depend on the capabilities of the compiler. With regard to the compilers — especially for Fortran, e.g. GNU, NAG, Intel, Cray, IBM — it is well-known that each compiler has its specific advantages or disadvantages concerning different purposes such as debugging, compiling, generating executable files, etc. Secondly, it is also understood that applications based on the same code but created with different compilers on the same machine can result in different efficiency rates. Applications created using the same compiler on different machines can produce quite different efficiency rates as well. Considering these findings — although all executions in this work were compiled using Intel Fortran on Intel machines — the computational strategies proposed do not rule out the possibility to be applied widely by developers of shallow water codes to increase the efficiency of their models.

According to the aforementioned research questions, the following hypotheses were put forward and tested:

1. Before developing a parallel model, it is necessary to thoroughly investigate a solver's behavior regarding its efficiency on a single-core setup (**Paper 5**).
2. Vectorization is non-trivial in attempts to increase performance, and can be achieved without having to perform any intrinsic function. To this regard, a reordering strategy — that can provide contiguous array patterns for the data structure of edges and cells — has to be developed (**Paper 3**).
3. A separate-way-parallelization based on the computational complexity of the SWEs' components can support a balanced load distribution among cores. A contiguous indexing system between edges and cells is thus of great importance (**Paper 3**).
4. The existence of wet-dry phenomena in a parallel model causes remarkable load imbalances among cores, hence a proper load balancing strategy is inevitable (**Paper 1**).
5. A two-equation turbulence model can be robustly integrated into a shallow water model for wet-dry simulations by developing a strategy near wet-dry interfaces as well as near wall boundaries, thus avoiding the difficulty of the conventional method of estimating the wall friction velocity prior to a simulation (**Paper 2**).
6. Not all common solvers — despite their robustness in modeling the most complex flow phenomena such as transcritical flows, shock waves, and moving wet-dry fronts — are capable of simulating all flow-past-structure cases properly, e.g. recirculating turbulent flows. Therefore, a new hybrid solver should be considered (**Paper 4**).
7. For the use of unstructured meshes, regardless of the domain shape, a strategy can be applied to enable a parallelization by simply storing edges and cells (independent of one another) in 1D configuration, thus avoiding the difficult task of considering the domain decomposition prior to starting a simulation. Since edges and cells are stored independently, the index relationships between them must be determined and SFCs can thus be utilized (**Paper 6**).

4.2. Research Accomplishments

This section provides a brief summary of the findings obtained in the six aforementioned papers, which are included in the Appendix. The results of the six papers are introduced chronologically based on the aforementioned hypotheses.

4.2.1. Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography (Paper 5)

The objective of this paper was firstly to investigate the behavior of the common shallow water solvers (HLLC and CU) regarding their efficiency for the vectorization support of a desktop computer with a Haswell architecture (Intel Xeon E3-1200 v3). All simulations were performed using a single-core setup. As there was no special technique applied to the code

(e.g. intrinsic functions, restructuration of data layout, etc.), the main optimization feature was expected to come from the vectorization. Note that all implementations in this paper relied on auto vectorization (the one that utilizes the ability of the compiler to automatically detect loops, which have potential to be vectorized).

Three benchmark cases were presented including dam-break cases and urban flood simulation for the use of up to 0.1 million cells (0.2 million edges). In terms of accuracy, both HLLC and CU solvers showed accurate results with only non-significant differences, and the strong gradients of the very complex wave-wave interactions in the dam-break cases were captured accurately, see cases 1 and 2 in **Paper 5**. Furthermore, the two schemes also showed similar results in the urban flood case, and the moving wet-dry fronts were stably simulated without any issue, see case 3 in **Paper 5**, for which NUFSAW2D was also compared with other shallow water models, e.g. ISIS 2D, MIKE FLOOD, SOBEK, TUFLOW, and XPSTORM, and the results of the HLLC and CU solvers were within the range of the results of these models.

In terms of efficiency, however, the HLLC scheme was on average 30% more expensive than the CU solver. This was because the algorithm of the HLLC scheme is quite complex: it consists of several branch statements (`if-then-else`), of which the pattern of true-or-false condition cannot be predicted in advance by the branch prediction logic of the processor. In contrast, the CU scheme is free of any branching, thus making vectorization feasible.

In addition to investigating the above solvers, another scheme based on an AV technique, drawing on the previous work of [17] — that would be more accurate, but with acceptably low cost — was developed. The AV technique was devised from a combination of a Laplacian and a biharmonic operator, for which the scaling factor variable was constructed using the spectral radius of the Jacobian matrix. This combination does not need any eigen-decomposition of the Jacobian matrix as required by Riemann solvers, which brings the AV technique to a class of Riemann-solver-free schemes. The AV technique itself is still rarely used for shallow water flow applications. Extensive applications were presented within the scope of aeronautics, see [33; 32; 48; 63; 61] for some pioneer works.

As expected, the AV technique was in some particular cases more accurate than the HLLC and CU solvers. For example, in the simulation of the L-shaped dam-break, the AV technique was able to produce less diffusive results, despite using a first-order spatial scheme. This shows that the AV technique was more accurate to capture the strong gradients of the complex wave-wave interactions. In terms of efficiency, a comparison between the three schemes revealed that the AV technique was on average 22% cheaper than the HLLC method, while being almost as efficient as the CU scheme. The results in **Paper 5** have shown that the computational complexity of a solver's algorithm is worth being investigated as a mandatory step in order to optimize node-level efficiency before the next step (parallelization at the block/process-level) is addressed. Therefore, the first hypothesis can be confirmed.

4.2.2. Comparison of Shallow Water Solvers: Applications for Dam-Break and Tsunami Cases with Reordering Strategy for Efficient Vectorization on Modern Hardware (Paper 3)

The main objective of this paper was to utilize the vectorization support at the level of shared-memory systems for three solvers (Roe, HLLC, and CU) more efficiently, in particular, to enable vectorization for the HLLC method that was not possible in the previous implementation. Additionally, this paper pointed out an advantage of an edge-based CCFV parallelization with a classification between internal and boundary edges in supporting a balanced load distribution among cores. Both single-core and multi-core computations were performed on three clusters with modern hardware architecture: Intel Xeon E5-2690/Sandy-Bridge-E (AVX), Intel Xeon E5-2697 v3/Haswell (AVX2), and Intel Xeon Phi/Knights Landing (AVX-512), each with 16 cores, 28 cores, and 64 cores inside a node, respectively.

In order to investigate the efficiency, a performance metric for the edge-driven level was introduced, namely Medge/s/core (million edges per second per core), representing a comparison between the total number of simulated edges that can be achieved per unit of time using one core. The implementation was restricted to single-precision arithmetic for the purpose of emphasizing the effect of vector width. For single-core computations, this means that the theoretical speed-ups of the AVX/AVX2 and AVX-512 machines should be $8\times$ and $16\times$, respectively. Meanwhile, for multi-core computations, the theoretical speed-ups of the AVX, AVX2, and AVX-512 machines are a total value obtained from a multiplication of the vector width for one core by the total number of cores, thus being $128\times$, $192\times$, and $1024\times$, respectively.

In order to achieve the main objective above as well as to facilitate the parallelization on the aforementioned hardware, the cell-edge reordering strategy proposed in [23] was employed. The core idea of the cell-edge reordering strategy is to determine the relationship patterns between edges and cells (for the edge-driven loop) and between cells and edges (for the cell-driven loop) so that loops with a similar computational procedure can be collected and then vectorized. This strategy allows for separate-way-parallelization by distinguishing the computations of internal edges from those of boundary edges. This classification is highly beneficial to maintaining the efficiency of the code, as internal edges require more expensive computations (e.g. the calculation of a solver), while boundary edges do not need such a solver computation (if they act as a wall boundary). Therefore, the performance decrease of the parallelization — due to the high ratio of the total number between internal edges and the boundary ones, whenever the loop is carried out directly from one boundary edge to another boundary edge — can be avoided with this classification. The cell-edge reordering strategy provides contiguous array patterns between edges and cells (edge-driven level) as well as between cells and edges (cell-driven level)—and was employed in 1D array configuration, thus yielding straightforward memory access patterns, easing unit stride, and conserving cache entries. Unlike the implementation in **Paper 5** that relied on auto vectorization, all applications in this paper employed guided vectorization (the one that utilizes some compiler

hints/pragmas and array notations).

Four benchmark cases were tested including dam-break and tsunami flows for the use of up to 0.4 million cells (0.8 million edges). In terms of accuracy, the three solvers investigated were able to simulate all cases accurately, showing only non-significant differences. In terms of efficiency, however, the solvers exhibited quite significant differences in performance, see Tables 1 and 2, in which the speed-up factors of all solvers are summarized. One can see that by means of the cell-edge reordering strategy, vectorization is possible for all solvers, even for the HLLC scheme that previously could not be vectorized but now experiences tremendous speed-ups. It is also observed in **Paper 3** that the CU scheme was the most efficient solver, whereas the HLLC scheme still became the most inefficient one among the others. This strongly indicates that the HLLC solver suffers from additional overhead due to the natural formulation of its branching condition.

	AVX	AVX2	AVX-512
HLLC	5.5 (75.7)	4.5 (108.4)	16.68 (928.9)
Roe	6.5 (89.4)	4.8 (115.6)	16.04 (892.9)
CU	6 (83.52)	5 (121.8)	16.42 (924.7)

Table 1 : Relative speed-up of the HLLC, Roe, and CU schemes with vectorization on different machines; multi-core relative speed-up is shown inside the bracket [22]. Note that each speed-up factor is calculated based on a comparison with the non-vectorized single-core performance of each solver. Therefore, the higher speed-up factors achieved by the HLLC method do not mean that it is more efficient than the CU scheme. See **Paper 3** for example, the HLLC scheme with AVX-512 achieved 1.01 Medge/s/core (non-vectorized on one core), 16.83 Medge/s/core (vectorized on one core), 0.81 Medge/s/core (non-vectorized on 64 cores), and 14.64 Medge/s/core (vectorized on 64 cores). Meanwhile, the CU method with AVX-512 achieved 1.38 Medge/s/core (non-vectorized on one core), 22.70 Medge/s/core (vectorized on one core), 1.12 Medge/s/core (non-vectorized on 64 cores), and 19.98 Medge/s/core (vectorized on 64 cores).

	AVX	AVX2	AVX-512
HLLC	0.69 (9.5)	0.73 (17.48)	0.74 (41.28)
Roe	0.83 (11.42)	0.79 (19.01)	0.78 (43.31)
CU	1 (13.92)	1 (24.36)	1 (56.33)

Table 2 : Absolute speed-up of the HLLC, Roe, and CU schemes with vectorization on different machines; multi-core absolute speed-up is shown inside the bracket [22]. Note that speed-up factor is calculated based on a comparison with the vectorized single-core performance of the CU scheme as the best sequential algorithm among the others.

In this paper, it was concluded that the branching condition of the HLLC solver will be the main issue when trying to achieve an efficient performance, where the CU solver as a Riemann-solver-free scheme would generally be able to outperform the Riemann solvers (HLLC and Roe schemes) even for simulations on the next generation of modern hardware. This paper has shown that the vectorization was non-trivial for boosting the performance of all solvers and could be achieved without any intrinsic function, as performed in [4], or the restructuration of data layout from arrays of structs (AoS) to structs of arrays (SoA), as conducted in [13]. It can also be seen in **Paper 3** that the separate-way-parallelization — which classifies the SWEs' components based on their computational complexity — led NUFSAW2D to an average efficiency of up to 90%. Therefore, the second and third hypotheses are confirmed.

4.2.3. Parallel Flood Simulations for Wet-Dry Problems Using Dynamic Load Balancing Concept (Paper 1)

Although the separate-way-parallelization in the previous implementation was able to allow for a balanced load distribution among cores, load imbalances can still occur in the existence of wet-dry phenomena. This is caused by a special treatment of edges and cells in case of wet-dry problems, where wet edges/cells essentially lead to more computational effort than the dry ones, see Algorithms 1 and 3 in **Paper 1**. For example, at the edge-driven level, if a wet-dry or a dry-dry interface is detected, the model turns to a first-order spatial scheme, thus requiring no MUSCL technique, which is cheaper than a second-order spatial scheme. In contrast, when a wet-wet interface appears, the model may have either second-order spatial accuracy (no discontinuity) or first-order spatial accuracy (with discontinuity). A similar problem also occurs at the cell-driven level. Due to the semi-implicit treatment of the friction term, unit discharges must be transformed back to velocities with a division by a depth. If a wet cell is detected, no problem needs to be considered. However, if a dry cell is detected, a division by a very low (almost zero) depth may produce oscillations. During runtime the total number of wet and dry edges/cells is always uncertain, making a simulation with wet-dry phenomena more difficult to achieve a balanced load distribution among cores. For this reason, **Paper 1** focused on developing a simple strategy, namely the WDLB technique, which can balance the load among cores dynamically during runtime for the use of static meshes. Further, this technique must be integrated into the data structure designed within the framework of the cell-edge reordering strategy.

The first step in this paper is to find a ratio of CPU time between the computations of wet and dry edges as well as those of wet and dry cells. For edges, it is quite simple where NUFSAW2D was tested to simulate cases (using one core without a wet–dry problem) for hydraulic jump, flow over transition channel, and dam-break flows, e.g. cases 1–5 in [17]. It was observed that using solely a first-order spatial scheme was typically $2\times$ faster than using the second-order one. Further, for cells, NUFSAW2D was tested (using one core with a second-order spatial scheme) against cases with wet-dry phenomena, e.g. cases 1 and 2 in **Paper 5** as well as the two cases in **Paper 1**. Note that knowing the CPU time ratio between wet and dry cells is more difficult because the ratio of the total number between wet and dry cells always changes during runtime, thus showing a nonlinear relationship. Nevertheless, it could be observed that the CPU time for dry cells was typically $2\times$ faster than that for wet cells.

The WDLB technique is sketched in Fig. 6 and its procedure is briefly explained as follows. At a certain time step, it is assumed that cells 1–3 and 10–14 are wet and then weighted by a factor of 2, whereas the others by a factor of 1. Using static load balancing, all cells can be distributed equally to the four cores employed so that each core receives four cells. Despite receiving the same amount of cells, cores 0 and 2 in fact suffer from more overhead because they contain more wet cells; the load imbalance therefore exists. To overcome this issue, the WDLB is employed firstly by detecting the total load of each core, collecting and summing all

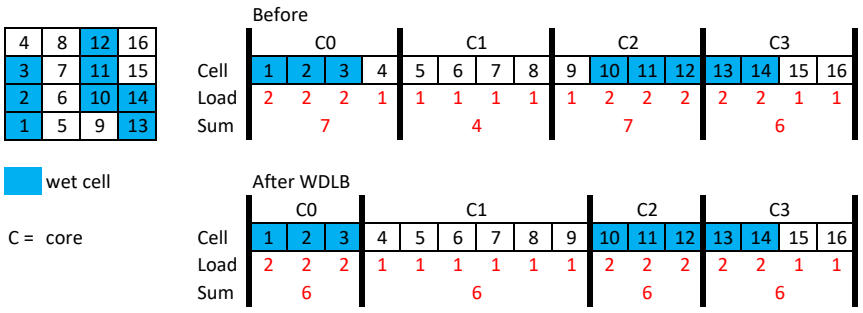


Figure 6 : Concept of the WDLB technique for wet-dry problems based on [23]

the load from the other cores, and then redistributing back the total amount of load equally to each core. The total amount of load is 24 units and, thus, each core must now receive 6 units. This translates to a condition where cores 0–3 obtain cells 1–3, 4–9, 10–12, and 13–16, respectively, and the load among cores turns out balanced. Note that only a sequential approach can be applied to redistribute the total amount of load back to each core. Therefore, it may not be efficient to apply the WDLB technique in every single time step. For this reason, the WDLB was applied to every 50^{th} total time step.

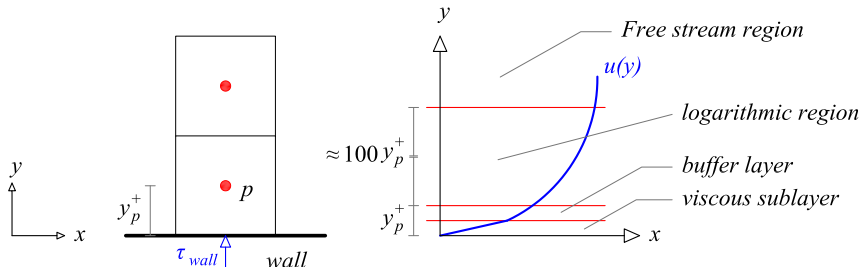
In this paper, the AVX and AVX-512 machines were employed to perform simulations with up to 6.4 million cells (12.8 million edges). It was observed that some cores still suffered from significant load imbalances using static load balancing, whereas the load was sufficiently balanced among cores using the WDLB technique, see Figures 14 and 15 in **Paper 1**. It was also found that the WDLB strategy was able to reduce the overhead of static load balancing by ratios of 19–20%, and led NUFSAW2D to parallel efficiencies of 88–97%. According to these results, the fourth hypothesis can be confirmed because in the existence of wet-dry phenomena, remarkable load imbalances occurred, which were successfully tackled by the WDLB strategy.

4.2.4. Central-Upwind Scheme for 2D Turbulent Shallow Flows using High-Resolution Meshes with Scalable Wall Functions (Paper 2)

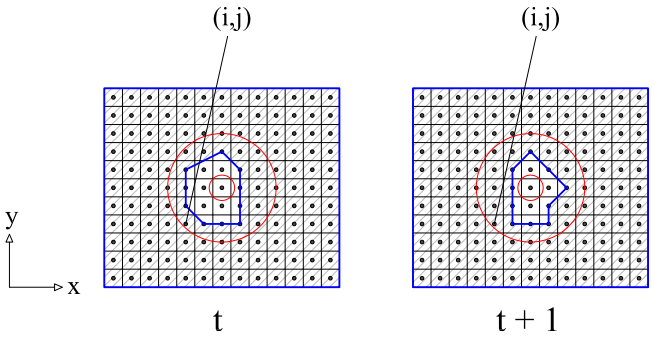
The objective of this paper was to include a two-equation turbulence model (ASM) into NUFSAW2D. This was because flow-past-structure phenomena together with moving wet-dry fronts often occur in a turbulent regime. Such phenomena cause mass, momentum, and heat transfers to exist and fluctuate with respect to time, which cannot be modeled properly by an inviscid shallow water model due to the neglect of the diffusive fluxes. Simulations in this paper were first motivated by a case (flow past a conical island in a surface-piercing condition) conducted both experimentally and numerically in [44], which concluded that the accuracy of the numerical model was not affected very much by reducing the mesh size. However, this statement should be argued and further investigated.

A presumption (of why such a statement was concluded) would be due to the impact of the wet–dry phenomena around the conical island on the wakes flows, for which, as this was

not of particular interest, no special treatment was explained in detail. Thus, a knowledge-gap — between wet-dry phenomena around boundary geometries (such as the ones that existed around the conical island) and the use of high-resolution meshes — must be found. In **Paper 2** it was argued that using high-resolution meshes can capture the wet-dry phenomena around the conical island better, and the flow properties, e.g. vortexes behind the island, will be modeled more accurately. Now, the problem concerns the task of determining the near-wall mesh size, which relates to the criterion of the dimensionless wall distance (y_p^+).



(a)



(b)

Figure 7 : Near-wall mesh and near-wall scaling of turbulent boundary layer (a) and wet-dry problems around a conical island from top view (b) [18]

In the conventional StWF, the criterion of $y_p^+ \geq 11.067$ indicates that the center of the first boundary cells must be placed in the logarithmic region, see Fig. 7a, to ensure the stability of the numerical model. This criterion may be contravened when using high-resolution meshes, unless a grid-independent study, such as performed in [35] has to be conducted in order to estimate the wall friction velocity before starting the simulation. However, such a study revolves situations of contradicting requirements to the discretization, in particular, if wet-dry problems appear. See Fig. 7b, for instance, at time step t, cell $(i, j+2)$ must be considered as a wall boundary cell, because its adjacent cell $(i+1, j+2)$ is computed as a dry cell. A proper function thus has to be employed to calculate the turbulent properties correctly. However, cell $(i, j+2)$ at time step $t+1$ cannot be treated as a wall boundary cell because its adjacent cell $(i+1, j+2)$ becomes a wet cell. Consequently, cell $(i+1, j+2)$ must be computed as a wall boundary cell. This shows that the moving boundary geometries appear with respect to time due to the existence of wet-dry problems. Here, a grid-independent study may not work as it is essentially employed for a condition of which the water level is relatively constant during runtime; in the presence of wet-dry phenomena, it is almost impossible to achieve a

constant water level for the whole simulation time. Note that even if the criterion of the StWF can somehow be fulfilled, the results will deteriorate as the meshes become finer and finer.

To remedy this problem, a novel approach was proposed by combining the hydrostatic and topography reconstructions [3] with the ScWF [50] so that the moving boundary geometries of the wet-dry phenomena can be captured robustly. At the same time, the turbulence values at the wet-dry interfaces (e.g. around the conical island) and at other wall boundaries can be computed properly. For moving boundary geometries like this, the ScWF are highly beneficial, allowing users to use high-resolution meshes without imposing the lower limit of the StWF. To prove the robustness of the proposed approach, four cases were tested against the model with amounts of up to 3.4 million cells (6.8 million edges). The simulations were performed on the AVX2 machine with OpenMP for parallelization.

It was observed that for the surface-piercing case investigated, wet-dry phenomena existed around the conical island, and simulations with cells from coarse (15 mm) to fine (2.5 mm) size could be performed stably without any grid-independent study or having to estimate the wall friction velocity in advance, where the results became better as the meshes turned finer and finer, see case 3 in **Paper 1**. The proposed strategy was also proven quite robust and accurate for the other cases, e.g. the turbulent dam-break cases with fluctuating wet-dry interfaces at the channel bank, see case 4 in **Paper 1**. Based on these results, the fifth hypothesis can be confirmed because the proposed approach served to robustly integrate a turbulence model into a shallow water model for wet-dry simulations, and it was possible to flexibly use high-resolution meshes without having to estimate the wall friction velocity before starting a simulation.

4.2.5. Hybrid Artificial Viscosity–Central-Upwind Scheme for Recirculating Turbulent Shallow Water Flows (Paper 4)

Despite its efficiency in the previous implementations, it was noticed that the CU scheme was not accurate enough for recirculating turbulent flows, unless high-resolution meshes were used. Albeit more accurate than the CU scheme, the HLLC solver sometimes failed to capture recirculating turbulent flows adequately. Therefore, **Paper 4** focused on the development of a new hybrid solver that can capture recirculating turbulent flows better than the common solvers—and several different approaches were thus investigated.

A first attempt was to employ the AV technique entirely for turbulence flows, namely both the convective fluxes of the SWEs and the turbulence model (κ-ϵ) were computed using this technique. However, the previously available expressions of the AV technique in [17] are not suitable for turbulence applications. Hypothetically, this is due to the fact that the AV technique only computes the spectral radius of the 3$\times$3 system Jacobian matrix of the 2D SWEs and a dimensionless depth-discontinuity sensor in order to fit the non-linearity of the SWEs, see, for instance, the forms of hu, huu, hvv, and huv in Eq. (2.2). Therefore, this technique does not match the non-linearity of the κ-ϵ model such as $h\kappa u$, $h\epsilon u$, $h\kappa v$, and $h\epsilon v$ in Eq. (2.4).

One possibility to solve this issue would be to consider the spectral radius of the 5×5 system Jacobian matrix simultaneously from the 2D SWEs and the κ-ϵ model. Even if this approach works, the overhead will increase significantly, thus losing the fundamental advantage of the AV technique itself in maintaining efficiency. To this end, a novel approach was proposed, based on computing the convective fluxes of the SWEs and the κ-ϵ model using the AV technique and the CU scheme, respectively. This new approach benefits from two points: (1) it resolves the inability/inaccuracy of the CU scheme in modeling recirculating turbulent flows and (2) it is still able to exploit compilers' optimizations on modern hardware, thus being efficient. Note that although in this approach the CU scheme has to be employed with the computation of the 5×5 system Jacobian matrix, the CPU time remains acceptably low because it is only used to calculate the convective fluxes of the κ-ϵ model.

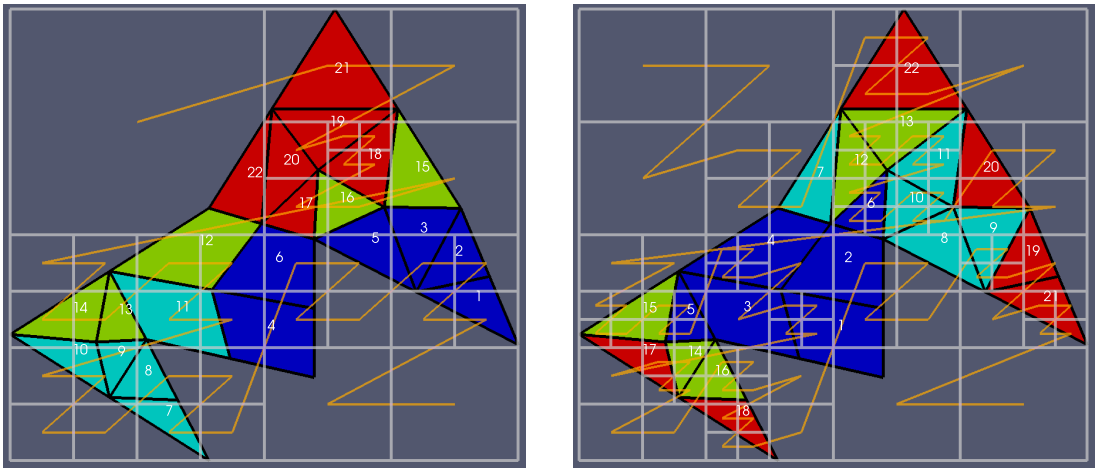
Similar to **Paper 2**, the ScWF were employed to avoid the limitation of the StWF, thus no grid study prior to a simulation was required such as the one performed with StarCCM+ in [26]. Three benchmark cases were included to compare the hybrid AV-CU scheme with the HLLC and CU solvers as well as with the other models in terms of accuracy and efficiency. OpenMP was used for parallelization on the AVX2 machine. In terms of accuracy, it was observed that the proposed hybrid AV-CU scheme could resolve the inaccuracy/inability of the CU scheme for simulations of recirculating turbulent shallow flows, and was remarkably more accurate than the HLLC scheme. The proposed scheme was able to significantly improve the results of the CU solver because the AV technique includes a biharmonic operator that is of third-order accuracy, activated in smooth flow-field regions. NUFSAW2D was also compared with two models (Lloyd & Stansby [45] and SCHISM-2D [70]), and became remarkably more accurate.

Regarding the efficiency, it was found that the hybrid AV-CU scheme remained as efficient as the CU scheme but could outperform the HLLC scheme by an average factor of 1.52. Thus, the proposed scheme is very promising for practical engineering purposes, especially when modeling recirculating turbulent flows. Based on these results, the sixth hypothesis can be confirmed because this paper showed that the common solvers were not always able to capture the recirculation in a turbulent regime, hence, a new hybrid solver was proposed.

4.2.6. Hybrid-Parallel Simulations and Visualisations of Real Flood and Tsunami Events using Unstructured Meshes on High-Performance Cluster Systems (Paper 6)

The objective of this paper was to extend the data structure in NUFSAW2D for applications with unstructured meshes (hybrid quadrilateral and triangular meshes) within the framework of hybrid OpenMP-MPI. This was motivated by the flexibility of unstructured meshes in handling the shape of complex domains that may not be achieved by (high-resolution) structured meshes. Also, this extension aimed to allow for a wider range of implementations with NUFSAW2D in shallow water flow modeling. The challenge was to design a strategy to parallelize domains with unstructured meshes without having to think of how the domain is decomposed

before starting a simulation. For this purpose, however, two important features of the existing format in NUFSAW2D — that have already been achieved for structured meshes — must be preserved: (1) the data structure of parallelizing a domain in two levels (edge-driven and cell-driven computations) according to the amount of cores/nodes used and (2) the applications of the WDLB technique for wet-dry problems.



(a) (b)

Figure 8 : Domain decomposition with the Lebesgue SFC: cell-reordering (a) and edge-reordering (b)

In order to achieve these two features, domains were decomposed employing SFCs, for which the goal is to provide a contiguous cell partition, thus giving contiguous communication patterns among the nodes. In this paper, the Lebesgue (Z-order) curve [54] was employed, the process of which is sketched in Fig. 8 for a domain with hybrid quadrilateral and triangular meshes that is supposed to be distributed into four nodes (denoted by four different colors). Two options are available: cell-reordering and edge-reordering. In the former approach, the process is started by renumbering the cells after the outermost boundary lines of the domain (bounding box) are detected. A similar quadtree-based voxelization process is also applied to the latter approach, but in order to capture a center of an edge before the Lebesgue curve can be formed. Note that the suitability of different reordering strategies for unstructured meshes depends on the particular problem, which is why one cannot rely merely on one strategy for domain decompositions, as this could lead to many different scenarios. Instead, a flexible data structure in NUFSAW2D was devised for the use of unstructured meshes in order to take various future applications into account.

In order to explain such a data structure, a domain with four cells, three internal edges, and eight boundary edges is given. Regardless of the domain shape and of the domain decomposition, the arrays for internal edges, boundary edges, and cells are always defined as 1D configuration, see STAGE 1 in Fig. 9a. In the next step, the user specifies the total number of nodes and threads intended, e.g. four nodes and two threads per node are used. Now, NUFSAW2D (by means of block-distribution) allocates the amount of arrays of internal edges, boundary edges, and cells into each node, see STAGE 2 in Fig. 9a. At this stage, the communication patterns among nodes and the memory access patterns inside each node still

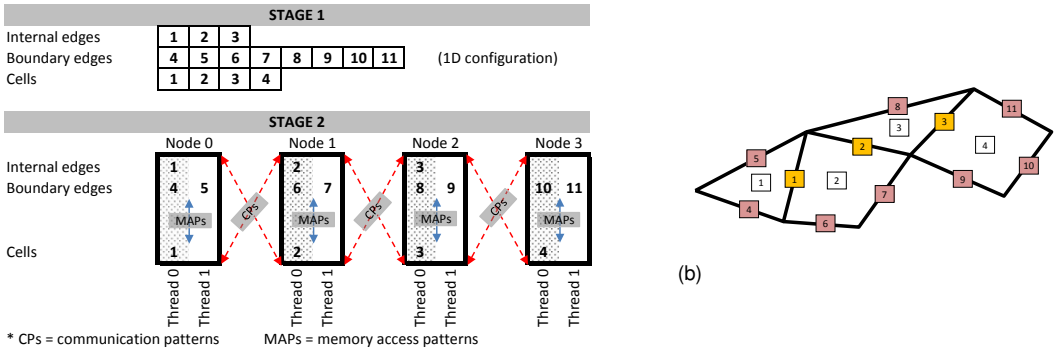


Figure 9 : Processes of design of data structure for (block-distribution) parallelization, communication patterns, and memory access patterns [25]

remain unknown. After this point, one can utilize SFCs or even any other suitable method (e.g. by using METIS [36]) to perform either cell-reordering or edge-reordering so that the actual indexing for the domain can be obtained, see for example Fig. 9b. Now, the communication patterns and memory access patterns can be determined, see Figure 5 in **Paper 6** for detail.

Due to the contiguous numbering, the WDLB technique in Fig. 6 can directly be extended to the data structure shown in Fig. 9a for the use of hybrid OpenMP-MPI. The procedure is similar. However, since the hybrid OpenMP-MPI is employed, each node must communicate with the others to compute the total load unit of the entire domain, once each node knows its load amount collected from all the corresponding threads. Afterwards, the reverse procedures must be carried out, namely the total load unit of the entire domain must be distributed back to each node and then to each thread. This process cannot be parallelized, which is why the WDLB technique is applied to every 50^{th} total time step for OpenMP similar to **Paper 1**, and every 100^{th} total time step for MPI.

Two cases of real river flood and tsunami simulations were carried out with up to 3.4 million cells (5.1 million edges). The AVX2 machine was employed. Here, NUFSAW2D has successfully maintained a good parallel speed-up of up to 305 for the use of 336 cores. This shows that the seventh hypothesis can be confirmed, as the proposed strategy led to a data structure that was flexible enough to support an independent and automatic parallelization between edges and cells.

5. Conclusions

In this work, the outcomes of the six papers have been briefly summarized. In the first investigation, one can see that it is quite important to profile the computational complexity with respect to a solver's node-level performance as a basic consideration prior to developing a parallel model. This statement is strongly supported by the fact that the investigated solvers showed differences in performance, especially with respect to the vectorization support from the compiler. It was observed that without any special technique (e.g. intrinsic functions, restructuration of data layout, etc.), auto vectorization was not possible for the HLLC method due to its branching formulas, thus making this method more expensive than the AV and CU schemes, for which auto vectorization was possible.

It was observed in the second investigation that the HLLC solver, for which vectorization was not possible in the previous implementation, could be vectorized by means of the cell-edge reordering strategy. This strategy plays an important role for the data structure to provide a contiguous memory access alignment and to exploit instruction pipelining, thus boosting the performance. Although the HLLC scheme has experienced a tremendous speed-up due to vectorization, its performance was still lower than the CU and Roe methods, indicating that the branching condition of the HLLC scheme will be the main issue when trying to achieve an efficient vectorization. In contrast, the CU scheme was observed to be able to outperform the HLLC and Roe methods even for simulations on the next generation of modern hardware. The cell-edge reordering strategy is one of the easiest approaches to exploit vectorization on modern hardware that can be applied to any CCFV shallow water model instead of explicitly using vector intrinsics for a low-level vectorization, which might be time-consuming and error-prone.

A separate-way-parallelization between internal and boundary edges — that distinguishes the former from the latter ones based on their computational complexity — was shown to be highly important to ensure that a balanced load proportion is assigned to each core. Because an internal edge typically requires more expensive computations than a boundary one, the high ratio between internal and boundary edges has a great influence on the parallel efficiency, thus parallelizing internal edges separately from the boundary ones is beneficial with regard to avoiding waiting time for one or more cores. The execution of the separate-way-parallelization on the shared-memory systems used was supported and eased by the cell-edge reordering strategy, which provides a contiguous indexing system between edges and cells. This led NUFSAW2D to a parallel efficiency of up to 90%.

In the third investigation, it was shown that wet-dry phenomena caused load imbalances among cores, in particular, at the edge-driven level among internal edges and at the cell-driven level for computing the friction terms. Such phenomena occur dynamically during runtime, which is why they cannot be predicted at the beginning of a simulation. According to

the numerical experiments, computations for dry edges at the edge-driven level were shown to be typically $2\times$ cheaper than those for the wet ones. Similarly, at the cell-driven level, computations for wet cells were observed to be typically $2\times$ more expensive than those for the dry ones. Using this ratio value, the WDLB was proposed and has been shown effective in tackling the load imbalances due to wet-dry problems by reducing up to 20% the overhead of static load balancing and by leading NUFSAW2D to an efficiency of up to 88%.

One can observe in the fourth investigation that an integration of a two-equation turbulence model into a shallow water model can be performed robustly in the scope of wet-dry modeling with high-resolution meshes if two main issues can be tackled, namely the dynamical-tracking of moving boundary geometries and the limitation of near-wall mesh spacing. The first issue relates to the dynamic wet-dry fronts due to which the positions of the wall boundary interfaces are not constant throughout simulations, while the second one corresponds to the criterion of the dimensionless wall distance in the StWF. As both issues in wet-dry modeling must be handled simultaneously — where using high-resolution meshes may destroy the second issue — a new approach was therefore proposed, based on combining the hydrostatic and topography reconstructions with the ScWF. With this approach, it was shown that the moving boundary geometries can be captured accurately, while the turbulence values both at wet-dry interfaces and at other wall boundaries can be properly estimated, even if high-resolution meshes are used, giving users flexibility in generating the meshes.

As one of the most common phenomena in flow-past-structure cases, recirculating turbulent flows are obviously of great importance, so a comprehensive investigation has to be conducted. In the fifth investigation, the numerical experiments have shown that — although common solvers were quite accurate in modeling the most complex and common phenomena in shallow flow modeling, such as transcritical flows, shock waves, and moving wet-dry fronts — such solvers were unable to properly simulate recirculating turbulent flows in all conditions. The CU scheme, despite its efficiency, was shown to be too diffusive. Albeit better than the CU scheme, it was observed that the HLLC method sometimes failed to capture the strong turbulent recirculation. A new approach (hybrid AV-CU scheme) was thus proposed and shown to be able to accurately capture recirculating turbulent flows in all simulations. The hybrid AV-CU scheme could resolve the inability/inaccuracy of the CU scheme because the AV technique includes a biharmonic operator that is of third-order accuracy, activated in smooth flow-field regions—and was more accurate than the HLLC solver. In terms of accuracy, this hybrid technique remained almost as efficient as the CU scheme but was approximately $1.52\times$ cheaper than the HLLC solver.

In order to avoid, prior to a simulation, a massive effort of decomposing a domain with unstructured meshes (for a hybrid OpenMP-MPI or an MPI parallelization), an according strategy was developed in the last investigation, based on storing edges and cells (independent of one another) in 1D configuration. Because edges and cells are stored in a consecutive manner, they can easily be parallelized by means of a block-distribution followed by a distribution

into cores/nodes. This tactic implies that regardless of the domain shape and of the domain decomposition, it is sufficient, prior to a simulation, only to know the total number of (internal and boundary) edges and cells for parallelization. Later on, after the distribution pattern of the parallelization has been identified, the communication patterns among nodes as well as the memory access patterns inside each node can be determined by means of SFCs. All strategies proposed have enabled a speed-up factor of up to 305 for the use of 336 cores.

Finally, one can draw a key conclusion that an “efficient” parallel simulation can intuitively be defined as a part of a modeling process that takes the efficiency issue into careful consideration at different levels of parallelization and performance optimization. The former can be ensured by tackling load imbalance problems, while the latter can be achieved by vectorization as the “finest” level of parallelism.

6. Outlook for Future Works

An obvious extension of the present work is a generalization to parallel non-hydrostatic computations of shallow water flows including wet-dry simulations. In [20], the applications of the AV technique were presented to simulate non-hydrostatic inviscid shallow flows by extending Eq. (2.1) (without the diffusive fluxes and the rainfall/infiltration term) to

$$\frac{\partial Q}{\partial t} + \frac{\partial C_x}{\partial x} + \frac{\partial C_y}{\partial y} = S_{bx} + S_{by} + S_f + S_p \,, \tag{6.1}$$

where S_p is the pressure term, and all the matrices are now defined by

$$Q = \begin{bmatrix} h \\ hu \\ hv \\ hw \end{bmatrix} \,, \quad C_x = \begin{bmatrix} hu \\ huu + \dfrac{gh^2}{2} \\ hvu \\ hwu \end{bmatrix} \,, \quad C_y = \begin{bmatrix} hv \\ huv \\ hvv + \dfrac{gh^2}{2} \\ hwv \end{bmatrix} \,, \quad S_{bx} = \begin{bmatrix} 0 \\ -gh\dfrac{\partial z_b}{\partial x} \\ 0 \\ 0 \end{bmatrix} \,,$$

$$S_{by} = \begin{bmatrix} 0 \\ 0 \\ -gh\dfrac{\partial z_b}{\partial y} \\ 0 \end{bmatrix} \,, \quad S_f = \begin{bmatrix} 0 \\ -c_f u\sqrt{u^2+v^2} \\ -c_f v\sqrt{u^2+v^2} \\ 0 \end{bmatrix} \,, \quad S_p = \begin{bmatrix} 0 \\ -\dfrac{1}{2}\dfrac{\partial h P_b}{\partial x} - P_b\dfrac{\partial z_b}{\partial x} \\ -\dfrac{1}{2}\dfrac{\partial h P_b}{\partial y} - P_b\dfrac{\partial z_b}{\partial y} \\ P_b \end{bmatrix} \,, \tag{6.2}$$

where w is the velocity in z direction and P_b is the non-hydrostatic pressure at bottom. The value of P_b must be computed iteratively by solving

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \,. \tag{6.3}$$

For Eq. (6.1), the computations in the x and y directions can first be carried out in the usual manner, similar to those of Eq. (2.1), so that the values of h, hu, and hv can be advanced. Thereafter, these values are corrected by the non-hydrostatic computations. The computations in z direction can be performed based on the Keller-box scheme [38] either in a 1-layer or a multi-layer way. In the case of 1-layer computation, as employed in [20], the convective fluxes on the fourth line of Eq. (6.2) vanished, thus letting the variable hw only be a function of P_b. For the sake of brevity, the entire formulations are not given here. Following a few mathematical operations, a system of linear equations is obtained, see [20], which can — for the use of rectangular meshes with the length in x and y directions Δx and Δy — be expressed as

$$K^1_{i,j} P_{b\,i,j}^{t+1} + K^2_{i,j} P_{b\,i+1,j}^{t+1} + K^3_{i,j} P_{b\,i-1,j}^{t+1} + K^4_{i,j} P_{b\,i,j+1}^{t+1} + K^5_{i,j} P_{b\,i,j-1}^{t+1} = K^6_{i,j} \,, \tag{6.4}$$

where K denotes the coefficients and follows $K = f(\eta, z_b, 1/h, hu, hv, w, \Delta x, \Delta y, \Delta t)$. Here, the indices (i, j) are used to denote the coordinates of cell-centers in a 2D array system, where i stands for the column index (left to right) and j for the row index (bottom to top).

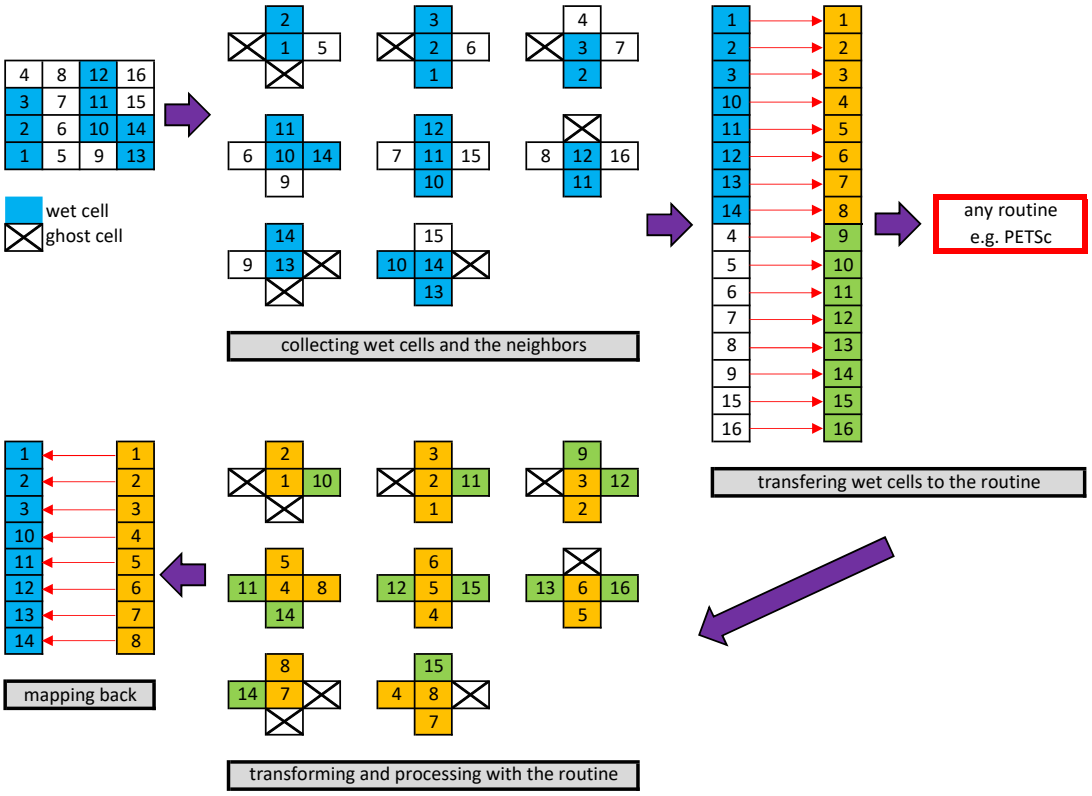


Figure 10 : Process of integrating an interface for a routine in non-hydrostatic computations

Considering a 4×4 domain in Fig. 10 (by assuming all wet cells), Eq. (6.4) shows a five-point stencil system that can be rewritten in matrix form as

$$\begin{bmatrix} K_1^1 & K_1^4 & 0 & 0 & K_1^2 & 0 & 0 & 0 & 0 & 0 & \ldots\ldots \\ K_2^5 & K_2^1 & K_2^4 & 0 & 0 & K_2^2 & 0 & 0 & 0 & 0 & \ldots\ldots \\ 0 & K_3^5 & K_3^1 & K_3^4 & 0 & 0 & K_3^2 & 0 & 0 & 0 & \ldots\ldots \\ 0 & 0 & K_4^5 & K_4^1 & 0 & 0 & 0 & K_4^2 & 0 & 0 & \ldots\ldots \\ K_5^3 & 0 & 0 & 0 & K_5^1 & K_5^4 & 0 & 0 & K_5^2 & 0 & \ldots\ldots \\ 0 & K_6^3 & 0 & 0 & K_6^5 & K_6^1 & K_6^4 & 0 & 0 & K_6^2 & \ldots\ldots \\ \ldots & & & & & & & & & & \end{bmatrix} \begin{bmatrix} P_{b1} \\ P_{b2} \\ P_{b3} \\ P_{b4} \\ P_{b5} \\ P_{b6} \\ \ldots \end{bmatrix} = \begin{bmatrix} K_1^6 \\ K_2^6 \\ K_3^6 \\ K_4^6 \\ K_5^6 \\ K_6^6 \\ \ldots \end{bmatrix} . \quad (6.5)$$

With all wet cells, Eq. (6.5) shows a 16×16 sparse matrix with a non-zero (main) diagonal and four sub-diagonals. In the scope of parallelization, it is necessary to not only consider the aspect of fast convergence but also simplicity for the solution of Eq. (6.5). There are several routines for highly-scalable parallelization available, e.g. PETSc [2], which, in addition to

scalable solvers, also provides scalable parallel preconditioners, and can easily be integrated into NUFSAW2D.

In cases with wet-dry problems, as $K^1_{\Omega_e} = f(1/h_{\Omega_e})$ in the main diagonal of Eq. (6.5), this linear equation system may have no solution, unless a proper strategy is considered such as restructuring the existing matrix to a new sparse matrix form. This is the new challenge, sketched in Fig. 10. In order to be able to fully exploit the high scalability provided by the routine PETSc, it is necessary to develop an interface that can translate the restructuration process of the existing matrix to the routine. Such an interface can be formed in two ways: the first one is to collect all wet cells within the domain and then transfer them into the routine, and the second one is to transform the results produced by the routine and then map them back to the actual domain. With regard to the wet-dry problems in Fig. 10, the 16$\times$16 matrix system is reduced to a 8$\times$8 system. In the course of this process, there is another challenging task to be considered, namely a trade-off between accuracy, CPU time, and computational stability, which relates to the first way when detecting and collecting all wet cells. The wet cells in this regard may be different from those computed in the edge-driven and cell-driven levels. For the computations in both of these levels, the wet cells are distinguished in the usual way by the limiter value of depth ranging from 10^{-6} to 10^{-10} m. However, employing this usual limiter value for non-hydrostatic computations can still lead to failures when trying to achieve convergence, hence a larger value has to be used. On the other hand, using a limiter value that is too large might reduce the accuracy significantly, despite reducing the CPU time. Therefore, this is another important aspect to be investigated.

Note that the non-hydrostatic computations have now become the most expensive part, significantly more expensive than the solver computations for the convective fluxes. Therefore, if the interface is not scalable enough, the efficient parallelization (which was already achieved using the WDLB technique) and the use of the highly-scalable routine will be useless. In summary, two topics are pointed out that would be very interesting for future works: (1) to investigate the proper value range for transferring wet cells to the routine that can satisfy the three aspects of accuracy, CPU time, and computational stability; and (2) to develop a scalable interface that can fully support and exploit the capability of the routine. Further, it could be of interest for future works to investigate these two aspects with a specific focus on turbulence simulations.

Bibliography

[1] https://ec.europa.eu/digital-single-market/en/policies/high-performance-computing. Accessed: May 2019.

[2] https://www.mcs.anl.gov/petsc. Accessed: May 2019.

[3] E. Audusse, F. Bouchut, M. Bristeau, R. Klein, and B. Perthame. A Fast and Stable Well-Balanced Scheme with Hydrostatic Reconstruction for Shallow Water Flows. *SIAM Journal on Scientific Computing*, 25(6):2050–2065, 2004. https://dx.doi.org/10.1137/S1064827503431090.

[4] M. Bader, A. Breuer, W. Hölzl, and S. Rettenberger. Vectorization of an Augmented Riemann Solver for the Shallow Water Equations. In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 193–201. IEEE, 2014. https://dx.doi.org/10.1109/HPCSim.2014.6903686.

[5] A. Bik, M. Girkar, P.M. Grey, and X. Tian. Automatic Intra-Register Vectorization for the Intel Architecture. *International Journal of Parallel Programming*, 2:65–98, 2002. https://dx.doi.org/10.1023/A:1014230429447.

[6] A. Bollermann, G. Chen, A. Kurganov, and S. Noelle. A Well-Balanced Reconstruction of Wet/Dry Fronts for the Shallow Water Equations. *Journal of Scientific Computing*, 56 (2):267–290, 2013. https://dx.doi.org/10.1007/s10915-012-9677-5.

[7] L. Cea, J. Puertas, and M.-E. Vázquez-Cendón. Depth Averaged Modelling of Turbulent Shallow Water Flow with Wet-Dry Fronts. *Archives of Computational Methods in Engineering*, 14(3):303–341, 2007. https://dx.doi.org/10.1007/s11831-007-9009-3.

[8] Y. Cheng, A. Chertock, M. Herty, A. Kurganov, and T. Wu. A New Approach for Designing Moving-Water Equilibria Preserving Schemes for the Shallow Water Equations. *Journal of Scientific Computing*, 80(1):538–554, 2019. https://dx.doi.org/10.1007/s10915-019-00947-w.

[9] A. Chertock, S. Cui, A. Kurganov, and T. Wu. Well-balanced Positivity Preserving Central-upwind Scheme for the Shallow Water System with Friction Terms. *International Journal for Numerical Methods in Fluids*, 78(6):355–383, 2015. https://dx.doi.org/10.1002/fld.4023.

[10] R. Comblen, S. Legrand, E. Deleersnijder, and V. Legat. A Finite Element Method for Solving the Shallow Water Equations on the Sphere. *Ocean Modelling*, 28(1):12–23, 2009. https://dx.doi.org/10.1016/j.ocemod.2008.05.004.

[11] A.I. Delis and I.K. Nikolos. A Novel Multidimensional Solution Reconstruction and Edge-based Limiting Procedure for Unstructured Cell-centered Finite Volumes with Application to Shallow Water Dynamics. *International Journal for Numerical Methods in Fluids*, 71 (5):584–633, 2013. https://dx.doi.org/10.1002/fld.3674.

[12] European Parliament and Council. Directive 2007/60/EC of the European Parliament and of the Council of 23 October 2007 on the Assessment and Management of Flood Risks. 2007.

[13] C. Ferreira, K. Mandli, and M. Bader. Vectorization of Riemann Solvers for the Single- and Multi-layer Shallow Water Equations. In *2018 International Conference on High Performance Computing Simulation (HPCS)*, pages 415–422. IEEE, 2018. https://dx.doi.org/10.1109/HPCS.2018.00073.

[14] M. Fišer, I. Özgen, R. Hinkelmann, and J. Vimmr. A Mass Conservative Well-balanced Reconstruction at Wet/Dry Interfaces for the Godunov-type Shallow Water Model. *International Journal for Numerical Methods in Fluids*, 82:893–908, 2016. https://dx.doi.org/10.1002/fld.4246.

[15] M. Fujihara and A.G.L. Borthwick. Godunov-Type Solution of Curvilinear Shallow-Water Equations. *Journal of Hydraulic Engineering (ASCE)*, 126(11):827–836, 2000. https://dx.doi.org/10.1061/(ASCE)0733-9429(2000)126:11(827).

[16] J.M. Gallardo, C. Parés, and M. Castro. On a Well-balanced High-order Finite Volume Scheme for Shallow Water Equations with Topography and Dry Areas. *Journal of Computational Physics*, 227(1):574–601, 2007. https://dx.doi.org/10.1016/j.jcp.2007.08.007.

[17] B.M. Ginting. A Two-dimensional Artificial Viscosity Technique for Modelling Discontinuity in Shallow Water Flows. *Applied Mathematical Modelling*, 45:653–683, 2017. https://dx.doi.org/10.1016/j.apm.2017.01.013.

[18] B.M. Ginting. Central-Upwind Scheme for 2D Turbulent Shallow Flows using High-Resolution Meshes with Scalable Wall Functions. *Computers & Fluids*, 179:394–421, 2019. https://dx.doi.org/10.1016/j.compfluid.2018.11.014.

[19] B.M. Ginting and H. Ginting. Hybrid Artificial Viscosity–Central-Upwind Scheme for Recirculating Turbulent Shallow Water Flows. *Journal of Hydraulic Engineering (ASCE)*, 145(12):04019041, 2019. https://dx.doi.org/10.1061/(ASCE)HY.1943-7900.0001639.

[20] B.M. Ginting and H. Ginting. Extension of Artificial Viscosity Technique for Solving 2D Non-Hydrostatic Shallow Water Equations. *European Journal of Mechanics B/Fluids, under review*, 2019.

[21] B.M. Ginting and R.-P. Mundani. Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography. In P. Gourbesville, J. Cunge, and G. Caignaert, editors, *Advances in Hydroinformatics*, chapter 4, pages 51–74. Springer Water, Springer, Singapore, 2018. https://dx.doi.org/10.1007/978-981-10-7218-5_4.

[22] B.M. Ginting and R.-P. Mundani. Comparison of Shallow Water Solvers: Applications for Dam-break and Tsunami Cases with Reordering Strategy for Efficient Vectorization on Modern Hardware. *Water*, 11(4):639, 2019. https://dx.doi.org/10.3390/w11040639.

[23] B.M Ginting and R.-P. Mundani. Parallel Flood Simulations for Wet-Dry Problems Using Dynamic Load Balancing Concept. *Journal of Computing in Civil Engineering (ASCE)*, 33(3):04019013, 2019. https://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000823.

[24] B.M. Ginting, R.-P. Mundani, and E. Rank. Parallel Simulations of Shallow Water Solvers for Modelling Overland Flows. In G. La Loggia, G. Freni, V. Puleo, and M. De Marchis, editors, *13th International Conference on Hydroinformatics*, volume 3 of *EPiC Series in Engineering*, pages 788–799. EasyChair, 2018. https://dx.doi.org/10.29007/wdn8.

[25] B.M. Ginting, P.K. Bhola, C. Ertl, R.-P. Mundani, M. Disse, and E. Rank. Hybrid-Parallel Simulations and Visualisations of Real Flood and Tsunami Events using Unstructured Meshes on High-Performance Cluster Systems. In *Advances in Hydroinformatics*. Springer, *accepted*, 2019.

[26] L. Han, E. Mignot, and N. Riviere. Shallow Mixing Layer Downstream from a Sudden Expansion. *Journal of Hydraulic Engineering (ASCE)*, 143(5):04016105, 2017. https://dx.doi.org/10.1061/(ASCE)HY.1943-7900.0001274.

[27] A. Harten, P. Lax, and B. Leer. On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws. *SIAM Review*, 25(1):35–61, 1983. https://dx.doi.org/10.1137/1025002.

[28] C. Hirsch. *Numerical Computation of Internal and External Flows*. Elsevier, 2007.

[29] Z. Horváth, J. Waser, R.A.P. Perdigão, A. Konev, and G. Blöschl. A Two-dimensional Numerical Scheme of Dry/Wet Fronts for the Saint-Venant System of Shallow Water Equations. *International Journal for Numerical Methods in Fluids*, 77(3):159–182, 2015. https://dx.doi.org/10.1002/fld.3983.

[30] J. Hou, Q. Liang, F. Simons, and R. Hinkelmann. A 2D Well-balanced Shallow Flow Model for Unstructured Grids with Novel Slope Source Term Treatment. *Advances in Water Resources*, 52:107–131, 2013. https://dx.doi.org/10.1016/j.advwatres.2012.08.003.

[31] J. Hou, F. Simons, M. Mahgoub, and R. Hinkelmann. A Robust Well-balanced Model on Unstructured Grids for Shallow Water Flows with Wetting and Drying over Complex Topography. *Computer Methods in Applied Mechanics and Engineering*, 257:126–149, 2013. https://dx.doi.org/10.1016/j.cma.2013.01.015.

[32] A. Jameson and D. Mavriplis. Finite Volume Solution of the Two-dimensional Euler Equations on a Regular Triangular Mesh. *AIAA Journal*, 24(4):611–618, 1986. https://dx.doi.org/10.2514/3.9315.

[33] A. Jameson, W. Schmidt, and E. Turkel. Numerical Solution of the Euler Equations by Finite Volume Methods using Runge Kutta Time Stepping Schemes. In *14th Fluid and Plasma Dynamics Conference*. 1981. https://dx.doi.org/10.2514/6.1981-1259.

[34] S. Jin and X. Wen. Two Interface-Type Numerical Methods for Computing Hyperbolic Systems with Geometrical Source Terms Having Concentrations. *SIAM Journal on Scientific Computing*, 26(6):2079–2101, 2005. https://dx.doi.org/10.1137/040605825.

[35] M. Karimi, G. Akdogan, and S.M. Bradshaw. Effects of Different Mesh Schemes and Turbulence Models in CFD Modelling of Stirred Tanks. *Physicochemical Problems of Mineral Processing*, 48(2):513–531, 2012. https://dx.doi.org/10.5277/ppmp120216.

[36] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. https://dx.doi.org/10.1137/S1064827595287997.

[37] G. Kasserwani and Q. Liang. Well-balanced RKDG2 Solutions to the Shallow Water Equations over Irregular Domains with Wetting and Drying. *Computers & Fluids*, 39(10):2040–2050, 2010. https://dx.doi.org/10.1016/j.compfluid.2010.07.008.

[38] H.B. Keller. A New Difference Scheme for Parabolic Problems. In B. HUbbard, editor, *Numerical Solutions of Partial Differential Equations II*, pages 327–350. Academic Press, New York, USA, 1971. https://dx.doi.org/10.1016/B978-0-12-358502-8.50014-1.

[39] A. Kurganov and G. Petrova. A Second-Order Well-Balanced Positivity Preserving Central-Upwind Scheme for the Saint-Venant System. *Communications in Mathematical Sciences*, 5(1):133–160, 2007. https://projecteuclid.org:443/euclid.cms/1175797625.

[40] A. Lacasta, M. Morales-Hernández, J. Murillo, and P. García-Navarro. GPU Implementation of the 2D Shallow Water Equations for the Simulation of Rainfall/Runoff Events. *Environmental Earth Sciences*, 74(11):7295–7305, 2015. https://dx.doi.org/10.1007/s12665-015-4215-z.

[41] Q. Liang. A Simplified Adaptive Cartesian Grid System for Solving the 2D Shallow Water Equations. *International Journal for Numerical Methods in Fluids*, 69(2):442–458, 2012. https://dx.doi.org/10.1002/fld.2568.

[42] Q. Liang, J. Hou, and X. Xia. Contradiction between the C-property and Mass Conservation in Adaptive Grid Based Shallow Flow Models: Cause and Solution. *International Journal for Numerical Methods in Fluids*, 78(1):17–36, 2015. https://dx.doi.org/10.1002/fld.4005.

[43] J.-Y. Liu, M.R. Smith, F.-A. Kuo, and J.-S. Wu. Hybrid OpenMP/AVX Acceleration of a Split HLL Finite Volume Method for the Shallow Water and Euler Equations. *Computers & Fluids*, 110:181–188, 2015. https://dx.doi.org/10.1016/j.compfluid.2014.11.011.

[44] P.M. Lloyd and P.K. Stansby. Shallow-Water Flow around Model Conical Islands of Small Side Slope. I: Surface Piercing. *Journal of Hydraulic Engineering (ASCE)*, 123(12):1057–1067, 1997. https://dx.doi.org/10.1061/(ASCE)0733-9429(1997)123:12(1057).

[45] P.M. Lloyd and P.K. Stansby. Shallow-Water Flow around Model Conical Islands of Small Side Slope. II: Submerged. *Journal of Hydraulic Engineering (ASCE)*, 123(12):1068–1077, 1997. https://dx.doi.org/10.1061/(ASCE)0733-9429(1997)123:12(1068).

[46] J. Lobeiras, M. Viñas, M. Amor, B.B. Fraguela, M. Arenaz, J.A. García, and M.J. Castro. Parallelization of Shallow Water Simulations on Current Multi-threaded Systems. *The International Journal of High Performance Computing Applications*, 27(4):493–512, 2013. https://dx.doi.org/10.1177/1094342012464800.

[47] P.A. Madsen, H.J. Simonsen, and C.-H. Pan. Numerical Simulation of Tidal Bores and Hydraulic Jumps. *Coastal Engineering*, 52(5):409–433, 2005. https://dx.doi.org/10.1016/j.coastaleng.2004.12.007.

[48] D. Mavriplis. Multigrid Solution of the Two-dimensional Euler Equations on Unstructured Triangular Meshes. *AIAA Journal*, 26(7):824–831. https://dx.doi.org/10.2514/3.9975.

[49] O. Meister, K. Rahnema, and M. Bader. Parallel Memory-Efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells. *ACM Transactions on Mathematical Software*, 43(3):19:1–19:27, 2016. https://dx.doi.org/10.1145/2947668.

[50] F. Menter and T. Esch. Elements of Industrial Heat Transfers Predictions. In *16th Brazilian Congress of Mechanical Engineering*, 2001.

[51] V. Michel-Dansac, C. Berthon, S. Clain, and F. Foucher. A Well-balanced Scheme for the Shallow-water Equations with Topography. *Computers & Mathematics with Applications*, 72(3):568–593, 2016. https://dx.doi.org/10.1016/j.camwa.2016.05.015.

[52] F. Mintgen and M. Manhart. A Bi-directional Coupling of 2D Shallow Water and 3D Reynolds-averaged Navier–Stokes Models. *Journal of Hydraulic Research*, 56(6):771–785, 2016. https://dx.doi.org/10.1080/00221686.2017.1419989.

[53] A. Mohamadian, D.Y. Le Roux, M. Tajrishi, and K. Mazaheri. A Mass Conservative Scheme for Simulating Shallow Flows over Variable Topographies using Unstructured Grid. *Advances in Water Resources*, 28(5):523–539, 2005. https://dx.doi.org/10.1016/j.advwatres.2004.10.006.

[54] G.M. Morton. A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. In *Technical Report*. IBM Ltd, Ottawa, Canada, 1966.

[55] J. Neal, T. Fewtrell, and M. Trigg. Parallelisation of Storage Cell Flood Models using OpenMP. *Environmental Modelling & Software*, 24(7):872–877, 2009. https://dx.doi.org/10.1016/j.envsoft.2008.12.004.

[56] D. Nuzman, I. Rosen, and A. Zaks. Auto-vectorization of Interleaved Data for SIMD. In *Proceedings of the 13th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 132–143. ACM, 2006. https://dx.doi.org/10.1145/1133981.1133997.

[57] I. Özgen, J. Zhao, D. Liang, and R. Hinkelmann. Urban Flood Modeling using Shallow Water Equations with Depth-dependent Anisotropic Porosity. *Journal of Hydrology*, 541: 1165–1184, 2016. https://dx.doi.org/10.1016/j.jhydrol.2016.08.025.

[58] W. Rodi. *Turbulence Models and Their Application in Hydraulics: A State-of-the Art Review*. A.A. Balkema, 1993.

[59] P.L. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, 135(2):250–258, 1997. https://dx.doi.org/10.1006/jcph.1997.5705.

[60] B.F. Sanders, J.E. Schubert, and R.L. Detwiler. Parbrezo: A Parallel, Unstructured Grid, Godunov-type, Shallow-water Code for High-resolution Flood Inundation Modeling at the Regional Scale. *Advances in Water Resources*, 33(12):1456–1467, 2010. https://dx.doi.org/10.1016/j.advwatres.2010.07.007.

[61] R.C. Swanson, R. Radespiel, and E. Turkel. On Some Numerical Dissipation Schemes. *Journal of Computational Physics*, 147(2):518–544, 1998. https://dx.doi.org/10.1006/jcph.1998.6100.

[62] E. Toro. *Shock-Capturing Methods for Free-Surface Shallow Flow*. John Wiley: Chichester, UK, 2001.

[63] J.W. Van Der Burg, J.G.M. Kuerten, and P.J. Zandbergen. Improved Shock-capturing of Jameson's Scheme for the Euler Equations. *International Journal for Numerical Methods in Fluids*, 15(6):649–671, 1992. https://dx.doi.org/10.1002/fld.1650150603.

[64] H.K Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method (second edition)*. Pearson Education Limited: Harlow, UK, 2007.

[65] R. Wittmann, H.-J. Bungartz, and P. Neumann. High Performance Shallow Water Kernels for Parallel Overland Flow Simulations based on FullSWOF2D. *Computers & Mathematics with Applications*, 74(1):110–125, 2017. https://dx.doi.org/10.1016/j.camwa.2017.01.005.

[66] W. Wu. Depth-Averaged Two-Dimensional Numerical Modeling of Unsteady Flow and Nonuniform Sediment Transport in Open Channels. *Journal of Hydraulic Engineering (ASCE)*, 130(10):1013–1024, 2004. https://dx.doi.org/10.1061/(ASCE)0733-9429(2004)130:10(1013).

[67] X. Xia and Q. Liang. A New Efficient Implicit Scheme for Discretising the Stiff Friction Terms in the Shallow Water Equations. *Advances in Water Resources*, 117:87–97, 2018. https://dx.doi.org/10.1016/j.advwatres.2018.05.004.

[68] C. Yu and J. Duan. Two-dimensional Depth-averaged Finite Volume Model for Unsteady Turbulent Flow. *Journal of Hydraulic Research*, 50(6):599–611, 2012. https://dx.doi.org/10.1080/00221686.2012.730556.

[69] S. Zhang, Z. Xia, R. Yuan, and X. Jiang. Parallel Computation of a Dam-break Flow Model using OpenMP on a Multi-core Computer. *Journal of Hydrology*, 512:126–133, 2014. https://dx.doi.org/10.1016/j.jhydrol.2014.02.035.

[70] Y.J. Zhang, G. Priest, J. Allan, and L. Stimely. Benchmarking An Unstructured-grid Model for Tsunami Current Modeling. *Pure and Applied Geophysics*, 173(12):4075–4087, 2016. https://dx.doi.org/10.1007/s00024-016-1328-6.

[71] J. Zhao, I. Özgen, D. Liang, and R. Hinkelmann. Improved Multislope MUSCL Reconstruction on Unstructured Grid for Shallow Water Equations. *International Journal for Numerical Methods in Fluids*, 87:401–436, 2018. https://dx.doi.org/10.1002/fld.4499.

[72] J. Zhao, I. Özgen-Xian, D. Liang, T. Wang, and R. Hinkelmann. An Improved Multislope MUSCL Scheme for Solving Shallow Water Equations on Unstructured Grids. *Computers and Mathematics with Applications*, 77:576–596, 2019. https://dx.doi.org/10.1016/j.camwa.2018.09.059.

[73] J.G. Zhou, D.M. Causon, D.M. Ingram, and C.G. Mingham. Numerical Solutions of The Shallow Water Equations with Discontinuous Bed Topography. *International Journal for Numerical Methods in Fluids*, 38(8):769–788, 2002. https://dx.doi.org/10.1002/fld.243.

APPENDIX
(Permission)

RE: Question about reusing papers (full part) for my dissertation

PERMISSIONS <permissions@asce.org>

Wed 7/17/2019 8:24 PM

To: Ginting, Bobby <bobbyminola.ginting@tum.de>;

Dear Bobby,
Thank you for your inquiry. As an original author of an ASCE journal article paper, you are permitted to reuse your own content (including figures and tables) for another ASCE or non-ASCE publication, provided it does not account for more than 25% of the new work. This email serves as permission to reuse your work in your thesis. Please make sure that the new work does in fact account for 75% new content.

A full credit line must be added to the material being reprinted. For reuse in non-ASCE publications, add the words "With permission from ASCE" to your source citation. For Intranet posting, add the following additional notice: "This material may be downloaded for personal use only. Any other use requires prior permission of the American Society of Civil Engineers. This material may be found at [URL/link of abstract in the ASCE Library or Civil Engineering Database]."

Each license is unique, covering only the terms and conditions specified in it. Even if you have obtained a license for certain ASCE copyrighted content, you will need to obtain another license if you plan to reuse that content outside the terms of the existing license. For example: If you already have a license to reuse a figure in a journal, you still need a new license to use the same figure in a magazine. You need a separate license for each edition.

For more information on how an author may reuse their own material, please view:
http://ascelibrary.org/page/informationforasceauthorsreusingyourownmaterial

Sincerely,

Leslie Connelly
Senior Marketing Coordinator
American Society of Civil Engineers
1801 Alexander Bell Drive
Reston, VA 20191

PERMISSIONS@asce.org

703-295-6169
Internet: www.asce.org/pubs | www.ascelibrary.org | http://ascelibrary.org/page/rightsrequests

A full credit line must be added to the material being reprinted. For reuse in non-ASCE publications, add the words "With permission from ASCE" to your source citation. For Intranet posting, add the following additional notice: "This material may be downloaded for personal use only. Any other use requires prior permission of the American Society of Civil Engineers. This material may be found at [URL/link of abstract in the ASCE Library or Civil Engineering Database]."

From: Ginting, Bobby <bobbyminola.ginting@tum.de>
Sent: Wednesday, July 17, 2019 9:03 AM
To: PERMISSIONS <permissions@asce.org>
Subject: Question about reusing papers (full part) for my dissertation

Dear Sir/Madam,

My name is Bobby Minola Ginting and I am the author of the papers entitled:

1. ***Parallel Flood Simulations for Wet-Dry Problems Using Dynamic Load Balancing Concept***, published by **Journal of Computing in Civil Engineering, Vol. 33(3), 2019**. DOI: 10.1061/(ASCE)CP.1943-5487.0000823.
2. ***Hybrid Artificial Viscosity–Central-Upwind Scheme for Recirculating Turbulent Shallow Water Flows***, (to be) published by **Journal of Hydraulic Engineering, (IN PRESS), 2019**. DOI: 10.1061/(ASCE)HY.1943-7900.0001639.

I would like to include this paper (**full part**) into my dissertation that will be published **electronically** by our library. Therefore, I have some questions:

1. Should I obtain an official permission first from you (the publisher)?
2. Am I allowed to include the paper with the ASCE's published version format or do I have to reformat it, e.g. by using my own layout format (but with **full content**)?

I look forward to your answer.

Many thanks.

Best wishes,
Bobby Minola Ginting

Title: Central-upwind scheme for 2D turbulent shallow flows using high-resolution meshes with scalable wall functions

Author: Bobby Minola Ginting

Publication: Computers & Fluids

Publisher: Elsevier

Date: 30 January 2019

RE: Question about reusing paper for my dissertation

Permissions Helpdesk <permissionshelpdesk@elsevier.com>

Thu 7/18/2019 12:57 PM

To: Ginting, Bobby <bobbyminola.ginting@tum.de>;

Dear Dr. Bobby Minola Ginting,

We hereby grant you permission to reprint the material below at no charge **in your thesis** subject to the following conditions:

1. If any part of the material to be used (for example, figures) has appeared in our publication with credit or acknowledgement to another source, permission must also be sought from that source. If such permission is not obtained then that material may not be included in your publication/copies.

2. Suitable acknowledgment to the source must be made, either as a footnote or in a reference list at the end of your publication, as follows:
"This article was published in Publication title, Vol number, Author(s), Title of article, Page Nos, Copyright Elsevier (or appropriate Society name) (Year)."

3. Your thesis may be submitted to your institution in either print or electronic form.

4. Reproduction of this material is confined to the purpose for which permission is hereby given

5. This permission is granted for non-exclusive world **English** rights only. For other languages please reapply separately for each one required. Permission excludes use in an electronic form other than submission. Should you have a

specific electronic project in mind please reapply for permission.

6. Should your thesis be published commercially, please reapply for permission.

This includes permission for the Library and Archives of Canada to supply single copies, on demand, of the complete thesis. Should your thesis be published commercially, please reapply for permission- Canada
This includes permission for UMI to supply single copies, on demand, of the complete thesis. Should your thesis be published commercially, please reapply for permission-ROW

7. Posting of the full article online is not permitted. You may post an abstract with a link to the Elsevier website www.elsevier.com , or to the article on ScienceDirect if it is available on that platform.

8. Article can used be in the University library if it is embedded in the thesis and not used commercially.

Permissions Helpdesk
ELSEVIER |Operations
permissionshelpdesk@elsevier.com

From: Ginting, Bobby <bobbyminola.ginting@tum.de>
Sent: Wednesday, July 17, 2019 4:23 PM
To: Permissions Helpdesk <permissionshelpdesk@elsevier.com>
Subject: Question about reusing paper for my dissertation

Dear Sir/Madam,

My name is Bobby Minola Ginting and I am the author of the paper entitled "Central-Upwind Scheme for 2D Turbulent Shallow Flows using High-Resolution Meshes with

Scalable Wall Functions" published by Computers & Fluids, Vol. 179, 2019, pp. 394-421. DOI: 10.1016/j.compfluid.2018.11.014.

I would like to include this paper (**full part**) into my dissertation that will be published electronically by our library. Therefore, I have some questions:

1. Should I obtain an official permission first from you (the publisher)?
2. Am I allowed to include the paper with the Elsevier's published version format directly or do I have to reformat it, e.g. by using my own layout format (but with full content)?

I look forward to your answer.

Many thanks.

Best wishes,
Bobby Minola Ginting

Order Completed

Thank you for your order.

This Agreement between Lehrstuhl für Computation in Engineering (Technical University of Munich) -- Bobby Minola Ginting ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

printable details

License Number	4637090896321
License date	Jul 27, 2019
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography
Licensed Content Author	Bobby Minola Ginting, Ralf-Peter Mundani
Licensed Content Date	Jan 1, 2018
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	print and electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	>50,000
Author of this Springer Nature content	yes
Title	Efficient Parallel Simulations of Flood Propagation including Wet-Dry Problems
Institution name	Chair for Computation in Engineering (Technical University of Munich)
Expected presentation date	Oct 2019
Requestor Location	Lehrstuhl für Computation in Engineering (Technical University of Munich) Arcisstr. 21 Munich, 80333 Germany Attn: Lehrstuhl für Computation in Engineering (Technical University of Munich)
Total	0.00 EUR

ORDER MORE CLOSE WINDOW

APPENDIX
(Papers 1 – 6)

Paper 1

B.M. Ginting and R.-P. Mundani. Parallel Flood Simulations for Wet-Dry Problems Using Dynamic Load Balancing Concept. *Journal of Computing in Civil Engineering* (ASCE), 33(3): 04019013, 2019.
https://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000823

Parallel Flood Simulations for Wet–Dry Problems Using Dynamic Load Balancing Concept

Bobby Minola Ginting[1] and Ralf-Peter Mundani[2]

Abstract: In this paper, a shared-memory parallel simulation of flood modeling is presented. The model used has second-order spatial and temporal accuracy, where the Monotonic Upwind Scheme for Conservation Laws (MUSCL) method is applied for spatial discretization and the Runge-Kutta second-order method is employed for temporal discretization. A cell-centered finite-volume model is used and solved in an edge-based data structure. The model is well-balanced and able to efficiently simulate flood cases on complex topography with wet–dry problems. A cell–edge reordering strategy is designed to ease vectorization and parallelization of the code. To tackle load imbalances among threads due to wet–dry problems, a novel weighted-dynamic load balancing is proposed. The model shows accurate results, and the strategy proposed shows very good parallel efficiencies for problems of different sizes (up to 6.4 million cells/12.8 million edges) on varying numbers of cores (up to 64 cores). As such, this load balancing technique could become a promising strategy for efficient parallel simulations of real flood cases. **DOI: 10.1061/(ASCE)CP.1943-5487.0000823.**

Introduction

Research topics in the field of computational hydraulics have become more and more complex. As a consequence, the computational requirements, particularly for computing very large domains, have been increasing rapidly. To deal with these problems, parallel computing can be an option. In general, there are two key standards used for parallelization: open multiprocessing (OpenMP) and message passing interface (MPI). MPI is a parallelization technique based on a distributed-memory platform in which a main domain should be manually divided into several subdomains, which are then allocated to several different processors. The communications at the interface between neighboring subdomains are, therefore, very important for this technique. OpenMP is a parallelization technique based on a shared-memory platform, in which a (simple) domain will automatically be divided into chunks and assigned to different threads for parallel processing. Therefore, no communication costs exist, but there may be additional synchronization costs. However, problems related to memory bottlenecks—e.g., for domains with billions of cells—are a typical disadvantage of this parallelization method. Also, parallelization only takes place inside one node consisting of several cores, thus hindering large-scale parallelization.

In real flood simulations, one has to deal with wet–dry problems, which have a significant impact on parallel efficiency because wet cells require more computational effort than dry cells. Therefore, a proper load balancing technique is required. However, comprehensive studies for such wet–dry load balancing are still quite rare. A few works for parallel simulations of the two-dimensional (2D) shallow-water equations (SWEs) have been conducted. Lobeiras et al. (2013) performed a parallelization of shallow flows coupled with pollutant transport on multicore central processing units (CPUs) using OpenMP. A speed-up of up to 4.6 on eight threads was obtained with hyperthreading (four cores with two threads per core). It was, however, not stated how the load balancing worked for wet–dry problems. Zhang et al. (2014) simulated a dam-break flow using less than 0.1 million quadrilateral meshes and achieved a speed-up of up to 8.64 on 16 threads using OpenMP. In the authors' opinion, even for a shared-memory platform, this total number of cells is still very small for a parallelization study considering the number of threads used (16 threads). The advantage of using parallel instructions cannot be exploited completely when using a small number of cells because revoking threads in OpenMP suffers from a computational overhead. The load balancing was not reported.

Using hybrid OpenMP/AVX parallelization, Liu et al. (2015) solved the SWEs by neglecting all source terms and did not present any load balancing strategy. Lacasta et al. (2015) implemented a code on graphics processing unit (GPU)s and a parallel code using OpenMP to solve the 2D SWEs for rainfall-runoff events. No load balancing was discussed where speed-up factors of 2.27–2.66 were achieved with four threads. These speed-up factors were still not promising enough for a (shared-memory) parallelization considering the small number of threads used. Neal et al. (2009) parallelized a LISFLOOD-FP model and obtained a parallel efficiency of up to 75% with OpenMP on four cores. It was stated that load balancing is required for further efficiency improvement.

Some works in the last decade focusing on load-balancing for parallelization of the SWEs were also conducted. Sanders et al. (2010) presented a ParBreZo model for parallelizing the SWEs and considered factors for even load balancing due to wet–dry problems. ParBreZo used static domain decomposition with a strategy to justify the distribution of wet–dry cells to each core. For this, one must first do a preliminary run to identify such a distribution, and the output is used later for correcting the previous decomposition. Some ratios were applied for wet:dry cells, e.g., 1:1, 2:1, 3:2, 3:1, and 5:1. No clear explanation was given for these ratios; they were probably only estimation and thus may not be sufficiently accurate. Meister et al. (2017) applied dynamic load balancing due to the adaptive mesh refinement (AMR), which caused the total number of cells assigned to each core to change during simulation time, leading to load imbalances. However, the technique proposed did not

[1]Chair, Computation in Engineering, Technical Univ. of Munich, Arcisstr. 21, Munich D-80333, Germany (corresponding author). ORCID: https://orcid.org/0000-0001-7825-5052. Email: bobbyminola.ginting@tum.de

[2]Chair, Computation in Engineering, Technical Univ. of Munich, Arcisstr. 21, Munich D-80333, Germany. Email: mundani@tum.de

Note. This manuscript was submitted on May 12, 2018; approved on September 21, 2018; published online on February 20, 2019. Discussion period open until July 20, 2019; separate discussions must be submitted for individual papers. This paper is part of the ***Journal of Computing in Civil Engineering***, © ASCE, ISSN 0887-3801.

tackle the load imbalances due to wet–dry problems and was not aimed for static domain decomposition. Recently, Wittmann et al. (2017) designed a mechanism for a full shallow water overland flow 2D (FullSWOF2D) model to tackle load imbalance from vectorization due to different algorithms of dry cells. The original code was changed into a branch-free vectorized version entirely with vector intrinsics. Vector permutations were employed for loading two neighboring vectors required for the computation. This technique is, however, too complex and still becomes very challenging for many branches condition, e.g., when using the Harten-Lax-van Leer-Contact (HLLC) solver.

In this paper, a new parallelization approach based on an edge-based type is presented that distributes the tasks based on the complexity level. For this, a cell–edge reordering strategy is designed easing the vectorization inside one core as well as the separate-way parallelization among the cores between the internal and boundary edges due to their different complexity levels. To tackle the load imbalance in all computational levels due to wet–dry problems, a novel weighted-dynamic load balancing is proposed (WDLB). This technique is very simple and balances the load distribution based on the complexity level, which requires no preliminary run. The authors will show that the WDLB proposed can efficiently distribute the loads among cores. The model is spatially second-order accurate to avoid diffusive results of a first-order scheme. Instead of using an implicit scheme, a second-order temporal scheme is employed to tackle instabilities for modeling very shallow depths on very rough beds and to ease the parallel implementation. An in-house code developed by the first author, Numerical Simulation of Free Surface Shallow Water 2D (NUFSAW2D) (Ginting et al. 2018; Ginting 2019) is used.

Governing Equations and Numerical Methods

The 2D SWEs are derived from the Navier-Stokes equations and are written in a conservative vector form as follows (Ginting and Mundani 2018; Ginting 2017):

$$\frac{\partial \mathbf{W}}{\partial t}+\frac{\partial \mathbf{F}}{\partial x}+\frac{\partial \mathbf{G}}{\partial y}=\mathbf{S}_w+\mathbf{S}_b+\mathbf{S}_f \tag{1}$$

where $\mathbf{W}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{S}_w$, $\mathbf{S}_b$, and $\mathbf{S}_f$ are vectors given by

$$\mathbf{W}=\begin{bmatrix} h \\ hu \\ hv \end{bmatrix},\quad \mathbf{F}=\begin{bmatrix} hu \\ huu+\frac{gh^2}{2} \\ hvu \end{bmatrix},\quad \mathbf{G}=\begin{bmatrix} hv \\ huv \\ hvv+\frac{gh^2}{2} \end{bmatrix},$$

$$\mathbf{S}_w=\begin{bmatrix} R-I \\ 0 \\ 0 \end{bmatrix},\quad \mathbf{S}_b=\begin{bmatrix} 0 \\ -gh\frac{\partial z}{\partial x} \\ -gh\frac{\partial z}{\partial y} \end{bmatrix},\quad \mathbf{S}_f=\begin{bmatrix} 0 \\ -c_f u\sqrt{u^2+v^2} \\ -c_f v\sqrt{u^2+v^2} \end{bmatrix} \tag{2}$$

where h, u, v, g, R, and I = water depth, velocity in x- and y-directions, acceleration due to gravity, rainfall, and infiltration, respectively; z = bed elevation; and $c_f = gn_m^2 h^{-1/3}$, where n_m is the Manning coefficient. The cell-centered finite-volume (CCFV) method used to discretize Eq. (1) is given in Eq. (3)

$$\frac{\partial}{\partial t}\iint_\Omega \mathbf{W}d\Omega+\sum_{i=1}^{N}(\mathbf{F}n_x+\mathbf{G}n_y)_i\Delta L_i=\iint_\Omega(\mathbf{S}_w+\mathbf{S}_b+\mathbf{S}_f)d\Omega \tag{3}$$

where all values of h, u, v, R, I, z, and n_m are defined at the center of each cell; N = total number of edges for a cell; and ΔL = length of edge. In this paper, rectangular grids are used; thus $N = 4$.

The numerical method used in this paper is in general similar to that of Ginting and Mundani (2018). To solve the Riemann problem in Eq. (3), the central-upwind scheme similar to that of Kurganov and Petrova (2007), Chertock et al. (2014), and Shirkhani et al. (2016) is used. The well-balanced property especially for wet–dry problems can be preserved using this scheme together with a method originally proposed by Mohammadian and Le Roux (2006) for the bed-slope terms, which is neither an extra upwinding scheme nor a Riemann solution. Based on the works of Hou et al. (2013), Audusse et al. (2004), and Duran (2015), both the hydrostatic and topography reconstructions are performed to correct the effect of the water depth h on the bed slopes $\partial z/\partial x$ and $\partial z/\partial y$ for all wet–wet, wet–dry, and dry–dry interfaces. A semi-implicit technique similar to that of Ginting and Mundani (2018) and Delis et al. (2011) is used for the friction terms to reduce numerical oscillations and to tackle problems with the Courant-Friedrichs-Lewy (CFL) condition, in the case of very shallow water flows on very rough beds.

Prior to applying all these treatments, the Monotonic Upwind Scheme for Conservation Laws (MUSCL) method pioneered by van Leer (1979) and van Albada et al. (1982) and similar to that of Hubbard (1999) is employed to achieve second-order spatial accuracy. The depth h and the conservative variables hu and hv are chosen to be reconstructed due to some advantages explained by Tan (1992) and Begnudelli et al. (2008). To enforce the monotonicity of the reconstructed variables, the novel edge-based limiter function presented by Delis and Nikolos (2013) is applied. As pointed out by Buffard and Clain (2010) and Hou et al. (2013), despite retaining second-order accuracy, some new local extrema may arise in the reconstruction process. This usually happens in simulating wet–dry problems because both the velocities u and v may not be monotone when transforming the unit discharges hu and hv back to the velocities due to a division by very low depth. To tackle this problem, a simple and efficient procedure given by Buffard and Clain (2010) is followed. The numerical model is not presented in detail in this paper. Interested readers are thus referred to the aforementioned publications.

When simulating wet–dry problems, one or more edges corresponding to a cell may become either wet–dry or dry–dry interface. For both cases, the variables of vector $\mathbf{W}$ at such edges are not reconstructed but simply defined according to the corresponding center, turning into a first-order scheme. As a consequence, the other edges that correspond to that center must not be reconstructed to satisfy the continuity property, even though those edges are wet–wet interfaces. This can be achieved, for example, by setting the limiter function to zero for such a center, by which the values at the edges turn into a first-order scheme although the MUSCL method's procedures are still calculated. For the sake of completeness, a pseudocode to distinguish between wet–wet and wet–dry interfaces is given in Algorithm 1.

Algorithm 1. Algorithm to distinguish between wet–wet and wet–dry interfaces

```
if wet–dry or dry–dry interfaces at edges then
    calculate first-order scheme
else
    calculate second-order scheme
    if velocities are not monotone then
        calculate first-order scheme
    end if
end if
```

The Runge-Kutta second-order (RKSO) method is used for temporal discretization, with which the convective fluxes, bed slope, and friction terms are easily incorporated, as shown in Algorithm 2, where A_p denotes the area of cell p, θ is the implicitness parameter, and Δt is the time step. The values of $\theta = 1$ and $\theta = 0$ give an implicit and an explicit scheme, respectively ($\theta = 0.5$ is used in this paper). In Algorithm 2, the asterisk defines the calculation step of the RKSO method. Thanks to the assumption of $(t+1) \approx *$, the asterisk now serves to incorporate the calculation of $(t+1)$ at every calculation step of the RKSO method, thus allowing the semi-implicit treatment of the friction source terms to work similarly to an explicit method. When transforming the conservative values back to the primitive values, division by a very low value of h^* may lead to a numerical error. This can be avoided by constructing a wet–dry treatment with a very low water depth as a limiter value. This treatment is written as Algorithm 3, where $h_{\min} = 1 \times 10^{-6}$ m. For the treatments of boundary conditions, readers are referred to Ginting (2017).

Algorithm 2. Concept of the calculation for temporal discretization

1: set the coefficients for the RKSO scheme
   $\alpha^a_{(*=1)} = 1$; $\alpha^b_{(*=1)} = 0$; $\alpha^a_{(*=2)} = 0.5$; $\alpha^b_{(*=2)} = 0.5$
2: set the initial value of $\mathbf{W}$ for all cells at each time step
3: **for** $(p = 1)$to$(p =$ total number of cells$)$ **do**
4: $\mathbf{W}_p^{(*=0)} = \mathbf{W}_p^t$; $\mathbf{W}_p^{(*=1)} = \mathbf{W}_p^t$
5: **end for**
6: **for** $(* = 1)$to$(* = 2)$ **do**
7: **for** $(p = 1)$to$(p =$ total number of cells$)$ **do**
8: $$\mathbf{W}_p^* = \alpha_*^a \left[\mathbf{W}_p^* + \frac{\Delta t}{A_p} \left(- \left[\sum_{i=1}^{N=4} (\mathbf{F} n_x + \mathbf{G} n_y)_i \Delta L_i \right]_p^{(*-1)} + \left[\iint_\Omega (\mathbf{S}_b) d\Omega \right]_p^{(*-1)} \right) \right] + \alpha_*^b \mathbf{W}_p^*$$
9: transforming back: conservative variables hu^* and $hv^* \rightarrow$ primitive variables u^* and v^*
10: updating $\mathbf{W}_p^* \leftarrow$ friction source terms are now taken into account
11: $\mathbf{W}_p^* \leftarrow \mathbf{W}_p^* - \Delta t g \mathbf{W}_p^* [(1-\theta)(n_m^2 h^{-\frac{4}{3}} \sqrt{u^2+v^2})_p^* + \theta (n_m^2 h^{-\frac{4}{3}} \sqrt{u^2+v^2})_p^{(*-1)}]$
12: saving the final values for the subsequent time step when $* = 2$ as $\mathbf{W}_p^{(t+1)} = \mathbf{W}_p^{(*=2)}$
13: **end for**
14: **end for**

Algorithm 3. Wet–dry treatment for a very low water depth less than $h_{\min}$

1: **if** $h^* \le h_{\min}$ **then**
2: hu^*, hv^*, u^*, and v^* are set to zero
3: **else**
4: $$u^* = \frac{hu^*}{h^*} \text{ and } v^* = \frac{hv^*}{h^*}$$
5: **end if**

Parallel Computing Implementation

Prior to explaining the cell–edge reordering strategy, a cell-based CCFV method is considered to distinguish it from an edge-based one. In the cell-based CCFV method, the loop is computed over the total number of cells so that Eq. (3) is directly computed within a single loop. This method can avoid synchronizations among the threads because there is no data dependency between cells in the same time step. Each thread computes all data associated with its block of volumes independently. However, computational cost may increase because the value of an edge shared between two cells is computed twice. The contribution of two cells to an edge has a similar scalar value but with a different algebraic sign. When the loop over a total number of cells is computed, some computations will be redundant.

In an edge-based CCFV method, the redundant computations given in the cell-based CCFV method can be avoided because both the convective fluxes and the bed-slope source terms are computed first within a single loop over the total number of edges prior to updating the values of the cells at the subsequent time step. Despite being cheaper, the edge-based CCFV method has the major drawback that it requires synchronization among threads to compute the values of the cells. The idea of the cell–edge reordering strategy emerged from the aforementioned problem that not only the computations to achieve an efficient thread concurrency are taken into consideration, but also the aspect that the overheads for the parallel code must be kept low. For the sake of completeness, both the cell-based and edge-based CCFV methods are depicted in Fig. 1.

Cell–Edge Reordering Strategy

The calculations in NUFSAW2D are classified into two parts: edge-driven and cell-driven computations. The edge-driven computations consist of the calculations of the MUSCL linear reconstruction, convective fluxes, and bed-slope source terms, whereas the cell-driven computation consists of the calculation of vector $\mathbf{W}$ for the subsequent time step together with the friction source terms. The basic idea of this reordering strategy is to determine the relationship patterns between edge and cell for the edge-driven loop and between cell and edge for the cell-driven loop, so that a similar computational procedure can be collected and then be successfully vectorized within a single loop.

The cell–edge reordering strategy is explained in Fig. 2 by giving a domain divided by 5×4 segments, creating 20 rectangular

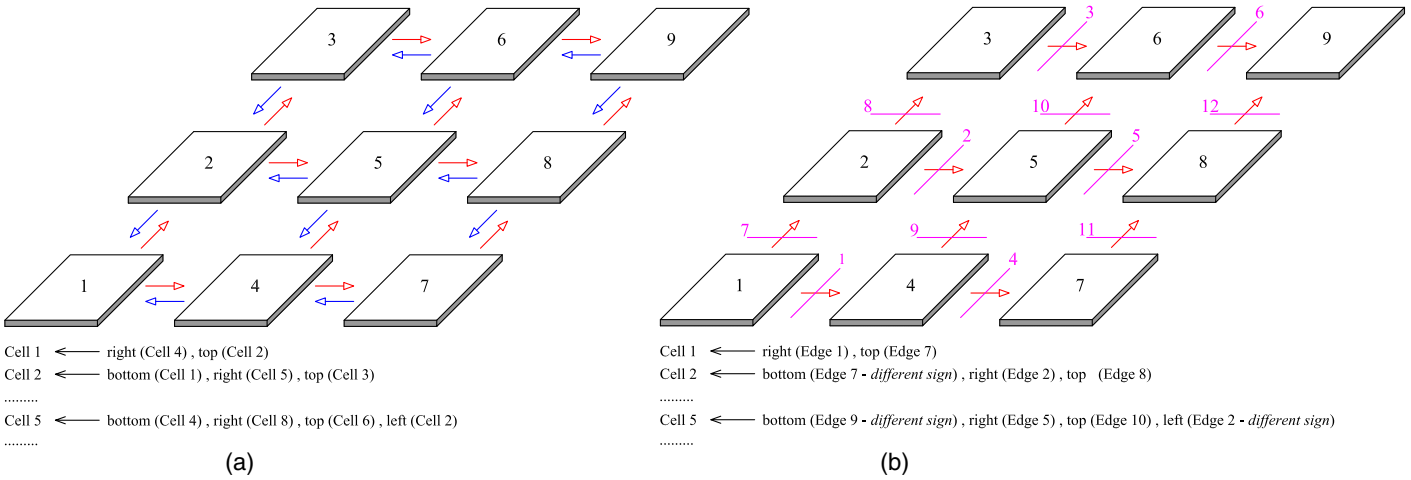


Fig. 1. Concept of computation in (a) cell-based; and (b) edge-based CCFV.

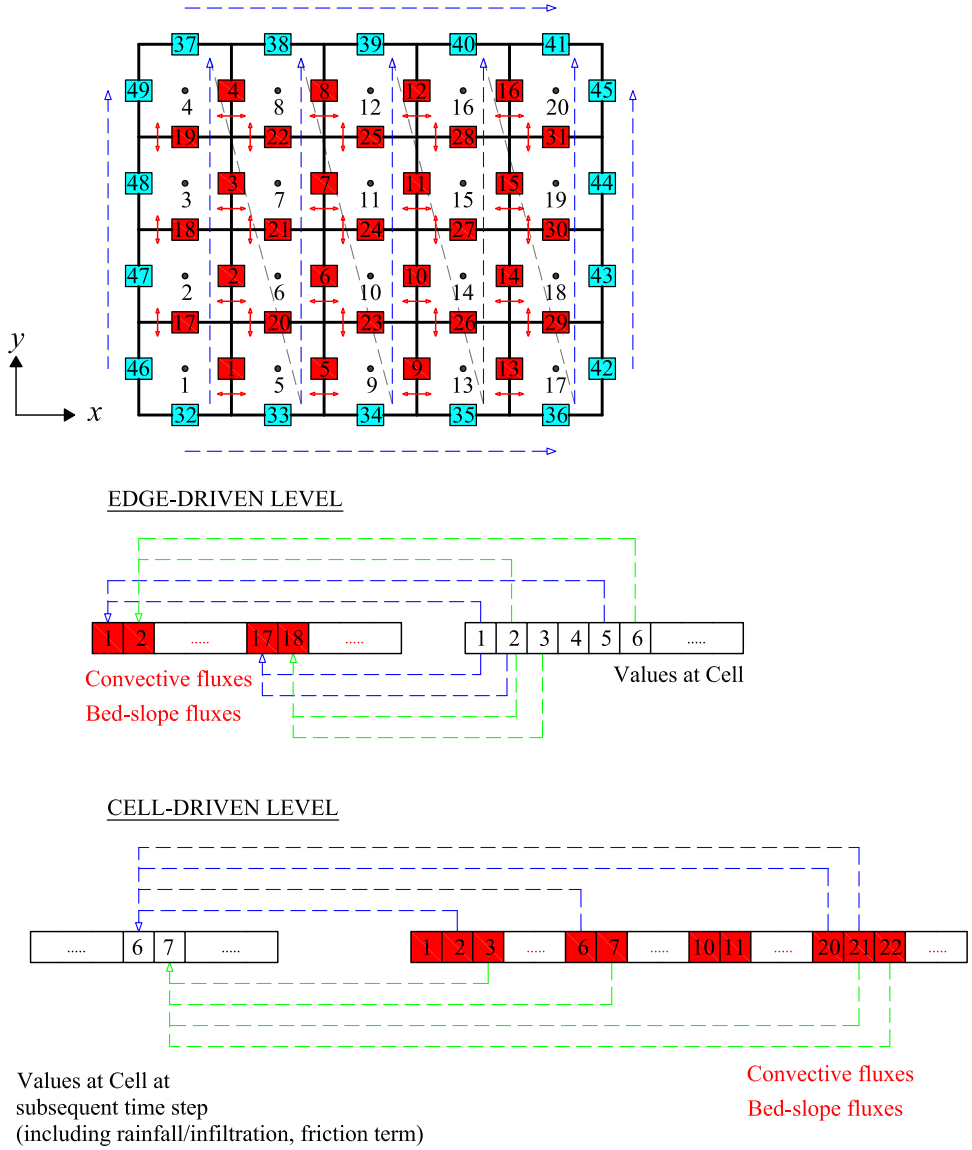


Fig. 2. Cell–edge reordering strategy for the edge-driven and cell-driven levels computations in an edge-based CCFV.

J. Comput. Civ. Eng., 2019, 33(3): 04019013

cells and 49 edges. First, the cell numbering is designed following the Z-pattern. This numbering is addressed for the cell-driven computations. The next step is to design the edge numbering for the edge-driven computations, where loops are carried out to compute the values of the edges by using the values of the cells as inputs. For this, the two cells corresponding to each edge must be known. Because the cells are rectangular, there are generally only two types of computation in the edge-driven level: computation of edges in x- and y-directions.

It is obvious that the relationship patterns between the edge and cell can easily be obtained by computing the edges in the x-direction and y-direction separately. However, because the computations of internal and boundary edges differ from each other, starting the loop from the boundary edge to another boundary edge may not be efficient. The high aspect ratio of the total number between the boundary edges to the internal edges has a great influence on the parallel efficiency. For example, there is no need to apply the MUSCL method to boundary edges or, in other words, all values at the boundary edges have first-order accuracy, thus requiring less CPU time. To solve this problem, the computations in every direction are classified again to be the calculations of internal and boundary edges.

As shown in Fig. 2, the first loop of the edge-driven computation is conducted in the y-direction for Internal edges 1–16 and in the x-direction for Internal edges 17–31. Now, every internal edge has a specific pattern for its two corresponding cells, for which no dependency exists between the edges, thus easing the parallel implementation. The second loop of the edge-driven computation is carried out in the x-direction for Boundary edges 32–41 and in the y-direction for Boundary edges 42–49. These boundary edges can either be wall boundaries or flow boundaries. In this edge-driven level, it is not required to add additional cells for the boundary edges.

With regard to Fig. 2, the convective fluxes for each edge i are saved, corresponding to 49 edges. However, a different strategy must be considered for the bed-slope terms. Because the bed-slope fluxes may have two different values for each edge i, those fluxes must be saved twice. For example, at Edge 6, the first flux of the bed-slope source terms is saved corresponding to Cell 6, and the second belongs to Cell 10. To accomplish this, in the similar loop to that of the convective fluxes, two arrays are provided for saving the bed-slope fluxes at each edge i. Let these two arrays be named as Bed slope A and Bed slope B. Only the internal edges have two corresponding cells. Thus, the first array of the bed-slope fluxes (Bed slope A) is calculated corresponding to 49 edges (31 internal edges and 18 boundary edges), whereas the second array of the bed-slope fluxes (Bed slope B) only belongs to 31 edges (31 internal edges).

The next step is to perform the computations in the cell-driven level after all computations in the edge-driven level are finished. Synchronization is thus required. Compared with the edge-driven level, the procedures in the cell-driven level are much simpler, only requiring summing both the convective and bed-slope fluxes with the proper signs from all the corresponding edges—and using rainfall/infiltration and friction terms from the previous time level. For example, for the convective fluxes computation, Cell 6 corresponds to Edges 20, 6, 21, and 2 with the flux signs of -1, $+1$, $+1$, and -1, respectively. For the bed-slope fluxes, Cell 6 corresponds to the similar edges to those of convective fluxes, which belong to Bed slope B, Bed slope A, Bed slope A, and Bed slope B, respectively. Similarly, for the convective fluxes computation, Cell 3 corresponds to Edges 18, 3, 19, and 48 with the flux signs of -1, $+1$, $+1$, and -1, respectively. For the bed-slope fluxes, Cell 3 corresponds to the same edges to those of convective fluxes, which belong to Bed slope B, Bed slope A, Bed slope A, and Bed slope A, respectively. In the proposed implementation, this classification is very useful, and it helps ease the compiler to exploit instruction pipelining and parallelization.

Causes of Load Imbalance Issues with Focus on Wet–Dry Problems

Using an edge-based data structure, three typical load imbalance issues faced by NUFSAW2D as well as common edge-based CCFV models are presented. This is illustrated in Fig. 3, where it is assumed to use four threads. The first reason is because the total number of edges or cells may not match the total number of threads. With a default static load balancing (SLB), each thread gets the same amount of loads (five cells) in the cell-driven level. In the edge-driven level, Threads 0–2 are, however, assigned the most amount of loads (eight internal plus five boundary edges), whereas Thread 3 receives, in contrast, the least amount of loads (seven internal plus three boundary edges). This condition leads to an unbalanced load distribution and decreases the parallel efficiency. However, the effect of this problem becomes less and less significant as the numbers of edges and cells increase, e.g., to millions of cells. This problem can also be tackled using a dynamic load balancing (DLB) that allows, for example Thread 3 to steal the work of Thread 0, 1, or 2 in the edge-driven level, so that a balanced load distribution can be ensured. Although more overheads appear in this dynamic way, the cost reduces as the problem size increases.

For wet–dry problems, the load imbalance issues become more complex. As shown in Algorithm 1, in the edge-driven level, if wet–dry or dry–dry interfaces at edges appear (which can only be known during runtime), the model turns to a first-order scheme, thus requiring no MUSCL technique and becoming cheaper than a second-order scheme—in contrast with wet–wet interfaces, which could either have second-order accuracy one (if no discontinuity exists) or first-order one (when dealing with discontinuity). This becomes the second reason of the load imbalance issue faced by NUFSAW2D. The third reason emerges in the cell-driven level, where due to the semi-implicit treatment of the friction terms, one needs to transform unit discharges back to velocities, for which a division by a very low depth (especially for dry/almost dry cells) may produce oscillations. To this regard, the nested branch statement (*if-then-else*) in Algorithm 3 becomes inevitable to anticipate such oscillations. Because the total number of wet and dry cells is uncertain during runtime, NUFSAW2D consequently suffers from a load imbalance issue.

Load Balancing Strategy

Prior to explaining the load balancing strategy, the ratio of CPU time required by the first-order (Line 2) and the second-order schemes (Lines 4–7) in Algorithm 1 as well as the ratio of CPU time between dry (Line 2) and wet cells (Line 4) in Algorithm 3 must be approximated. For the former, it is quite simple to do where NUFSAW2D was tested for simulating several cases (using one core without a wet–dry problem) including hydraulic jump, flow over transition channel, and dam-break cases [e.g., Cases 1–5 in Ginting (2017)]. It was found that for Algorithm 1, using solely the first-order scheme was typically 1.7–2 times faster than using the second-order scheme. For the latter, knowing the CPU time ratio between wet and dry cells in Algorithm 3 is harder because the ratio of the total number between wet and dry cells always changes during runtime, showing a nonlinear relationship. To approximate this, some cases dealing with wet–dry problems [e.g., Cases 1–2 of Ginting and Mundani (2018) as well as the two cases presented in this paper] were simulated using one core with the second-order

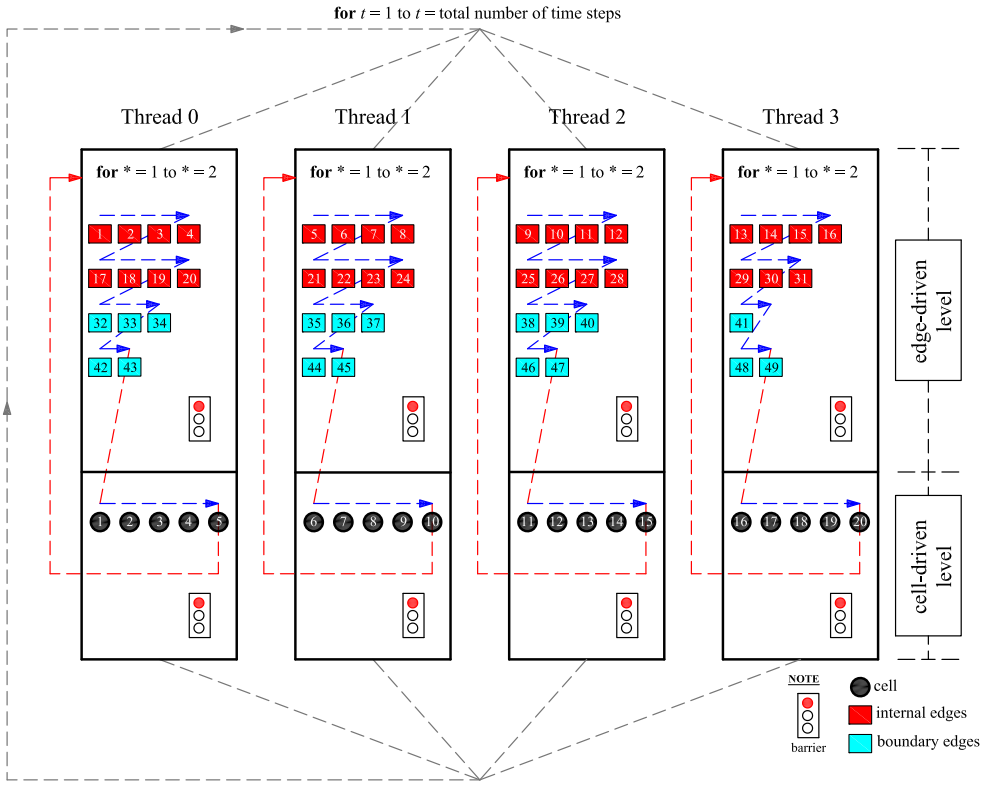


Fig. 3. Load distributions among threads with a static load balancing in NUFSAW2D for the domain in Fig. 2.

scheme. It was found that the CPU time for dry cells in Algorithm 3 was typically 1.9–2.1 times faster than that for wet cells. Based on these facts and for the sake of simplicity, a factor of 2 was chosen for the load balancing purpose.

To explain the load balancing strategy, the domain in Fig. 2 is now considered where only some areas are inundated by water, as shown in Fig. 4. One can see that Cells 1–3, 6–7, 14–15, and 18–19 are wet cells and correspondingly, Internal edges 2–3, 14–15, 17–18, 21, 27, and 30 are classified as wet–wet interfaces. First, the load balancing strategy is explained for the internal edges, where four threads are used. A factor of 2 is assigned to the aforementioned edges, whereas the others are assigned 1. In the y-direction, each thread receives the same total number of internal edges (four edges) using the SLB. However, the loads assigned are not same due to wet–dry problems. Thread 0 gets six loads (Edges 1–4 = 1, 2, 2, and 1 loads, respectively). Similarly, Thread 3 obtains six loads. Meanwhile, Threads 1–2 only have four loads. This condition causes a load imbalance issue.

Using the DLB with a proper chunk size (CS) will not significantly help tackle the load imbalance issue. First, this is because the DLB assigns the edges to the threads sequentially based on the given CS, so that in the end, it is still possible that some threads must wait for the others. Second, it is difficult to set the proper value for CS because the total number of wet–dry cells changes during runtime. This is how the idea of WDLB emerged. For the internal edges in the y-direction, the total amount of load is summed up to be 20 loads. This can easily be done in parallel, e.g., using !$omp do reduction (+:value). These 20 loads must then be divided by the total number of threads (four), thus each threads must have five loads, as shown in Fig. 4; it is easy to do, e.g., Edges 1–3 contain $1 + 2 + 2 = 5$ loads. However, this procedure is difficult to be parallelized; thus, only a sequential way is employed. After the WDLB, Threads 0–3 are now assigned the new edge indexes of 1–3, 4–8, 9–13, and 14–16, respectively. A similar way is also applied to the internal edges in the x-direction.

For the x and y boundary edges, there is no need to employ the WDLB because no MUSCL method is applied to these edges. The cost of each boundary edge is almost similar, and the standard SLB or DLB with a proper CS can thus be applied. There were no significant differences found using the SLB and DLB for the boundary edges. However, if one is not aware of assigning the CS value, one or more threads may not be assigned a task at all, thus decreasing the parallel efficiency. For example, if one uses the SLB with CS = 4 for the y boundary edges, Threads 0–3 will receive four, four, two, and zero loads, respectively, because the round-robin way applies. To tackle this problem, a block-distribution type is used, so that it is always ensured that each thread is assigned the loads. For this, a small program of subroutine para_range from Aoyama and Nakano (1999) was adopted.

The WDLB is also applied to cells in the cell-driven level. A similar procedure to that of internal edges is thus employed to check the total amount of loads, where 29 loads are obtained. These 29 loads are distributed to four threads giving approximately seven loads to each thread. A sequentially way is again employed to determine the new cell indexes for Threads 0–3, which are 1–4, 5–9, 10–15, and 16–20, having seven, seven, eight, and seven loads, respectively. In this case, the results before and after load balancing are accidentally similar.

One can see that the cell–edge reordering strategy has helped ease the WDLB procedures by providing a contiguous array structure. Because the procedures for all internal edges as well as for cells are performed sequentially, it may not be efficient to apply the WDLB in every single time level. For this reason, the WDLB is applied after several time levels. In all the implementations, it

J. Comput. Civ. Eng., 2019, 33(3): 04019013

was found that the most efficient step is about every 1/50 of total time levels. For the sake of clarity, the structure of NUFSAW2D is illustrated in Algorithm 4, where tot_threads is the total number of threads, thread_ID is thread's identity number, tot_num_t_s is the total number of time steps, iedg_sta and iedg_end are internal edges' indexes, bedg_sta and bedg_end are boundary edges' indexes, and cell_sta and cell_end are cells' indexes. The first parallel region is given on Lines 1–3 to perform the cell–edge reordering strategy and the second one is given on Lines 4–26. The CPU time between the first and second parts are distinguished because the authors focus on the efficiency of the second part without considering input/output files. CPU time of the first part is very small even for millions of cells because the computation required in the cell–edge reordering strategy is very simple.

Algorithm 4. Code structure of NUFSAW2D with OpenMP parallel technique

```
1: !$omp parallel
2:    calculate cell–edge reordering strategy
3: !$omp end parallel
4: !$omp parallel
5: tot_threads = omp_get_max_threads(); thread_ID = omp_get_thread_num()
6: for (t = 1) to (t = tot_num_t_s) do
7:  if mod (t − 1, 1/50 * tot_num_t_s) = 0 then
8:     perform weighted-dynamic load balancing for internal edges and cells
9:     apply subroutine para_range for boundary edges
10:    internal & boundary edges' and cells' indexes start – end (1: tot_threads) are obtained
11:  end if
12:  for (∗ = 1)to(∗ = 2) do
13:    for (i = iedg_sta(thread_ID + 1))to(i = iedg_end(thread_ID + 1)) do
14:       calculate Algorithm 1 for all internal edges
15:    end for
16:    for (i = bedg_sta(thread_ID + 1))to(i = bedg_end(thread_ID + 1)) do
17:       apply boundary condition treatments for all boundary edges
18:    end for
19:    !$omp barrier
20:    for (i = cell_sta(thread_ID + 1))to(i = cell_end(thread_ID + 1)) do
21:       calculate values at cells based on Algorithms 2 and 3
22:    end for
23:    !$omp barrier
24:  end for
25: end for
26: !$omp end parallel
```

Test Cases

Two cases dealing with complex wet–dry mechanisms, rough and complex topography, and well-balanced property are considered to verify the efficiency of NUFSAW2D. The first case is a dam-break case propagating over an initially dry bed with three humps and is proposed to check the model ability in modeling wave–bed wall interactions and drying mechanism on the humps for the well-balanced property. The second case is considered to simulate a real flood case involving a very complex topography, very low water depth on a very rough bed, and complex wet–dry mechanisms. NUFSAW2D was executed on two Linux computing clusters: Sandstorm operated at the Chair for Computation in Engineering, Technical University of Munich (CiE-Sandstorm 2018) and CoolMUC-3 operated by Leibniz Supercomputing Center (LRZ 2018). Sandstorm contains nodes with Intel Xeon E5-2690 (Sandy-Bridge-E) and CoolMUC-3 provides nodes with Intel Xeon-Phi 7210F (Knights-Landing). The Intel Fortran compiler 16 was used. The parallel computation was implemented by compiling both the sequential and parallel codes with –O2 optimization flag and using double-precision arithmetic (64-bit). The weak and strong scaling are performed using 16 cores (Sandstorm) and 64 cores (CoolMUC-3) for 0.1–6.4 million cells or 0.2–12.8 million edges. The CPU time for the cell–edge reordering strategy is not discussed here because it is not significant in all implementations (it took less than 0.3 s to reorder 3.2/6.4 million cells using 16/64 cores and less than 0.012 s to reorder 200,000 cells using a single core). For each case, the simulation was performed 10 times to obtain an average CPU time. A performance metric is used and expressed in million cells per second per core (Mcell/s/core), which is a comparison between the numbers of cells for a total number of calculation steps that can be processed per unit of time using one core. A metric of Gflop/s/core is also presented and compared with the theoretical peak performance of each hardware.

Case 1: Dam Break over Three Bumps

This case was first proposed by Kawahara and Umetsu (1986) and also tested numerically in several papers, e.g., by Brufau et al. (2002) and Liang and Borthwick (2009). This case deals with a wave propagation due to a dam-break case flowing over an initially

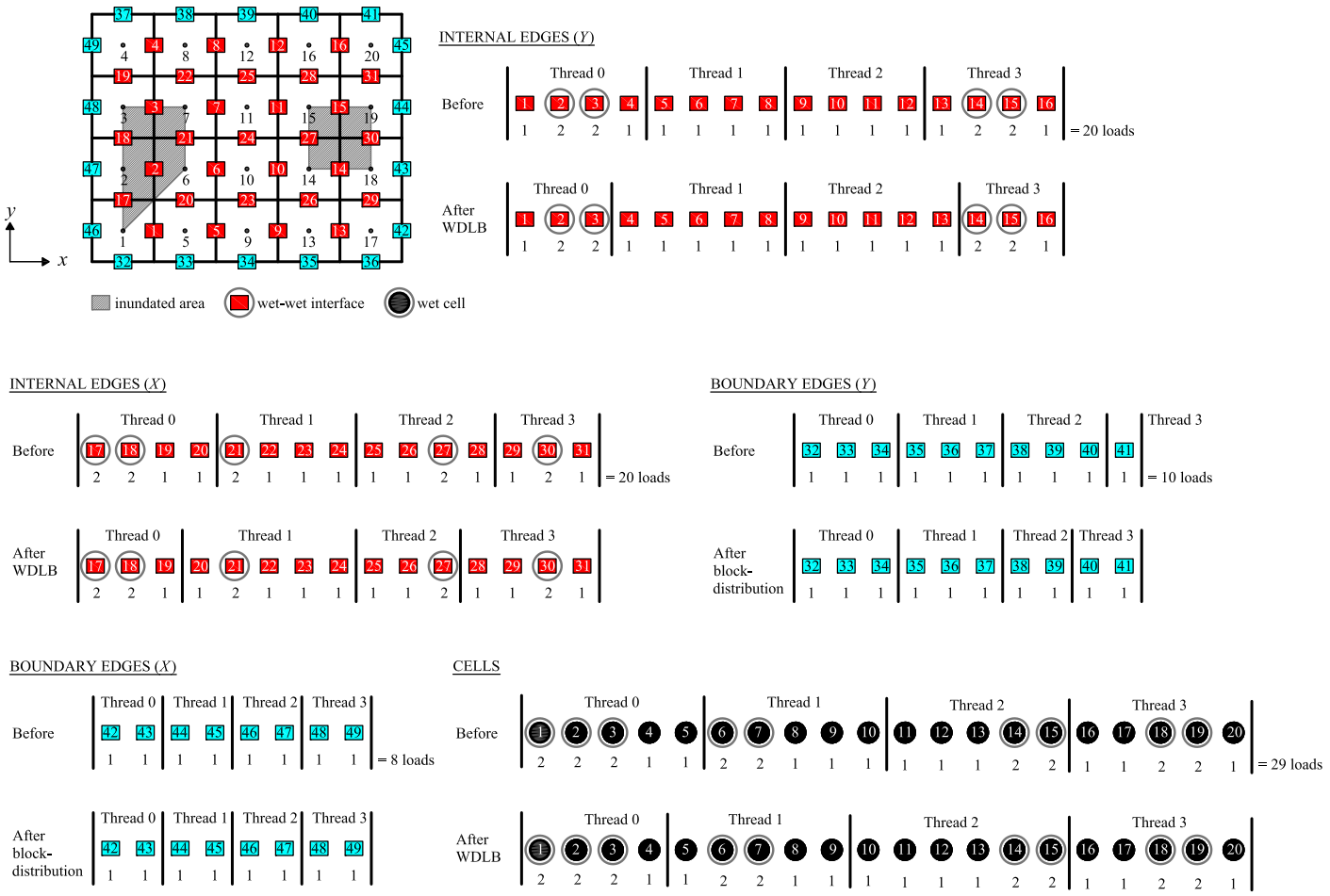


Fig. 4. Proposed load balancing strategy: weighted-dynamic load balancing.

dry bed with a size of 75 × 30 m. Three humps are formed on the domain, the shape of which follows

$$z(x,y) = \max\left[0, 1 - \frac{1}{8}\sqrt{(x-30)^2 + (y-6)^2}, 1 - \frac{1}{8}\sqrt{(x-30)^2 + (y-24)^2}, 3 - \frac{3}{10}\sqrt{(x-47.5)^2 + (y-15)^2}\right] \quad (4)$$

The domain configurations tested are (1a) 707 × 283, (1b) 1,000 × 400, (1c) 1,414 × 566, (1d) 2,000 × 800, (1e) 2,449 × 980, and (1f) 2,830 × 1,131 for Sandstorm and (2a) 500 × 200, (2b) 707 × 283, (2c) 1,000 × 400, (2d) 1,414 × 566, (2e) 2,000 × 800, (2f) 2,830 × 1,131, (2g) 3,464 × 1,386, (2h) 3,741 × 1,497, and (2i) 4,000 × 1,600 for CoolMUC-3. Along the domain from $x = 0$ to $x = 16$ m, the initial water elevation is set to 1.875 m, whereas the rest contains no water. A uniform Manning coefficient of 0.0125 $\mathrm{sm}^{-1/3}$ is given for the whole domain. This case aims to study the characteristics of wave–wave, wave–bed, and wave–wall interactions as well as to investigate the capability of simulating the drying mechanism on the humps until a steady-state condition is achieved. The simulation time is set to 300 s. The constant Δt values of 0.002 and 0.001 s are used, thus giving 150,000 and 300,000 calculation steps for Sandstorm and CoolMUC-3, respectively. The maximum CFL number of 0.68 is produced for the case with the maximum number of cells.

The results are visualized in Figs. 5 and 6. NUFSAW2D has shown a stable computation for simulating the complex wave–bed–wall interactions. No problem was found to simulate the highly discontinuous shock waves due to such interactions. Also, NUFSAW2D is able to model the wet–dry mechanisms on the humps properly until the steady-state condition. The results are in accordance to those of Brufau et al. (2002) and Liang and Borthwick (2009). The mass conservation has always been guaranteed for this case.

For Sandstorm with 150,000 steps, NUFSAW2D requires 13,060 s (for 1a with one core) and 15,010 s (for 1f with 16 cores) giving the performance metrics of 2.3 and 2 Mcell/s/core, respectively. This allows NUFSAW2D to achieve 1.81 Gflop/s/core, which translates to approximately 31% of the theoretical peak performance (TPP) of 5.8 Gflop/s/core. Meanwhile, for CoolMUC-3 with 300,000 steps, NUFSAW2D requires 46,154 s (for 2a with one core) and 52,632 s (for 2i with 64 cores), giving performance metrics of 0.65 and 0.57 Mcell/s/core, respectively. This leads NUFSAW2D to obtain 0.54 Gflop/s/core, which is about 21% of the TPP of 2.6 Gflop/s/core. Fig. 7 shows the parallel efficiency of the model for weak and strong scaling. For weak scaling, the mesh configurations 1a–1f are tested using 1, 2, 4, 8, 12, and 16 cores, respectively, and 2a–2i are tested using 1, 2, 4, 8, 16, 32, 48, 56, and 64 cores, respectively. For strong scaling, each 1c, 1d, and 1f are tested using 1–16 cores, whereas each 2e, 2f, and 2i are tested using 1–64 cores. The results show NUFSAW2D scales very well on both hardware configurations. On Sandstorm, it achieves efficiencies

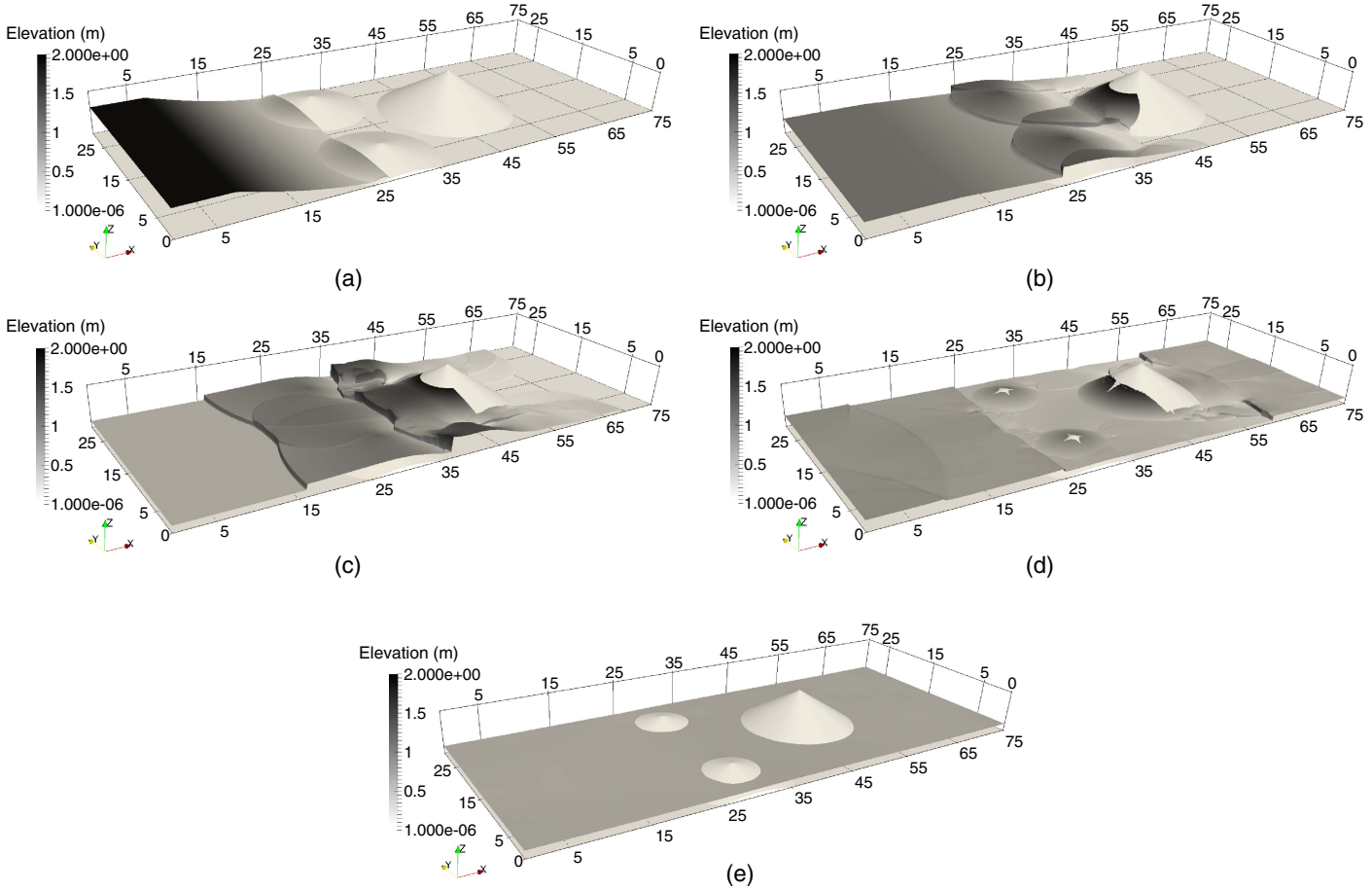


Fig. 5. Case 1: numerical results at 2, 6, 12, 30, and 300 s using 2,830 × 1,131 cells; side view (elevation).

more than 98% with eight cores and 87% with 16 cores. On CoolMUC-3, up to 97% of efficiency is obtained with 56 cores and 88% with 64 cores. NUFSAW2D also achieves excellent speed-up factors, which are almost linear up to 56 cores (for 2i). These strong scaling results show NUFSAW2D becomes more efficient for the larger number of cells.

Case 2: Urban Flood in Glasgow, United Kingdom

In this case, a simulation dealing with very low water depth on very rough beds is considered. The aim of this case is to simulate the real flood case in Glasgow, United Kingdom, which was caused by strong rainfall and an overloaded culvert. More information on the event has been given by Neelz and Pender (2013), according to whom, the domain was taken on August 13, 2009. and provided in a digital elevation model (DEM) format with a size of 965 × 401 m, as shown in Fig. 8. The ground elevations of the DEM vary within the range of +21 to +37.61 m. Following Neelz and Pender (2013), to simulate the bare earth of the DEM, the buildings at the actual location are ignored.

There are no observation data provided for this case; instead, several one-dimensional (1D), 2D, and three-dimensional (3D) numerical models were used to simulate the event, and the results were compared at nine points. For the sake of comparison in this paper, the results of the 2D models are selected: ISIS 2D, MIKE FLOOD, SOBEK, two-dimensional unsteady FLOW finite volume (TUFLOW), and XPSTORM; the details of these 2D models are given in the aforementioned publication.

The domain configurations tested are (1a) 695 × 288, (1b) 982 × 408, (1c) 1,390 × 576, (1d) 1,966 × 814, (1e) 2,403 × 999, and (1f) 2,779 × 1,152 for Sandstorm and (2a) 491 × 204, (2b) 695 × 288, (2c) 982 × 408, (2d) 1,390 × 576, (2e) 1,966 × 814, (2f) 2,779 × 1,152, (2g) 3,402 × 1,411, (2h) 3,674 × 1,525, and (2i) 3,927 × 1,630 for CoolMUC-3. The initial boundary condition is set as a dry bed at all cells, and all boundaries of the modeled area are set as wall boundaries. The Manning coefficients are set to 0.02 $s\,m^{-1/3}$ for roads and pavements and 0.05 $s\,m^{-1/3}$ everywhere else [Neelz and Pender (2013) have provided more details]. These values show that this case deals with very rough beds, which may produce excessive drag forces that can reverse the flow. With regard to the sources of water, the rainfall event is uniformly spread across all cell centers, and the surcharge flow of the culvert is located at (919.75, 337.75) m, of which the values are given in Fig. 9. The simulation time is set to 5 h. The constant Δt of 0.03 s and 0.015 s are used, thus requiring 600,000 and 1.2 million calculation steps for Sandstorm and CoolMUC-3, respectively. The maximum CFL number of 0.48 is yielded for the case with the maximum number of cells.

The results of NUFSAW2D using 695 × 288 and 2,779 × 1,152 cells are compared with those of the aforementioned 2D models of Neelz and Pender (2013) (where approximately 98,000 cells were used) at Point 2 (561.75, 255.75) m and Point 6 (595.75, 145.75) m, as given in Figs. 10 and 11, respectively. As a general result, NUFSAW2D produces a double-peaked shape hydrograph similar to the aforementioned 2D models. As discussed by Neelz and Pender (2013), this phenomenon is due to the very intense rainfall

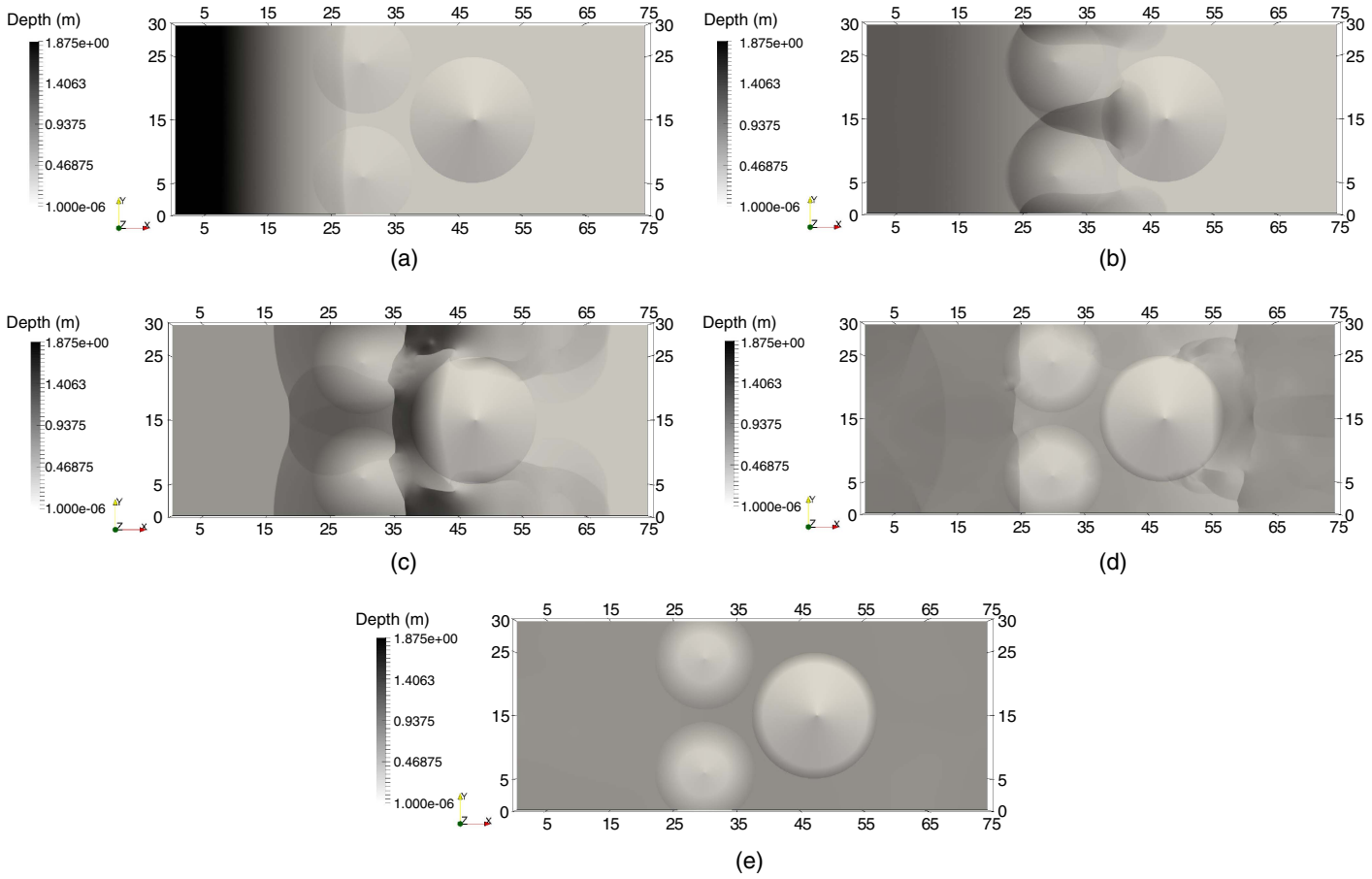


Fig. 6. Case 1: numerical results at 2, 6, 12, 30, and 300 s using 2,830 × 1,131 cells; top view (depth).

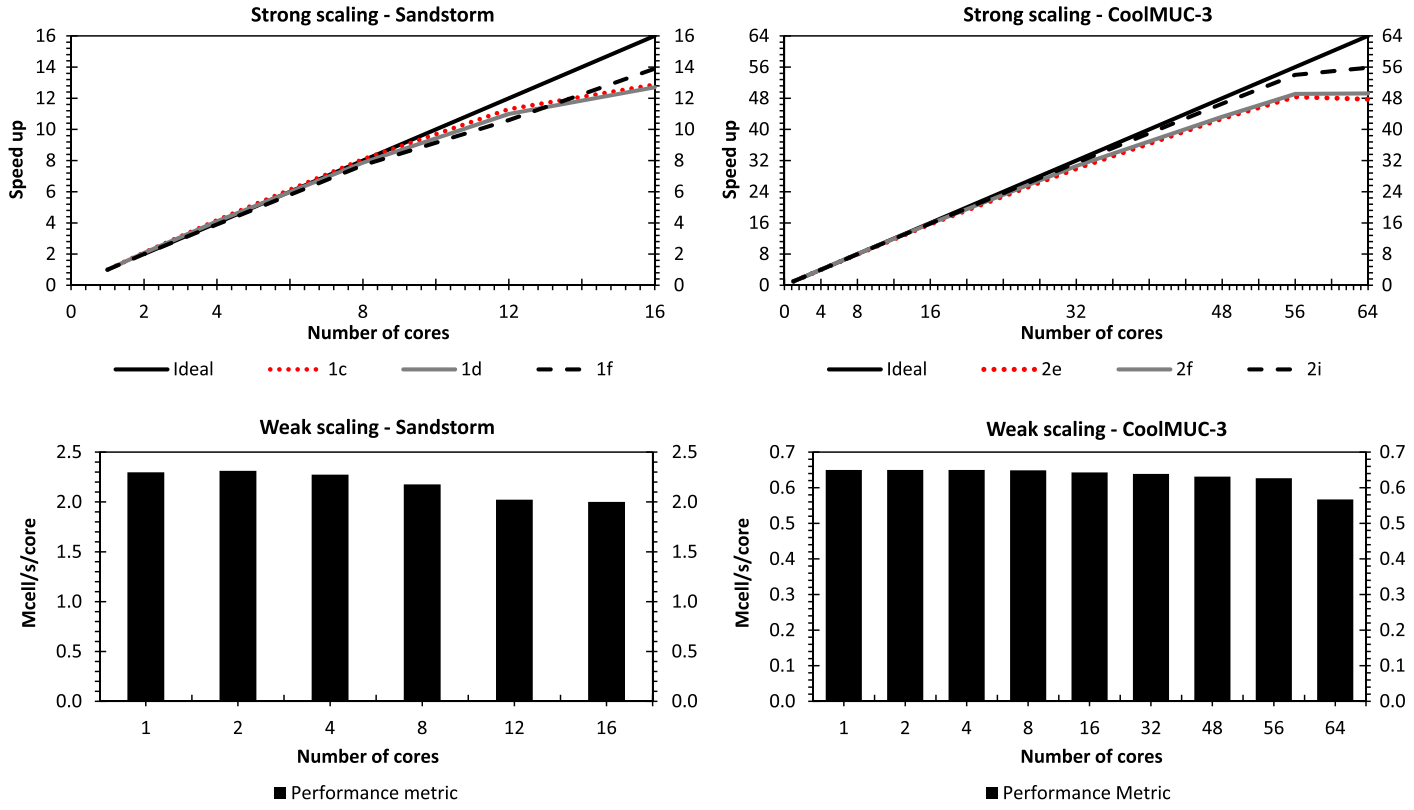


Fig. 7. Case 1: weak and strong scaling.

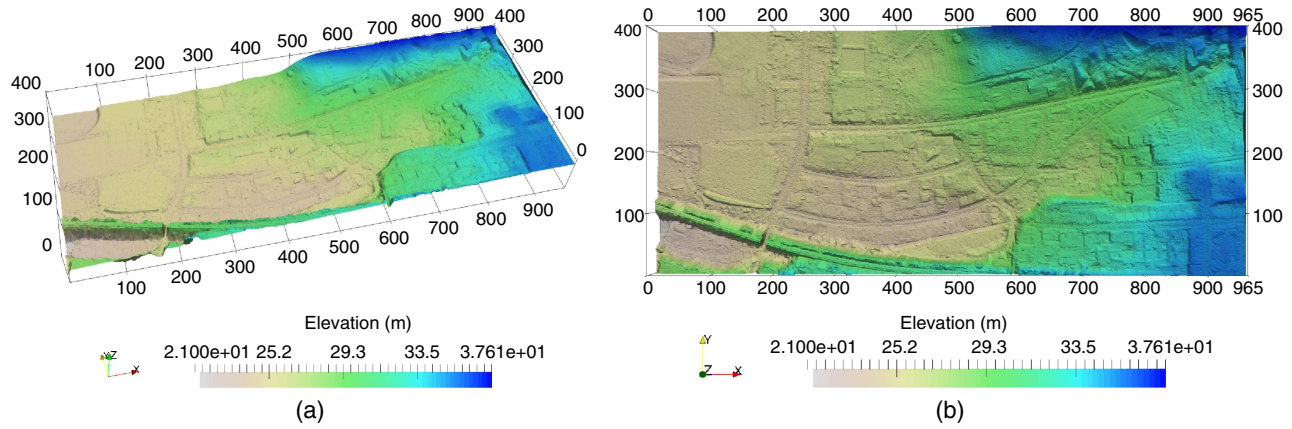


Fig. 8. Case 2: domain in Glasgow, United Kingdom.

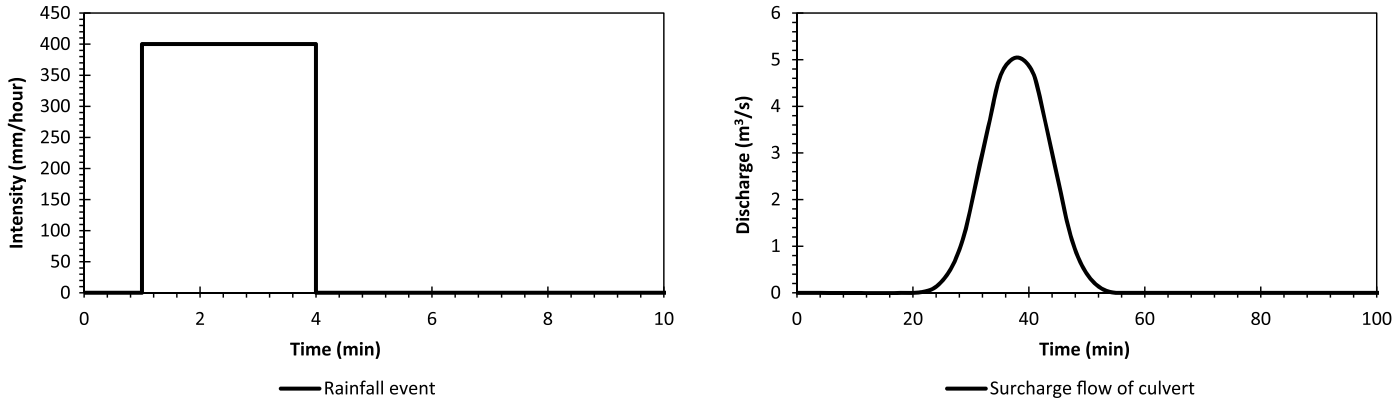


Fig. 9. Case 2: rainfall event and surcharge flow of culvert.

during 1–4 min. At Point 2, for the first peak flow, both results of NUFSAW2D exhibit a similar characteristic of the water level, with a value of +28.72 m at around $t = 11$ min. The second peak flow turns out to be almost identical as well, with the highest water level being +28.83 m at around $t = 43.5$ min, and there are merely nonsignificant differences after $t = 100$ min.

At Point 6 around $t = 47$ min, there is a 3-cm difference in water level for the peak flow. The highest water level of +27.08 m is computed using 695×288 cells, whereas the use of $2{,}779 \times 1{,}152$ cells leads to a highest water level of +27.11 m. Meanwhile, a significant difference is shown for the velocity. NUFSAW2D with 695×288 and $2{,}779 \times 1{,}152$ cells computes the highest velocities of 1.27 and 0.96 m/s, respectively. This is due to the fact that in the wet–dry mechanisms, more cells were captured and classified as a wet cell when $2{,}779 \times 1{,}152$ cells are used. However, the authors cannot determine which one is the more accurate result for this case because no observed results were provided. Nevertheless, it has been shown that NUFSAW2D is very stable and capable of simulating very shallow water on a very rough bed, which is in accordance with the results of Neelz and Pender (2013). In Fig. 12, the inundation area in 2D view is presented using $2{,}779 \times 1{,}152$ cells, showing that the result is in line with the aforementioned publication.

For Sandstorm with 600,000 steps, NUFSAW2D needs 52,215 s (for 1a with one core) and 59,988 s (for 1f with 16 cores) giving performance metrics of 2.3 and 2 Mcell/s/core, respectively. For CoolMUC-3 with 1,200,000 steps, NUFSAW2D requires 184,615 s (for 2a with 1 core) and 210,526 s (for 2i with 64 cores) giving the performance metrics of 0.65 and 0.57 Mcell/s/core, respectively. Similar to Case 1, NUFSAW2D achieves 1.81 Gflop/s/core for Sandstorm (31% of the TPP) and 0.54 Gflop/s/core for CoolMUC-3 (21% of the TPP). Fig. 13 shows the parallel efficiency of the model for weak and strong scaling with the mesh configurations similar to Case 1. Again, NUFSAW2D scales very well with efficiencies more than 98% (eight cores) and 87% (16 cores) for Sandstorm and up to 97% (56 cores) and 88% (64 cores) for CoolMUC-3. Excellent speed-up factors are achieved as well, being almost linear up to 56 cores (for 2i).

Investigation of Efficiency for each Thread

It was shown from Cases 1–2 that NUFSAW2D can achieve a sufficient average parallel efficiency of 87% (16 cores) and 88% (64 cores). To investigate the performance losses of 12%–13%, the CPU time for each thread is measured separately for each edge-driven and cell-driven levels. For example, according to Algorithm 4, for each single level of the asterisk, the CPU time of Lines 13–18 and of Lines 20–22 are measured (both before !$omp barrier). For this, Case 2 is selected, for which the first 1 h (120,000 calculation steps) is simulated using 1f (Sandstorm) and 2i (CoolMUC-3).

The results of WDLB and SLB are presented in Table 1 only for Sandstorm for simplicity. The dimensionless time is calculated by dividing the CPU time of each thread by the minimum value for each level; ideally, the value must be 1. One can see in Figs. 14 and 15

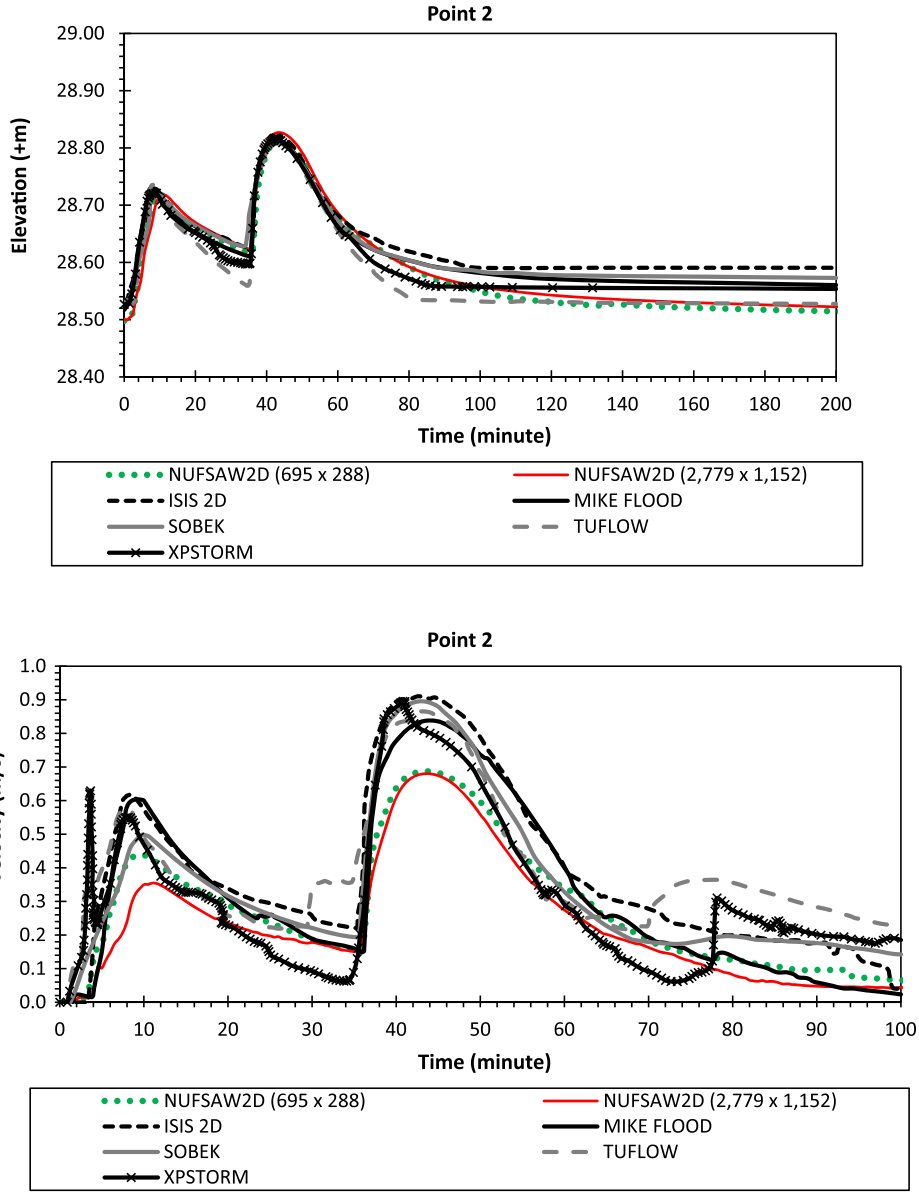


Fig. 10. Case 2: comparison between NUFSAW2D using 695 × 288 and 2,779 × 1,152 cells and the other models at Point 2.

that each thread gives almost similar values using the WDLB for both edge-driven and cell-driven levels. Meanwhile, some threads suffer from load imbalances quite significantly using the SLB. This shows that for a wet–dry-dominated case like this, the load distributions are not well-distributed due to the different complexity levels of the algorithm, which always change during runtime. The proposed WDLB is able to tackle such load imbalances, giving more efficient computations.

The efficiencies of 87% (16 cores) and 88% (64 cores) are the comparative values obtained by a comparison with one core for an unbiased measurement. Because Figs. 14–15 show relatively similar load distributions among threads, the performance losses of 12%–13% are thus not caused by an inefficient load distribution, but this is probably due to the nonuniform memory access (NUMA) effects, which depend on the memory location relative to the processor. In NUMA, its own local memory can be accessed by a processor faster than nonlocal memory shared between processors.

Fig. 16 shows a comparison between the WDLB and SLB for weak and strong scaling. For Sandstorm, using SLB with more than four cores starts producing lower performance metrics and speed-up factors. Meanwhile, the performance metrics and speed-up for CoolMUC-3 start decreasing using more than 12 cores with SLB. This shows that the WDLB can increase the performance of the SLB about 19%–20% for a high-load-imbalance case due wet–dry problems like this.

Conclusion

A shallow-water code NUFSAW2D has been presented using an edge-based CCFV method to solve the 2D SWEs with the OpenMP parallelization technique. The model is spatially and temporally second-order accurate, well-balanced, and able to efficiently simulate flood cases with wet–dry problems. Two main levels were presented in the edge-based data structure: edge-driven and cell-driven levels. The edge-driven level consists of the fluxes calculations for all edges including the convective and bed-slope fluxes. Meanwhile, the cell-driven level consists of the calculations for

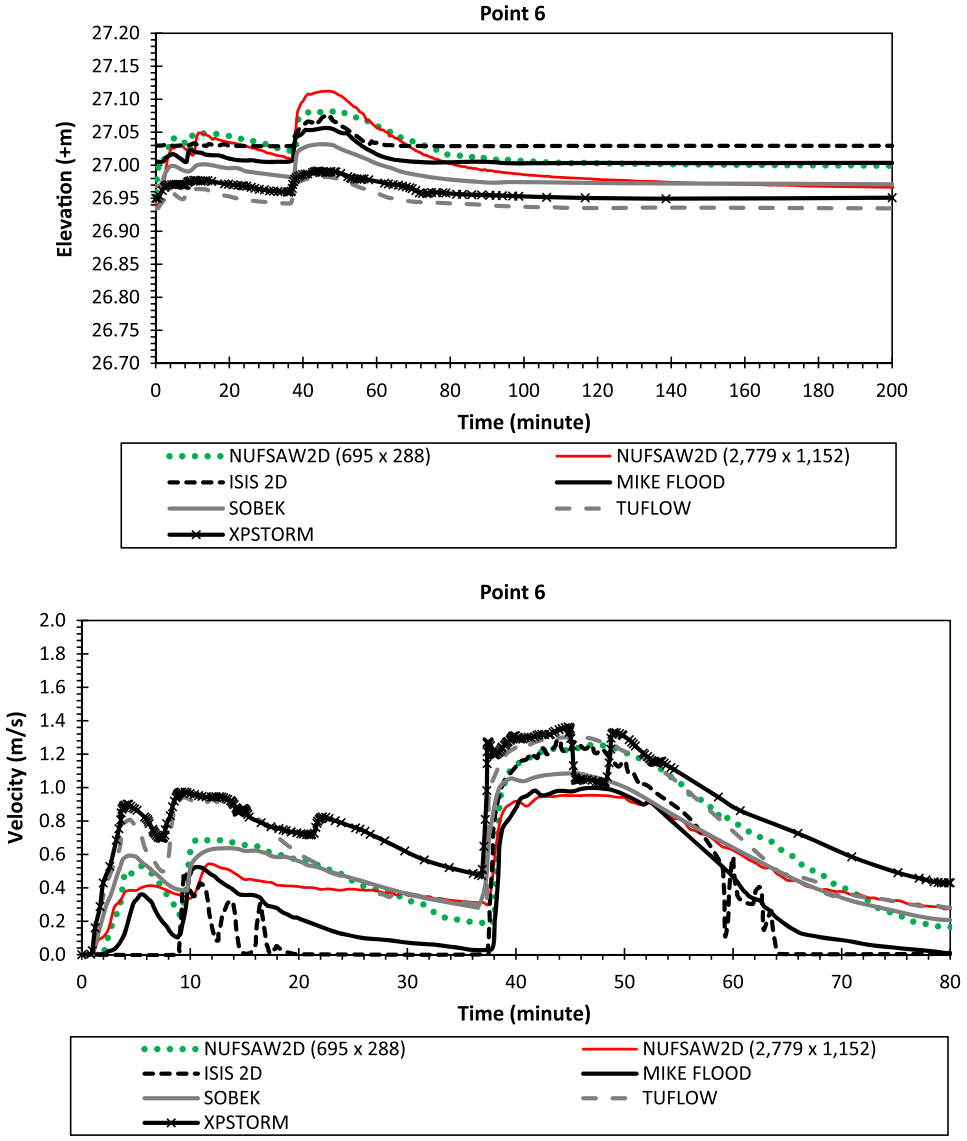


Fig. 11. Case 2: comparison between NUFSAW2D using 695 × 288 and 2,779 × 1,152 cells and the other models at Point 6.

summing both the convective and bed-slope fluxes with the proper signs from all the corresponding edges and using rainfall/infiltration and friction terms from the previous time level.

A simple cell–edge reordering strategy was proposed that provides an index relationship between edge and cell so that better memory access was achieved in both edge-driven and cell-driven levels and random memory access could be avoided. This strategy was highly beneficial to help the compiler to exploit the instruction pipelining and parallelization of the model. No difficulty was found in implementing this cell–edge reordering strategy because it took less than 0.3 s to reorder 3.2/6.4 million cells using 16/64 cores and less than 0.012 s to reorder 200,000 cells using a single core. This cell–edge reordering strategy is essentially a preprocessing strategy that can be applied to any CCFV scheme. Because such a colocated scheme is very flexible, it is possible to apply this strategy and the WDLB to unstructured meshes using indirect array indexing. A good scalability can be ensured at the expense of (more) CPU time than that of the structured meshes because knowing array patterns for such meshes might not be a simple task.

In addition to better memory access, it has been shown that the cell–edge reordering strategy proposed could help ease the implementation of the WDLB for wet–dry problems. The proposed WDLB was shown to be easily implemented and has been proven able to sufficiently achieve good parallel efficiency as well. For Sandstorm using up to 3.2 million cells (6.4 million edges), excellent parallel efficiencies of more than 98% (eight cores) and 87% (16 cores) for the weak scaling, and the speed-up factors of 7.8 (eight cores) and 13.95 (16 cores) for the strong scaling were achieved. Meanwhile, for CoolMUC-3 with up to 6.4 million cells (12.8 million edges), NUFSAW2D obtained efficiencies of 97% (56 cores) and 88% (64 cores) for the weak scaling, and the speed-up factors of 54 (56 cores) and 55.8 (64 cores) for the strong scaling. NUFSAW2D was able to achieve 1.81 Gflop/s/core for Sandstorm (31% of the TPP) and 0.54 Gflop/s/core for CoolMUC-3 (21% of the TPP). An investigation of the performance losses of 12%–13% for 16/64 cores was performed, which showed a relatively similar load distribution among threads. It could therefore be concluded such losses were not caused by an inefficient load distribution but were probably due to the NUMA effects. It was also investigated that the WDLB could gain the performance of the SLB about 19%–20%.

With both the cell–edge reordering strategy and WDLB, quite efficient performance metrics of 2–2.3 and 0.57–0.65 Mcell/s/core could be achieved by NUFSAW2D considering the second-order

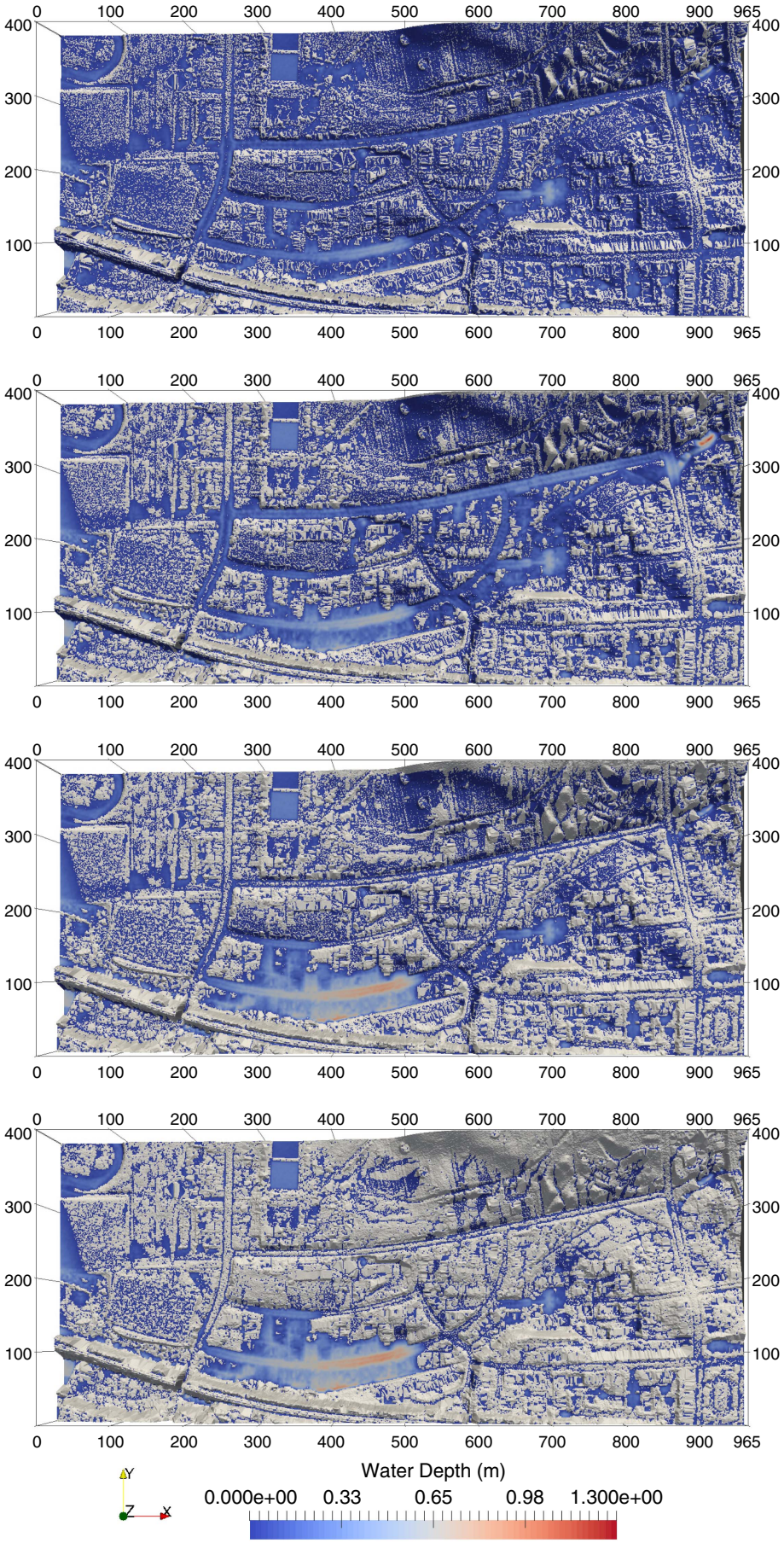


Fig. 12. Case 2: two-dimensional view of inundation area at 15, 45, 90, and 300 min using 2,779 × 1,152 cells.

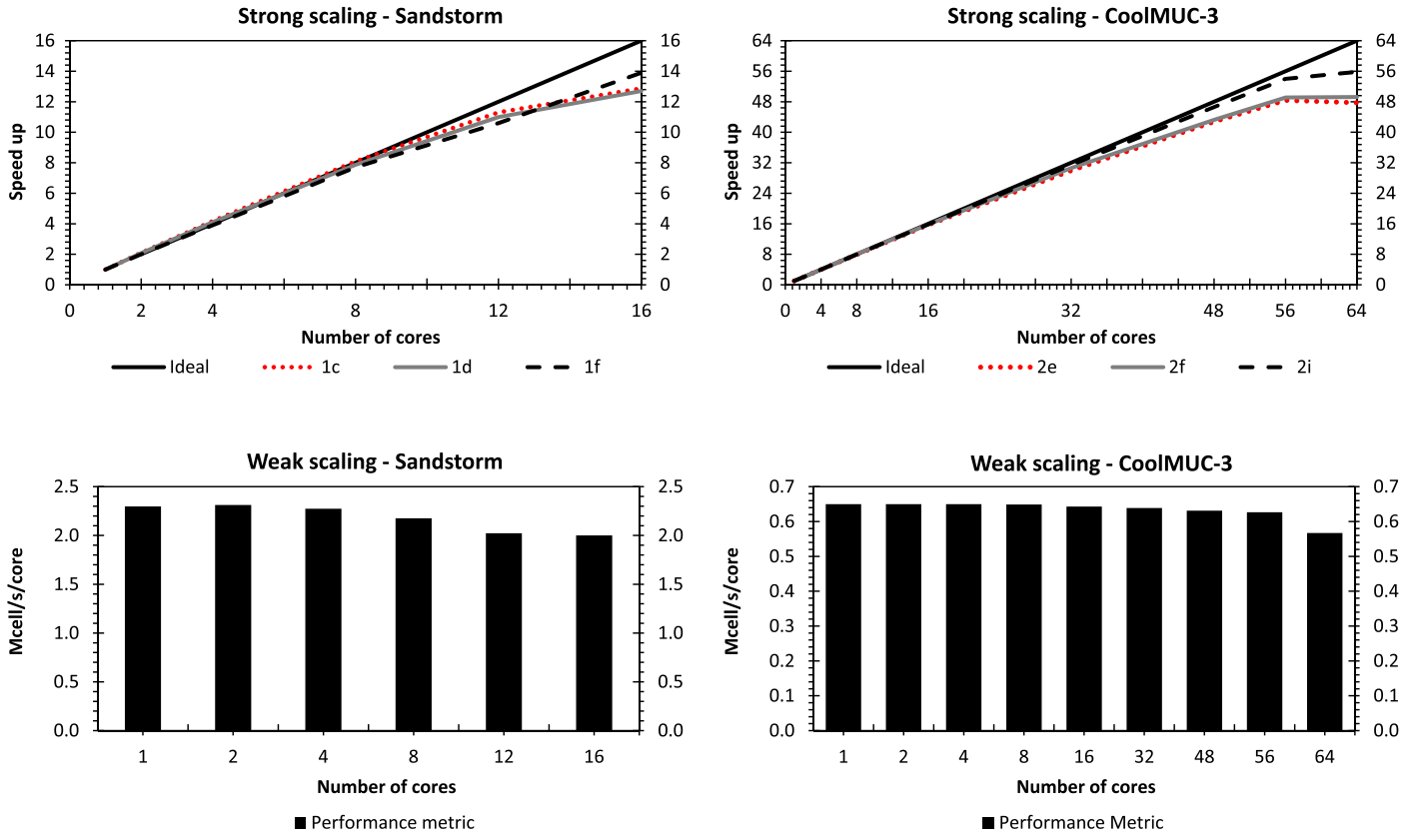


Fig. 13. Case 2: weak and strong scaling.

Table 1. Summary of average CPU time for edge-driven and cell-driven levels (Sandstorm)

Thread ID	CPU time (ms)		Dimensionless time	
	Edge-driven level (WDLB/SLB)	Cell-driven level (WDLB/SLB)	Edge-driven level (WDLB/SLB)	Cell-driven level (WDLB/SLB)
0	36.832/37.981	18.329/19.065	1.008/1.066	1.001/1.069
1	36.700/37.086	18.544/18.320	1.004/1.041	1.012/1.027
2	36.940/35.859	18.586/18.248	1.011/1.007	1.015/1.023
3	36.732/38.168	18.628/18.966	1.005/1.072	1.017/1.064
4	36.763/35.799	18.582/18.235	1.006/1.005	1.014/1.023
5	37.073/36.304	18.694/18.938	1.014/1.019	1.021/1.062
6	37.151/35.619	18.639/17.833	1.017/1.000	1.018/1.000
7	36.945/36.557	18.508/18.169	1.011/1.026	1.010/1.019
8	36.933/36.089	18.413/18.339	1.011/1.013	1.005/1.028
9	36.984/36.098	18.350/18.156	1.012/1.013	1.002/1.018
10	37.083/35.935	18.450/18.150	1.015/1.009	1.007/1.018
11	37.119/36.375	18.429/18.541	1.016/1.021	1.006/1.040
12	36.773/36.307	18.317/18.697	1.006/1.019	1.000/1.048
13	37.000/35.671	18.421/18.052	1.012/1.001	1.006/1.012
14	36.548/36.444	18.404/19.069	1.000/1.023	1.005/1.069
15	36.876/37.630	18.365/18.460	1.009/1.056	1.003/1.035
MIN	36.548/35.619	18.317/17.833	—	—

Note: MIN = minimum.

spatial and temporal accuracy employed as well as the double-precision arithmetic (64-bit) used. Higher performance metrics of 5.7–6 and 1.7–2 Mcell/s/core can be achieved if, for example, NUFSAW2D is employed using a first-order spatial and temporal model in the double-precision arithmetic. However, results of

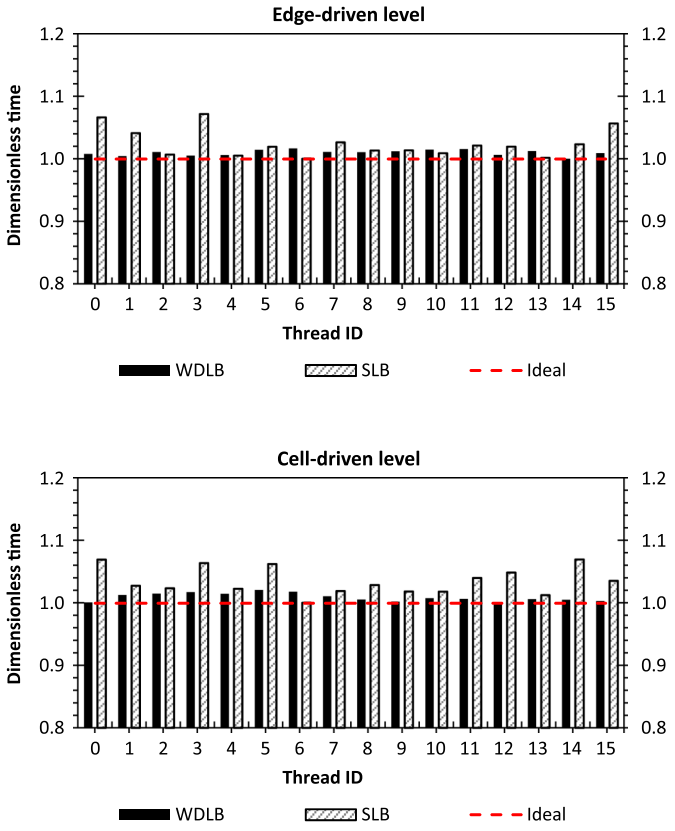


Fig. 14. Case 2: load distributions for each thread (Sandstorm).

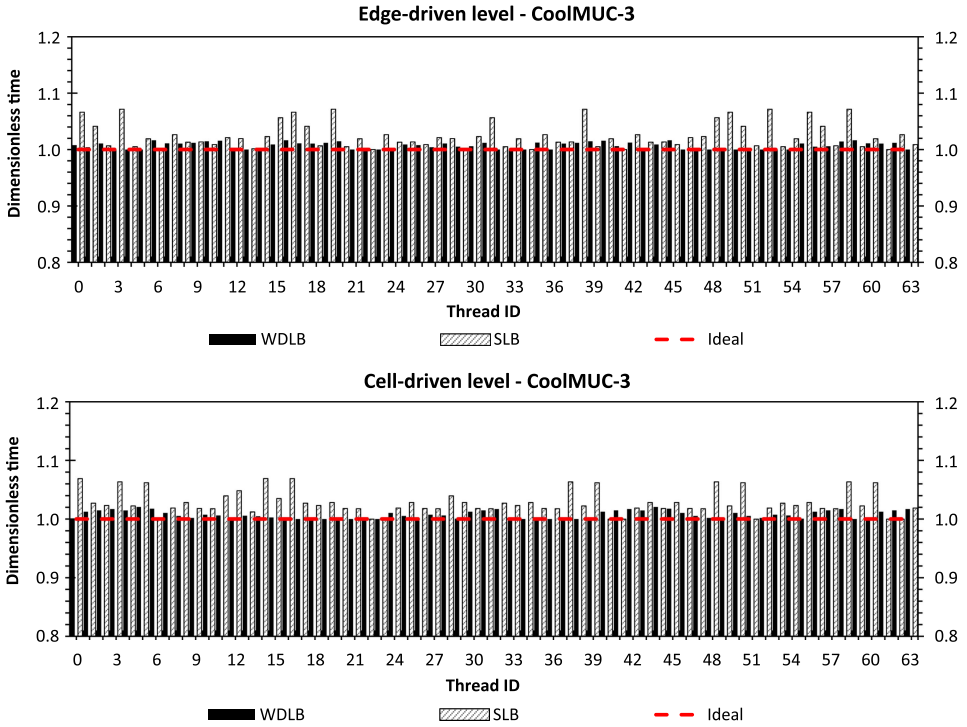


Fig. 15. Case 2: load distributions for each thread (CoolMUC-3).

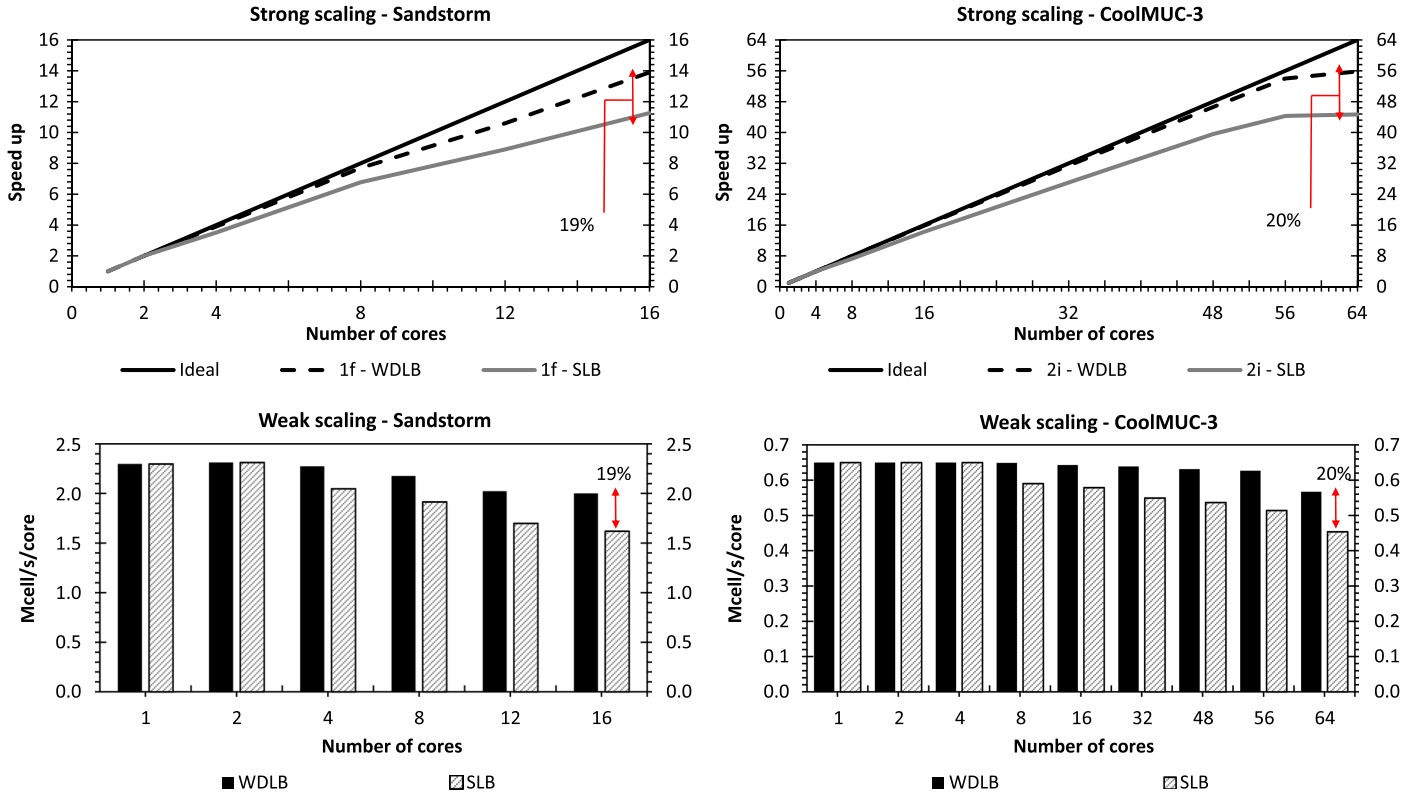


Fig. 16. Case 2: performance comparison between WDLB and SLB.

first-order spatial models may be too diffusive for general applications, and stability issues may occur using first-order temporal models. If the single-precision arithmetic (32-bit) is preferred, up to two times higher performance metrics can even be achieved by NUFSAW2D. However, a problem in terms of a mass-balance error may appear using the single-precision arithmetic. A similar phenomenon was also investigated by Smith and Liang (2013), who recommended the double-precision arithmetic for flood modeling.

The cases considered here were 2D problems. For more complex applications, say 3D baroclinic/nonhydrostatic cases including large-size problems, OpenMP may not be appropriate anymore for large-scale parallelization; thus, MPI should be considered. For an (extremely) large number of cores, using pure OpenMP will suffer from NUMA effects. Here this phenomenon was visible in wet–dry problems due to the data access by a processor, which is faster than the nonlocal memory shared between processors, especially when applying the WDLB. The authors assumed using pure MPI tends to be imbalanced due to inhomogeneous kernel execution, and the communication costs (for the WDLB) also become more significant. For this, the first author is currently working for a hybrid parallelization combining OpenMP and MPI where the cell–edge reordering strategy and the WDLB can easily be applied together with any space-filling-curve so that load imbalance issues for simulations with thousand cores can be tackled more efficiently. Simulations with AMR such as those of LeVeque et al. (2011) and Berger et al. (2011) using a hybrid parallelization technique would also be a challenging task to be investigated.

Acknowledgments

The first author gratefully acknowledges the DAAD (German Academic Exchange Service) for supporting the research in the scope of the research grants/doctoral programs in Germany for 2015–2016 (Grant No. 57129429). The authors would also like to thank Professor Michael Manhart and Florian Mintgen from the Chair of Hydromechanics, Technical University of Munich, for invaluable discussions on the topic and for providing data concerning Case 2. Suggestions of Professor Ernst Rank and helps of Christoph Ertl from the Chair for Computation in Engineering, Technical University of Munich, are appreciated. The compute and data resources provided by the Leibniz Supercomputing Centre are acknowledged. The authors are also grateful to all the anonymous reviewers who provided many constructive comments and suggestions.

Notation

The following symbols are used in this paper:

c_f = friction factor;
$\mathbf{F}$ = vector of convective terms in x-direction;
$\mathbf{G}$ = vector of convective terms in y-direction;
g = gravity acceleration;
h = depth;
I = infiltration;
N = total number of edges for a cell;
n_m = Manning coefficient;
R = rainfall;
$\mathbf{S}_b$ = vector of bed-slope terms;
$\mathbf{S}_f$ = vector of friction terms;
$\mathbf{S}_w$ = vector of rainfall/infiltration;
t = time;
u = velocity in x direction;
v = velocity in y direction;
$\mathbf{W}$ = vector of conservative variables;
z = bed elevation;
ΔL = length of edge;
Δt = time step;
θ = implicitness parameter; and
Ω = discretized domain.

References

Aoyama, Y., and J. Nakano. 1999. *RS/6000 SP: Practical MPI programming*. Austin, TX: IBM, International Technical Support Organization.

Audusse, E., F. Bouchut, M.-O. Bristeau, R. Klein, and B. Perthame. 2004. "A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows." *SIAM J. Sci. Comput.* 25 (6): 2050–2065. https://doi.org/10.1137/S1064827503431090.

Begnudelli, L., B. F. Sanders, and S. F. Bradford. 2008. "Adaptive Godunov-based model for flood simulation." *J. Hydraul. Eng.* 134 (6): 714–725. https://doi.org/10.1061/(ASCE)0733-9429(2008)134:6(714).

Berger, M. J., D. L. George, R. J. LeVeque, and K. T. Mandli. 2011. "The Geoclaw software for depth-averaged flows with adaptive refinement." *Adv. Water. Resour.* 34 (9): 1195–1206. https://doi.org/10.1016/j.advwatres.2011.02.016.

Brufau, P., P. Garcia-Navarro, and M. E. Vazquez-Cendon. 2002. "A numerical model for the flooding and drying of irregular domains." *Int. J. Numer. Methods Fluids* 39 (3): 247–275. https://doi.org/10.1002/fld.285.

Buffard, T., and S. Clain. 2010. "Monoslope and multislope MUSCL methods for unstructured meshes." *J. Comput. Phys.* 229 (10): 3745–3776. https://doi.org/10.1016/j.jcp.2010.01.026.

Chertock, A., A. Kurganov, and Y. Liu. 2014. "Central-upwind schemes for the system of shallow water equations with horizontal temperature gradients." *Numer. Math.* 127 (4): 595–639. https://doi.org/10.1007/s00211-013-0597-6.

CiE-Sandstorm. 2018. "Sandstorm-cluster of the chair for computation in engineering—Technical University of Munich." Accessed March 1, 2018. https://sandstorm.inf.bauwesen.tu-muenchen.de/wiki/index.php/Sandstorm-Cluster_(En).

Delis, A. I., and I. K. Nikolos. 2013. "A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered finite volumes with application to shallow water dynamics." *Int. J. Numer. Methods Fluids* 71 (5): 584–633. https://doi.org/10.1002/fld.3674.

Delis, A. I., I. K. Nikolos, and M. Kazolea. 2011. "Performance and comparison of cell-centered and node-centered unstructured finite volume discretizations for shallow water free surface flow." *Arch. Comput. Method E.* 18 (1): 57–118. https://doi.org/10.1007/s11831-011-9057-6.

Duran, A. 2015. "A robust and well-balanced scheme for the 2D Saint-Venant system on unstructured meshes with friction source term." *Int. J. Numer. Methods Fluids* 78 (2): 89–121. https://doi.org/10.1002/fld.4011.

Ginting, B. M. 2017. "A two-dimensional artificial viscosity technique for modelling discontinuity in shallow water flows." *Appl. Math. Model.* 45: 653–683. https://doi.org/10.1016/j.apm.2017.01.013.

Ginting, B. M. 2019. "Central-upwind scheme for 2D turbulent shallow flows using high-resolution meshes with scalable wall functions." *Comput. Fluids* 179: 394–421. https://doi.org/10.1016/j.compfluid.2018.11.014.

Ginting, B. M., and R.-P. Mundani. 2018. "Artificial viscosity technique: A Riemann-solver-free method for 2D urban flood modelling on complex topography." In *Advances in hydroinformatics*, edited by P. Gourbesville, J. Cunge, and G. Caignaert. Singapore: Springer.

Ginting, B. M., R.-P. Mundani, and E. Rank. 2018. "Parallel simulations of shallow water solvers for modelling overland flows." In Vol. 3 of *Proc., 13th Int. Conf. on Hydroinformatics*, edited by G. La Loggia, G. Freni, V. Puleo, and M. De Marchis, 788–799. Palermo, Italy: University Campus of Palermo.

Hou, J., Q. Liang, F. Simons, and R. Hinkelmann. 2013. "A 2D well-balanced shallow flow model for unstructured grids with novel slope source term treatment." *Adv. Water. Resour.* 52: 107–131. https://doi.org/10.1016/j.advwatres.2012.08.003.

Hubbard, M. E. 1999. "Multidimensional slope limiters for MUSCL-type finite volume schemes on unstructured grids." *J. Comput. Phys.* 155 (1): 54–74. https://doi.org/10.1006/jcph.1999.6329.

Kawahara, M., and T. Umetsu. 1986. "Finite element method for moving boundary problems in river flow." *Int. J. Numer. Methods Fluids* 6 (6): 365–386. https://doi.org/10.1002/fld.1650060605.

Kurganov, A., and G. Petrova. 2007. "A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system." *Commun. Math. Sci.* 5 (1): 133–160. https://doi.org/10.4310/CMS.2007.v5.n1.a6.

Lacasta, A., M. Morales-Hernandez, J. Murillo, and P. Garcia-Navarro. 2015. "GPU implementation of the 2D shallow water equations for the simulation of rainfall/runoff events." *Environ. Earth Sci.* 74 (11): 7295–7305. https://doi.org/10.1007/s12665-015-4215-z.

LeVeque, R. J., D. L. George, and M. J. Berger. 2011. "Tsunami modelling with adaptively refined finite volume methods." *Acta. Numer.* 20: 211–289. https://doi.org/10.1017/S0962492911000043.

Liang, Q., and A. G. L. Borthwick. 2009. "Adaptive quadtree simulation of shallow flows with wet-dry fronts over complex topography." *Comput. Fluids* 38 (2): 221–234. https://doi.org/10.1016/j.compfluid.2008.02.008.

Liu, J. Y., M. R. Smith, F. A. Kuo, and J. S. Wu. 2015. "Hybrid OPENMP/AVX acceleration of a split HLL finite volume method for the shallow water and Euler equations." *Comput. Fluids* 110: 181–188. https://doi.org/10.1016/j.compfluid.2014.11.011.

Lobeiras, J., M. Vinas, M. Amor, B. B. Fraguela, M. Arenaz, J. A. Garcia, and M. J. Castro. 2013. "Parallelization of shallow water simulations on current multi-threaded systems." *Int. J. High Perform. Comput. Appl.* 27 (4): 493–512. https://doi.org/10.1177/1094342012464800.

LRZ (Leibniz-Rechnenzentrum). 2018. "Leibniz supercomputing Centre of the Bavarian Academy of Sciences and Humanity." Accessed July 1, 2018. https://www.lrz.de.

Meister, O., K. Rahnema, and M. Bader. 2017. "Parallel memory-efficient adaptive mesh refinement on structured triangular meshes with billions of grid cells." *ACM T. Math. Software* 43 (3): 1–27. https://doi.org/10.1145/2947668.

Mohammadian, A., and D. Y. Le Roux. 2006. "Simulation of shallow flows over variable topographies using unstructured grids." *Int. J. Numer. Methods Fluid.* 52 (5): 473–498. https://doi.org/10.1002/fld.1167.

Neal, J. C., T. J. Fewtrell, and M. Trigg. 2009. "Parallelisation of storage cell flood models using OpenMP." *Environ. Modell. Software* 24 (7): 872–877. https://doi.org/10.1016/j.envsoft.2008.12.004.

Neelz, S., and G. Pender. 2013. *Benchmarking the latest generation of 2D hydraulic modelling packages*. Bristol, UK: Environment Agency.

Sanders, B. F., J. E. Schubert, and R. L. Detwiler. 2010. "Parbrezo: A parallel, unstructured grid, Godunov-type, shallow-water code for high-resolution flood inundation modeling at the regional scale." *Adv. Water Resour.* 33 (12): 1456–1467. https://doi.org/10.1016/j.advwatres.2010.07.007.

Shirkhani, H., A. Mohammadian, O. Seidou, and A. Kurganov. 2016. "A well-balanced positivity-preserving central-upwind scheme for shallow water equations on unstructured quadrilateral grids." *Comput. Fluids* 126: 25–40. https://doi.org/10.1016/j.compfluid.2015.11.017.

Smith, L. S., and Q. Liang. 2013. "Towards a generalised GPU/CPU shallow-flow modelling tool." *Comput. Fluids* 88: 334–343. https://doi.org/10.1016/j.compfluid.2013.09.018.

Tan, W. Y. 1992. *Shallow water hydrodynamics: Mathematical theory and numerical solution for a two-dimensional system of shallow-water equations*. Beijing: Water & Power Press.

van Albada, G. D., B. van Leer, and W. W. Roberts. 1982. "A comparative study of computational methods in cosmic gas dynamics." *Astron. Astrophys.* 108 (1): 76–84.

van Leer, B. 1979. "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method." *J. Comput. Phys.* 32 (1): 101–136. https://doi.org/10.1016/0021-9991(79)90145-1.

Wittmann, R., H.-J. Bungartz, and P. Neumann. 2017. "High performance shallow water kernels for parallel overland flow simulations based on fullSWOF2D." *Comput. Math. Appl.* 74 (1): 110–125. https://doi.org/10.1016/j.camwa.2017.01.005.

Zhang, S., Z. Xia, R. Yuan, and X. Jiang. 2014. "Parallel computation of a dam-break flow model using OpenMP on a multi-core computer." *J. Hydrol.* 512: 126–133. https://doi.org/10.1016/j.jhydrol.2014.02.035.

Paper 2

B.M. Ginting. Central-Upwind Scheme for 2D Turbulent Shallow Flows using High-Resolution Meshes with Scalable Wall Functions. *Computers & Fluids*, 179:394–421, 2019. https://dx.doi.org/10.1016/j.compfluid.2018.11.014

Computers and Fluids 179 (2019) 394–421

Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

Benchmark solutions

Central-upwind scheme for 2D turbulent shallow flows using high-resolution meshes with scalable wall functions

Bobby Minola Ginting

Chair for Computation in Engineering, Technical University of Munich Arcisstr. 21, Munich D-80333, Germany

ARTICLE INFO

Article history:
Received 12 July 2018
Revised 31 October 2018
Accepted 13 November 2018
Available online 14 November 2018

Keywords:
Central-upwind
High-resolution meshes
Turbulence
Scalable wall functions
Shallow flow

ABSTRACT

In this paper, the Reynolds-averaged Navier Stokes equations supplemented by the algebraic stress model are solved using the central-upwind scheme for simulating 2D turbulent shallow flows. The model is of spatially and temporally second-order accurate. To increase the accuracy, high-resolution meshes up to 3.4 million cells (6.8 million edges) are used. Consequently, a strategy combining the hydrostatic and topography reconstructions and the scalable wall functions – is proposed to accurately simulate wet–dry phenomena near the interfaces (moving boundary geometries) thus ensuring a proper calculation for the turbulence properties. This strategy has been proven to be accurate and to not deteriorate the results for such very fine meshes thus giving flexibility to users in generating meshes.

1. Introduction

In reality, almost all flows in the scope of hydraulics are turbulent where the true flow conditions are highly irregular, fluctuating, and always exhibit a three-dimensional eddying motion [1]. It remains a challenge to find a proper numerical model to comprehensively understand such turbulence effects on flows. For this, 3D turbulence models, e.g. Reynolds-averaged Navier Stokes (RANS), Large-Eddy Simulation (LES), or Direct Numerical Simulation (DNS) models, may be the choice. Using such 3D models, the complex turbulent characteristics may be better understood than if 2D models are employed. However, for some specific cases – which are of (absolute) convective instabilities or where approximate flow predictions are quite insensitive to bed shear and secondary effects thus the dispersive part plays a non-significant role – the 2D turbulence models still become favorable as much less computational time is required but they keep ensuring sufficiently accurate results. In [2,3] where the vertical mixing was instantaneous, it was even shown that the 3D turbulence model gave poorer representations than did the 2D one. Some successful works employing the 2D turbulence models for weakly-dispersive flows are noted here [4–11].

Due to their simplicity, RANS models are in general more powerful than LES or DNS models considering the advent of computers [1] and therefore become more popular for practical purposes in environmental hydraulic modeling. A turbulence model is required in RANS models to calculate the eddy viscosity which is not a real fluid property but an artificially introduced one acting as new stresses on the water in addition to the viscous stresses given by the kinematic viscosity. For shallow flows, the simplest turbulence model might be the parabolic eddy viscosity (PEV) model, e.g. applied by Ding et al. [12], or the mixing length (ML) model, e.g. used by Stansby [13], which assumes that the turbulent length scale depends on the water depth. In the PEV model, the turbulence is generated by bed-friction, whereas the horizontal strain-rate production of turbulence is taken into account in the ML model. A more advanced turbulence model is the $\kappa - \epsilon$ model, proposed by Rodi [14] and then applied, e.g. in [4,6,7]. All the aforementioned turbulence models are based on the Boussinesq's approximation [15] which assumes an isotropic eddy viscosity for the Reynolds stresses. This becomes the main weakness of these models [16]. As an alternative, the algebraic stress model (ASM) is used here which can be regarded as a non-linear extension of eddy viscosity models [6].

Especially with regard to finite volume models, Riemann solvers e.g. Roe, HLL, and HLLC schemes were commonly used in the past decade for turbulent shallow flow simulations, e.g. Roe scheme in [5,6], HLL scheme in [8] and HLLC scheme in [7,9,17]. Meanwhile, another robust (Riemann-solver-free) scheme such as the central-upwind (CU) scheme was mainly addressed for inviscid shallow flow simulations, see [18–24], among others – and to the best of our knowledge, this scheme was never used previously to model turbulent shallow flows. Ginting and Mundani [25] showed the CU scheme was averagely 1.3× cheaper than the HLLC scheme using the first-order finite volume scheme – and in our recent work

E-mail address: bobbyminola.ginting@tum.de

https://doi.org/10.1016/j.compfluid.2018.11.014
0045-7930/

(*unpublished*), it was noted that the CU scheme became averagely 1.2× cheaper than both the Roe and HLLC schemes using the second-order finite volume scheme, where the Roe and HLLC schemes required almost similar CPU time.

In general, using sufficiently fine grid resolution for RANS models can increase accuracy, since such a resolution may minimize the effects of numerical uncertainties from the turbulence properties [26]. However, the results sensitivity to the grid resolution should be taken into careful consideration. We illustrate this by a well-known example: recirculating wakes flow around a conical island with a surface-piercing condition [2]. For this, two major problems appear when performing simulations using high-resolution meshes. The first is the effects of the wet–dry problems existing around the conical island become significant on the numerical results of the wakes characteristics behind the island. This is because the wet–dry phenomena deal with the moving boundary geometries around the island which obviously affect the turbulence properties. For instance, when using a RANS model with the $\kappa-\epsilon$ or the ASM model, one requires a special treatment near the boundary geometries to calculate the turbulent kinetic energy. This normally relates to the standard wall functions (StWF) that impose several limitations in terms of near-wall mesh resolution. Typically, the near-wall mesh spacing must satisfy the criterion of $y^+ \geq 11.067$ – where y^+ is a dimensionless wall distance. Consequently, the centers of the first boundary cells must be placed in the logarithmic region. The second problem emerges from this situation. Such a criterion may be contravened by the use of very fine near-wall meshes – and particularly for wet–dry problems, this becomes even more complex since at the initial step one does not know precisely the boundary geometry around the conical island.

Karimi et al. [27] studied the effect of mesh size by comparing the StWF and the enhanced wall treatments. The latter required y^+ as low as possible and the meshes near the wall were thus adaptively refined. This adaptive technique may become a main issue in simultaneously preserving the *C*-property and the mass conservation during runtime for inviscid shallow flow simulations [28]. Even though a solution has been proposed and was successful, the extended application in turbulence modeling may turn out to be complex. Also, the grid independent study done in [27] is too complex for practical purposes especially when dealing with wet–dry problems.

We try to summarize some findings related to the aforementioned example according to the previous publications [2,3,10,13,29]. Firstly, it was stated that the accuracy of the numerical model was not much affected by reducing the mesh size. Secondly, the impacts of the wet–dry phenomena around the conical island on the wakes flows were not of particular interest and the special treatments for this problem were therefore never explained in detail. However, we believe the mesh size plays a significant role and therefore the wet–dry phenomena effects around the conical island become significant on the accuracy. This becomes our main focus in this paper. To this end, a strategy – combining the hydrostatic and topography reconstructions and the scalable wall functions (ScWF) to accurately simulate wet–dry phenomena at the interfaces, e.g. near the conical island, and to simultaneously compute the proper values of the turbulence properties around such interfaces and other wall boundaries – is proposed. With very fine meshes, the ScWF become beneficial and more flexible for such a moving boundary geometry due to wet–dry interfaces. We will show that the proposed strategy can avoid the limitations of the StWF thus allowing users to generate meshes without imposing a lower limit from the StWF.

For the sake of simplicity, we employ here high-resolution structured meshes to obtain accurate solutions. Indeed, structured meshes have a shortcoming for representing real (complex) geometries and unstructured meshes can thus be used to capture such complex geometries more properly. However, generating millions of unstructured cells might not be a simple task. Note that none of these mesh types can always guarantee to accurately capture moving boundary geometries at wet–dry interfaces unless very fine meshes are used. Although we only present computations with structured meshes here, our proposed strategy can obviously be employed for unstructured meshes.

Previously, some notable works had dealt with wet–dry problems for turbulent shallow flows. Wu [4] used a threshold value (e.g. 0.02 m) to judge drying and wetting; when the depth is below this value, the cell is dry and the StWF are applied at the wet–dry edge. Yu and Duan [7] added an extra flux to the bed-slope terms calculations to determine wet–dry interfaces and then applied the StWF. Cea et al. [6] employed the upwind discretization for the bed-slope terms to capture wet–dry interfaces, where the limit of $y^+ \approx 100$ was always kept. All these approaches still exhibit the limitation of the wall functions and are thus still not suitable for the use of high-resolution meshes. In [4], only the wet–dry interfaces of the dry cells are treated with the StWF, whereas no wall functions are applied to the wet cells although such cells may have wet–dry interfaces. Also, the technique in [6] only works when using the Riemann solvers (the Roe scheme). The strategy proposed here is a Riemann-solver-free scheme computed separately from the convective fluxes, thus is applicable to any solver. We consider here four test cases to show the ability of our proposed strategy for simulating turbulent shallow flows with very fine meshes and wet–dry problems. Our in-house code NUFSAW2D (Numerical simUlation of Free surface ShAllow Water 2D) is used.

2. Governing equations

2.1. 2D RANS equations for shallow flows

Assuming negligible dispersive terms, the 2D RANS equations for shallow flows are expressed as

$$\frac{\partial Q}{\partial t}+\frac{\partial C_x}{\partial x}+\frac{\partial C_y}{\partial y}=\frac{\partial D_x}{\partial x}+\frac{\partial D_y}{\partial y}+S_b+S_f, \tag{1}$$

where Q are the conservative variables, C_x and C_y are the convective fluxes (or the convective terms), D_x and D_y are the diffusive fluxes (or the viscous terms) obtained in connection with the Boussinesq's assumption, S_b are the bed-slope terms, and S_f are the bed friction terms. Here, U are denoted as the primitive variables. All the matrices are given by

$$Q=\begin{bmatrix} h \\ hu \\ hv \end{bmatrix},\quad U=\begin{bmatrix} h \\ u \\ v \end{bmatrix},\quad C_x=\begin{bmatrix} hu \\ huu+\dfrac{gh^2}{2} \\ hvu \end{bmatrix},\quad C_y=\begin{bmatrix} hv \\ huv \\ hvv+\dfrac{gh^2}{2} \end{bmatrix},$$

$$D_x=\begin{bmatrix} 0 \\ 2h(\nu_e+\nu_t)\dfrac{\partial u}{\partial x}-\dfrac{2}{3}h\kappa \\ h(\nu_e+\nu_t)\left(\dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\right) \end{bmatrix},\quad D_y=\begin{bmatrix} 0 \\ h(\nu_e+\nu_t)\left(\dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\right) \\ 2h(\nu_e+\nu_t)\dfrac{\partial v}{\partial y}-\dfrac{2}{3}h\kappa \end{bmatrix},$$

$$S_b=\begin{bmatrix} 0 \\ -gh\dfrac{\partial z}{\partial x} \\ -gh\dfrac{\partial z}{\partial y} \end{bmatrix},\qquad S_f=\begin{bmatrix} 0 \\ -c_f u\sqrt{u^2+v^2} \\ -c_f v\sqrt{u^2+v^2} \end{bmatrix}, \tag{2}$$

where h, u, and v are the flow depth and velocities in x and y directions, respectively, g is the gravity acceleration, ν_e and ν_t are the kinematic viscosity and the eddy viscosity, respectively, κ is the kinetic energy, z is the bed contour, n_m is the Manning coefficient, and $c_f = g n_m^2 h^{-\frac{1}{3}}$.

2.2. 2D ASM equations

A turbulence model is required to compute ν_t and κ in Eq. (2). In this paper, the ASM is used being a non-linear extension of eddy viscosity models and here is correlated with the $\kappa - \epsilon$ model. First, the $\kappa - \epsilon$ model is expressed as

$$\frac{\partial \Phi}{\partial t} + \frac{\partial C_{\Phi,x}}{\partial x} + \frac{\partial C_{\Phi,y}}{\partial y} = \frac{\partial D_{\Phi,x}}{\partial x} + \frac{\partial D_{\Phi,y}}{\partial y} + S_{\kappa-\epsilon}. \tag{3}$$

For the energy dissipation rate ϵ, the conservative variables Φ, the primitive variables Y, the convective fluxes $C_{\Phi,x}$ and $C_{\Phi,y}$, the diffusive fluxes $D_{\Phi,x}$ and $D_{\Phi,y}$, and the turbulence source terms $S_{\kappa-\epsilon}$, are expressed as

$$\Phi = \begin{bmatrix} h\kappa \\ h\epsilon \end{bmatrix}, \quad \Upsilon = \begin{bmatrix} \kappa \\ \epsilon \end{bmatrix}, \quad C_{\Phi,x} = \begin{bmatrix} h\kappa u \\ h\epsilon u \end{bmatrix}, \quad C_{\Phi,y} = \begin{bmatrix} h\kappa v \\ h\epsilon v \end{bmatrix},$$

$$D_{\Phi,x} = \begin{bmatrix} \sigma_\kappa^{-1} h \nu_t \frac{\partial \kappa}{\partial x} \\ \sigma_\epsilon^{-1} h \nu_t \frac{\partial \epsilon}{\partial x} \end{bmatrix}, \quad D_{\Phi,y} = \begin{bmatrix} \sigma_\kappa^{-1} h \nu_t \frac{\partial \kappa}{\partial y} \\ \sigma_\epsilon^{-1} h \nu_t \frac{\partial \epsilon}{\partial y} \end{bmatrix}, \tag{4}$$

$$S_{\kappa-\epsilon} = \begin{bmatrix} P_h + P_{\kappa b} - h\epsilon \\ c_{\epsilon 1} \frac{\epsilon}{\kappa} P_h + P_{\epsilon b} - c_{\epsilon 2} h \frac{\epsilon^2}{\kappa} \end{bmatrix}.$$

The terms P_h, $P_{\kappa b}$, and $P_{\epsilon b}$ are defined as

$$P_h = h\nu_t \left(2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2 \right),$$
$$P_{\kappa b} = c_f^{-0.5} U_*^3, \quad P_{\epsilon b} = \frac{c_{\epsilon\Gamma}\, c_{\epsilon 2}\, c_\mu^{0.5}\, c_f^{-0.75}}{h}, \tag{5}$$

where $U_* = \sqrt{c_f\left(u^2 + v^2\right)}$ is the bed friction velocity. The eddy viscosity is computed by

$$\nu_t = c_\mu \left(\frac{\kappa^2}{\epsilon} \right). \tag{6}$$

All coefficients in Eqs. (4)–(6) are given by [14]

$$c_\mu = 0.09, \quad c_{\epsilon 1} = 1.44, \quad c_{\epsilon 2} = 1.92, \quad \sigma_\kappa = 1.0, \quad \sigma_\epsilon = 1.3, \quad c_{\epsilon\Gamma} = [1.8, 3.6]. \tag{7}$$

The ASM is now employed to recalculate P_h. Here this model is briefly presented; the complete explanation can be read in [5,6]. Only three components of the Reynolds stresses appear in shallow water flows: $\overline{u'^2}$, $\overline{u'v'}$, and $\overline{v'^2}$. A new formula P_h^* is applied to account for the production of turbulent energy due to horizontal velocity gradient expressed as

$$P_h^* = h\left[-\overline{u'^2}\left(\frac{\partial u}{\partial x}\right) - \overline{v'^2}\left(\frac{\partial v}{\partial y}\right) - \overline{u'v'}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \right] + \left[\frac{P_{uu,b} + P_{vv,b}}{2} \right], \tag{8}$$

where $P_{uu,b}$ and $P_{vv,b}$ account for the vertical production of turbulent energy due to bed shear.

All $\overline{u'^2}$, $\overline{u'v'}$, and $\overline{v'^2}$ are first calculated. Considering the constant $c_1 = 1.8$, a tensorial form can be written to calculate these stresses as

$$m_{ij} r_j = c_{11} b_i, \tag{9}$$

where $i = 1$–3, $j = 1$–3, $r_j = \left[\overline{u'^2}, \overline{v'^2}, \overline{u'v'}\right]$, $c_{11} = c_1 + \frac{P_h}{\epsilon} - 1$.

The matrix m_{ij} is defined as

$$m_{ij} = c_{11}\delta_{ij} - (1 - c_2)\frac{\kappa}{\epsilon} a_{ij}, \tag{10}$$

where $c_2 = 0.6$ and δ_{ij} is Kronecker delta. The matrices a_{ij} and b_i are given respectively by

$$\begin{bmatrix} -\frac{4}{3}\frac{\partial u}{\partial x} & \frac{2}{3}\frac{\partial v}{\partial y} & -\frac{4}{3}\frac{\partial u}{\partial y} + \frac{2}{3}\frac{\partial v}{\partial x} \\ -\frac{2}{3}\frac{\partial u}{\partial x} & -\frac{4}{3}\frac{\partial v}{\partial y} & -\frac{4}{3}\frac{\partial v}{\partial x} + \frac{2}{3}\frac{\partial u}{\partial y} \\ -\frac{\partial v}{\partial x} & -\frac{\partial u}{\partial y} & -\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \end{bmatrix},$$
$$\begin{bmatrix} \frac{2}{3}\kappa + \frac{\kappa}{\epsilon}\frac{(1-C_2)}{c_{11}}\left(\frac{2}{3}P_{uu,b} - \frac{1}{3}P_{vv,b}\right) \\ \frac{2}{3}\kappa + \frac{\kappa}{\epsilon}\frac{(1-C_2)}{c_{11}}\left(\frac{2}{3}P_{vv,b} - \frac{1}{3}P_{uu,b}\right) \\ \frac{\kappa}{\epsilon}\frac{(1-C_2)}{c_{11}}P_{uv,b} \end{bmatrix}. \tag{11}$$

The values of $P_{uu,b}$, $P_{vv,b}$, and $P_{uv,b}$ are computed by

$$P_{uu,b} = 2\frac{c_f u^2 |\mathbf{U}|}{h}, \quad P_{vv,b} = 2\frac{c_f v^2 |\mathbf{U}|}{h}, \quad P_{uv,b} = 2\frac{c_f uv |\mathbf{U}|}{h}, \tag{12}$$

where $|\mathbf{U}|$ can be computed by the Keulegan's law. For the sake of simplicity, it is assumed here to be similar to the depth-averaged velocity.

3. Numerical model

3.1. Spatial discretization

Integrating over a control cell Ω and applying the Gauss divergence theorem, both Eqs. (1) and (3) are now written, respectively, as

$$\iint_\Omega \frac{\partial Q}{\partial t} d\Omega + \oint_\Gamma \left(C_x + C_y - D_x - D_y\right) \cdot \vec{n}\, d\Gamma = \iint_\Omega \left(S_b + S_f\right) d\Omega, \tag{13}$$

$$\iint_\Omega \frac{\partial \Phi}{\partial t} d\Omega + \oint_\Gamma \left(C_{\Phi,x} + C_{\Phi,y} - D_{\Phi,x} - D_{\Phi,y}\right) \cdot \vec{n} d\Gamma = \iint_\Omega S_{\kappa-\epsilon}\, d\Omega, \tag{14}$$

where Γ denotes the line boundary of the control cell Ω and $\vec{n}$ is the unit normal vector pointing outward of the boundary. All values h, u, v, z, n_m, κ, and ϵ are defined at the center of each cell as a cell-centered finite volume (CCFV) method is used. Further, the water elevation is defined as $\eta = h + z$.

3.1.1. MUSCL linear reconstruction

First the spatial discretization is explained for the convective fluxes C_x, C_y, $C_{\Phi,x}$, and $C_{\Phi,y}$. For the sake of simplicity, F_x =

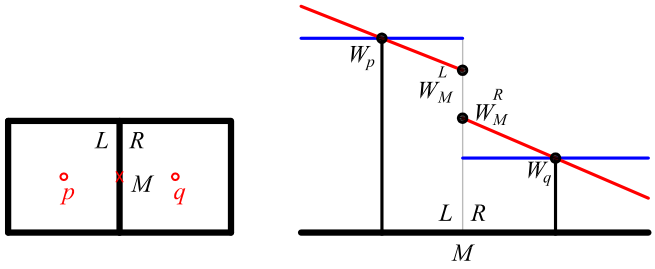


Fig. 1. Representation of the values of L and R states at midpoint M of edge i.

$[C_x, C_{\Phi,x}]$ and $F_y = [C_y, C_{\Phi,y}]$. The line integrals in Eqs. (13) and (14) are estimated by

$$\oint_\Gamma \Big(F_x + F_y\Big)\cdot \vec{n}\, d\Gamma \approx \sum_{i=1}^{N} \Big(F_x\, n_x + F_y\, n_y\Big)_i\, \Delta L_i, \tag{15}$$

where N is the total number of edges for a cell p and ΔL_i is the length of the edge. In this paper, rectangular grids are used, hence $N = 4$. The approximation in Eq. (15) creates a local Riemann problem: a proper solver is required to calculate the flux values at the center of each edge. Prior to applying such a solver, the values at every edge are interpolated from its two corresponding cell center's values using the MUSCL linear reconstruction technique pioneered by van Leer [30]. In the framework of the edge-based finite volume scheme, Delis and Nikolos [31] proposed a reconstruction technique limited by some novel edge-based limiter functions, such as the Van Albada, MinMod, and Superbee methods. NUFSAW2D has an edge-based data structure and the edge-based MinMod limiter function is chosen. In our implementations, no significant differences were shown for accuracy between these limiters. In addition, the MinMod limiter is simpler to implement. The values of the left (L) and the right (R) states at midpoint M of edge i in Eq. (1) can be computed as

$$\begin{aligned} W_M^L &= W_p + \frac{\|\, r_{pM}\,\|}{\|\, r_{pq}\,\|} LIM\Big[\nabla W_p^{upw}\cdot \mathbf{r}_{pq}\,,\ \nabla W^{cent}\cdot \mathbf{r}_{pq}\Big], \\ W_M^R &= W_q - \frac{\|\, r_{Mq}\,\|}{\|\, r_{pq}\,\|} LIM\Big[\nabla W_q^{upw}\cdot \mathbf{r}_{pq}\,,\ \nabla W^{cent}\cdot \mathbf{r}_{pq}\Big], \end{aligned} \tag{16}$$

where $W = [\eta, U, \Upsilon]$. $\|r_{pM}\|$, $\|r_{Mq}\|$, and $\|r_{pq}\|$ are the scalar lengths between the center points p and q and the midpoint M at edge, given in Fig. 1.

The variables ∇W_p^{upw}, ∇W_q^{upw}, and ∇W^{cent} are the delta gradients corresponding to the two adjacent cells p and q, computed as

$$\begin{aligned} &\nabla W_p^{upw} = 2\ \nabla W_p - \nabla W^{cent}, \qquad \nabla W_q^{upw} = 2\ \nabla W_q - \nabla W^{cent}, \\ &\nabla W^{cent}\cdot \mathbf{r}_{pq} = W_q - W_p. \end{aligned} \tag{17}$$

The symbol *LIM* denotes the edge-based MinMod limiter function used in this paper to preserve the monotonicity of W_M^L and W_M^R, expressed as

$$LIM[a,\ b] = b \max\left[0,\ \min\left(\frac{a}{b},\ 1\right)\right], \tag{18}$$

where $a = \nabla W_p^{upw}\cdot \mathbf{r}_{pq}$ and $b = \nabla W^{cent}\cdot \mathbf{r}_{pq}$, e.g. for computing W_M^L. The complete formulas can be found in [31].

As shown in Eq. (16), the reconstructed values W are obtained in this paper based on η and the primitive variables U and Y. There is another option, probably being a more common approach, which is to compute Eq. (16) using η and the conservative variables Q and Φ. Each approach has its own advantages and disadvantages. Using the conservative variables, a relationship between conservation laws, symmetric system, and hyperbolic system are well-defined particularly when dealing with highly discontinuous flows [32]. However, if a wet–dry problem appears, this approach may cause a serious problem, since it is required to transform both the unit discharges hu and hv in the matrix Q back to the velocities u and v by a division of the small value of h. This results in very high values of u and v. If no wet–dry problem exists, both approaches yield similar results.

On the other side, using the primitive variables may suffer from an energy loss problem in subcritical flows as investigated by Begnudelli et al. [33]. However, this technique is admittedly cheaper as the calculations for transforming the unit discharges back to the velocities are not required. It should be noted, Begnudelli et al. [33] also concluded that the primitive variable reconstructions may be advantageous in some cases, as followed by Delis and Nikolos [31] by reconstructing the primitive variables to successfully simulate complex shallow water flows. Obviously, some techniques had been developed to tackle the problem using the conservative variables, see for example [19], which has been proven to be accurate. However, this technique requires more CPU time. Therefore, the conservative variables are not reconstructed here. Another reason is that the gradients used to compute the primitive variables can be used directly to calculate the diffusive fluxes.

3.1.2. Hydrostatic and topography reconstructions

The next step – prior to calculating the convective fluxes using a proper solver – is to apply the hydrostatic and topography reconstructions at each edge (see point M in Fig. 1). Similar to [25,34], the bed elevation at each edge is obtained by

$$z_M = \max\left(\eta_M^L - h_M^L,\ \eta_M^R - h_M^R\right). \tag{19}$$

Now, the hydrostatic reconstruction is applied to correct the depth at edge, thus

$$h_M^L = \max\left(\eta_M^L - z_M, 0\right), \qquad h_M^R = \max\left(\eta_M^R - z_M, 0\right). \tag{20}$$

It is obvious that Eq. (20) will always ensure non-negative water depth, despite facing wet–dry problems. To preserve the flux balance for the calculation of the bed-slope terms, z_M must be remodified by applying the bed topography reconstruction as [25]

$$z_M \leftarrow z_M - \max\left(0,\ z_M - \eta_M^L,\ z_M - \eta_M^R\right). \tag{21}$$

Eq. (21) will be explained later in Section 3.3.2.

3.1.3. Convective fluxes calculation

Using the reconstructed values in Eq. (16) and after the hydrostatic reconstruction in Eq. (20), the convective fluxes in Eq. (15) are computed for each edge i using the CU scheme by

$$\begin{aligned} &\Big(F_x\, n_x + F_y\, n_y\Big)_i\, \Delta L_i \\ &= \frac{\Delta L_i}{\left(a^{in} + a^{out}\right)_i} \begin{bmatrix} \Big[\big(a^{in}F_x(W_M^R) + a^{out}F_x(W_M^L)\big)\ n_x + \\ \big(a^{in}F_y(W_M^R) + a^{out}F_y(W_M^L)\big) n_y - \\ a^{in}a^{out}(W_M^R - W_M^L)\Big] \end{bmatrix}_i, \end{aligned} \tag{22}$$

where a^{in} and a^{out} are the local one-sided propagation speeds computed by

$$\begin{aligned} a^{in} &= -\min(\Lambda_{1_M^L},\quad \Lambda_{1_M^R},\quad 0), \\ a^{out} &= \max(\Lambda_{5_M^L},\quad \Lambda_{5_M^R},\quad 0). \end{aligned} \tag{23}$$

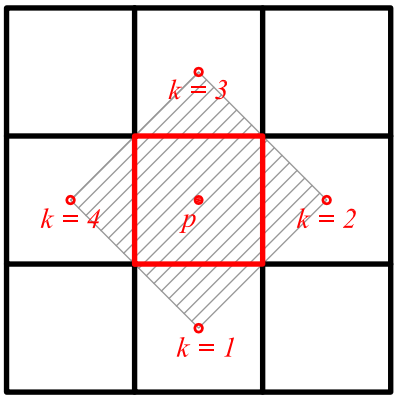


Fig. 2. Properties for calculating the gradient of W for rectangular grids.

$\Lambda_1 \leq \ldots \leq \Lambda_5$ are the eigenvalues of the 5×5 global system Jacobian matrix for Eqs. (1) and (3). This matrix is expressed as

$$\begin{bmatrix} 0 & n_x & n_y & 0 & 0 \\ (-u^2+gh)\ n_x - uvn_y & 2un_x + vn_y & un_y & 0 & 0 \\ -uvn_x + (-v^2+gh)\ n_y & vn_x & un_x + 2vn_y & 0 & 0 \\ -(\kappa un_x + \kappa vn_y) & \kappa n_x & \kappa n_y & un_x + vn_y & 0 \\ -(\epsilon un_x + \epsilon vn_y) & \epsilon n_x & \epsilon n_y & 0 & un_x + vn_y \end{bmatrix}. \tag{24}$$

3.1.4. Diffusive fluxes calculation

For the diffusive fluxes, the discretization of the derivative form of the primitive variables is fundamental. First, it is necessary to calculate the derivations of those variables with respect to x and y directions at cells. Second, these derivation values are used to compute the derivations at the edges. For the calculations at the cells, as previously explained in Section 3.1.1, the gradients computed for the MUSCL reconstruction technique in Eq. (16) with the values of U and Y can be used directly. For rectangular grids, the gradients of W at cell p are simply calculated as (see Fig. 2)

$$\left(\frac{\partial W}{\partial x}\right)_p = \frac{W_{k=2} - W_{k=4}}{2\ \Delta x}, \qquad \left(\frac{\partial W}{\partial y}\right)_p = \frac{W_{k=3} - W_{k=1}}{2\ \Delta y}. \tag{25}$$

For the sake of simplicity, $G_x = [D_x,\ D_{\Phi,x}]$ and $G_y = [D_y,\ D_{\Phi,y}]$ and they are calculated as

$$-\oint_\Gamma \left(G_x + G_y\right) \cdot \vec{n}\ d\Gamma \approx -\sum_{i=1}^{4} \left(G_x\ n_x + G_y\ n_y\right)_i\ \Delta L_i. \tag{26}$$

The centered method computed by averaging the values of two cells corresponding to an edge (see points p and q in Fig. 1) – is employed to calculate the gradient at edges. Sufficient accuracy can be achieved at no extra effort, while computational time remains acceptably low. This method is written as

$$\begin{aligned} &(G_x\ n_x + G_y\ n_y)_i\ \Delta L_i \\ &= \frac{\Delta L_i}{2}[(G_x(W_p) + G_x(W_q))\ n_x + (G_y(W_p) + G_y(W_q))\ n_y\ ]_i. \end{aligned} \tag{27}$$

Other discretization schemes are also possible for the diffusive fluxes, see [35]. As one can see, Eq. (27) can easily be computed after the gradient values at the cells are known. According to Cea et al. [6], a different formulation of the centered scheme may be applied by averaging the eddy viscosity, water depth, and velocity gradient at an edge of its two corresponding cells. It is also possible to employ the upwind discretization for the diffusive fluxes; however, this technique is more complex, since one needs to know the eigenstructure of the Jacobian matrix – as given in the Roe solver – which is not required in the CU scheme (as a Riemann-solver-free method) nor in the discretization of the bed-slope terms presented in Section 3.1.5.

3.1.5. Bed-slope terms calculation

The bed-slope terms are discretized similar to [25,36]. This method is a Riemann-solution-free technique which can be computed separately from the convective fluxes. In [25], it was proven that this technique was suitable to balance the fluxes for wet–dry problems. This technique for x direction (S_{bx}) is expressed as

$$\begin{aligned} \iint_\Omega S_{bx}\ d\Omega = -\iint_\Omega \left(gh\frac{\partial z}{\partial x}\right) d\Omega \approx -\frac{g}{2}\sum_{i=1}^{4}\Bigg[\left(\frac{\left(h^L_{Mi} + h^R_{Mi}\right)}{2} + h_p\right) \\ z_{Mi} - \frac{\left(h^L_{Mi} + h^R_{Mi}\right)}{2}\, z_p\Bigg]\, n_{xi}\ \Delta L_i. \end{aligned} \tag{28}$$

It should be noted that Eq. (28) is applied in this paper in an edge-based manner, where h^L_M and h^R_M are computed by Eq. (20) and z_M is obtained using Eq. (21). Eq. (21) is valid to compute wet–dry and dry-dry interfaces and evidently gives no change to wet-wet interfaces as investigated in [25]. This will also be explained further in Section 3.3.2. A similar approach can also be applied to y direction (S_{by}).

3.1.6. Friction terms calculation

As pointed out in [25,31], a fully explicit treatment of the friction source terms may give oscillations due to very low water depths on rough beds. To tackle this problem, one may use varying time step so that a very small value is achieved ensuring a proper limitation of the Courant–Friedrichs–Lewy (CFL) number or one can refine the computational grids during simulation time. Both approaches require, however, high computational cost. Another possibility is to apply a semi-implicit technique, so that the friction source terms in x direction can be expressed as

$$\begin{aligned} &(hu)^{t*}_p \leftarrow \Pi^{-1}(hu)^{t*}_p, \qquad \Pi = 1 \\ &+\ g\ \Delta t\left[(1-\theta)\left(\frac{n_m^2\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}\right)^{t*}_p + \theta\left(\frac{n_m^2\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}\right)^{(t*-1)}_p\right], \end{aligned} \tag{29}$$

where Δt defines the time step and the symbol p corresponds to the values at the cell. θ is an implicitness coefficient, which is set to 0.5 in this paper. t^* defines a calculation level for the Runge–Kutta second-order (RKSO) method. A similar approach can also be applied to y direction.

3.1.7. Turbulence source terms calculation

The technique presented in [6,37] is employed according to which the diffusive fluxes of the ASM in Eq. (27) are split into two parts – the orthogonal and the non-orthogonal terms. The former was computed semi-implicitly and the latter was calculated explicitly. For rectangular grids, the latter is zero. Instead of being computed as the normal fluxes, like the convective fluxes of the 2D RANS equations, the former is treated as a source term, which is computed together with the source term $S_{\kappa-\epsilon}$. According

Algorithm 1 Concept of the calculation for temporal discretization.

set the coefficients for the RKSO scheme
$\alpha^a_{(t*=1)} = 1;\quad \alpha^b_{(t*=1)} = 0;\quad \alpha^a_{(t*=2)} = 0.5;\quad \alpha^b_{(t*=2)} = 0.5$
set the initial value of Q for all cells at each time step
for $(p = 1)$ to $(p =$ total number of cells$)$ **do**
  $Q_p^{(t*=0)} = Q_p^t \quad ; \quad \Phi_p^{(t*=0)} = \Phi_p^t$
end for
for $(t* = 1)$ to $(t* = 2)$ **do**
  for $(p = 1)$ to $(p =$ total number of cells$)$ **do**
    $Q_p^{t*} = \alpha^a_{t*}\left[Q_p^{(t*=0)} - \dfrac{\Delta t}{A_p}\left(\sum_{i=1}^{4}\left(\left(C_x - D_x - S_{bx}\right) n_x + \left(C_y - D_y - S_{by}\right) n_y\right)_i \Delta L_i\right)^{(t*-1)}\right] + \alpha^b_{t*}\, Q_p^{(t*-1)}$
    transforming back :conservative variables hu^{t*} and $hv^{t*} \rightarrow$ primitive variables u^{t*} and v^{t*}
    updating $Q_p^{t*} \leftarrow$ friction source terms are now taken into account, so that: $Q_p^{t*} \leftarrow \Pi^{-1}\, Q_p^{t*}$
    $\Phi_p^{t*} = \alpha^a_{t*}\left[\Phi_p^{(t*=0)} - \dfrac{\Delta t}{A_p}\left(\sum_{i=1}^{4}\left(C_{\Phi,x}\, n_x + C_{\Phi,y}\, n_y\right)_i \Delta L_i\right)^{(t*-1)}\right] + \alpha^b_{t*}\, \Phi_p^{(t*-1)}$
    updating $\Phi_p^{t*} \leftarrow$ turbulent source terms are now taken into account, so that: $\Phi_p^{t*} \leftarrow \Phi_p^{t*} + \Delta t \quad \mathbf{H}_p^{(t*-1)}$
    saving the final values for the subsequent time step when $* = 2$ as : $Q_p^{(t+1)} = Q_p^{(*=2)}$ and $\Phi_p^{(t+1)} = \Phi_p^{(*=2)}$
  end for
end for

to Eqs. (4) and (5), the new symbols are noted

$$\mathbf{H}_1 = \begin{bmatrix} \oint_\Gamma \left[\sigma_\kappa^{-1} h\nu_t\left(\dfrac{\partial \kappa}{\partial x} + \dfrac{\partial \kappa}{\partial y}\right)\right]\cdot \vec{n}\, d\Gamma \\ \oint_\Gamma \left[\sigma_\epsilon^{-1} h\nu_t\left(\dfrac{\partial \epsilon}{\partial x} + \dfrac{\partial \epsilon}{\partial y}\right)\right]\cdot \vec{n}\, d\Gamma \end{bmatrix}, \quad \mathbf{H}_2 = \begin{bmatrix} P_h \\ c_{\epsilon 1}\dfrac{\epsilon}{\kappa} P_h \end{bmatrix}, \quad \mathbf{H}_3 = \begin{bmatrix} P_{\kappa b} \\ P_{\epsilon b} \end{bmatrix}, \quad \mathbf{H}_4 = \begin{bmatrix} -\dfrac{\epsilon}{\kappa} \\ -c_{\epsilon 2}\dfrac{\epsilon}{\kappa} \end{bmatrix}. \tag{30}$$

The total of the diffusive fluxes and the source term $S_{\kappa-\epsilon}$ at cell p is now denoted by $\mathbf{H}_p$ and is computed semi-implicitly by

$$\mathbf{H}_p^{(t*-1)} = \left[\max\left(\frac{\mathbf{H}_1^{(t*-1)}}{A_p}, 0\right) + \mathbf{H}_2^{(t*-1)} + \mathbf{H}_3^{(t*-1)}\right] + \Phi^{t*}\left[\min\left(\frac{\mathbf{H}_1^{(t*-1)}}{\left(A_p \Phi^{(t*-1)}\right)}, 0\right) + \mathbf{H}_4^{(t*-1)}\right], \tag{31}$$

where A_p is the area of cell p.

3.1.8. Some limitations for turbulence properties

To ensure computational stability, the turbulence properties must be limited, so that negative values of the Reynolds stresses can be avoided. These limitations include limiter values for both P_h and $P_{\kappa b}$ – and realizability conditions for ν_t, $\overline{u'^2}$, $\overline{u'v'}$, and $\overline{v'^2}$. The details are not given here; interested readers are thus referred to [5,6,38].

3.2. Temporal discretization

Using the RKSO method, the procedure of the temporal discretization is explained in Algorithm 1 and following [39] is written as

$$\mathbf{K}_p^t = -\frac{\Delta t}{A_p}\left[\sum_{i=1}^{4}\left(\left(C_x - D_x - S_{bx}\right) n_x + \left(C_y - D_y - S_{by}\right) n_y\right)_i^t \Delta L_i\right],$$
$$Q_p^{t*} = Q_p^t + \mathbf{K}_p^t, \qquad Q_p^{t*} \leftarrow \Pi^{-1} Q_p^{t*},$$
$$Q_p^{t+1} = \frac{1}{2}\left(Q_p^t + Q_p^{t*} + \mathbf{K}_p^{t*}\right), \tag{32a}$$

$$\mathbf{K}_{\Phi}{}_p^t = -\frac{\Delta t}{A_p}\left[\sum_{i=1}^{4}\left(C_{\Phi,x}\, n_x + C_{\Phi,y}\, n_y\right)_i^t \Delta L_i\right],$$
$$\Phi_p^{t*} = \Phi_p^t + \mathbf{K}_{\Phi}{}_p^t, \qquad \Phi_p^{t*} \leftarrow \Phi_p^{t*} + \Delta t\, \mathbf{H}_p^t, \tag{32b}$$
$$\Phi_p^{t+1} = \frac{1}{2}\left(\Phi_p^t + \Phi_p^{t*} + \mathbf{K}_{\Phi}{}_p^{t*}\right).$$

The calculations are started by explicitly calculating h_p^{t*}, hu_p^{t*}, and hv_p^{t*} in the matrix Q_p^{t*} without the friction terms. Both u_p^{t*} and v_p^{t*} can also be calculated without the friction terms by dividing hu_p^{t*} and hv_p^{t*} with h_p^{t*}. These velocities are then used to update the values of hu_p^{t*} and hv_p^{t*} using Eq. (29) so that the friction terms are now included. To reduce computational time, h_p^{t*} is used as a criterion to determine whether hu_p^{t*}, hv_p^{t*}, $h\kappa_p^{t*}$, and $h\epsilon_p^{t*}$ should be computed or not, see Algorithm 2. If h_p^{t*} is calculated as a dry

Algorithm 2 Wet-dry treatment for a very low water depth less than $h_{min} = 10^{-10}$ m.

if $h^* \le h_{min}$ **then**
  hu^*, hv^*, u^*, v^*, κ^*, and ϵ^* are set to 10^{-15}
else
  $u^* = \dfrac{hu^*}{h^*}$, $v^* = \dfrac{hv^*}{h^*}$, $\kappa^* = \dfrac{h\kappa^*}{h^*}$, and $\epsilon^* = \dfrac{h\epsilon^*}{h^*}$
end if

cell $\left(h_p^{t*} \le 10^{-10}\ \text{m}\right)$, all the corresponding variables hu_p^{t*}, hv_p^{t*}, u_p^{t*}, v_p^{t*}, κ_p^{t*}, and ϵ_p^{t*} are simply set to a very small value (in this paper 10^{-15}). If h_p^{t*} is calculated as a wet cell, both hu_p^{t*} and hv_p^{t*} in the matrix Q_p^{t*} and both $h\kappa_p^{t*}$ and $h\epsilon_p^{t*}$ in the matrix Φ_p^{t*} are computed. Note that we have monitored the mass conservation

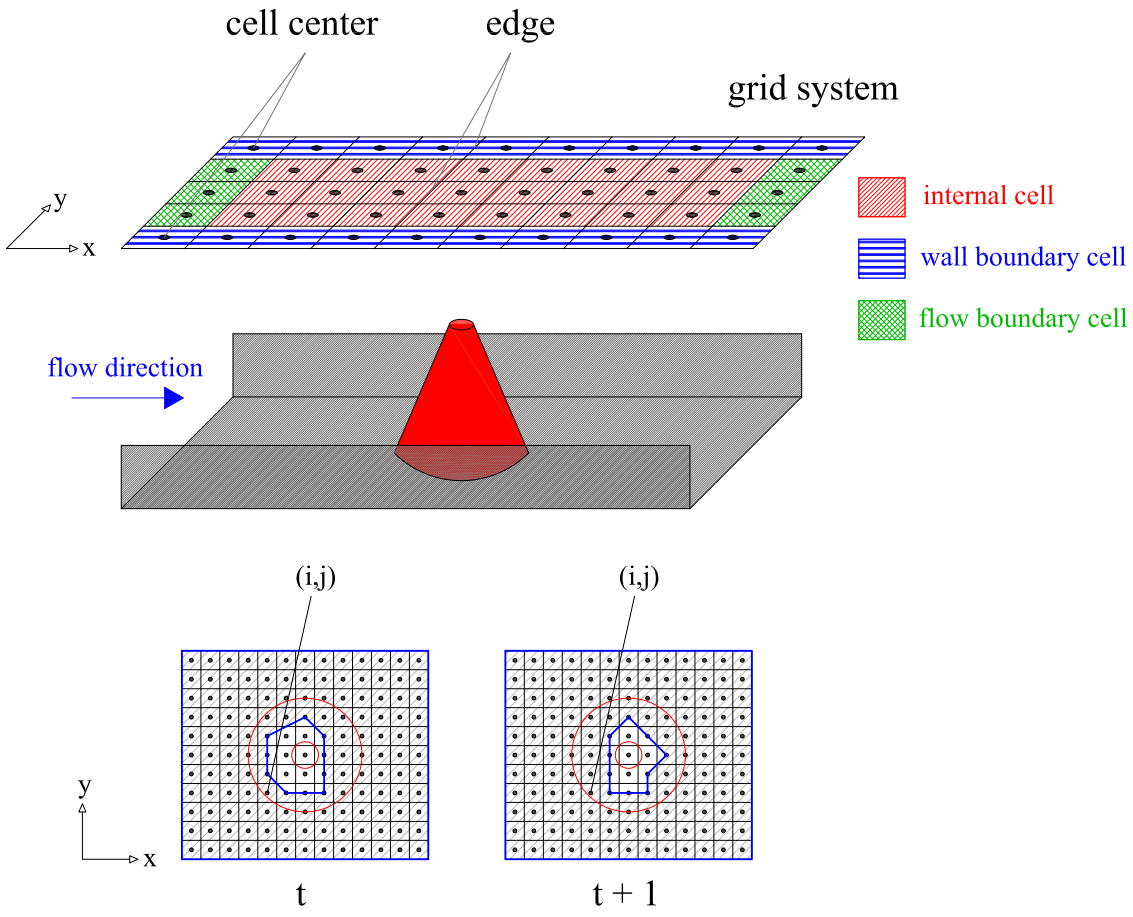


Fig. 3. Meshing system in a domain with a conical island.

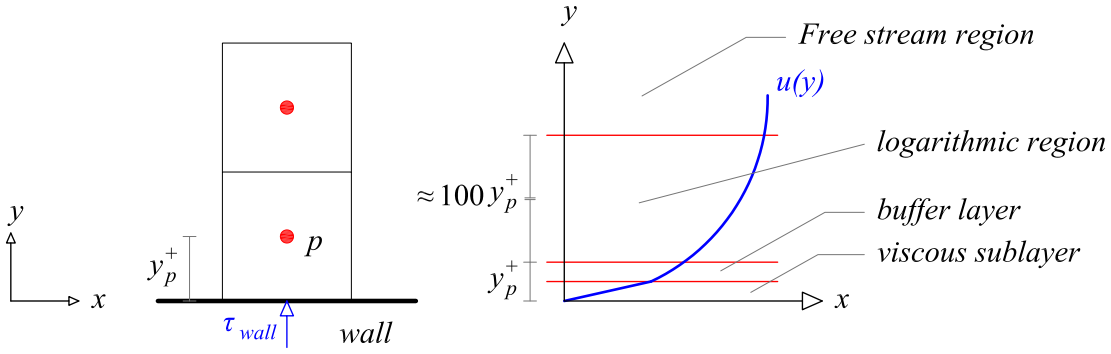


Fig. 4. Near-wall mesh and near-wall scaling of turbulent boundary layer: τ_{wall} is the wall shear stress.

with the threshold parameter $h_{min} = 10^{-10}$ m in all simulations presented in this paper. Here the time step Δt is limited by the Courant–Friedrichs–Lewy (CFL) number of 0.5 and by considering the Peclet (Pe) number $\leq$ 2/CFL, see [40]. We also consider the CFL limitation for the local one-sided propagation speeds of the CU scheme in [19].

3.3. Boundary conditions

3.3.1. Flow (open) boundary

Following [41], the characteristic method is applied for subcritical flow boundaries. When h_M^R is given, u_M^R is computed as

$$u_M^R = u_M^L + 2\sqrt{g\, h_M^L} - 2\sqrt{g\, h_M^R}. \tag{33}$$

In a similar way when u_M^R is specified, h_M^R is calculated by

$$h_M^R = \frac{\left(u_M^L - u_M^R + 2\sqrt{g\, h_M^L}\right)^2}{4\, g}. \tag{34}$$

When unit discharge hu_M^R is given as a boundary condition, the corresponding h_M^R and u_M^R can be calculated by applying $u_M^R = \frac{hu_M^R}{h_M^R}$, so

$$\frac{hu_M^R}{h_M^R} = u_M^L + 2\sqrt{g\, h_M^L} - 2\sqrt{g\, h_M^R}. \tag{35}$$

Since a non-linear relationship appears, an iteration technique, e.g. the Newton–Raphson method, is required to solve Eq. (35). In the

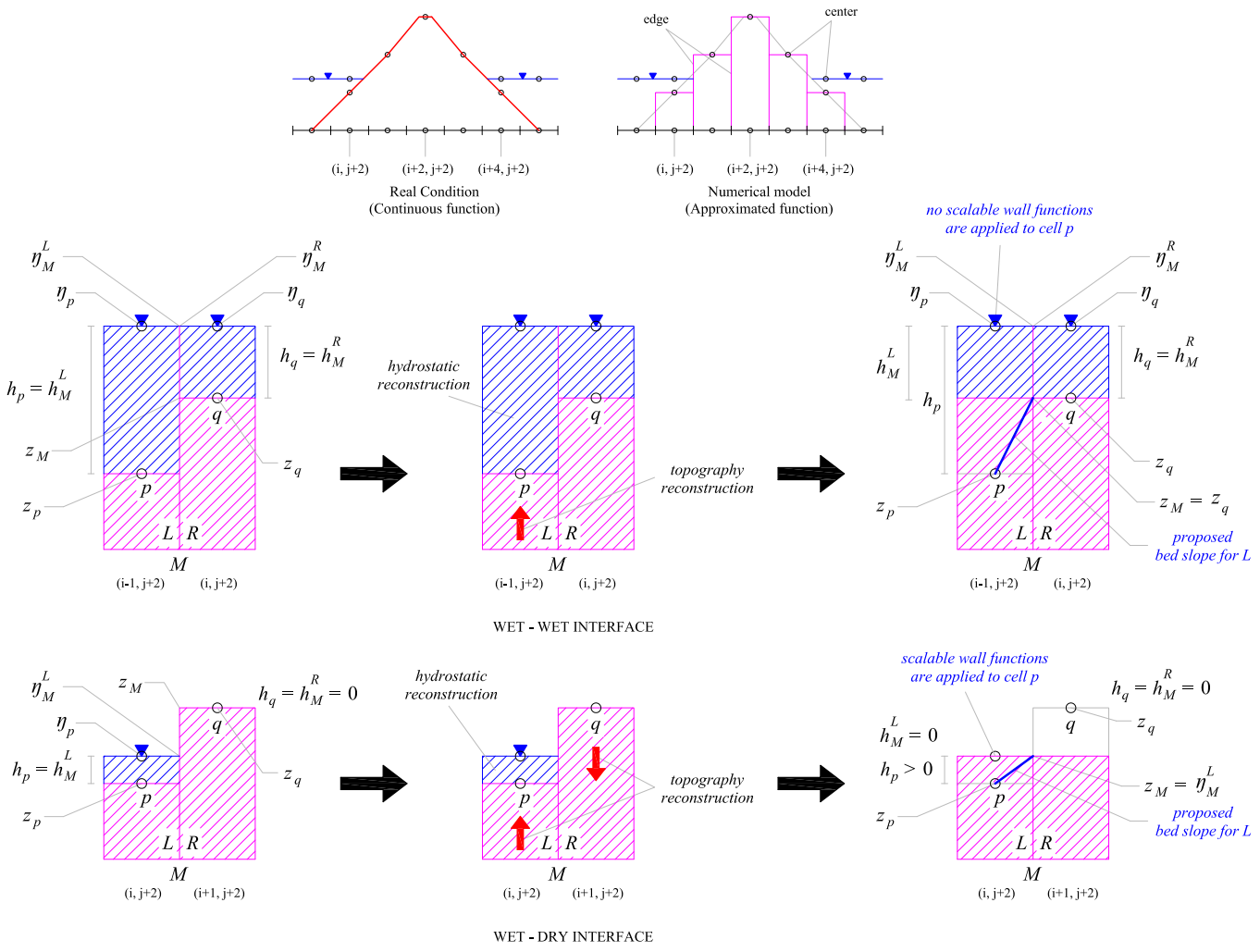


Fig. 5. Strategy of the hydrostatic and topography reconstructions.

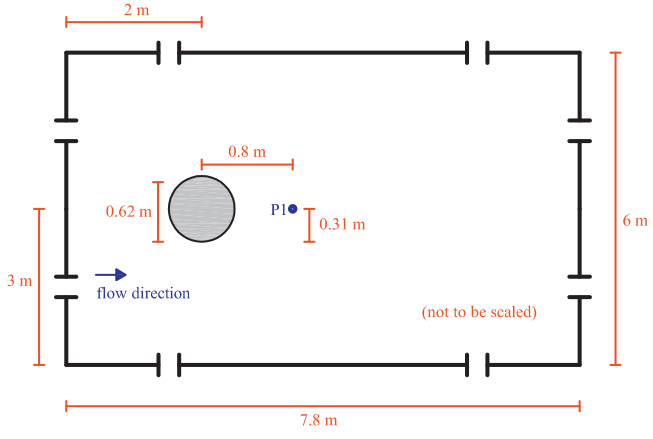


Fig. 6. Case 1: Domain of circular cylinders.

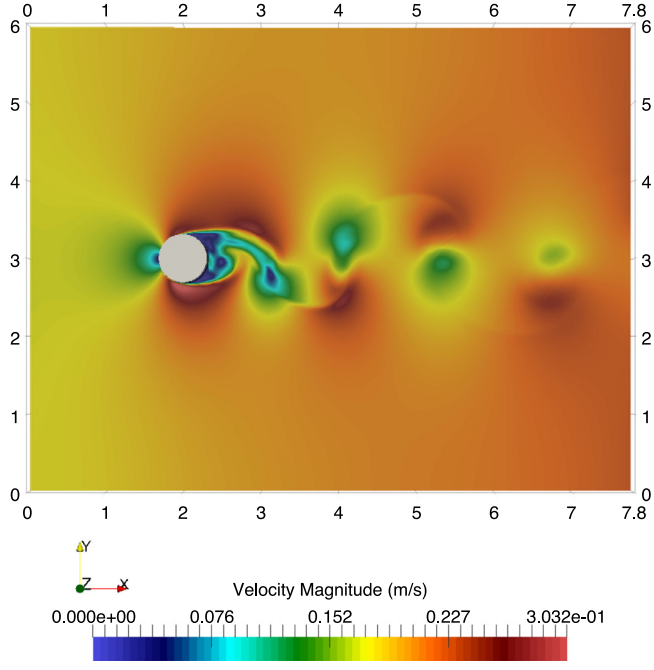


Fig. 7. Case 1: Visualization of wakes along the channel at 283.2 s.

case of supercritical flows, all the values h_M^R, hu_M^R, and hv_M^R are simply set similar to h_M^L, hu_M^L, and hv_M^L, respectively.

Regarding the turbulence model, the turbulent intensity could be specified at flow boundary cells (see Fig. 3) within the range of 1–10% for the low–high turbulent cases in order to compute κ, whereas the turbulent length scale could be set to 0.22 of the inlet boundary layer thickness to compute ϵ. In general, it might, how-

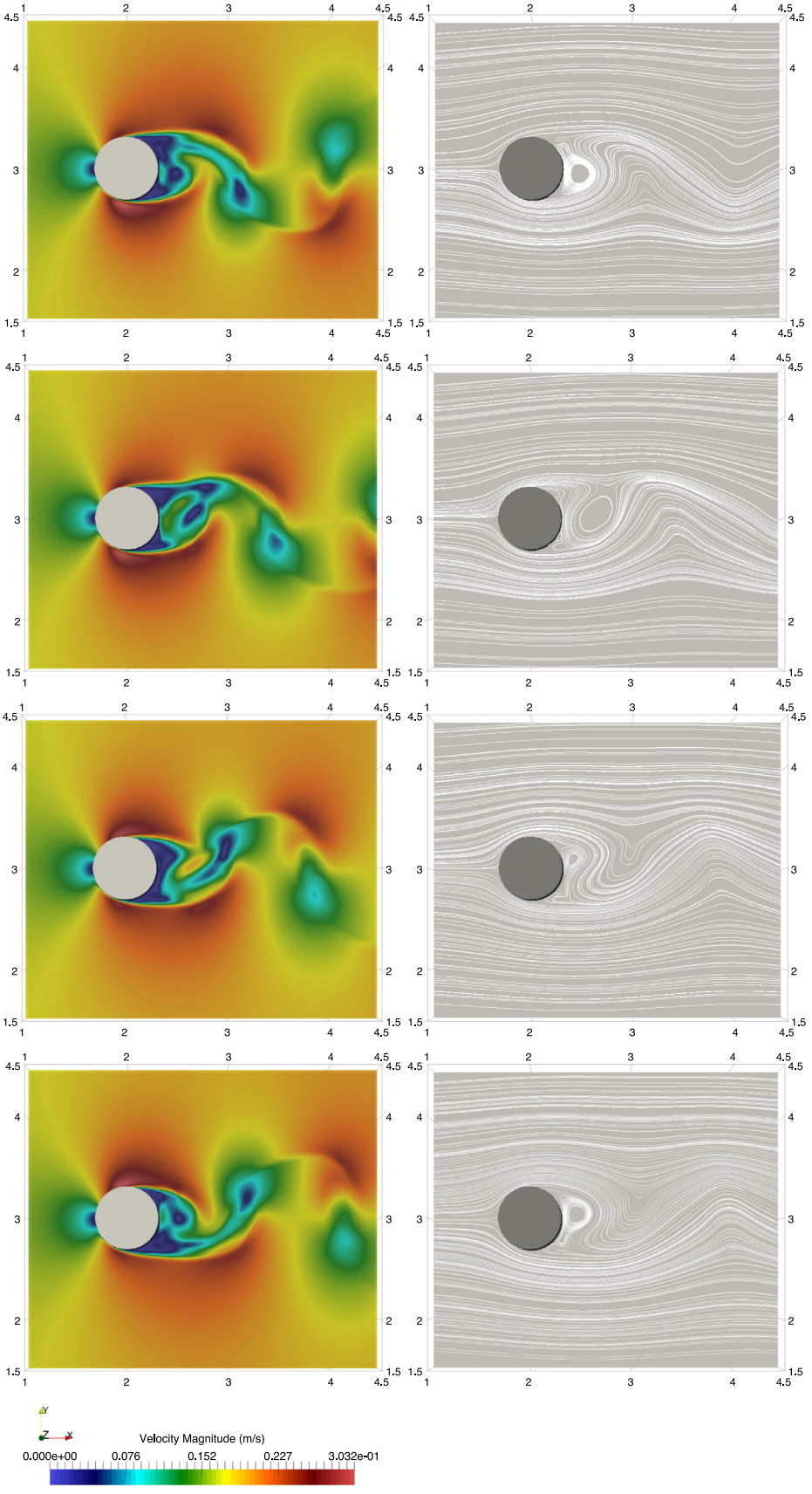


Fig. 8. Case 1: Visualization of wakes near the cylinder at 283.2 s, 286.4 s, 289.6 s, and 292.9 s.

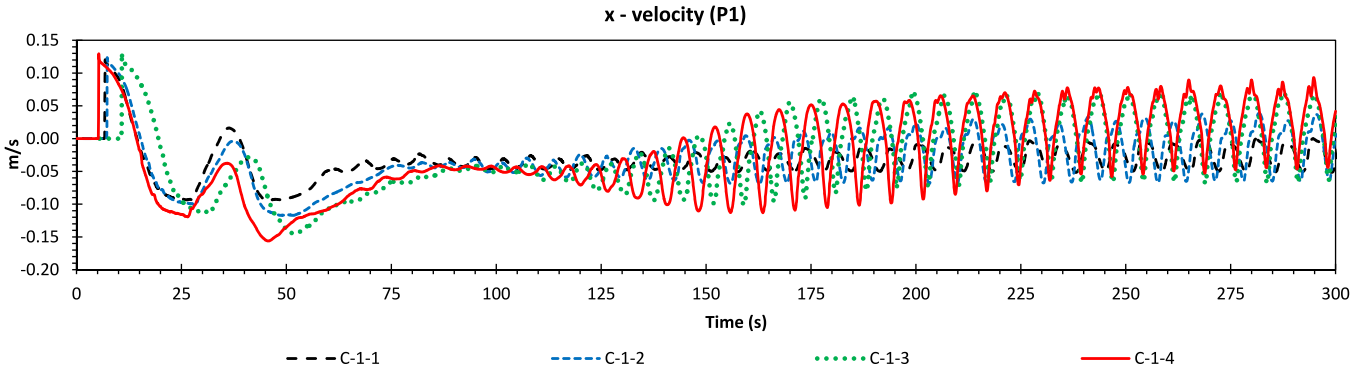


Fig. 9. Case 1: Convergence histories at point P1.

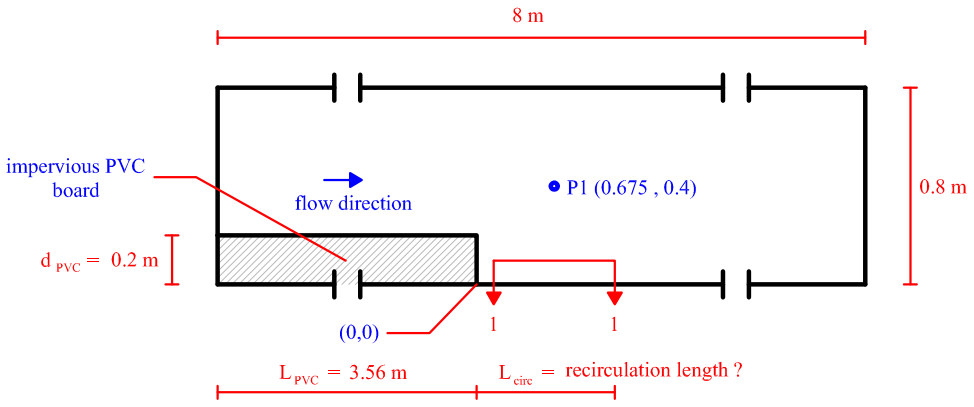


Fig. 10. Case 2: Domain of channel expansion.

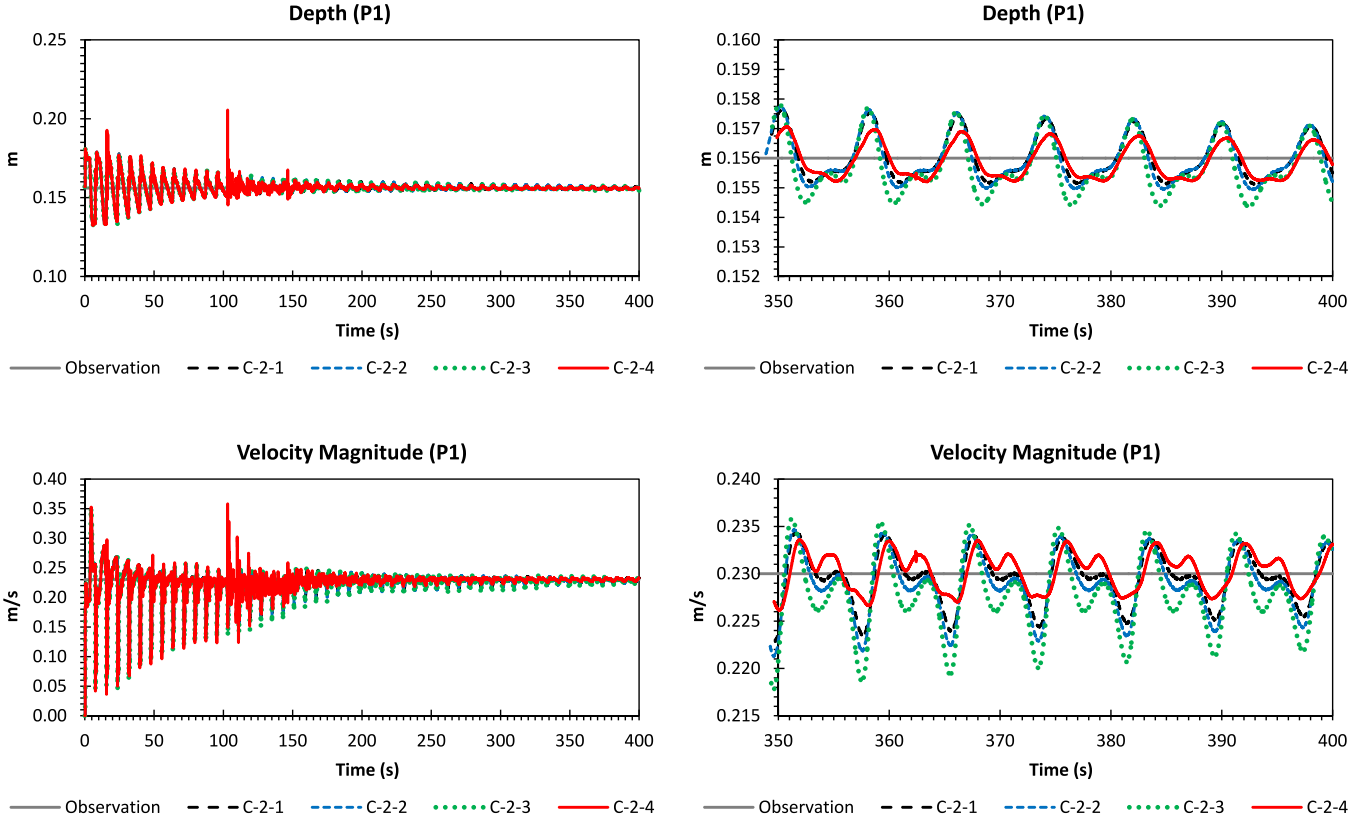


Fig. 11. Case 2: Convergence histories at point P1.

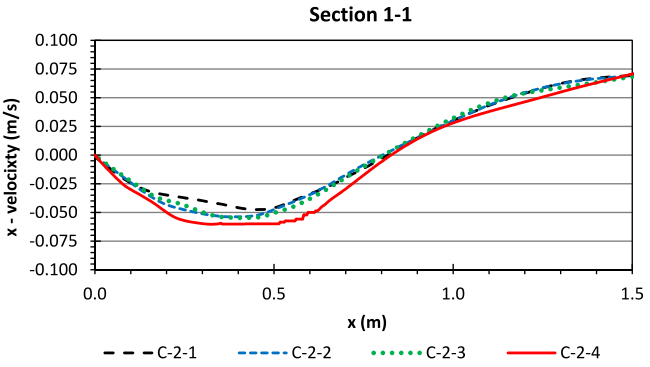


Fig. 12. Case 2: Comparisons along section 1-1.

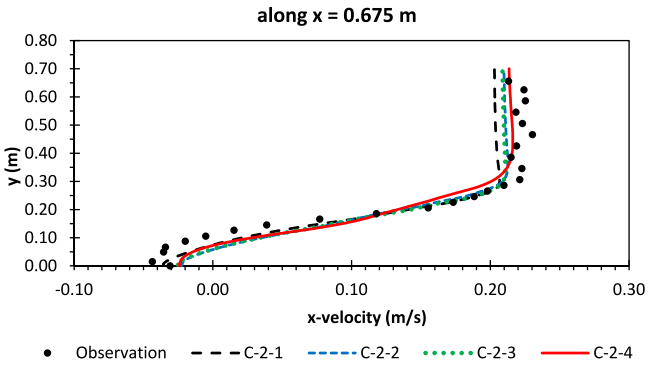


Fig. 13. Case 2: Comparisons and along $x = 0.675$ m.

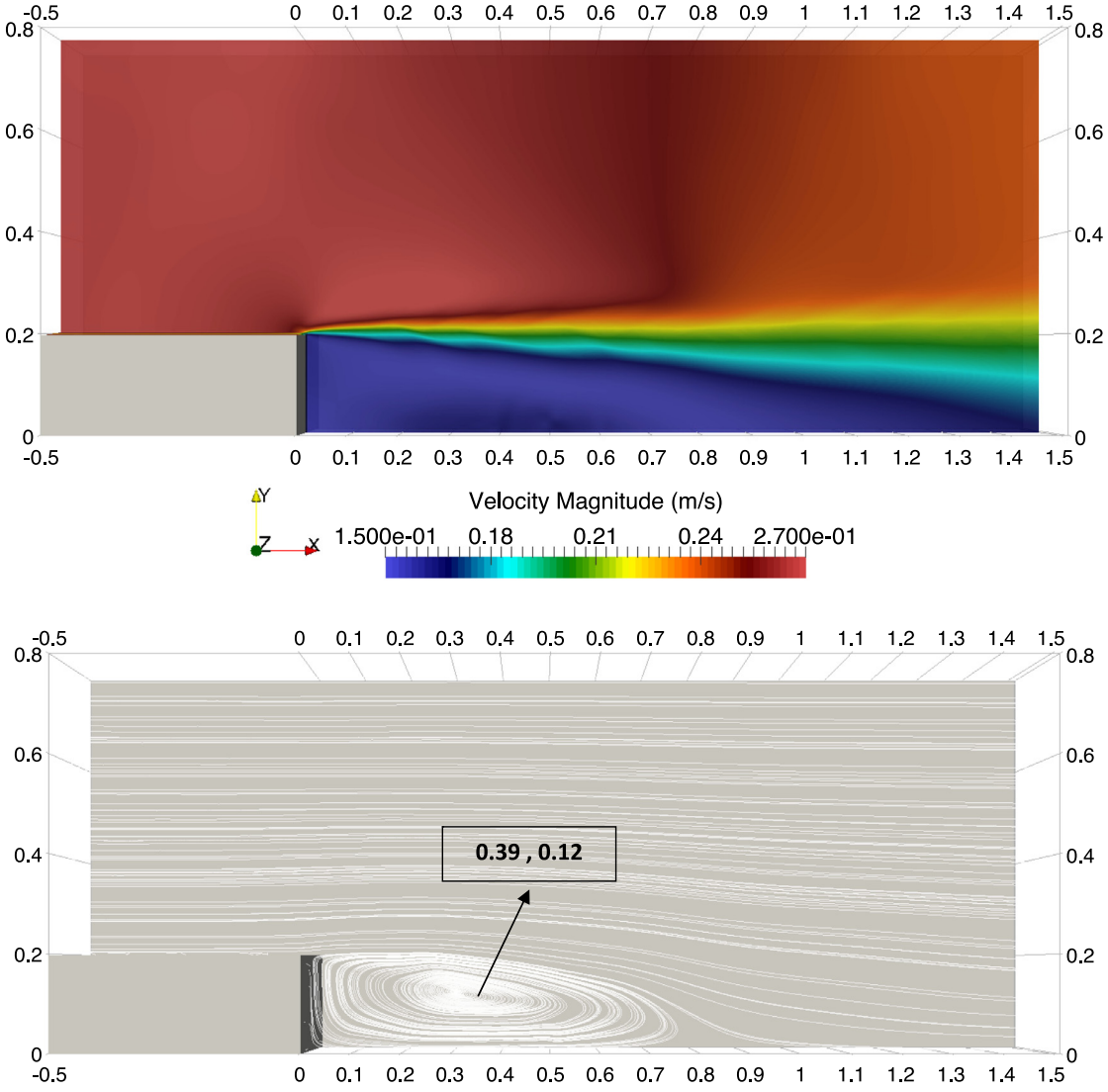


Fig. 14. Case 2: Visualization of the velocity magnitude near the recirculation zone at 400 s.

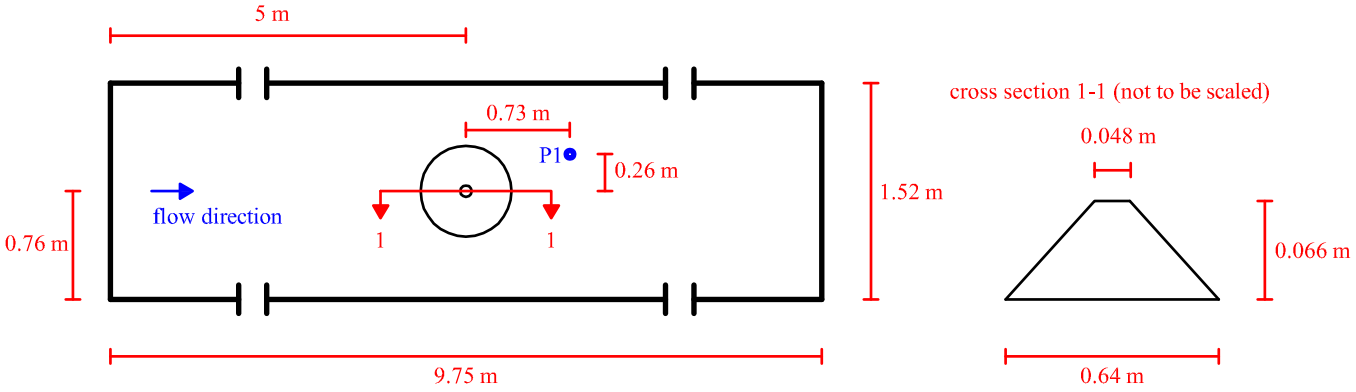


Fig. 15. Case 3: domain of conical island.

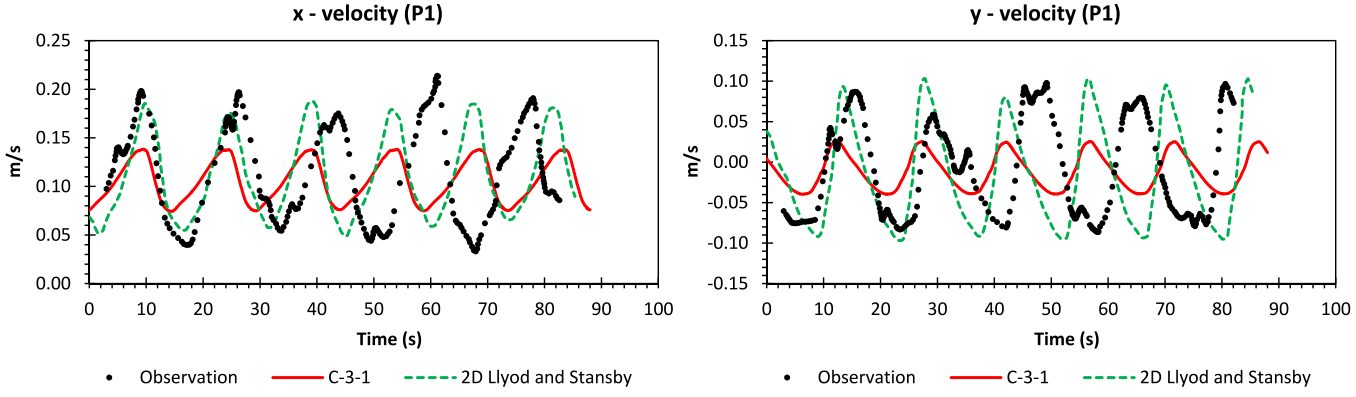


Fig. 16. Case 3: Results of C-3-1 at the measurement point P1.

ever, be difficult to know such ranges accurately. Therefore, an alternative way given in [6] is applied to approximate the turbulence parameters at flow boundary cells as

$$\kappa_p^{t+1} = \left[\frac{\left(g\, n_m^2\, {h_p}^{-\frac{1}{3}}\right)^{0.75}\left({u_p}^2+{v_p}^2\right)}{c_\mu\, c_{\epsilon\Gamma}}\right]^{t+1},$$
$$\epsilon_p^{t+1} = \left[\frac{g\, n_m^2\, {h_p}^{-\frac{1}{3}}\left({u_p}^2+{v_p}^2\right)^{1.5}}{h_p}\right]^{t+1}, \tag{36}$$
$$\nu_{t_p}^{t+1} = \left[0.19\,\upsilon\, h_p\sqrt{g\, n_m^2\, {h_p}^{-\frac{1}{3}}\left({u_p}^2+{v_p}^2\right)}\right]^{t+1},$$

where υ is the von-Karman coefficient (0.41). At outlet boundaries, there is no need to calculate the turbulence properties, where κ_p^{t+1}, ϵ_p^{t+1}, and $\nu_{t_p}^{t+1}$ can simply be set similar to the values of the closest adjacent cell.

3.3.2. Wall boundary with the ScWF

To explain Eqs. (21) and (28) as well as the meshing system in NUFSAW2D, a domain with a conical island is sketched in Fig. 3. The meshes are built with rectangular cells for the whole domain. At all wall boundaries, the fluxes normal to the wall are set to zero, whereas $h_M^R = h_M^L$ is employed. It is now assumed that in the certain time levels t and $t+1$, the flow characteristics become similar to that shown in Fig. 3. An example is taken for cell $(i, j+2)$. At time level t, cell $(i, j+2)$ must be treated as a wall boundary cell, since its adjacent cell $(i+1, j+2)$ is calculated as a dry cell. Therefore, a proper wall function must be applied to give the correct values of the turbulence properties. At time level $t+1$, one cannot, however, treat cell $(i, j+2)$ as a wall boundary cell, since its adjacent cell $(i+1, j+2)$ becomes a wet cell and consequently, cell $(i+1, j+2)$ must now be calculated as a wall boundary cell.

As previously mentioned in Section 1, the cell center $(i, j+2)$ at time level t must be placed in the logarithmic region to satisfy the criteria $y^+ \geq 11.067$, when the StWF are preferred (see Fig. 4). Since y^+ is a function of the wall friction velocity that may change during simulation time, it is unfortunately not an easy task to do, unless one can predict the wall friction velocity at the initial time level appropriately prior to starting the simulation and ensure such a wall friction velocity to be relatively constant during simulation time. Even, for cases dealing with wet–dry problems – in which very high-resolution meshes are used thus the water fluctuations become very sensitive to determine the wall boundary geometries – this task becomes more difficult. The extent of the logarithmic region in Fig. 4 is much larger for high Reynolds numbers (turbulent cases) than for low Reynolds numbers (laminar cases) [42]. Therefore, for turbulent cases, one might still obtain proper results despite still using the StWF for wall boundary cells. However, as the meshes become finer and finer, results will deteriorate.

To remedy the shortcoming of the StWF, Menter and Esch [42] proposed the ScWF for a cell p (as a wall boundary cell) written as

$$w_p^{tg} = \frac{w_p^*}{\upsilon}\ln\left(y_p^+\, E\right), \qquad y_p^+ = \max\left(11.067\,,\ \frac{\triangle y_p^{nor}\, w_p^*}{\nu_e}\right), \tag{37}$$

where w_p^{tg} is the velocity component of cell p parallel to the wall (it can either be u_p or v_p), w_p^* is the wall friction velocity, y_p^+ denotes the dimensionless wall distance, $\triangle y_p^{nor}$ is the normal distance from cell center p to the wall, and E denotes the roughness parameter.

In this paper, E is set to 8.432 as the material in all benchmark cases are assumed to be smooth wall. Another formula is also possible to calculate E, e.g. see [43]. Eq. (37) prevents the values at

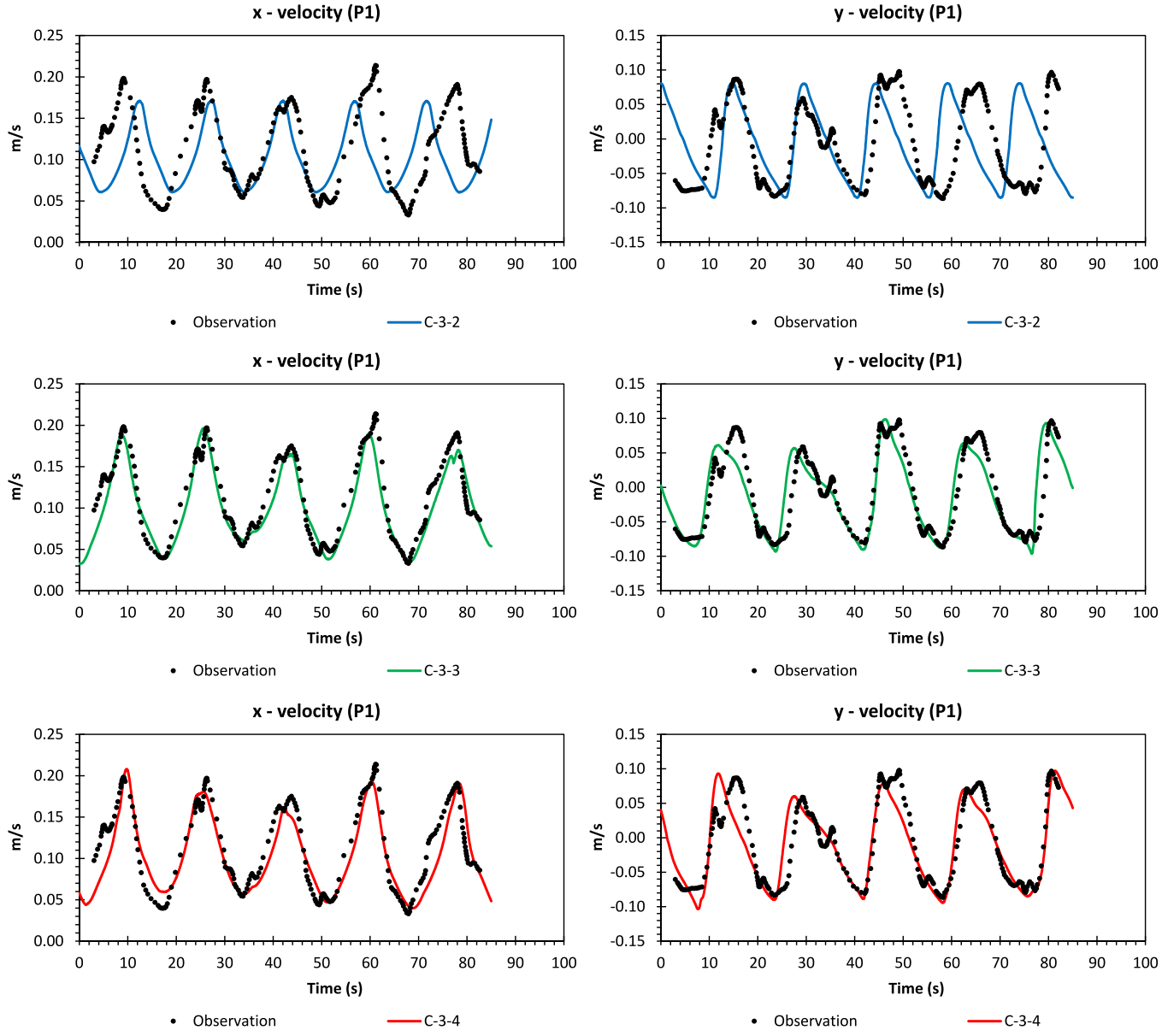


Fig. 17. Case 3: Results of C-3-2, C-3-3, and C-3-4 at the measurement point P1.

wall boundary cell centers to slide into the linear profile area (viscous sublayer) as y_p^+ must always be $\geq$ 11.067, so that the definition of y_p^+ is now separated from the mesh size. Physically, this phenomenon is interpreted for very fine meshes as a limitation that the cell p will be treated as a wall boundary cell, only if it acts as a boundary edge of the viscous sublayer, thus giving a linear relationship between w_p^{tg} and w_p^*. Obviously, this becomes a major drawback, in which the results may not depend on the logarithmic profile assumption, so that the effect of the displacement of the viscous sublayer is not taken into account anymore. However, one should accept this shortcoming, as all wall functions formulations never accurately resolve the sublayer [42]. To solve Eq. (37), the Newton–Raphson iteration technique is employed, since a nonlinear relationship exists, expressed as Algorithm 3. Normally, the value of $w_p^{*\,\text{trial }0}$ can be chosen freely and the iteration procedure ends if the absolute error between the current and previous trials becomes less than a specific value, e.g. 10^{-4}. If this absolute error value is used, the total number of trials becomes unpredictable and the compiler can fail to vectorize the loop. In terms of the simulation efficiency using millions of cells, this is, however, undesirable, since the computation time becomes extremely high. Therefore, if the solution goes to line 4 in Algorithm 3, it is suggested to set the value of $w_p^{*\,\text{trial }0}$ similar to the value of w_p^* given on line 2. This gives a faster convergence, which is less than 20 trials for all cases simulated.

As also done in [5], the turbulence properties κ and ϵ at the centers of wall boundary cells for the subsequent time level are not calculated by solving Eq. (3), but their values are simply computed by

$$\kappa_p^{t+1} = \left(\frac{w_p^{*\,2}}{\sqrt{c_\mu}}\right)^{t+1}, \qquad \epsilon_p^{t+1} = \left(\frac{w_p^{*\,3}}{\upsilon\ \triangle y_p^{nor}}\right)^{t+1}. \tag{38}$$

Since the surface piercing case is simulated, the total incoming and outgoing fluxes for all dry cells – e.g. cell $(i+1, j+2)$ at time level t or cell $(i+2, j+2)$ at the top of the island in Fig. 3 – must be ensured to be zero. This can be achieved by several approaches, e.g. applying a conditional statement to distinguish such dry cells using a limiter depth value of 10^{-10} m at those wall boundary cells.

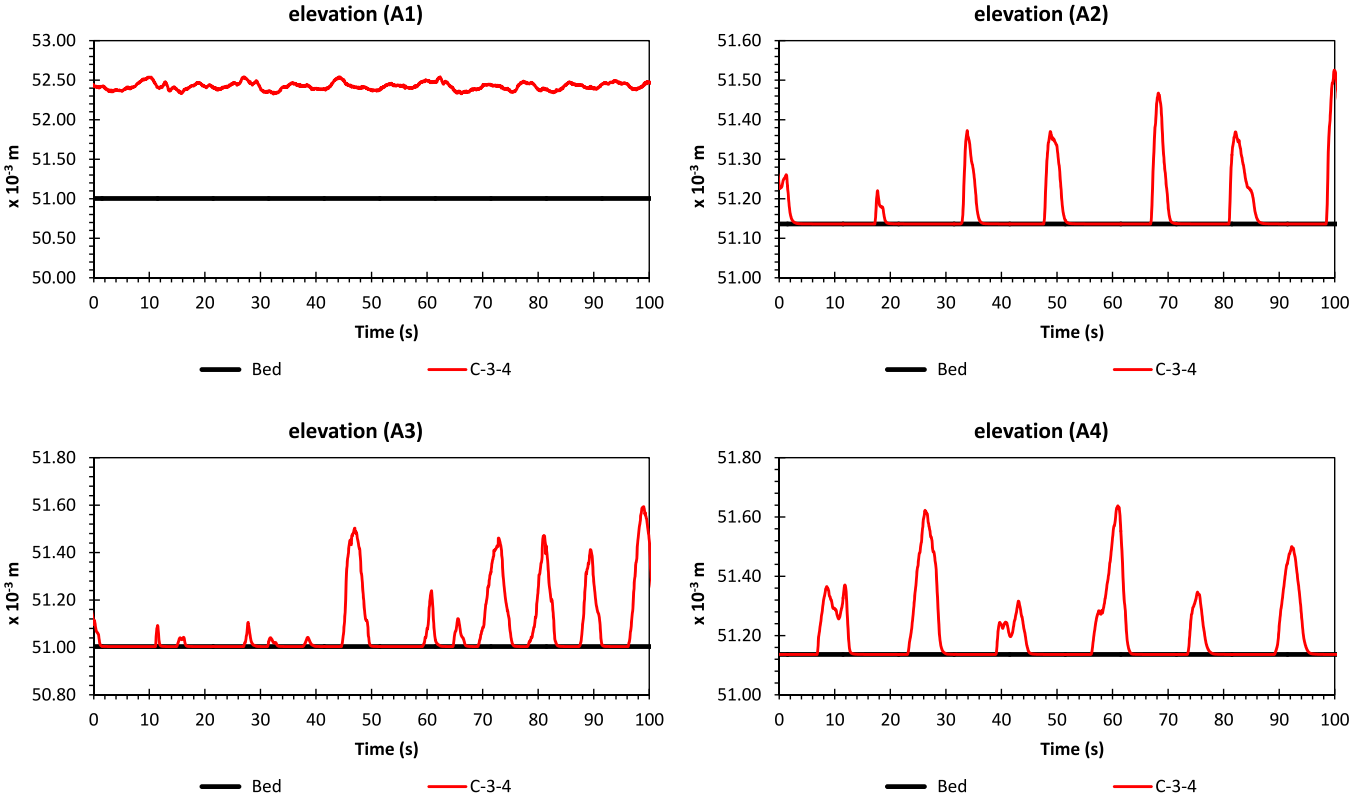


Fig. 18. Case 3: Results of C-3-4 at points A1, A2, A3, and A4.

Algorithm 3 Algorithm for calculating w_p^*.

1: **if** $\left[w_p^{tg} \le 11.067 \, \dfrac{\ln\left(11.067\ E\right)\ \nu_e}{\triangle y_p^{nor}\ \upsilon} \right]$ **then**

2: $\quad w_p^* = \dfrac{\upsilon\ w_p^{tg}}{\ln\left(11.067\ E\right)}$

3: **else**

4: $\quad$ solve the Newton–Raphson iteration as follows:

5: $$w_p^{*\,\text{trial }1} = w_p^{*\,\text{trial }0} - \frac{w_p^{*\,\text{trial }0} \ln\left(\dfrac{\triangle y_p^{nor}\ E}{\nu_e}\ w_p^{*\,\text{trial }0}\right) - \upsilon\ w_p^{tg}}{\ln\left(\dfrac{\triangle y_p^{nor}\ E}{\nu_e}\ w_p^{*\,\text{trial }0}\right) - 1},$$

6: $$w_p^{*\,\text{trial }2} = w_p^{*\,\text{trial }1} - \frac{w_p^{*\,\text{trial }1} \ln\left(\dfrac{\triangle y_p^{nor}\ E}{\nu_e}\ w_p^{*\,\text{trial }1}\right) - \upsilon\ w_p^{tg}}{\ln\left(\dfrac{\triangle y_p^{nor}\ E}{\nu_e}\ w_p^{*\,\text{trial }1}\right) - 1},$$

7: $\quad w_p^{*\,\text{trial }3} = \ldots$

8: **end if**

However, this may end up with many complex $if-then-else$ statements in addition to that given in Algorithm 3 which destroy any possibility for vectorizing the loops due to repeated branchings. For this reason, the hydrostatic and topography reconstructions given in Eqs. (21) and (28) and sketched in Fig. 5 – are utilized. For wet-wet interface, the hydrostatic reconstruction in Eq. (20) must be applied to cell p, so that the conservativeness of the convective and diffusive fluxes is achieved, where now $h_M^L < h_p$ and h_p itself does not change. Consequently, z_M must also be recalculated to have a proper bed slope value according to h_M^L. It should be noted z_p is obviously not altered. The value given by Eq. (19) will be similar to that given by Eq. (21) for this wet-wet case. Since wet-wet interface exists, cell p is not categorized as a wall boundary cell, thus no ScWF are applied. For wet–dry interface, it is clearly shown that no-flux-transfer condition between cells p and q must be satisfied. This can be achieved by the hydrostatic reconstruction that now $h_M^L = 0$, whereas h_p still does not change. In this case, the topography reconstruction must be applied to both cells p and q, where $z_M = \eta_M^L$. This can only be done by Eq. (21) and not by Eq. (19). Therefore, Eq. (21) must be used for all cases, which evidently gives no effect to wet-wet interface as previously mentioned in Section 3.1.2. It would easily be understood that these hydrostatic and topography reconstructions will also give no incoming nor outgoing fluxes for dry-dry interface, so that a zero-flux condition can always be ensured for dry cells, although these cells are included in all calculation levels.

For the sake of completeness, the proposed strategy is written as pseudo-code in Algorithm 4, which is applied in each calculation step of the RKSO method. It is shown in Algorithm 4 that since h_p^{t*} can be computed in a fully explicit way for the subsequent time level, this variable is used to decide whether the corresponding unit discharges, velocities, and turbulence properties are computed or not; the computational cost can thus be reduced especially for cases dominated by wet–dry problems. For internal cells (see Fig. 3), where the wall boundary geometries may change, the value of h_p^t is used as benchmark whether to apply the ScWF in Eq. (37) or to calculate the turbulence properties by solving the corresponding fluxes in Algorithm 1. It should be noted that the criterion on line 14 in Algorithm 4 is done by the hydrostatic and topography reconstructions.

4. Results

NUFSAW2D is written in Fortran and was compiled using Intel Fortran compiler 17.0.6. The target machine is a Linux com-

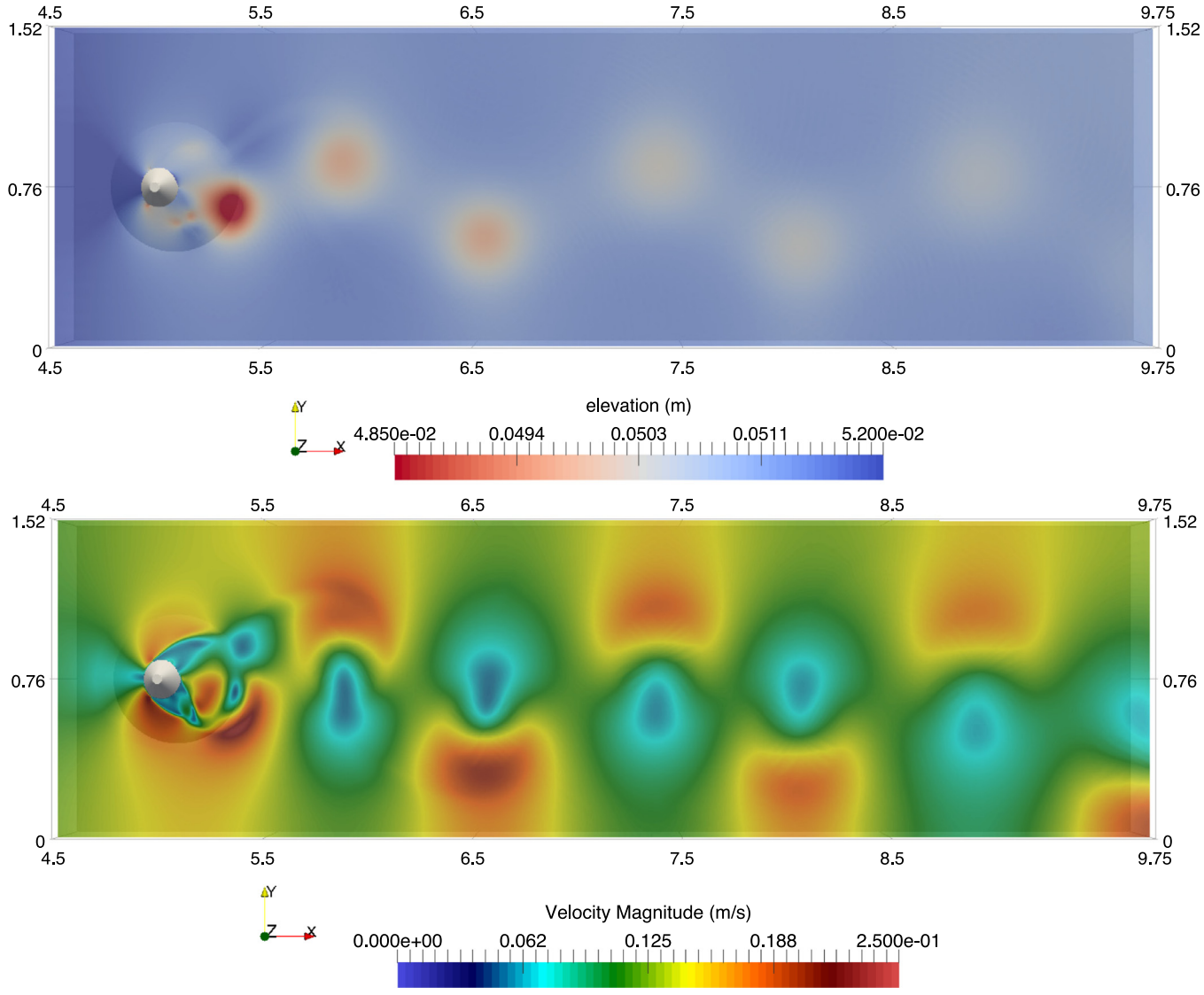


Fig. 19. Case 3: Visualization of wakes flows along the channel using C-3-4 at 45 s.

puting cluster "CoolMUC-2" operated by the Leibniz Supercomputing Centre [44]. Our simulations were performed with Open Multi-Processing (OpenMP) parallelization technique on 28 cores, where the optimization flag "–O3–xHost" was applied and double precision arithmetic (64-bit) was used. An overview of the data structure, parallel efficiency, and accuracy of our code can be found in our recent works [45,46]. $\nu_e = 10^{-6}$ m^2 s^{-1}, $\rho = 998.2$ kg m$^{-\frac{1}{3}}$, and $g = 9.81$ m s^{-2} are applied. All the turbulence coefficients are set similar to those given in Eq. (7), where $c_{\epsilon\Gamma} = 3.6$ is used.

4.1. Case 1: Flow past circular cylinder

We adopt this case from [47], who experimentally investigated turbulent wakes in a shallow water layer. This case is selected to test the ability of our model to achieve convergence with the ScWF for the use of very fine meshes. The experiment was performed on a 7.8 × 6 m channel, where a circular cylinder with a diameter of 0.62 m was located as shown in Fig. 6. A depth of 0.02 m and a velocity of 0.295 m/s were set constant at upstream, while free outflow condition was conducted at downstream. This produced a fully developed turbulent condition with a Reynold number of 5900 for the free stream flow or 183,000 based on the cylinder dimension. The Manning coefficient was 0.01 s m$^{-\frac{1}{3}}$. No measurement data at a specific point was presented; however, a Strouhal number of 0.215 was observed experimentally for indicating the vortex-shedding frequency. We present our results at point P1 in Fig. 6 only to show the convergence rates of our model.

For our simulation, we set 0.02 m depth with zero velocity initially. Four mesh configurations are tested: (1) 780 × 600, (2) 1040 × 800, (3) 1560 × 1200, (4) 2080 × 1600 cells, which give the square sizes of 0.01 m, 0.0075 m, 0.005 m, and 0.00375 m denoted by C-1-1, C-1-2, C-1-3, and C-1-4, respectively. The simulation is performed for 300 s (although NUFSAW2D can achieve convergence at about 200 s) to show the results do not deteriorate for the whole simulation. The result of C-1-4 at 283.2 s is visualized in Fig. 7, which shows the establishment of a vortex-street pattern. In general, a fuzzy and streaky pattern is shown due to small-scale turbulent mixing created by bottom shear and lateral instabilities.

To know the Strouhal number of the numerical results, the vortex-shedding frequency must be estimated. In [47], the oscillation periods of the vortex-street pattern were observed averagely over 10 cycles from the video record. For this, we observe the visualizations of C-1-1, C-1-2, C-1-3, and C-1-4 for the last 20 s

Algorithm 4 The proposed strategy in pseudo-code.

define an index-relationship between cell and edge
for total number of edges **do**
  calculate the convective, diffusive, and bed-slope fluxes using Eqs. (22), (27) and (28) in an edge-based manner F_{x_i} , F_{y_i} , G_{x_i} , G_{x_i} , S_{bx_i} , S_{by_i}
end for
for total number of cells **do**
  calculate depth h_p^{t*} using Algorithm 1
  using h_p^{t*} to decide whether unit discharges must be calculated or not
  if $h_p^{t*} > 10^{-10}$ m **then**
    calculate unit discharges without friction source term hu_p^{t*} and hv_p^{t*} using Algorithm 1
    transforming unit discharges hu_p^{t*} and hv_p^{t*} back to velocities u_p^{t*} and v_p^{t*}
    including the friction source terms to velocities using Eq. (29) and reupdating unit discharges using Algorithm 1
    calculating the turbulence properties based on the cell type (see Fig. 3) as follows:
    if internal cells **then**
      if $h_p^{(t*-1)} > 10^{-10}$ m **and** (at least) one of the corresponding convective fluxes (normal to edge) is zero **then**
        calculate κ_p^{t*} and ϵ_p^{t*} using the ScWF in Algorithm 3
      else if $h_p^{(t*-1)} > 10^{-10}$ m **then**
        calculate κ_p^{t*} and ϵ_p^{t*} by solving Eq. (3)
      else
        set κ_p^{t*} and ϵ_p^{t*} to 10^{-15}
      end if
    else if wall boundary cells **then**
      calculate κ_p^{t*} and ϵ_p^{t*} using the ScWF in Algorithm 3
    else if flow (open) boundary cells **then**
      calculate κ_p^{t*} and ϵ_p^{t*} without solving the ASM
    end if
  else
    set hu_p^{t*}, hv_p^{t*}, u_p^{t*}, v_p^{t*}, κ_p^{t*}, and ϵ_p^{t*} to 10^{-15}
  end if
end for

of our simulations every 0.1 s. For the sake of brevity, only the results of C-1-4 are visualized here, see Fig. 8. One can see the vortex-shedding appears by the fuziness of the turbulent mixing. The lateral instabilities keep continuing after 283.2 s until a similar vortex-street pattern but with a different phase appears at about 292.9 s. To this end, the period is approximated to be 9.7 s. A similar way is observed for C-1-1, C-1-2, and C-1-3 as well giving the periods of 8.5 s, 8.9 s, and 9.3 s, respectively. C-1-1, C-1-2, C-1-3, and C-1-4 thus calculate the Strouhal numbers of 0.247, 0.236, 0.226, and 0.217, respectively. C-1-4 becomes the most accurate one for representing the Strouhal number in agreement with the observation. This shows that the mesh size refinement plays an important role for the accuracy.

The convergence histories at point P1 are shown in Fig. 9. In general, all the configurations achieve a similar convergence approximately at 200 s. C-1-1 computes the lowest magnitudes, where C-1-3 and C-1-4 exhibit similar magnitudes. One can also see that our results do not deteriorate using the very fine meshes and are always able to achieve convergence.

4.2. Case 2: Turbulent recirculating flow due to channel expansion

This case is adopted from [48,49], where both experiment and numerical investigations (with StarCCM+) were performed to investigate the recirculation zone length (L_{circ}). Similar to case 1, we select this case to test our model's ability to achieve convergence for very fine meshes with the ScWF – and to show the accuracy of our model in predicting L_{circ} as well. The experiment was conducted with a 8×0.8 m channel which has a symmetrical rectangular section and a streamwise mean slope of 0.18%, see Fig. 10.

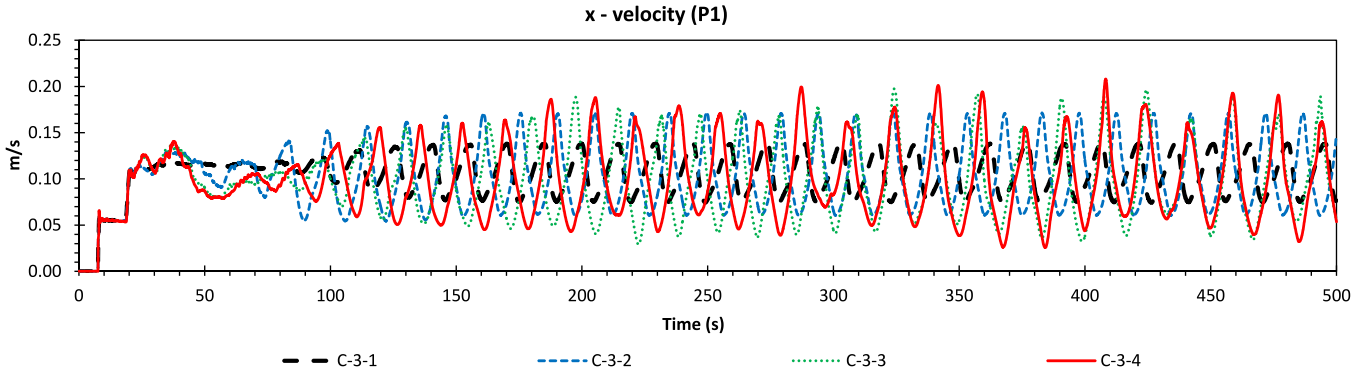


Fig. 21. Case 3: Convergence history for quasi-steady states at point P1.

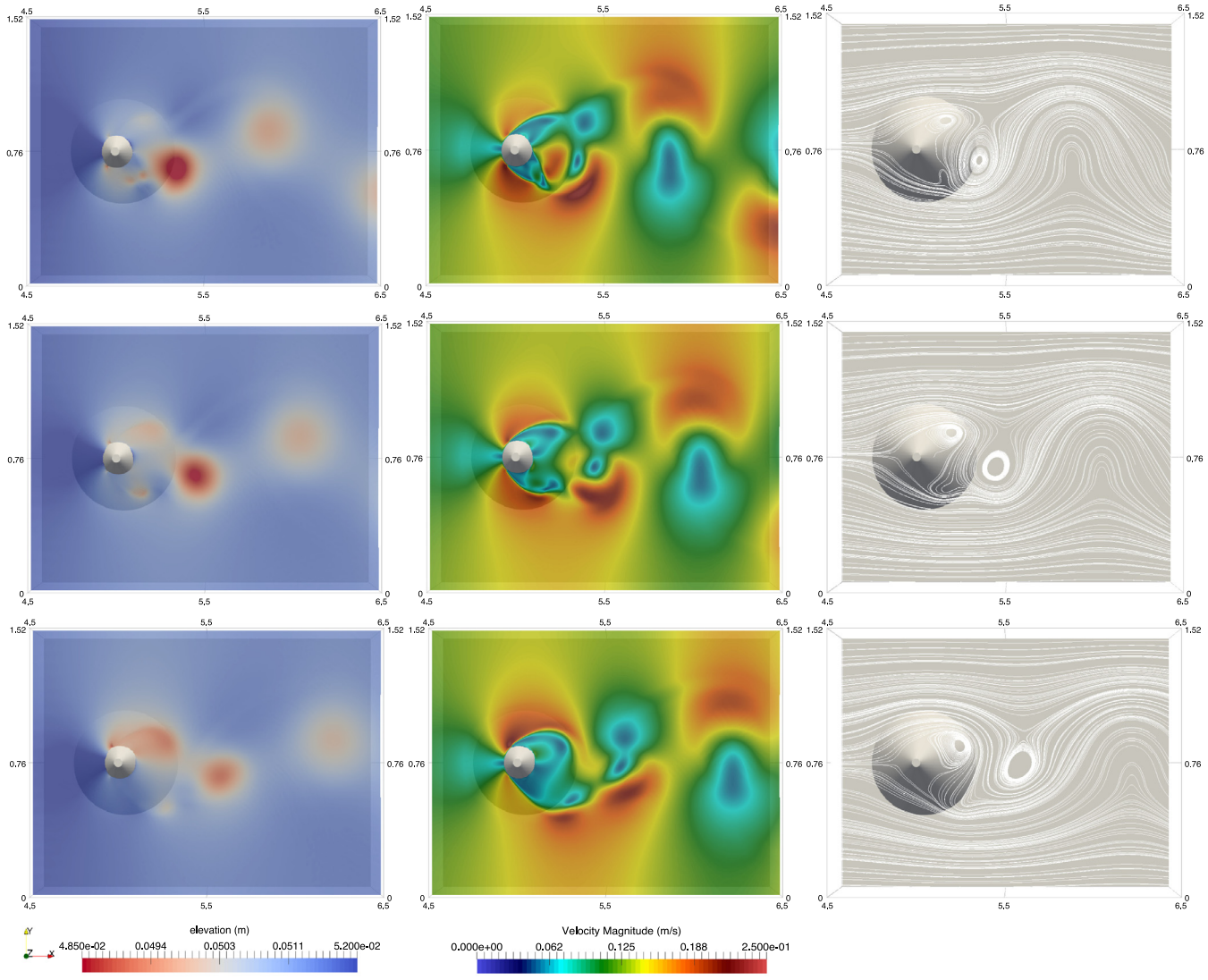


Fig. 20. Case 3: Visualization of wakes flows near the conical island using C-3-4 at 45 s, 47 s, and 49 s.

The bed was PVC made with a smooth surface. To conduct an expansion in the channel, a PVC board was added to flexibly change the channel width. A constant discharge of 20 l/s was given at upstream, whereas outflow condition was applied at downstream. The Reynolds number was 94,000 for the free stream flow. A bottom roughness value of 3.4 $\times 10^{-5}$ m was used; we thus convert this value here using the Colebrook-White equation and the Darcy–Weisbach coefficient and assume as a constant value resulting a Manning coefficient of 0.0094 s $m^{\frac{1}{3}}$. StarCCM+ was employed in [48] for numerical simulations with a rigid-lid (constant water-depth) assumption, thus the effect of free-surface fluctuations was not taken into account. Instead of simulating with the original channel condition in Fig. 10, a 20 m long channel was modeled to ensure that the inlet and outlet boundary conditions did not affect the flow patterns at the expansion region. The StWF with a condition of $30 < y^+ < 500$ were chosen. For this, a grid study was undertaken to know a proper mesh size, with which the 20 m long channel was divided into three parts for one simulation: coarse (0.0125 m), average (0.00625 m), and fine meshes (0.003125 m) for achieving convergence.

In this work, such a grid study is not required. To prove that NUFASW2D can achieve convergence for all mesh sizes, we simply choose four mesh configurations: (1) 800 × 80, (2) 1600 × 160, (3) 2000 × 200, (4) 2670 × 267 cells, which give the square sizes of 0.01 m, 0.005 m, 0.004 m, and 0.003 m denoted by C-2-1, C-2-2, C-2-3, and C-2-4, respectively. In the experiment, the recirculating zone was always in an unsteady condition, thus L_{circ} changed with respect to time. Averagely, it was observed that L_{circ} = 1.29 m with the recirculation center at (0.735, 0.104) m – and a depth of 0.156 m and a velocity of 0.23 m/s at point P1. First, we must estimate a proper steady state condition for this simulation: by observing the convergence history at P1 and by ensuring a small difference between the inflow and outflow discharges. For this, we perform our computations for 400 s and present in Fig. 11 the convergence histories at P1 for all mesh sizes. It shows all configurations achieve similar convergence rates approximately at 150 s. Note for the velocity magnitude at P1, C-2-4 gives the smallest error amplitudes from the observed data (0.23 m/s) among the others. Fig. 11 shows stable period fluctuations meaning the convergence is ensured and no chaotic oscillation is exhibited. We also observe after 150 s small differences of the inflow-outflow discharges in the order of 10^{-4} m^3/s. Based on these facts, comparing the recirculation zone at 400 s is thus acceptable.

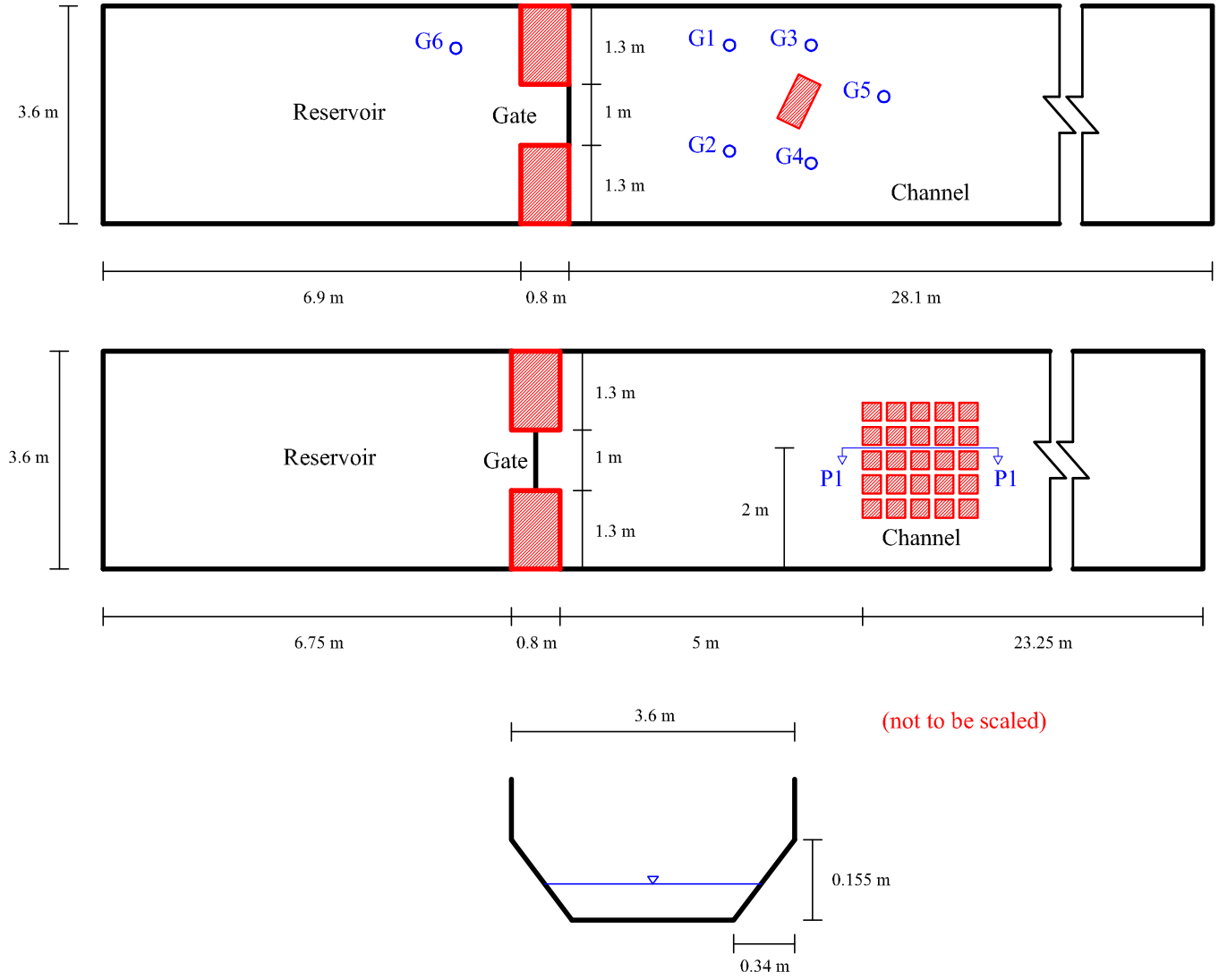


Fig. 22. Case 4: Domain of dam-break flows through obstacle(s) with its channel shape.

To compare L_{circ}, we plot the x velocities along section 1-1 (at the centers of the first boundary cells); when the sign of such velocities changes from negative to positive, the reattachment point can be known. Fig. 12 shows all configurations almost give a similar result of $L_{circ} = 0.85$ m; this value slightly differs from the observed one. To compare the y velocity distributions, we plot in Fig. 13 the velocities along $x = 0.675$ m showing that no significant differences are exhibited where C-2-4 becomes slightly more accurate among the others at $y > 0.2$ m. The velocity magnitude near the recirculation zone at 400 s for C-2-4 are visualized in Fig. 14. Along the separating streamlines, the mixing layers appear which ends approximately at (1 , 0) m. The recirculation center is created at (0.39 , 0.12) m. Although L_{circ} computed by our model still slightly differs from the observed data, we have shown that our computations can always ensure convergence using very fine meshes.

4.3. Case 3: Recirculating wakes around a surface-piercing conical island

This case is selected based on [2], who experimentally and numerically investigated recirculating wakes behind a conical island. With this case we show the ability of our proposed strategy to accurately simulate the wakes properties behind the island, to deal with wet–dry problems in the case of fluid-structure-interactions near the island, and to ensure that our results do not deteriorate for the use of very fine meshes by achieving convergence. The experiment was conducted in a 9.75 × 1.52 m channel, see Fig. 15. One measurement point – P1 (5.73, 1.02) m – was installed behind the conical island. The Manning coefficient was 0.01 s $m^{\frac{1}{3}}$. The Reynolds number was 5865 for the free stream flow or 47,296 based on the representative island dimension, showing that the flow was in a fully developed turbulent condition. At upstream, the depth and the velocity were set to 0.051 m and 0.115 m/s, constantly. Meanwhile, the outflow boundary condition was applied at downstream.

In [2], the observation results were only given for a timeframe of about 100 s. These results are supposed to be obtained during 100 s after a stable condition had been reached. To this regard, a total simulation time of 500 s is settled here. The results obtained during the last 100 s are thus compared with the observation results (after slightly shifting the time series due to different transient phases). Actually, NUFSAW2D could achieve the stable wakes patterns at the timeframe of 200–300 s. However, this case is still performed for 500 s, since we would like also to com-

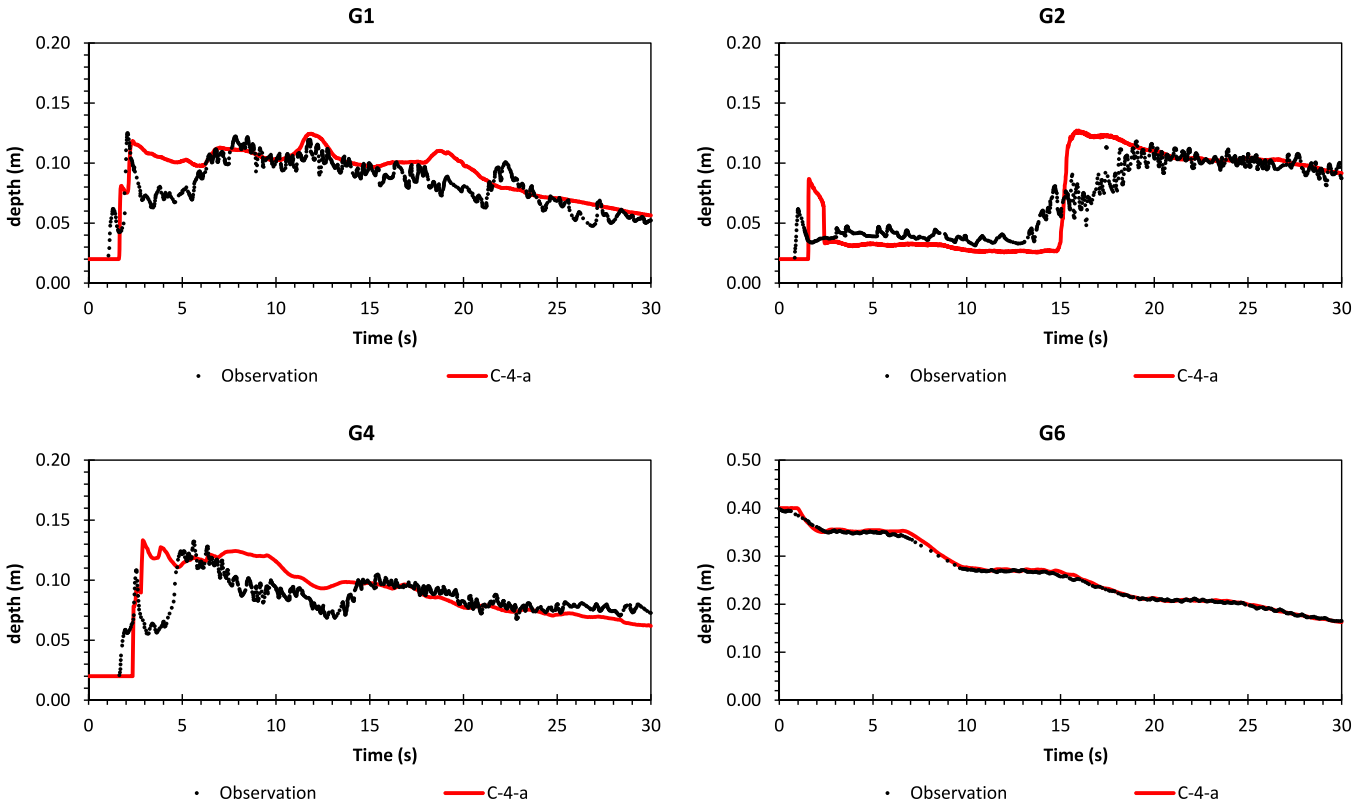


Fig. 23. Case 4: Comparison of depths between observation and numerical results for C-4-a.

pare the convergence rate and to show that the results do not deteriorate using the four mesh configurations due to the ScWF, instead of the StWF. Four mesh configurations are selected for our simulations: (1) 650 × 102, (2) 1300 × 204, (3) 2600 × 408, (4) 3900 × 612 cells, which give the square cell sizes of 0.015 m, 0.0075 m, 0.00375 m, and 0.0025 m, respectively denoted by C-3-1, C-3-2, C-3-3, and C-3-4, respectively. The first configuration was chosen to approximately follow the mesh size used for numerical model in [2] (a square cell size of 0.0152 m). We compare our result C-3-1 with both observation data and the 2D model of [2] (Lloyd and Stansby) to give a fair comparison with the mesh size used. Meanwhile, C-3-2, C-3-3, and C-3-4 are only compared with the observation data. For all simulations, we set the depth to 0.051 m and the velocities to zero, initially.

The results of C-3-1 are given in Fig. 16 showing that for C-3-1 NUFSAW2D detects a similar characteristic of wakes velocity periods to that computed by Lloyd and Stansby. Both C-3-1 and their model produce a maximum deviation of approximately 10 s for the wakes periods from the observed data. However, the wakes velocity amplitudes are underestimated by NUFSAW2D approximately half of the observation results. The 2D model of Lloyd and Stansby is more proper in predicting such amplitudes. These errors indicate that the CU scheme has a quite deficient result for simulating recirculating flows, which is in accordance with the fact investigated in [50]. Up to here, we assume that the mesh resolution will play a significant role in increasing the accuracy of the results in contrast to that investigated by Lloyd and Stansby. To prove this assumption, the meshes are refined and the results of C-3-2, C-3-3, and C-3-4 are shown in Fig. 17. One can clearly see that the results especially C-3-4 now becomes better. Using the mesh size 36 × smaller than that of C-3-1, NUFSAW2D with C-3-4 can accurately predict both the amplitudes and the periods of the wakes velocity. The maximum magnitudes of the wakes velocity in every phase are captured correctly. In general, a similar characteristic is shown by C-3-3 for both the amplitudes and periods. Only some non-significant differences exist particularly for the velocity amplitudes, e.g. in x direction, C-3-3 computes the amplitude slightly better than does C-3-4 during 14–18 s, but C-3-4 becomes more accurate to predict the amplitude during 74–78 s. Although C-3-2 still shows some errors for the wakes velocity periods (a maximum deviation of approximately 6 s), it has established a quite significant improvement from C-3-1 in calculating the maximum amplitudes of the wakes velocity. Using the mesh size 4 × smaller than that of C-3-1, NUFSAW2D with C-3-2 can detect the maximum wakes velocity amplitudes in y direction accurately. The velocity amplitudes in x direction are also calculated appropriately by C-3-2 and much better than the results of C-3-1.

Based on these facts one can see that refining the mesh size from C-3-1 to C-3-2 (4 × smaller) results outstandingly in more accurate results for the amplitudes (more than 2 ×), in which the maximum deviation of the period becomes 4 s smaller (from 10 s to 6 s). The mesh size refinement from C-3-2 to C-3-3 does resolve the inaccuracy of detecting the wakes periods that now the maximum deviation becomes only 1 s. Although the velocity magnitudes of C-3-3 are better than those of C-3-2, the accuracy accretion for such magnitudes is, however, not as significant as that from C-3-1 to C-3-2. It is also shown that the refinement from C-3-3 to C-3-4 only gives slightly different results. To this regard, it is agreeable to say that the mesh size has a significant effect on the accuracy especially with the 16 × smaller cell. Indeed, no more significant improvements are shown with the 36 × smaller cell. Nevertheless, NUFSAW2D has shown its ability with the ScWF to not deteriorate the results from C-3-3 to C-3-4.

To show the wet–dry phenomena near the conical island, four points – A1 (4.91, 0.76) m, A2 (5, 0.67) m, A3 (5.091, 0.76) m, and A4 (5, 0.851) m – are selected to present the water surface fluctua-

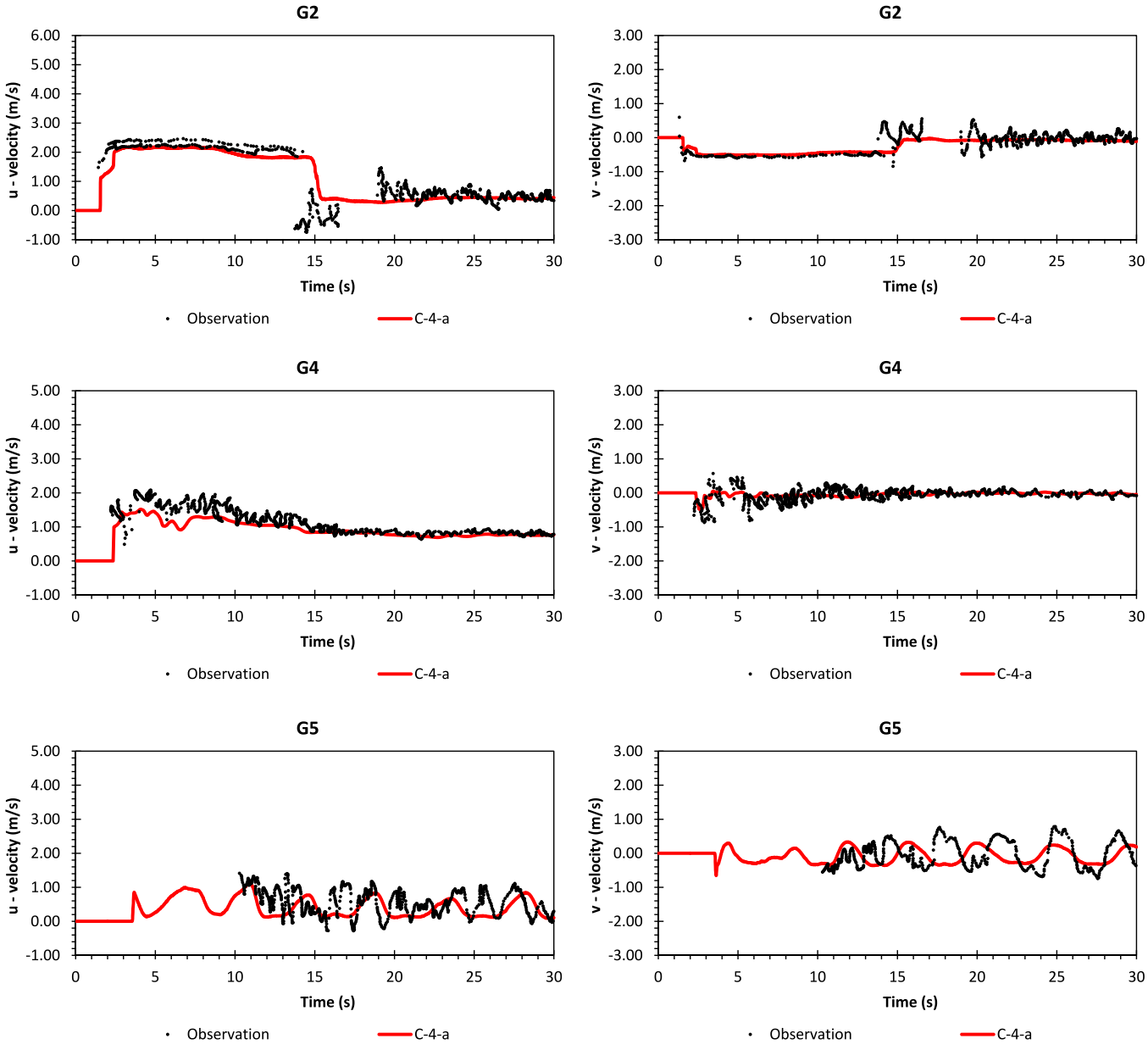


Fig. 24. Case 4: Comparison of velocities between observation and numerical results for C-4-a.

tions at those cell center locations. The results are given in Fig. 18. As expected, at point A1 (in front of the conical island), the water surface elevation is always greater than the bed elevation causing this cell to be a wet cell and no ScWF are required if none of the corresponding convective fluxes (normal to edge) is zero. The turbulence properties of this cell will be treated based on line 17 in Algorithm 4. If (at least) one of such corresponding convective fluxes is zero, the ScWF must be applied to this cell, as this cell becomes a wall boundary cell, so that the turbulence properties are calculated based on line 15 in Algorithm 4. Different to point A1, at points A2, A3, and A4 (at the left side, behind, and right side of the conical island), the wet–dry phenomena occur periodically throughout the simulation. Therefore, when these cells become dry, all the turbulence properties must be computed according to line 19 in Algorithm 4 – and as soon as these cells become wet and all the corresponding (normal) convective fluxes are non-zero, the calculation on line 17 in Algorithm 4 must be performed. Similarly, when these cells become wet but (at least) one of corresponding (normal) convective fluxes is zero, the ScWF must be applied. Following this algorithm, NUFSAW2D has produced highly accurate results to model the recirculating wakes flows dealing with wet–dry problems.

The wakes flows are visualized along the channel for $x = 4.5$–9.75 m in Fig. 19 for the water surface elevation and velocity magnitude – and the details around the conical island for $x = 4.5$–6.5 m are given in Fig. 20 for the water surface elevation, velocity magnitude, and streamline pattern using C-3-4. In general, the results shows the recirculating flows characterized by vigorous and periodic vortex shedding. We denote here the first, second, third, and fourth quadrants in a cartesian plane in connection with the center of the conical island. As shown in Fig. 20, at the second quadrant, the water flows mostly parallel to the x axis, except close to the flat island apex, the water tends to flow following the shape of the island. At the first quadrant, water starts to separate from the flow of the second quadrant and produces mixing close to the bed. It is shown that a horizontal shear layer exists in this region.

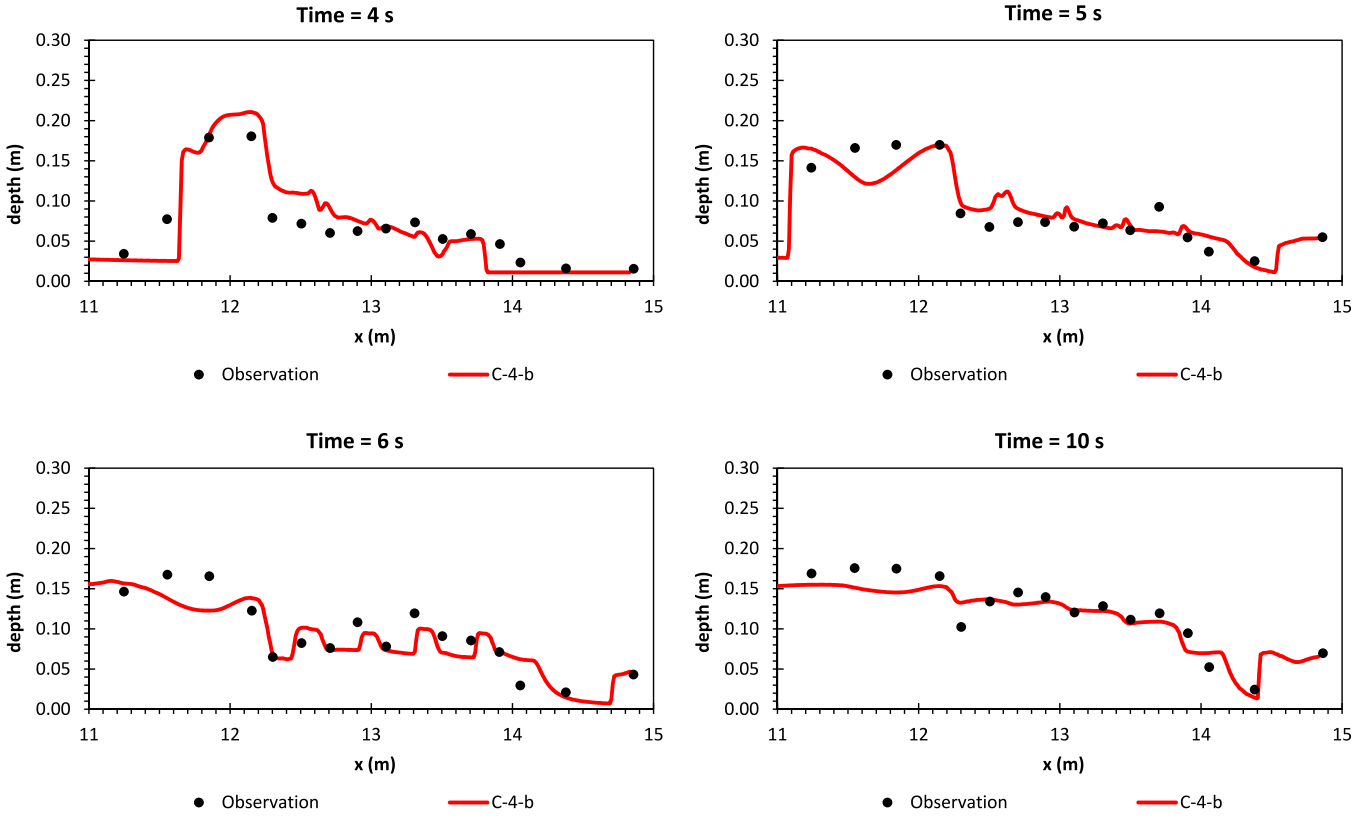


Fig. 25. Case 4: Comparison of depths between observation and numerical results for C-4-b.

At the fourth quadrant, the flow is affected by the water flowing from the first quadrant, thus creating vortex, in which the wakes flow is clearly shown. This leads to a condition where the velocity magnitude is relatively low behind the island (at the center). A typical secondary flow close at the bed appears at this quadrant. These results demonstrate a good qualitative agreement with the experiment of [2].

Finally, the convergence history is presented in Fig. 21. The x velocities at point P1 are selected. Obviously, all the corresponding y velocities show a similar characteristic. It is shown that both C-3-1 and C-3-2 can achieve a faster convergence (approximately) after 150 s. Meanwhile, both C-3-3 and C-3-4 reach the convergence slightly later (approximately) after 200 s. Despite reaching a faster convergence, both C-3-1 and C-3-2 are not better than C-3-3 and C-3-4, since C-3-1 and C-3-2 cannot capture the maximum wakes amplitudes appropriately. Also, the wakes periods are still overestimated by both C-3-1 and C-3-2 as shown in Figs. 16 and 17. The ScWF applied have ensured the convergence/stability for such (extremely) mesh refinements, where no chaotic wakes pattern is observed.

4.4. Case 4: Dam-break flows through obstacle(s)

We consider here two experimental test cases dealing with dam-break flows: through one obstacle [51] and through 25 obstacles [52]. These cases are selected to show the ability of our proposed strategy in simulating discontinuous turbulent flow characteristics using very fine meshes which interact with structure(s) and are affected by wet–dry phenomena. The channel was 35.8 m long and 3.6 m wide, with a trapezoidal cross section, sketched in Fig. 22. The water elevation was set to 0.4 m initially at the reservoir. Meanwhile, the initial water elevations of 0.02 m and 0.011 m were set for the first and second cases, respectively giving a dry bed initial condition at the sloping parts of the trapezoidal channel. The Manning coefficient was set to 0.01 s $m^{\frac{1}{3}}$ for both cases.

In this paper, we discretize the domains for both cases into 3600×360 cells, which give a square size of 0.01 m. We denote the first and second cases as C-4-a and C-4-b. The simulations are performed for 30 s and 10 s for the first and second cases, respectively.

We show in Fig. 23 the comparison of depths between observation and numerical results for C-4-a. At G1, our model detects the maximum bore at 2 s accurately; however, at 2–5 s it predicts the elevations slightly higher. After 5 s, sufficiently accurate results are shown by NUFSAW2D. At G2, our model calculates the first incoming wave properly at around 1.5 s and the second incoming wave approximately at 15 s is detected properly as well. Between 15–20 s, NUFSAW2D exhibits slightly higher results but after 20 s it computes the elevations accurately. A similar characteristic with G1 is shown at G4, where the first bore at 2 s is appropriately simulated and higher elevations are shown at 2–5 s and 7–12 s. However, our model exhibits accurate results after 12 s. At G6, NUFSAW2D predicts accurate results showing the incoming discharge from the reservoir to the channel is accurately computed. We present in Fig. 24 the comparison of velocities between observation and numerical results for C-4-a. At G2 and G4, again our model shows its ability to calculate velocities in x and y directions accurately. At G5, velocity fluctuations appear showing the turbulent wakes occurs periodically behind the obstacle. Our model is able to detect such fluctuations although non-significant oscillations still appear.

In Figs. 25 and 26, we show the comparison of depths and velocities between observation and numerical results for C-4-b. At 4–10 s NUFSAW2D predicts proper values for elevation along section 1–1. Good agreements are shown between the numerical re-

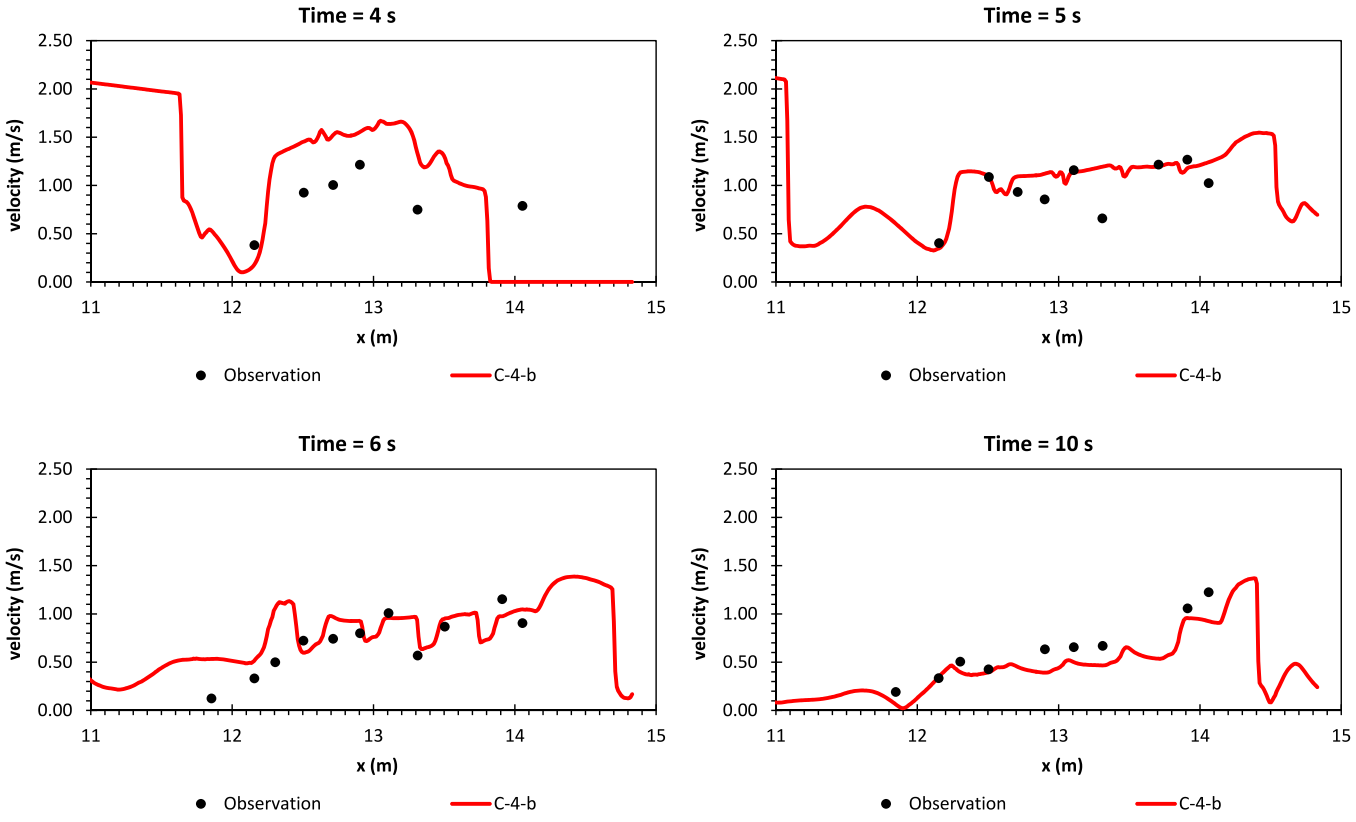


Fig. 26. Case 4: Comparison of velocities between observation and numerical results for C-4-b.

sults and the observed data for velocity as well. We have shown that our model is in general capable of simulating both cases properly.

We visualize the wave propagations for both cases in Figs. 27–30. In Figs. 27 and 28, the water propagates from the reservoir to channel at 1 s, where the channel banks are still dry. At this period, the cells classified as wall boundary are still similar to those in initial condition. Suddenly, after 1 s, the water starts to inundate the channel banks showing this case deals with moving boundary geometries; the cells classified as wall boundary thus (always) change during runtime. Especially with the fine meshes used, the transformations from internal to boundary cells (vice versa) become even more sensitive. Our model is able to detect such transformations and to compute the proper turbulence values, thus ensuring stability. The interactions between water and the obstacle(s) are depicted in detail in Figs. 29 and 30. One can see the moving boundary geometries, e.g. at $x = 9$ m for C-4-a or at $x = 13$ m for C-4-b, change during runtime. See also, e.g. at $x = 14$ m for C-4-a, the velocity magnitudes near the wall boundaries at the channel bank fluctuate noticeably; it is still dry at 1 s, increases up to 2.2 m/s at 4 s, and decreases to 0.5 m/s at 10 s. One can also observe that the vortexes periodically appear behind the obstacle for C-4-a and the hydraulic jumps exist behind the obstacles for C-4-b.

Finally, we present the streamline patters for both cases in Figs. 31. For C-4-a, the recirculating flows appear behind the obstacle at 6 s and become larger at 10 s. For C-4-b, the recirculating flows exist in front of the obstacles at 6 s, whereas no recirculation is detected behind. However, at 10 s the recirculation becomes smaller at upstream, where the small recirculation appears at downstream.

4.5. Performance comparison

In each case, Δt may change during runtime for each time level due to the limitations previously mentioned in Section 3.2 and depending on the mesh size; the total number of simulation steps is different thus giving non-linear relationships of CPU time for each case. For example, to reach 300 s in case 1, C-1-4 requires smaller Δt than that for C-1-1; therefore, the ratio of CPU time between both of them depends not only on the proportion of the total number of cells, but also on the proportion of the total number of simulation steps. Prior to giving the CPU time for each case simulated, it is important to note that we use Mcell/s/core (million cells per second per core) being a comparison between the number of cells for a total number of calculation steps that can be processed per unit of time using one core – to express a comparable performance metric (PM) for our simulations as well as to show the scalability of our model. Our code NUFSAW2D has been tested on several machines, e.g. Sandstorm with Intel Xeon E5-2690 (Sandy-Bridge-E) [53], CoolMUC-2 with Intel Xeon E5-2697 v3 (Haswell), and CoolMUC-3 with Intel Xeon Phi (Knights Landing) [44]. In our recent work [45], we proposed a novel load balancing strategy for parallelizing shallow flow simulations to ensure the scalability of our code, where we were able to obtain the efficiencies of 98% (8 cores) and 87% (16 cores) with Sandstorm – and of 97% (56 cores) and 88% (64 cores) with CoolMUC-3 showing NUFSAW2D scaled very well. This leads our model to achieve 1.81 Gflop/s/core and 0.54 Gflop/s/core which are about 31% and 21% of the theoretical peak performances of Sandstorm and CoolMUC-3, respectively.

In this paper, we can achieve the average efficiencies of 98% (16 cores) and 89% (28 cores) using CoolMUC-2 for all simulations leading to an average PM of 2.2 Mcell/s/core. So, we present the CPU time for each case simulated here according to this PM value.

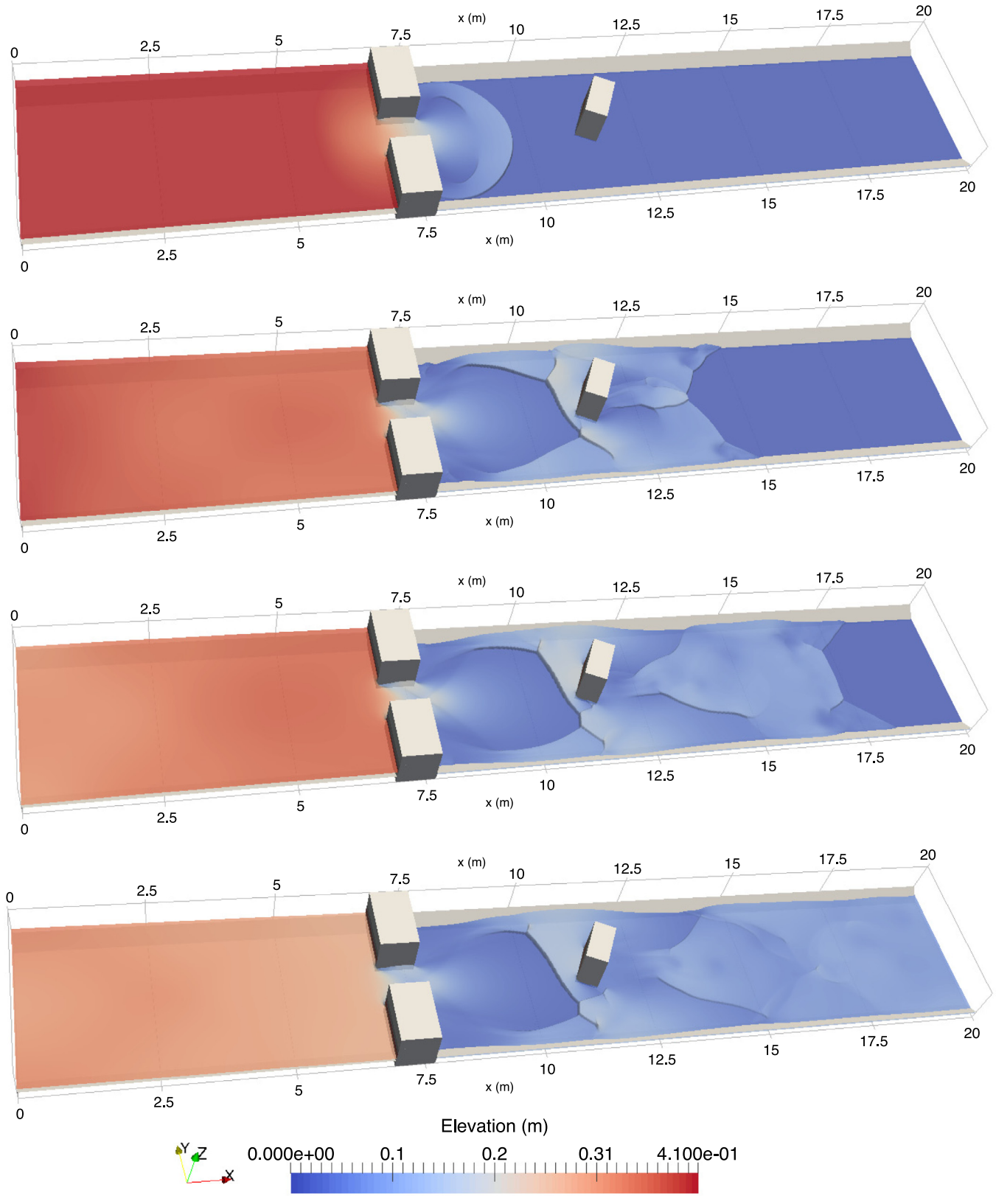


Fig. 27. Case 4: Visualization of depth propagation at 1 s, 4 s, 6 s, and 10 s for C-4-a.

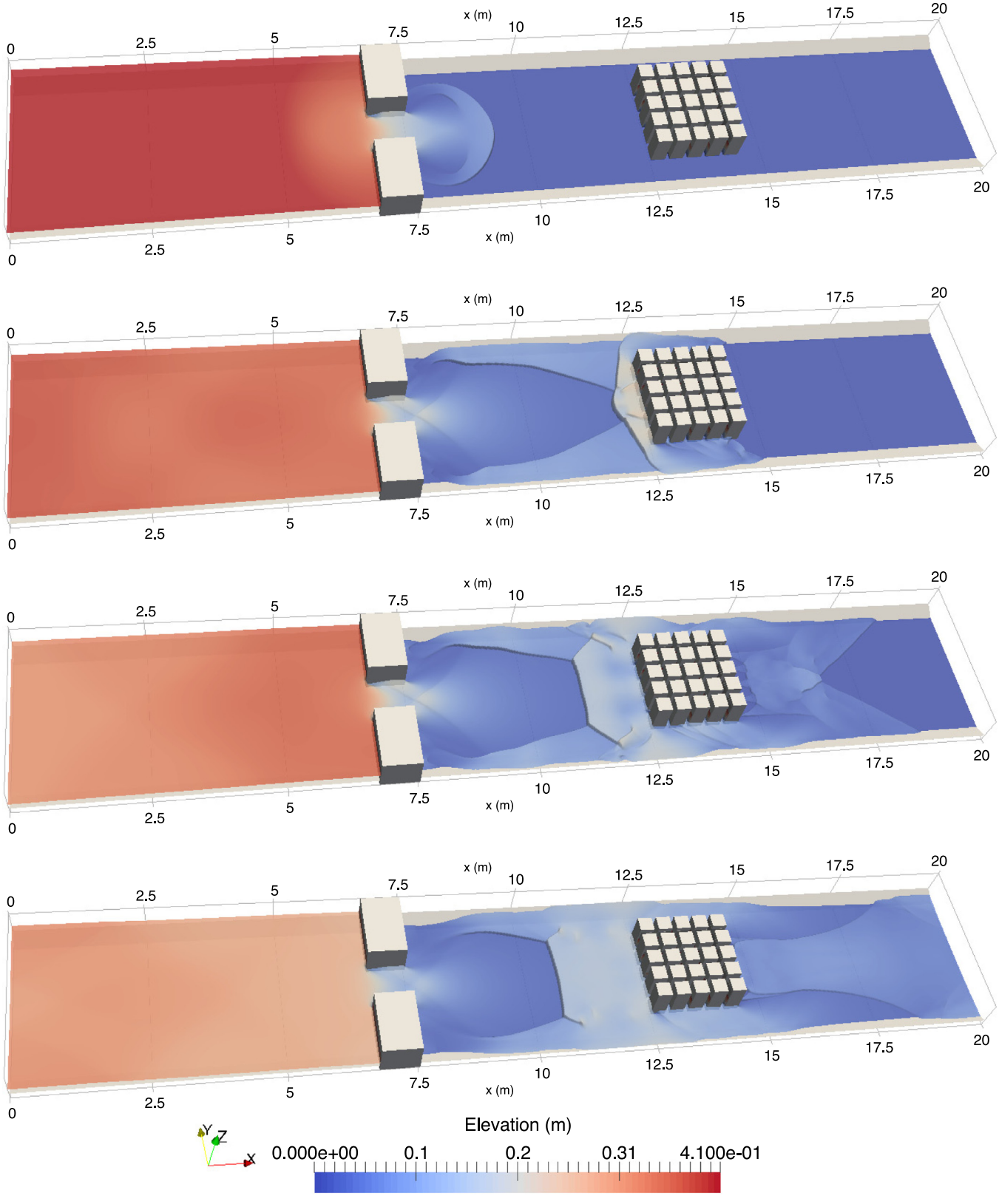


Fig. 28. Case 4: Visualization of depth propagation at 1 s, 4 s, 6 s, and 10 s for C-4-b.

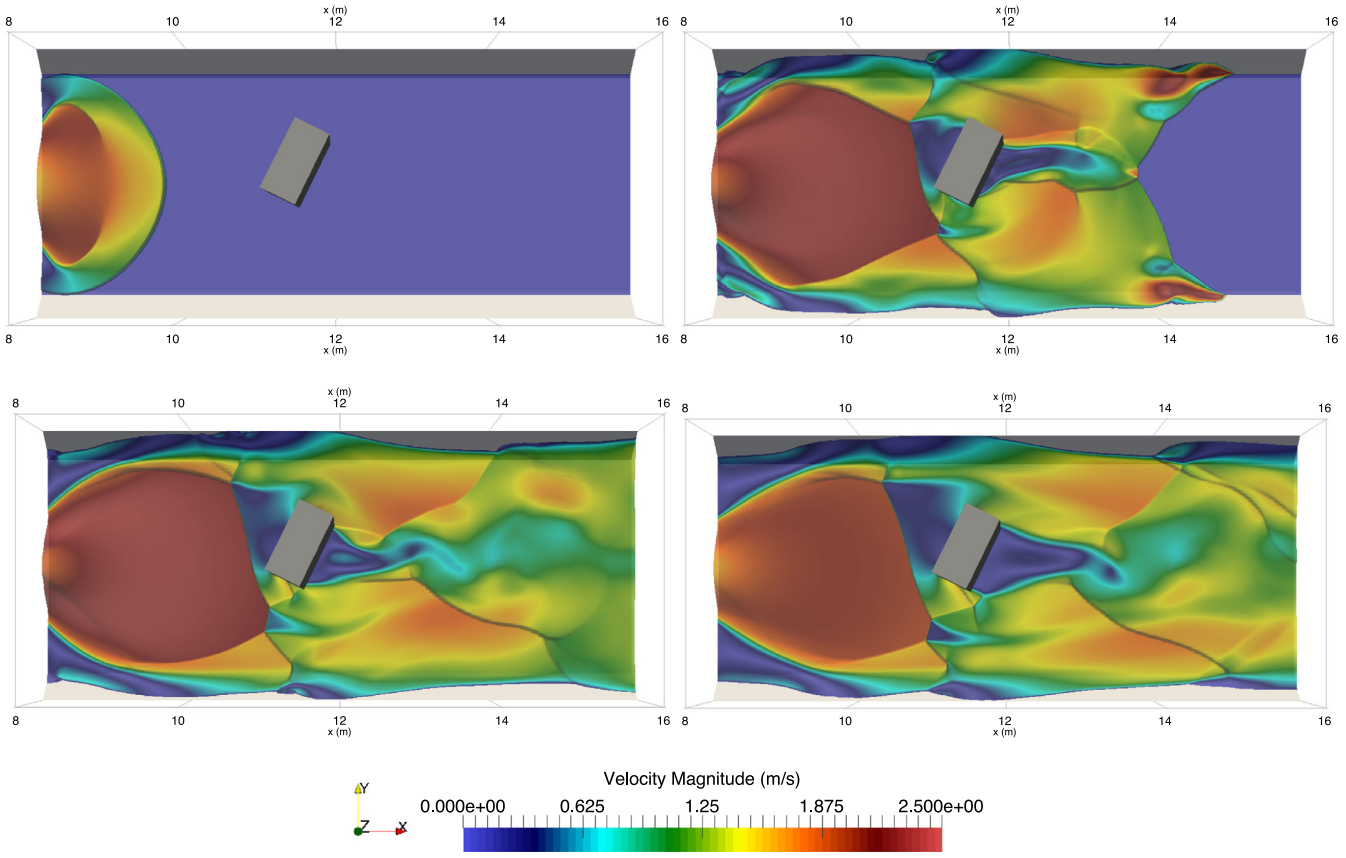


Fig. 29. Case 4: Visualization of velocity propagation at 1 s, 4 s, 6 s, and 10 s for C-4-a (top left, top right, bottom left, bottom right).

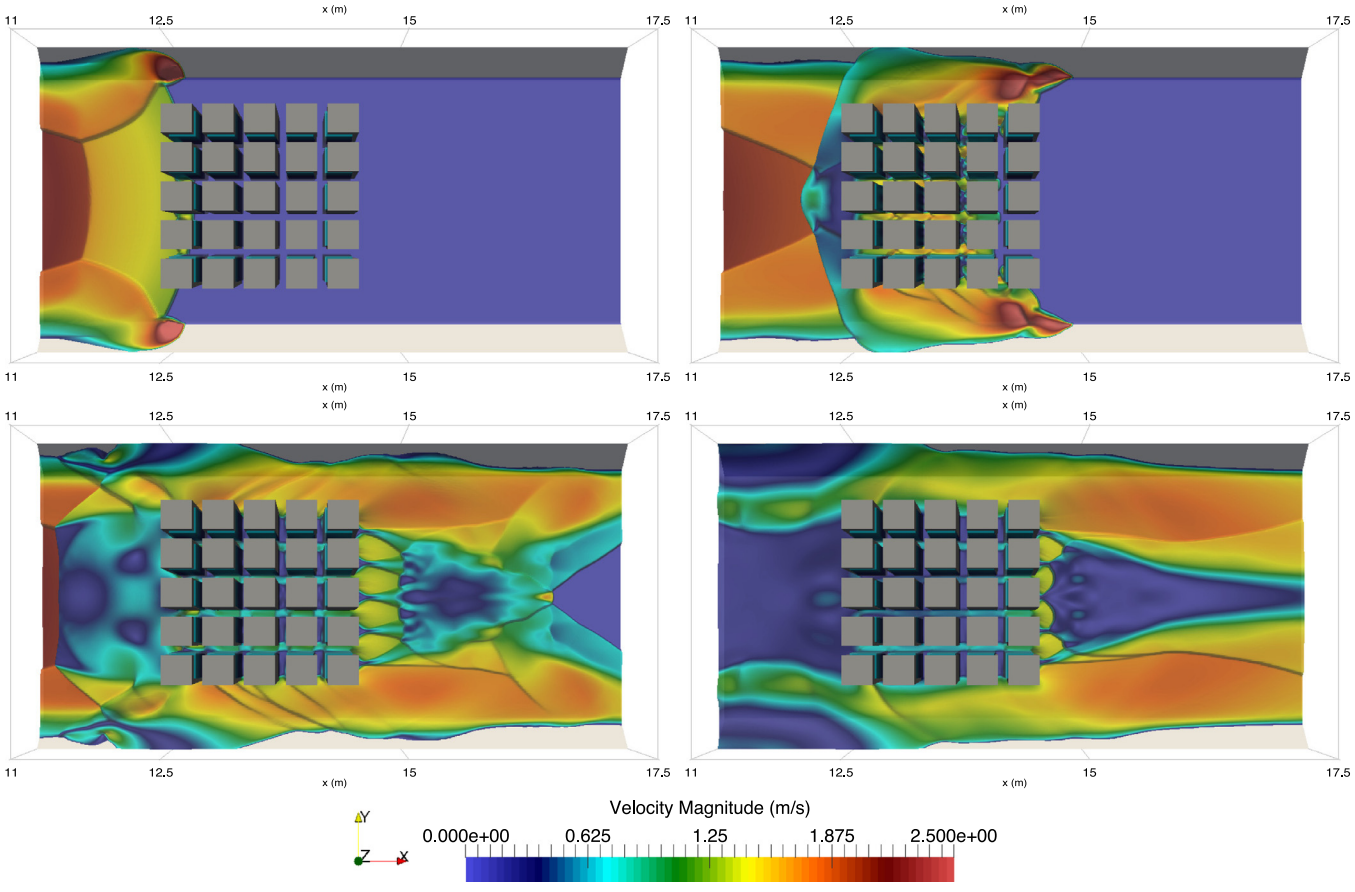


Fig. 30. Case 4: Visualization of velocity propagation at 1 s, 4 s, 6 s, and 10 s for C-4-b (top left, top right, bottom left, bottom right).

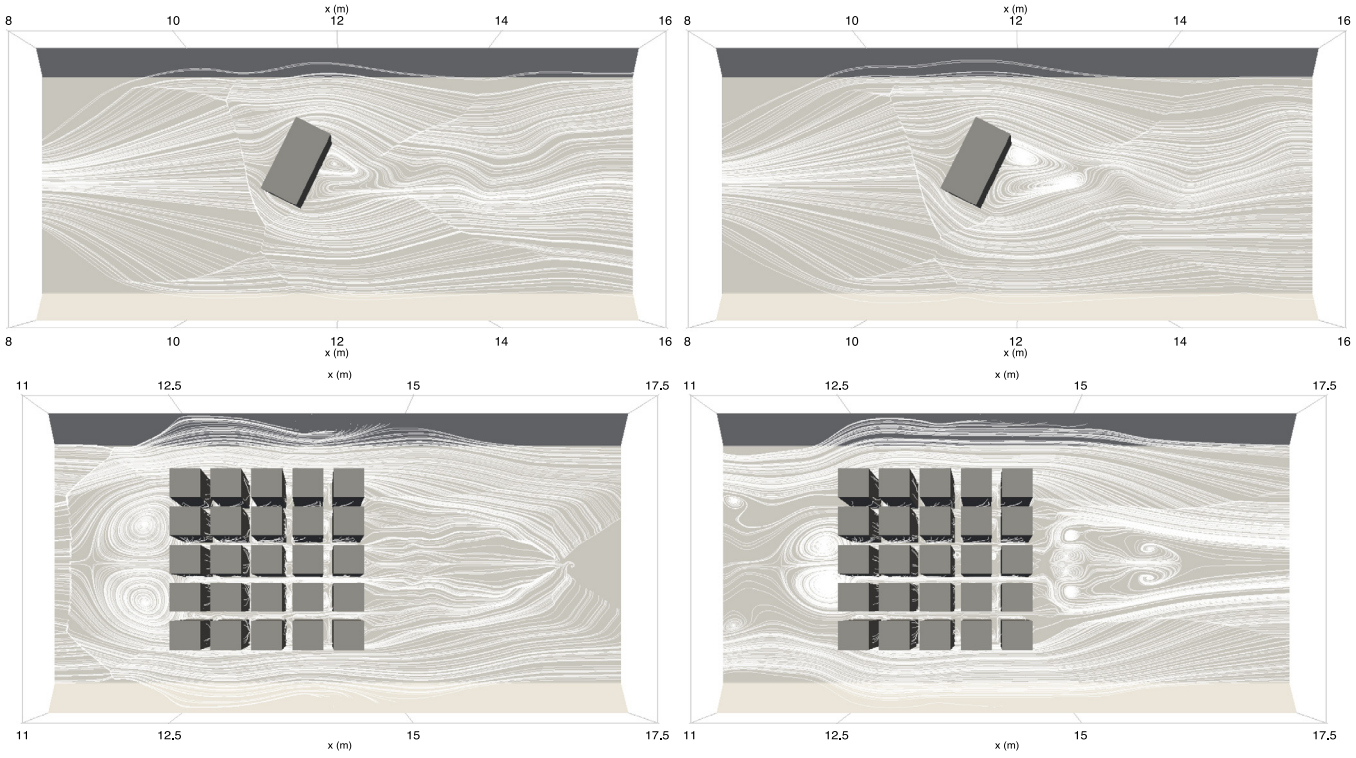


Fig. 31. Case 4: Visualization of streamlines for C-4-a (top) and C-4-b (bottom) at 6 s and 10 s.

Table 1
Summary of CPU time and total number of simulation steps.

Case	Number of cells ($\times 10^6$)	CPU time (s)	Number of simulation steps
C-1-1	0.468	2118	278,812
C-1-2	0.832	6695	495,686
C-1-3	1.872	33,892	1,115,348
C-1-4	3.328	64,269	1,189,598
C-2-1	0.064	1072	1,031,809
C-2-2	0.256	17,152	4,128,236
C-2-3	0.400	41,875	6,449,806
C-2-4	0.713	79,831	6,896,998
C-3-1	0.067	51	46,803
C-3-2	0.266	792	183,830
C-3-3	1.061	12,629	733,245
C-3-4	2.387	38,354	989,777
C-4-a	1.296	980	46,588
C-4-b	1.296	319	15,184

In other words, each CPU time shown in Table 1 gives a similar PM value.

5. Conclusions

Within the framework of our in-house code NUFSAW2D, a 2D RANS and ASM equations based-model has been presented to simulate the turbulent shallow water flows. Our model is an edge-based CCFV model, which has second-order spatial accuracy, achieved by employing the MUSCL method with the edge-based MinMod slope limiter function. We have shown that the CU method, as a Riemann-solver-free scheme, was robust for calculating the convective fluxes of both the 2D RANS and ASM equations. The bed-slope terms were solved using a Riemann-solution-free method, which has been proven to be capable of balancing the fluxes after being combined with the hydrostatic and topography reconstructions. The diffusive fluxes of the 2D RANS equations were solved using the centered method, whereas the diffusive fluxes of the ASM equations were treated in a similar manner, but as a source term. This source term was later treated semi-implicitly together with the friction and the turbulence source terms. No stability issue was found when dealing with the turbulence model, since some limitations have been applied to the turbulence properties, e.g. a realizability condition for ν_t, $\overline{u'^2}$, $\overline{u'v'}$, and $\overline{v'^2}$ and the limiter values of P_h and P_{kb}.

Using high-resolution meshes up to millions of cells, we have performed our simulations with the mesh sizes of up to 2.5 mm. Highly accurate results were shown, for example in case 3, where our model resulted outstandingly in better results than those of the previous publication. Using $4\times$ smaller cell, the wakes amplitudes became significantly more accurate and the maximum deviation for the wakes periods was 4 s smaller. Refining the mesh to be $16\times$ smaller yielded highly accurate results, which was demonstrated by the fact that all the wakes characteristics (amplitudes and periods) were calculated correctly in every phase with this cell size. Further refinement with the $36\times$ smaller cell produced, however, only non-significant differences. Since the increase of the mesh resolution – which could give not only the proper ampli-

tudes, but also the accurate periods – was up to 16 ×, it is agreeable to say that the mesh size has a significant effect on the accuracy of the results, especially for case 3. Sufficiently accurate results were also exhibited in case 4 which dealt with wet–dry problems on the channel banks. Our model simulated this case stably and could capture the recirculation and the wakes properly near the obstacle(s).

The accurate results obtained using such finer meshes are inseparable from a proper wall boundary treatment for the wet–dry problems around the structures or channel banks. These problems related to the calculation of the turbulence properties for moving boundary geometries. This was clearly shown by the results at some points which experienced periodic wet–dry phenomena throughout the simulation, e.g. in cases 3 and 4. To this regard, our model required no grid study especially to know the proper near-wall mesh size. Instead, we simply generated the very fine meshes – and with the ScWF our model was always able to achieve convergence. Our proposed strategy – a combination of the hydrostatic and topography reconstructions calculated in an edge-based manner at edges and the ScWF – was proven quite robust and accurate for calculating the turbulence properties at all cells, for which three types of cells are considered: internal, wall boundary, and flow boundary cells. This strategy was applied for simulating such wet–dry problems; although the change of wall boundary geometries for wet–dry phenomena becomes more sensitive for the use of very fine meshes, the proposed strategy was able to accurately detect such moving boundary geometries, to properly compute the turbulence values and to not deteriorate the results. The reconstruction techniques employed in our strategy served to correct all the values of the convective, diffusive, and bed-slope fluxes based on the values of depth. Thereafter, the ScWF were applied at cells to calculate the turbulence properties, when those cells were calculated as wet cell and (at least) one of the corresponding convective fluxes (normal to the edge) was zero. Since no grid study is needed, our approach gives flexibility to users in generating meshes for their simulations without requiring estimating the wall friction velocity at the initial step (as if the StWF are used). Obviously, the proposed algorithm can be extended for other types of turbulence shallow water modeling, such as dealing with wet–dry problems for real flood simulations on river banks, in which the water fluctuates leading to non-constant wall boundary geometries. This algorithm can also be applied to any CCFV scheme with other robust and well-balanced solvers, such as Roe, HLL, and HLLC schemes.

Acknowledgments

The author gratefully acknowledges the DAAD (*German Academic Exchange Service*) for supporting his research in the scope of the Research Grants – Doctoral Programmes in Germany 2015/16 (57129429) – and the compute and data resources provided by the Leibniz Supercomputing Centre. The author would like to thank Prof. Michael Manhart from the Chair of Hydromechanics (Technical University of Munich) for interesting and fruitful discussions about turbulence modeling. Suggestions from Prof. Ernst Rank and Dr.rer.nat.(habil) Ralf-Peter Mundani from the Chair for Computation in Engineering (Technical University of Munich) are appreciated. The author is also grateful to all the anonymous reviewers who provided many constructive comments and suggestions.

References

[1] Rodi W. Turbulence modeling and simulation in hydraulics: a historical review. J Hydraul Eng 2017;143(5):1–20. doi:10.1061/(ASCE)HY.1943-7900.0001288.
[2] Lloyd PM, Stansby PK. Shallow-water flow around model conical islands of small side slope. i: surface piercing. J Hydraul Eng 1997;123(12):1057–67. doi:10.1061/(ASCE)0733-9429(1997)123:12(1057).
[3] Lloyd PM, Stansby PK. Shallow-water flow around model conical islands of small side slope. ii: submerged. J Hydraul Eng 1997;123(12):1068–77. doi:10.1061/(ASCE)0733-9429(1997)123:12(1068).
[4] Wu W. Depth-averaged two-dimensional numerical modeling of unsteady flow and nonuniform sediment transport in open channels. J Hydraul Eng 2004;130(10):1013–24. doi:10.1061/(ASCE)0733-9429(2004)130:10(1013).
[5] Cea L. An unstructured finite volume model for unsteady turbulent shallow water flow with wet-dry fronts: numerical solver and experimental validation. Ph.D. thesis, Universidade da Corua; 2005.
[6] Cea L, Puertas J, Vazquez-Cendon M-E. Depth averaged modelling of turbulent shallow water flow with wet-dry fronts. Arch Comput Methods Eng 2007;14(3):303–41. doi:10.1007/s11831-007-9009-3.
[7] Yu C, Duan J. Two-dimensional depth-averaged finite volume model for unsteady turbulent flow. J Hydraul Res 2012;50(6):599–611. doi:10.1080/00221686.2012.730556.
[8] Kim D-H, Lynett PJ. Turbulent mixing and passive scalar transport in shallow flows. Phys Fluids 2011;23(1):016603. doi:10.1063/1.3531716.
[9] Kim D-H, Lynett PJ, Socolofsky SA. A depth-integrated model for weakly dispersive, turbulent, and rotational fluid flows. Ocean Modell 2009;27(3):198–214. doi:10.1016/j.ocemod.2009.01.005.
[10] https://coastal.usc.edu/currents_workshop/index.html; Accessed: 5 May 2018.
[11] Zhang YJ, Priest G, Allan J, Stimely L. Benchmarking an unstructured-grid model for tsunami current modeling. Pure Appl Geophys 2016;173(12):4075–87. doi:10.1007/s00024-016-1328-6.
[12] Ding Y, Jia Y, Wang SSY. Identification of manning's roughness coefficients in shallow water flows. J Hydraul Eng 2004;130(6):501–10. doi:10.1061/(ASCE)0733-9429(2004)130:6(501).
[13] Stansby PK. Limitations of depth-averaged modeling for shallow wakes. J Hydraul Eng 2006;132(7):737–40. doi:10.1061/(ASCE)0733-9429(2006)132:7(737).
[14] Rodi W. Turbulence models and their application in hydraulics: a state-of-the art review. A.A. Balkema; 1993.
[15] Boussinesq J. Essai sur la theorie des eaux courantes. Memoires presentes par divers savants a l'Academie des Sciences, Paris; 1887.
[16] Rodi W. A new algebraic relation for calculating the reynolds stresses. Z Angew Math Mech 1976;56:219–21.
[17] Cao Z, Hu P, Hu K, Pender G, Liu Q. Modelling roll waves with shallow water equations and turbulent closure. J Hydraul Res 2015;53(2):161–77. doi:10.1080/00221686.2014.950350.
[18] Kurganov A, Levy D. Central-upwind schemes for the saint-venant system. ESAIM: Math Modell Numer Anal 2002;36(3):397–425. doi:10.1051/m2an:2002019.
[19] Kurganov A, Petrova G. A second-order well-balanced positivity preserving central-upwind scheme for the saint-venant system. Commun Math Sci 2007;5(1):133–60. https://projecteuclid.org/euclid.cms/1175797625.
[20] Wu G, He Z, Liu G. Development of a cell-centered godunov-type finite volume model for shallow water flow based on unstructured mesh. Math Probl Eng 2014;2014:1–15. doi:10.1155/2014/257915.
[21] Horvath Z, Waser J, Perdigao RAP, Konev A, Bloeschl G. A two-dimensional numerical scheme of dry/wet fronts for the saint-venant system of shallow water equations. Int J Numer Methods Fluids 2015;77(3):159–82. doi:10.1002/fld.3983.
[22] Chertock A, Kurganov A, Liu Y. Central-upwind schemes for the system of shallow water equations with horizontal temperature gradients. Numer Math 2014;127(4):595–639. doi:10.1007/s00211-013-0597-6.
[23] Liu X, Mohammadian A, Kurganov A, Sedano JAI. Well-balanced central-upwind scheme for a fully coupled shallow water system modeling flows over erodible bed. J Comput Phys 2015;300(Suppl C):202–18. doi:10.1016/j.jcp.2015.07.043.
[24] Shirkhani H, Mohammadian A, Seidou O, Kurganov A. A well-balanced positivity-preserving central-upwind scheme for shallow water equations on unstructured quadrilateral grids. Comput Fluids 2016;126(Suppl C):25–40. doi:10.1016/j.compfluid.2015.11.017.
[25] Ginting BM, Mundani R-P. Artificial viscosity technique: a Riemann-solver-free method for 2d urban flood modelling on complex topography. Advances in hydroinformatics. Gourbesville P, Cunge J, Caignaert G, editors. Singapore: Springer Water; 2018. ISBN 9789811072178. doi:10.1007/978-981-10-7218-5_4.
[26] Coroneo M, Montante G, Paglianti A, Magelli F. Cfd prediction of fluid flow and mixing in stirred tanks: numerical issues about the rans simulations. Comput Chem Eng 2011;35(10):1959–68. doi:10.1016/j.compchemeng.2010.12.007.
[27] Karimi M, Akdogan G, Bradshaw SM. Effects of different mesh schemes and turbulence models in cfd modelling of stirred tanks. Physicochem Probl Mineral Process 2012;48(2):513–31. doi:10.5277/ppmp120216.
[28] Liang Q, Hou J, Xia X. Contradiction between the c-property and mass conservation in adaptive grid based shallow flow models: cause and solution. Int J Numer Methods Fluids 2015;78(1):17–36. doi:10.1002/fld.4005.
[29] Stansby PK. A mixing-length model for shallow turbulent wakes. J Fluid Mech 2003;495:369â384. doi:10.1017/S0022112003006384.
[30] van Leer B. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. J Comput Phys 1979;32(1):101–36. doi:10.1016/0021-9991(79)90145-1.
[31] Delis AI, Nikolos IK. A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered finite volumes with application to shallow water dynamics. Int J Numer Methods Fluids 2013;71(5):584–633. doi:10.1002/fld.3674.

[32] Tan WY. Shallow water hydrodynamics: mathematical theory and numerical solution for a two-dimensional system of shallow-water equations. Elsevier; 1992.
[33] Begnudelli L, Sanders BF, Bradford SF. Adaptive Godunov-based model for flood simulation. J Hydraul Eng 2008;134(6):714–25. doi:10.1061/(ASCE)0733-9429(2008)134:6(714).
[34] Audusse E, Bouchut F, Bristeau M-O, Klein R, Perthame B. A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. SIAM J Sci Comput 2004;25(6):2050–65. doi:10.1137/S1064827503431090.
[35] Castanedo S, Medina R, Mendez FJ. Models for the turbulent diffusion terms of shallow water equations. J Hydraul Eng 2005;131(3):217–23. doi:10.1061/(ASCE)0733-9429(2005)131:3(217).
[36] Mohammadian A, Le Roux DY. Simulation of shallow flows over variable topographies using unstructured grids. Int J Numer Methods Fluids 2006;52(5):473–98. doi:10.1002/fld.1167.
[37] Davidson L. Implementation of a $\kappa - \epsilon$ model and a Reynold stress model into a multiblock code. Tech. Rep CRS4-APPMATH-93-21. Applied Mathematics and Simulation Group CRS4, Cagliari, Italy; 1993.
[38] Durbin PA. On the k-3 stagnation point anomaly. Int J Heat Fluid Flow 1996;17(1):89–90. doi:10.1016/0142-727X(95)00073-Y.
[39] Liang Q, Borthwick AGL. Adaptive quadtree simulation of shallow flows with wet-dry fronts over complex topography. Comput Fluids 2009;38(2):221–34. doi:10.1016/j.compfluid.2008.02.008.
[40] Hirsch C. Numerical computation of internal and external flows. Elsevier; 2007. ISBN 9780750665940.
[41] Ginting BM. A two-dimensional artificial viscosity technique for modelling discontinuity in shallow water flows. Appl Math Model 2017;45(Suppl C):653–83. doi:10.1016/j.apm.2017.01.013.
[42] Menter F, Esch T. Elements of industrial heat transfers predictions. 16th Brazilian congress of mechanical engineering; 2001.
[43] Duan J. Simulation of flow and mass dispersion in meandering channels. J Hydraul Eng 2004;130(10):964–76. doi:10.1061/(ASCE)0733-9429(2004)130:10(964).
[44] https://www.lrz.de, Accessed: 1 October 2018.
[45] Ginting BM, Mundani R-P. Parallel flood simulations for wet-dry problems using dynamic load balancing concept. J Comput Civ Eng 2018 (in press). doi:10.1061/(ASCE)CP.1943-5487.0000823.
[46] Ginting BM, Mundani R-P, Rank E. Parallel simulations of shallow water solvers for modelling overland flows. In: La Loggia G, Freni G, Puleo V, De Marchis M, editors. 13th international conference on hydroinformatics, 3. EPiC series in engineering; 2018. p. 788–99. doi:10.29007/wdn8.
[47] Chen D, Jirka G. Experimental study of plane turbulent wakes in a shallow water layer. Fluid Dyn Res 1995;16(1):11–41. doi:10.1016/0169-5983(95)00053-G.
[48] Han L. Recirculation zone developing downstream of an expansion in a shallow open-channel flow. Ph.D. thesis, INSA de Lyon; 2015.
[49] Han L, Mignot E, Riviere N. Shallow mixing layer downstream from a sudden expansion. J Hydraul Eng 2017;143(5). doi:10.1061/(ASCE)HY.1943-7900.0001274.
[50] Mohamadian A, Le Roux DY, Tajrishi M, Mazaheri K. A mass conservative scheme for simulating shallow flows over variable topographies using unstructured grid. Adv Water Resour 2005;28(5):523–39. doi:10.1016/j.advwatres.2004.10.006.
[51] Soares-Frazao S, Zech Y. Experimental study of dam-break flow against an isolated obstacle. J Hydraul Res 2007;45:27–36. doi:10.1080/00221686.2007.9521830.
[52] Soares-Frazao S, Zech Y. Dam-break flow through an idealised city. J Hydraul Res 2008;46(5):648–58. doi:10.3826/jhr.2008.3164.
[53] https://sandstorm.cie.bgu.tum.de/wiki/index.php/Main_Page, Accessed: 1 May 2018.

Paper 3

B.M. Ginting and R.-P. Mundani. Comparison of Shallow Water Solvers: Applications for Dam-break and Tsunami Cases with Reordering Strategy for Efficient Vectorization on Modern Hardware. *Water*, 11(4):639, 2019. https://dx.doi.org/10.3390/w11040639

water

MDPI

Article

Comparison of Shallow Water Solvers: Applications for Dam-Break and Tsunami Cases with Reordering Strategy for Efficient Vectorization on Modern Hardware

Bobby Minola Ginting * and Ralf-Peter Mundani

Chair for Computation in Engineering, Technical University of Munich, Arcisstr. 21, D-80333 Munich, Germany; mundani@tum.de

* Correspondence: bobbyminola.ginting@tum.de; Tel.: +49-89-289-23044

Received: 13 February 2019; Accepted: 21 March 2019; Published: 27 March 2019

Abstract: We investigate in this paper the behaviors of the Riemann solvers (Roe and Harten-Lax-van Leer-Contact (HLLC) schemes) and the Riemann-solver-free method (central-upwind scheme) regarding their accuracy and efficiency for solving the 2D shallow water equations. Our model was devised to be spatially second-order accurate with the Monotonic Upwind Scheme for Conservation Laws (MUSCL) reconstruction for a cell-centered finite volume scheme—and be temporally fourth-order accurate using the Runge–Kutta fourth-order method. Four benchmark cases of dam-break and tsunami events dealing with highly-discontinuous flows and wet–dry problems were simulated. To this end, we applied a reordering strategy for the data structures in our code supporting efficient vectorization and memory access alignment for boosting the performance. Two main features are pointed out here. Firstly, the reordering strategy employed has enabled highly-efficient vectorization for the three solvers investigated on three modern hardware (AVX, AVX2, and AVX-512), where speed-ups of 4.5–6.5× were obtained on the AVX/AVX2 machines for eight data per vector while on the AVX-512 machine we achieved a speed-up of up to 16.7× for 16 data per vector, all with singe-core computation; with parallel simulations, speed-ups of up to 75.7–121.8× and 928.9× were obtained on AVX/AVX2 and AVX-512 machines, respectively. Secondly, we observed that the central-upwind scheme was able to outperform the HLLC and Roe schemes 1.4× and 1.25×, respectively, by exhibiting similar accuracies. This study would be useful for modelers who are interested in developing shallow water codes.

Keywords: central-upwind; efficiency; finite volume; HLLC; modern hardware; Roe; shallow water equations; vectorization

1. Introduction

Dam-break or tsunami flows cause not only potential dangers to human life, but also great losses of property. These phenomena can be triggered by some natural hazards, such as earthquakes or heavy rainfall. When a dam breaks, a large amount of water is released instantaneously from the dam and will propagate rapidly to the downstream area. Similarly, tsunami waves flowing rapidly from the ocean bring a large volume of water to coastal areas, which endangers human life as well as damages infrastructure. Since natural hazards have very complex characteristics, in terms of the spatial and temporal scales, they are quite difficult to predict precisely. Therefore, it is highly important to study the evolution of such flows as a part of a disaster management, which will be useful for the related stakeholders in decision-making. Such study can be done by developing a mathematical model based on the 2D shallow water equations (SWEs).

Recent numerical models of the 2D SWEs rely, almost entirely, on the computations of (approximate) Riemann solvers, particularly in the applications of the high-resolution Godunov-type methods. The simplicity, robustness, and built-in conservation properties of the Riemann solvers, such as the Roe and HLLC schemes, had led to many successful applications in shallow flow simulations, see [1–5], among others. Highly discontinuous flows, including transcritical flows, shock waves and moving wet–dry fronts were accurately simulated.

Generally speaking, a scheme can be regarded as a class of Riemann solvers if it is proposed based on a Riemann problem. The Roe scheme was originally devised by [6] and was proposed to estimate the interface convective fluxes between two adjacent cells on a spatially-and-temporally discretized computational domain by linearizing the Jacobian matrix of the partial differential equations (PDEs) with regard to its left and right eigenvectors. This linearized part contributes to the computation of the convective fluxes of the PDEs, as a flux difference for the average value of the considered edge taken from its two corresponding cells. Since the eigenstructure of the PDEs—which leads to an approximation of the interface value in connection with the local Riemann problem—must be known in the calculation of the flux difference, the Roe scheme is regarded as an approximate Riemann solver.

More than 20 years later, Toro [7] then developed a new approximate Riemann solver—HLLC scheme—to simulate shallow water flows, which was an extended version of the Harten-Lax-van Leer (HLL) scheme proposed in [8]. In the HLL scheme, the solution is approximated directly for the interface fluxes by dividing the region into three parts: left, middle, and right. Both the left and right regions correspond to the values of the two adjacent cells, whereas the middle region consists of a single value separated by intermediate waves. One major flaw of the HLL scheme is related to both contact discontinuities and shear waves leading to a missing contact (middle) wave. Therefore, Toro [7] fixed this scheme in the HLLC scheme by including the computation of the middle wave speed that now the solution is divided into four regions. There are several ways to calculate the middle wave speed, see [9–11]. All the calculations deal with the eigenstructure of the PDEs, which is related to the local Riemann problem, and obviously, this brings the HLLC scheme back to a class of Riemann solvers.

Opposite to the Riemann solvers, Kurganov et al. [12] proposed the central-upwind (CU) method as a Riemann-solver-free scheme, in which the eigenstructure of the PDEs is not required to calculate the convective fluxes. Instead, the local one-sided speeds of propagation at every edge, which can be computed in a straight-forward manner, are used. This scheme has been proven to be sufficiently robust and at the same time can satisfy both the well-balanced and positivity preserving properties, see [13–15].

To solve the time-dependent SWEs, all the aforementioned schemes must be temporally discretized either by using an implicit or an explicit time stepping method. Despite its simplicity, the latter may, however, suffer from a stability computational issue particularly when simulating a very low water on a very rough bed [16,17]. The former is unconditionally stable and even is very flexible to use a large time step. However, the computation is admittedly complex. Another way that can be used to overcome the stability issue of the explicit method and to avoid the complexity of the implicit method—is to perform a high-order explicit method, such as the Runge–Kutta high-order scheme. This method is more stable than the explicit method, while the computation remains simple and acceptably cheap as that of the explicit method.

As the high-order time stepping method is now considered, the selection of solvers included in models must be taken into careful consideration, since such solvers—which are the most expensive part in SWEs simulations—need to be computed several times in a single time step. For example, the Runge–Kutta fourth-order (RKFO) method requires the updating of a solver four times to determine the value at the subsequent time step. The more complex the algorithm of a solver is, the more CPU time one obtains.

Nowadays, performing SWE simulations is becoming more and more common on modern hardware/CPUs towards high-performance computing (HPC) using advanced features such as

AVX, AVX2, and AVX-512, which support the algorithm vectorization for executing some operations in a single instruction—known as single instruction multiple data (SIMD)—so that a significant computation speed-up can be achieved. Vectorization on such modern hardware employs vector instructions, which can dramatically outperform scalar instructions, thus being quite important for having more efficient computations. Among the other compilers' optimizations, vectorization can even be regarded as the common ways for utilizing vector-level parallelism, see [18,19]. Such a speed-up, however, can only exist if the algorithm formulation is suitable for vectorization instructions either automatically (by compilers) or manually (by users) [20].

Typically, there are three classes of vectorization: auto vectorization, guided vectorization, and low-level vectorization. The first type is the easiest one utilizing the ability of the compiler to automatically detect loops, which have a potential to be vectorized. This can be done at compiling time, e.g., using the optimization flag `-O2` or higher. However, some typical problems, e.g., non-contiguous memory access and data-dependency, make vectorization difficult. For this, the second type may be a solution utilizing some compiler hints/pragmas and array notations. This type may successfully vectorize the loops that cannot be auto-vectorized by the compiler. However, if not used carefully, it gives no significant performance or even the results can be wrong. The last type is probably the hardest one since it requires deep-knowledge about intrinsics/assembly programming and vector classes, thus not so popular.

Especially in simulating complex phenomena such as dam-break or tsunami flows as part of disaster planning, accurate results are obviously of particular interest for modelers. However, focusing only on numerical accuracy but ignoring performance efficiency is no longer an option. For instance, in addition to relatively large-sized domains, most of real dam-break and tsunami simulations require performing long real-time computations, e.g., days or even up to weeks. Wasting the performance either due to the complexity level of the solver selected or the code's inability to utilize the vectorization, is thus undesirable. This becomes our focus in this paper. We compare three common shallow water solvers (HLLC, Roe, and CU schemes) here, where two main findings are pointed out. Firstly, to enable highly-efficient vectorization for all solvers on all the aforementioned hardware, we employ a reordering strategy that we have recently applied in [21]. This strategy supports guided vectorization and memory access alignment for the array loops attempted in the SWEs' computations, thus boosting the performance. Secondly, we observe that the CU scheme is capable of outperforming the performance of the HLLC and Roe schemes by exhibiting similar accuracies. These findings would be useful for modelers as a reference to select the numerical solvers to be included in their models as well as to optimize their codes for vectorization.

Some previous studies reporting about vectorization of shallow water solvers are noted here. In [20], the augmented Riemann solver implemented in a source code Geo Conservation Laws (GeoCLAW) was vectorized using a low-level vectorization with SSE4 and AVX intrinsics. The average speed-up factors of 2.7× and 4.1× (both with single-precision arithmetic) were achieved with SSE4 and AVX machines, respectively. Also using GeoCLAW, the augmented Riemann solver was vectorized in [22] by changing the data layouts from arrays of structs (AoS) to structs of arrays (SoA), thus requiring a considerably huge task for rewriting the code—and then applying a guided vectorization with `!$omp simd`. The average speed-up factors of 1.7× and 4.4× (both with double-precision arithmetic) were achieved with AVX2 and AVX-512 machines, respectively. In [23], the split HLL Riemann solver was vectorized and parallelized for the flux computation and state computation modules of the SWEs employing low-level vectorization with SSE4 and AVX intrinsics. To the best of our knowledge, this is the first attempt to report the efficiency comparisons of common solvers (both Riemann and non-Riemann solvers) regarding the vectorization on the three modern hardwares without having to perform complex intrinsic functions or to require a lot of work for rewriting the code. We use here an in-house code of the first-author—numerical simulation of free surface shallow water 2D (NUFSAW2D). Some successful applications were shown using NUFSAW2D for varying shallow water-type simulations, e.g., dam-break cases, overland flows, and turbulent flows, see [17,21,24,25].

This paper is organized as follows. The governing equations and the numerical model are briefly explained in Section 2. An overview of data structures in our code is presented in Section 3. The model verifications against the benchmark cases and its performance evaluations are given in Section 4. Finally, conclusions are given in Section 5.

2. Governing Equations and Numerical Models

The 2D SWEs are written in conservative form according to [26] as

$$\frac{\partial \mathbf{W}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{S}_b + \mathbf{S}_f \,, \tag{1}$$

where the vectors $\mathbf{W}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{S}_b$, and $\mathbf{S}_f$ are expressed as

$$\begin{gathered} \mathbf{W} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} \,, \quad \mathbf{F} = \begin{bmatrix} hu \\ huu + \dfrac{gh^2}{2} \\ hvu \end{bmatrix} \,, \quad \mathbf{G} = \begin{bmatrix} hv \\ huv \\ hvv + \dfrac{gh^2}{2} \end{bmatrix} \,, \\ \mathbf{S}_b = \begin{bmatrix} 0 \\ -gh\dfrac{\partial z_b}{\partial x} \\ -gh\dfrac{\partial z_b}{\partial y} \end{bmatrix} \,, \quad \mathbf{S}_f = \begin{bmatrix} 0 \\ -g\,h\dfrac{n_m^2\,u\,\sqrt{u^2+v^2}}{h^{4/3}} \\ -g\,h\dfrac{n_m^2\,v\,\sqrt{u^2+v^2}}{h^{4/3}} \end{bmatrix} \,. \end{gathered} \tag{2}$$

The water depth, velocities in x and y directions, gravity acceleration, bottom elevation, and Manning coefficient are denoted by h, u, v, g, z_b, and n_m, respectively. Using a cell-centered finite volume (CCFV) method, Equation (1) is spatially discretized over a domain Ω as

$$\frac{\partial}{\partial t}\iint_\Omega \mathbf{W} d\Omega + \iint_\Omega \left(\frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y}\right) d\Omega = \iint_\Omega \left(\mathbf{S}_b + \mathbf{S}_f\right) d\Omega \,. \tag{3}$$

Applying the Gauss divergence theorem, the convective fluxes of Equation (3) can be transformed into a line-boundary integral Γ as

$$\frac{\partial}{\partial t}\iint_\Omega \mathbf{W} d\Omega + \oint_\Gamma \left(\mathbf{F}\,n_x + \mathbf{G}\,n_y\right)\, d\Gamma = \iint_\Omega \left(\mathbf{S}_b + \mathbf{S}_f\right) d\Omega \,, \tag{4}$$

leading to a flux summation for the convective fluxes by

$$\oint_\Gamma \left(\mathbf{F}n_x + \mathbf{G}n_y\right)\, d\Gamma \approx \sum_{i=1}^{N} \left(\mathbf{F}\,n_x + \mathbf{G}\,n_y\right)_i \,\Delta L_i \,, \tag{5}$$

where n_x and n_y are the normal vectors outward Γ, N is the total number of edges for a cell, and ΔL is the edge length. We will investigate the accuracy and efficiency of the three solvers for solving Equation (5). The in-house code NUFSAW2D used here implements the modern shock-capturing Godunov-type model, which supports the structured as well as unstructured meshes by storing the average values in each cell-center. Here we use structured rectangular meshes, hence $N = 4$. The second-order spatial accuracy was achieved with the MUSCL method utilizing the MinMod limiter function to enforce the monotonicity in multiple dimensions. The bed-slope terms were computed using a Riemann-solution-free technique, with which the bed-slope fluxes can be computed separately from the convective fluxes, thus giving a fair comparison for the three aforementioned

solvers. The friction terms were treated semi-implicitly to ensure stability for wet–dry simulations. The RKFO method is now applied to Equation (4) as

$$\begin{aligned} \mathbf{W}^{p=0} &= \mathbf{W}^{t}\,, \qquad \texttt{for}\;\; p = 1, \ldots, 4 \;\; \texttt{then} \\ \mathbf{W}^{p} &= \mathbf{W}^{p=0} + \alpha_p \left[-\frac{\Delta t}{A} \sum_{i=1}^{4} \left(\mathbf{F}\, n_x + \mathbf{G}\, n_y \right)_i \Delta L_i + \Delta t \iint_{\Omega} \left(\mathbf{S}_b + \mathbf{S}_f \right) d\Omega \right]^{p-1}, \\ \mathbf{W}^{t+1} &= \mathbf{W}^{p=4}\,, \end{aligned} \tag{6}$$

where A is the cell area, Δt is the time step, α_p is the coefficient being $1/4$, $1/3$, $1/2$, and 1 for p = 1–4, respectively. The numerical procedures for Equations (4) and (6) are given in detail in [17,25,26], thus are not presented here.

3. Overview of Data Structures

3.1. General

Here we explain in detail how the data structures of our code are designed to advance the solutions of Equation (6). Note this is a typical data structure used in many shallow water codes (with implementations of modern finite volume schemes). As shown in Figure 1, a domain is discretized into several sub-domains (rectangular cells). We call this step the pre-processing stage. Each cell now consists of the values of z_b and n_m located at its center. Initially, the values of h, u, and v are given by users at each cell-center.

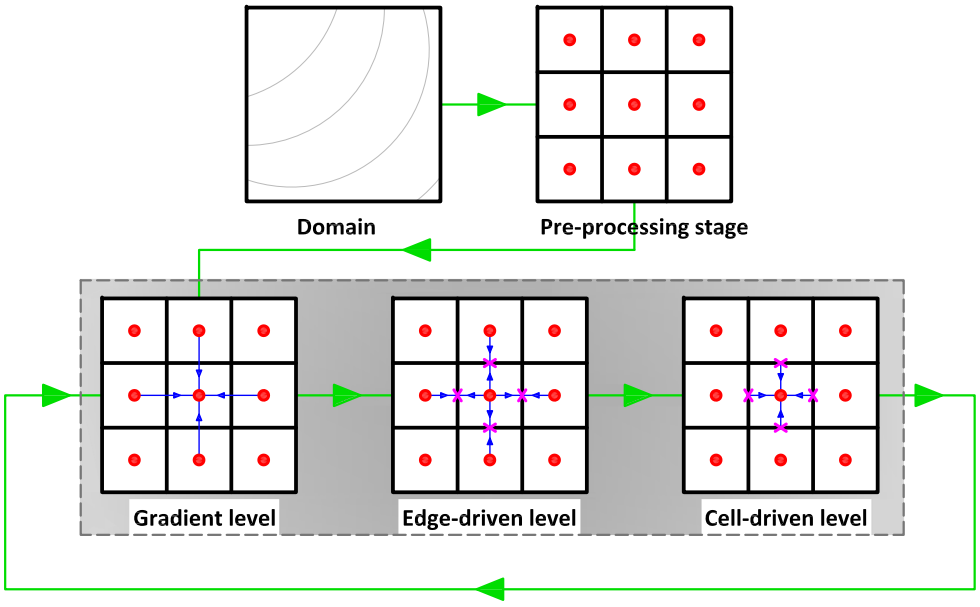


Figure 1. Typical process in shallow flow modeling (with implementations of modern finite volume schemes).

As our model employs a reconstruction process to spatially achieve second-order accuracy with the MUSCL method, it requires the gradient values at cell-center. Therefore, these gradient values must firstly be computed. This step is called the gradient level. Hereafter, one requires to calculate the values at each edge using the values of its two corresponding cell-centers. This stage is then called the edge-driven level. In this level, a solver, e.g., HLLC, Roe, or CU scheme, is required to compute the non-linear values

of **F** and **G** at edges. Prior to performing such a solver, the aforementioned reconstruction process with the MUSCL method was employed. Note the values of $\mathbf{S}_b$ are also computed at the edge-driven level. After the values of all edges are known, the solution can be advanced for the subsequent time level by also computing the values of $\mathbf{S}_f$. For example, the solutions of **W** at the subsequent time level for a cell-center are updated using the **F**, **G**, and $\mathbf{S}_b$ values from its four corresponding edges—and using $\mathbf{S}_f$ values located at the cell-center itself. We call this stage the cell-driven level.

Note that the edge-driven level is the most expensive stage among the others; one should thus pay extra attention to its computation. We also point out here that we apply the computation for the edge-driven level in an edge-based manner rather than in a cell-based one, namely we compute the edge values only once per single calculation level. Therefore, one does not need to save the values of $\left[\sum_{i=1}^{N}\left(\mathbf{F}\,n_x+\mathbf{G}\,n_y\right)_i\Delta L_i\right]$ in arrays for each cell-center; only the values of $\left[\left(\mathbf{F}\,n_x+\mathbf{G}\,n_y\right)_i\Delta L_i\right]$ are saved corresponding to the total number of edges, instead. The values of an edge are only valid for one adjacent cell—and such values are simply multiplied by (-1) for another cell. It is now a challenging task to design an array structure that can ease vectorization and exploit memory access alignment in both the edge-driven and cell-driven levels.

3.2. Cell-Edge Reordering Strategy for Supporting Vectorization and Memory Access Alignment

We focus our reordering strategy here on tackling the two common problems for vectorization: non-contiguous memory access and data-dependency. Regarding the former, a contiguous array structure is required to provide contiguous memory access giving an efficient vectorization. Typically, one finds this problem when dealing with an indirect array indexing, e.g., using `x(y(i))` forces the compiler to decode `y(i)` for finding the memory reference of `x`. This is also a typical problem for a non-unit strided access to array, e.g., incrementing a loop by a scalar factor, where non-consecutive memory locations must be accessed in the loop. The vectorization is sometimes still possible for this problem type. However, the performance gain is often not significant. The second problem relates to usage of arrays identical to the previous iteration of the loop, which often destroy any possibility for vectorization, otherwise a special directive should be used.

See Figure 2, for advancing the solution of **W** in Equation (1) for `k`, one requires **F**, **G**, and $\mathbf{S}_b$ from `i`, where `i` = `index_function(j)` and [`j` ← `1-4`]—and $\mathbf{S}_f$ from `k` itself. Opting `index_function` as an operator for defining `i` leads to a use of an indirect reference in a loop. This is not desired since it may avoid the vectorization. This may be anticipated by directly declaring `i` into the same array to that of `k`, e.g., `W(k)` ← [`W(k+m)`,`W(k-m)`,`W(k+n)`,`W(k-n)`], where `m` and `n` are scalar. This, however, leads to a data-dependency problem that makes vectorization difficult.

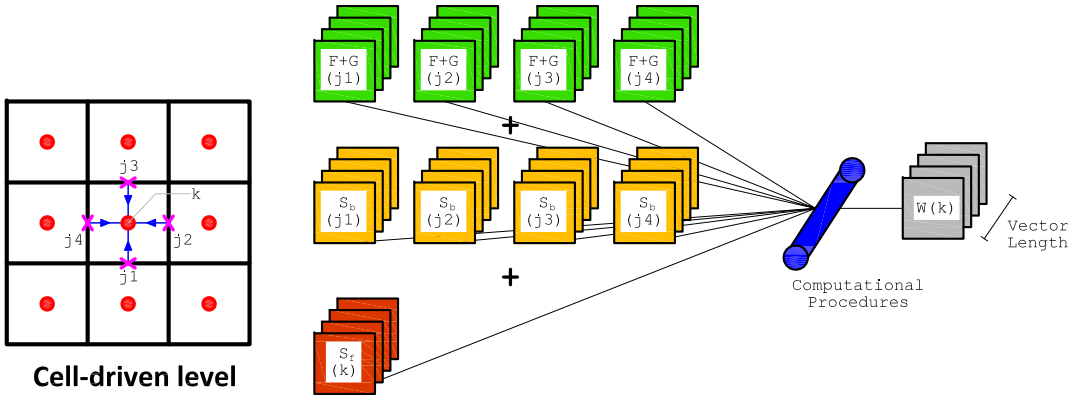


Figure 2. Vectorization for advancing the solution in the cell-driven level.

To avoid these problems, we have designed a cell-edge reordering strategy, see Figure 3, where the loops with similar computational procedures are collected to be vectorized. Note that this strategy

is only applied once at the pre-processing stage in Figure 1. The core idea of this strategy is to build contiguous array patterns between edges and cells for the edge-driven level as well as between cells and edges for the cell-driven level. We point out here that we only employ 1D array configuration in NUFSAW2D, so that the memory access patterns are straightforward, thus easing unit stride and conserving cache entries. The first step is to devise the cell numbering following the Z-pattern, which is intended for the cell-driven level. Secondly, we design the edge numbering for the edge-driven level by classifying the edges into two types: internal and boundary edges in the most contiguous way; the former is the edges that have two neighboring cells (e.g., edges 1–31), whereas the latter is the edges with only one corresponding cell (e.g., edges 32–49). The reason for this classification is the computational complexity between the internal and boundary edges differs from each other, e.g., (1) no reconstruction process is required for the latter, thus having less CPU time than the former—and (2) due to corresponding to two neighboring cells, the former accesses more memories than does the latter; declaring all edges only in one single loop-group therefore deteriorates the memory access patterns, thus decreasing the performance.

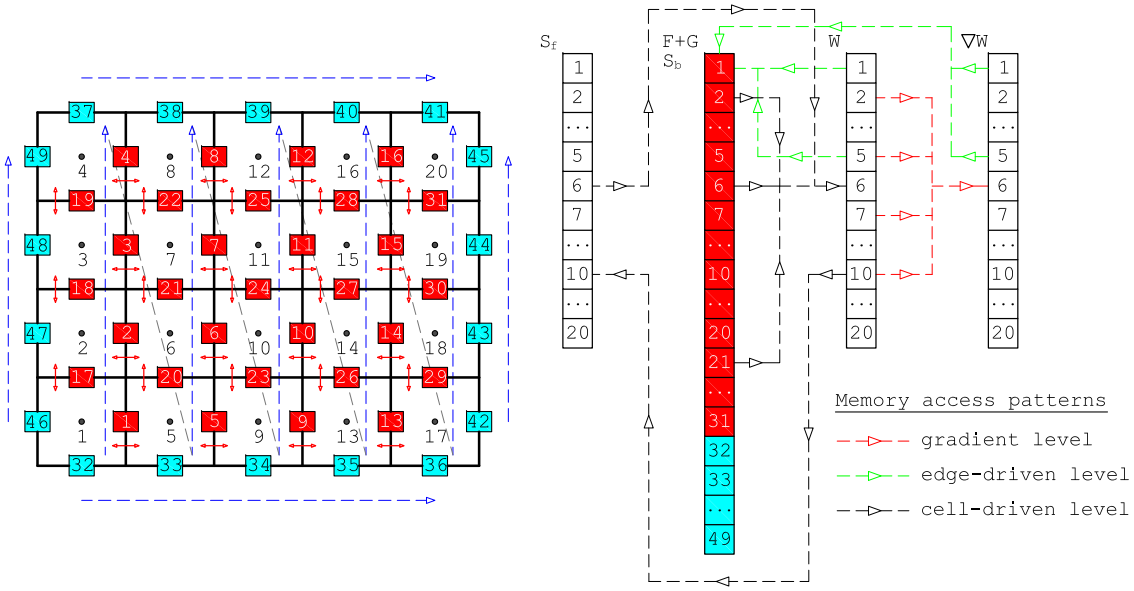


Figure 3. Cell-edge reordering strategy [21] and an example of memory access patterns.

For the sake of clarity, we write in Algorithm 1 the pseudo-code of the model's `SUBROUTINE` employed in NUFSAW2D. Note that Algorithm 1 is a typical form applied in many common and popular shallow water codes. First, we mention that `seg_x = 5`, `seg_y = 4`, and `Ncells = 20` according to Figure 3, where `seg_x`, `seg_y`, and `Ncells` are the total number of domain segments in x and y directions, and the total number of cells, respectively. We now explain the `SUBROUTINE gradient`. The cells are now classified into two groups: internal and boundary cells. Internal cells, e.g., cells 6, 7, 10, 11, 14, and 15 are cells whose gradient computations require accessing two cell values in each direction. For example, computing the x-gradient of **W** of cell 6 needs the values of **W** of cells 2 and 10; this is denoted by $[\nabla \mathtt{W_x(6)} \leftarrow \mathtt{W(2),W(10)}]$ and similarly $[\nabla \mathtt{W_y(6)} \leftarrow \mathtt{W(5),W(7)}]$. Boundary cells, e.g., cells 1–4, 5, 8, 9, 12, 13, 16, and 17–20, are cells affiliated with boundary edges. These cells may not always require accessing two cell values in each direction for the gradient computation, e.g., $[\nabla \mathtt{W_x(8)} \leftarrow \mathtt{W(4),W(12)}]$ but $[\nabla \mathtt{W_y(8)} \leftarrow \mathtt{W(7),W(8)}]$ showing that a symmetric boundary condition is applied to cell 8 in y direction. Considering the fact that the total number of internal cells is significantly larger than that of boundary cells, we group the internal cells into a single loop and distinguish them from the boundary cells, see Algorithm 2.

Algorithm 1 Typical algorithm for shallow water code (within the Runge–Kutta fourth-order (RKFO) method's framework)

```
1:  for t = 1 ← [total number of time step] do
2:    !  within the RKFO method from [p=1] to [p=4]
3:    for p = 1 ← 4 do
4:      CALL gradient
5:        → compute gradient
6:      CALL edge-driven_level
7:        → compute MUSCL_method
8:        → compute bed_slope
9:        → compute shallow_water_solver
10:     CALL cell-driven_level
11:       → compute friction_term
12:       → compute update_variables
13:   end for
14: end for
```

Algorithm 2 Pseudo-code for `SUBROUTINE gradient`

```
1:  for k=1 ← [seg_x-2] do
2:    l = (seg_y+2)+(k-1)*seg_y
3:    !$omp simd simdlen(VL) aligned(∇W_x,∇W_y :Nbyte)
4:    for i=l ← [l+seg_y-3] do
5:      .........
6:      ∇W_x(i) ← W(i-seg_y),W(i+seg_y)  ;  ∇W_y(i) ← W(i-1),W(i+1)
7:    end for
8:  end for
9:  !$omp simd simdlen(VL) aligned(∇W_x,∇W_y :Nbyte)
10: for i=1 ← [seg_y] do
11:   j=Ncells-seg_y+i
12:   i1=i-1 OR i1=i     ;     i2=i+1 OR i2=i
13:   i3=j-1 OR i3=j     ;     i4=j+1 OR i4=j
14:   .........
15:   ∇W_x(i) ← W(i),W(i+seg_y)  ;  ∇W_y(i) ← W(i1),W(i2)
16:   ∇W_x(j) ← W(j-seg_y),W(j)  ;  ∇W_y(j) ← W(i3),W(i4)
17: end for
18: !=== This loop is not vectorized due to non-unit strided access ===!
19: for i=1 ← [seg_x-2] do
20:   j=i*seg_y+1         ;     k=(i+1)*seg_y
21:   i1=j-1 OR i1=j     ;     i2=j+1 OR i2=j
22:   i3=k-1 OR i3=k     ;     i4=k+1 OR i4=k
23:   .........
24:   ∇W_x(j) ← W(j-seg_y),W(j+seg_y)  ;  ∇W_y(j) ← W(i1),W(i2)
25:   ∇W_x(k) ← W(k-seg_y),W(k+seg_y)  ;  ∇W_y(k) ← W(i3),W(i4)
26: end for
```

Algorithm 2 shows three typical loops in the `SUBROUTINE gradient`. The first loop (lines 1–8) is designed sequentially with a factor of `seg_x-2` for its outer part to exclude all boundary cells. For its inner part, this loop is constructed based on the outer loop in a contiguous way, thus making vectorization efficient. Each element of array ∇W_x accesses two elements from array `W` with the farthest alignment of `seg_y`, while each element of array ∇W_y also accesses two elements of array `W` but only with the farthest alignment of `1`. The second loop (lines 10–17) is also designed similarly to the first one, but since this loop includes boundary cells, each element of arrays ∇W_x and ∇W_y only accesses one array with the farthest alignment of `seg_y` and `1`, respectively—whereas the other elements from array `W` required are contiguously accessed by each element of both ∇W_x and ∇W_y. Note in our implementation, none of these two loops can be auto-vectorized by the compiler. Therefore, we apply a guided vectorization with OpenMP directive instead of the Intel one, namely `!$omp simd simdlen(VL) aligned(var1,var2,... :Nbyte)`; this will be explained later in Section 4.5. The third loop (lines 19–26) is designed for the rest cells, which are not included in the previous two loops. This loop is not devised in a contiguous manner, thus disabling auto vectorization or, although a guided vectorization is possible, it still does not give any significant performance

improvement due to non-unit strided access. Despite being unable to be vectorized, the third loop does not significantly decrease the performance of our model for the entire simulation as it only has an array dimension of `2*[seg_x-2]` (quite small compared to the other two loops).

We now discuss the `SUBROUTINE edge-driven_level` and sketch it in Algorithm 3. Note for the sake of brevity, only the pseudo-code for internal edges is represented in Algorithm 3; for boundary edges, the pseudo-code is similar but computed without `MUSCL_method`. The first loop corresponds to the edges 1–16 and the second one to the edges 17–31. In the first loop (lines 1–7), each flux computation accesses the array with the farthest alignment of `seg_y`, whereas the arrays are designed in the second loop (lines 8–17) to have contiguous patterns. Every edge has a certain pattern for its two corresponding cells, where no data-dependency exists, thus enabling an efficient vectorization. Note with this pattern, both loops can be auto-vectorized; however, we still implement a guided vectorization as it gives a better performance.

Finally, we sketch the `SUBROUTINE cell-driven_level` in Algorithm 4. Again, for the sake of brevity only the pseudo-code for internal cells is given. Similar to the internal cell in the `SUBROUTINE gradient`, the loop is designed sequentially with a factor of `seg_x-2` for the outer part. In the inner part the arrays access patterns are, however, different to those of the gradient computation, where `W` accesses `F`, `G`, and S_b from the corresponding edges—and S_f from the corresponding cell; in other words, more array accesses are required in this loop. Nevertheless, the vectorization gives a significant performance improvement since the array accesses patterns are contiguous. However, there is a part that cannot be vectorized in this cell-driven level due to non-unit strided access, similar to that shown in Algorithm 2. Again, since the dimension of this non-vectorizable loop is considerably smaller than the others, there is no significant performance alleviation for the entire simulation.

Algorithm 3 Pseudo-code for `SUBROUTINE edge-driven_level` (only for internal edges)

```
1:  !$omp simd simdlen(VL) aligned(∇W_x,W,z_b,F,G,S_b :Nbyte)
2:  for i=1 ← [seg_y*(seg_x-1)] do
3:    j=i    ;    k=i+seg_y
4:    .........
5:    compute MUSCL_method + bed_slope + shallow_water_solver
                [∇W_x(j),∇W_x(k),W(j),W(k),z_b(j),z_b(k),...,F_x^L,F_x^R,G_x^L,G_x^R,S_bx^L,S_bx^R]
6:    F+G(i) ← F_x^L,F_x^R,G_x^L,G_x^R    ;    S_b(i) ← S_bx^L,S_bx^R
7:  end for
8:  for l=1 ← [seg_x] do
9:    m=seg_y*(seg_x-1)+1+(l-1)*(seg_y-1)    ;    n=m+seg_y-2    ;    o=(l-1)*seg_y
10:   !$omp simd simdlen(VL) aligned(∇W_y,W,z_b,F,G,S_b :Nbyte)
11:   for i=m ← n do
12:     j=(i-m+1)+o    ;    k=j+1
13:     .........
14:     compute MUSCL_method + bed_slope + shallow_water_solver
                [∇W_y(j),∇W_y(k),W(j),W(k),z_b(j),z_b(k),...,F_y^L,F_y^R,G_y^L,G_y^R,S_by^L,S_by^R]
15:     F+G(i) ← F_y^L,F_y^R,G_y^L,G_y^R    ;    S_b(i) ← S_by^L,S_by^R
16:   end for
17: end for
```

Algorithm 4 Pseudo-code for `SUBROUTINE cell-driven level` (only for internal cells)

```
1:  for k=1 ← [seg_x-2] do
2:    j = (seg_y+2)+(k-1)*seg_y    ;    l = (seg_y*(seg_x-1)+seg_y)+(k-1)*(seg_y-1)
3:    !$omp simd simdlen(VL) aligned(W,F,G,n_m,S_b,S_f :Nbyte)
4:    for i=j ← [j+seg_y-3] do
5:      i1 = l+(i-j)    ;    i2 = i    ;    i3 = i1+1    ;    i4=i-seg_y
6:      .........
7:      compute friction_term [W(i),n_m(i),...,S_f(i)]
8:      compute update_variables
9:      W(i) ← F+G(i1),F+G(i2),F+G(i3),F+G(i4),S_b(i1),S_b(i2),S_b(i3),S_b(i4),S_f(i)
10:   end for
11: end for
```

3.3. Avoiding Skipping Iteration for Vectorization of Wet–Dry Problems

In reality, almost all shallow flow simulations deal with wet–dry problems. To this end, the computations of both solver and bed-slope terms in the `SUBROUTINE edge-driven level` must satisfy the well-balanced and positivity-preserving properties as well, see [27,28], among others. Similarly, the calculations of the friction terms in the `SUBROUTINE cell-driven level` must also consider the wet–dry phenomena, otherwise errors are obtained. For example, in the edge-driven level, a wet–dry or dry–dry interface of an edge may exist since one or two cell-centers consist of no water; for both cases, the MUSCL method for achieving second-order accuracy is sometimes not required or even if this method is still computed, it must be turned back to first-order accuracy to ensure computational stability by simply defining the edge values according to the corresponding centers. Another example is in the cell-driven level, where the transformation of the unit discharges (hu and hv) back to the velocities (u and v) are required for computing the friction terms by a division of a water depth (h); very low water depth may thus cause significant errors. To anticipate these problems, one often employs some skipping iterations in the loops, see Algorithm 5.

Algorithm 5 Pseudo-code of some possible skipping iterations

```
!== This is a typical skipping iteration in the SUBROUTINE edge-driven level ==!
if [wet-dry or dry-dry interfaces at edges] then
  NO MUSCL_method: calculate first-order scheme
else
  compute MUSCL_method:  calculate second-order scheme
  if [velocities are not monotone] then
    back to first-order scheme
  end if
  .........
end if
!== This is a typical skipping iteration in the SUBROUTINE cell-driven level ==!
if [depths at cell-centers > depth limiter] then
  compute friction_term
else
  unit discharges and velocities are set to very small values
  .........
end if
```

Typically, the two skipping iterations in Algorithm 5 are important to ensure the correctness of shallow water models. Unfortunately, such layouts may destroy auto vectorization—or although a guided vectorization is possible, it does not give any significant improvement or may even decrease the performance significantly. This is because the SIMD instructions simultaneously work only for sets of arrays, which have contiguous positions. In our experiences, a guided vectorization was indeed possible for both iterations; the speed-up factors, however, were not so significant. Borrowing the idea of [22], we therefore change the layouts in Algorithm 5 to those in Algorithm 6, where the early exit condition is moved to the end of the algorithm. Using the new layouts in Algorithm 6, we significantly observed up to 48% more improvements of the vectorization from those given in Algorithm 5. Note that the results given by Algorithms 5 and 6 should be similar because no computational procedure is changed but only the layouts.

Algorithm 6 Pseudo-code of the solutions of the skipping iterations in Algorithm 5

```
!== A solution for the skipping iteration in the SUBROUTINE edge-driven level ==!
compute MUSCL_method:  calculate second-order scheme
.........
if [velocities are not monotone] then
  back to first-order scheme
end if
.........
if [wet-dry or dry-dry interfaces at edges] then
  NO MUSCL_method: calculate first-order scheme
end if
!== A solution for the skipping iteration in the SUBROUTINE cell-driven level ==!
compute friction_term
.........
if [depths at cell-centers ≤ depth limiter] then
  unit discharges and velocities are set to very small values
  .........
end if
```

3.4. Parallel Computation

We explain briefly here the parallel computing implementation of NUFSAW2D according to [21]. Our idea is to decompose and parallelize the domain based on its complexity level. NUFSAW2D employs hybrid MPI-OpenMP parallelization, thus is applicable to parallel simulations with multi-nodes. However, as we focus here on the vectorization, which no longer influences the scalability beyond one node [20], we limit our study on single-node implementations and thus only employ OpenMP for parallelization. Further, we examine the memory bandwidth effect when using only one core or 16 cores (AVX), 28 cores (AVX2), and 64 cores (AVX-512).

In Figure 4 we show an example of the decomposition of the domain in Figure 3 using four threads; for the sake of brevity, the illustration is given only for the edge-driven level. The parallel directive, e.g., `!$omp do`, can easily be added to each loop, thus according to Algorithm 2, in the gradient level the domain is decomposed as: thread 0 (cells 6, 7, 1, 17, 5, 8), thread 1 (cells 10, 11, 2, 18, 9, 12), thread 2 (cells 14, 15, 3, 19, 13, 16), and thread 3 (cells 4, 20). Similarly, regarding Algorithm 3 it gives in the edge-driven level: thread 0 (edges 1–4, 17–22, 32–33, 37–38, 42, 46), thread 1 (edges 5–8, 23–25, 34, 39, 43, 47), thread 2 (edges 9–12, 26–28, 35, 40, 44, 48), and thread 3 (edges 13–16, 29–31, 36, 41, 45, 49). Meanwhile, the cell-driven level applies a similar decomposition to that of the gradient level. One can see, the largest loop components, e.g., internal edges 1–4, 5–8, etc., are decomposed in a contiguous pattern easing the vectorization implementation, thus efficient. Note the decomposition in Figure 4 is based on static load balancing that causes load imbalance due to the non-uniform amount of loads assigned to each thread; this load imbalance will become less and less significant as the domain size increases, e.g., to millions of cells. However, another load imbalance issue—which can only be recognized during runtime—appears, namely the one caused by wet–dry problems, where wet cells are computationally more expensive than dry cells. For this, we have developed in [21] a novel weighted-dynamic load balancing (WDLB) technique that was proven effective to tackle load imbalance due to wet–dry problems. All the parallel and load balancing implementations are described in detail in [21], thus are not explained here. We also note that we have successfully applied this cell-edge reordering strategy in [24,25] for parallelizing the 2D shallow flow simulations using the CU scheme with good scalability. Yet, we will show in the next section that the cell-edge reordering strategy proposed can help in easing all the vectorization implementations.

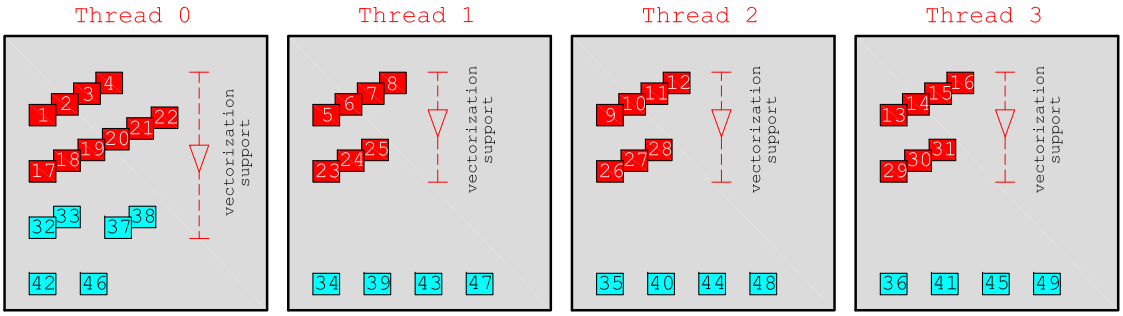


Figure 4. Illustration of load distribution using static load balancing with vectorization support for the edge-driven level based on Figure 3.

4. Results and Discussions

We validate our model against four benchmark tests: two dam-break cases and two tsunami cases. Each case was simulated using a constant Δt that satisfies the Courant-Friedrichs-Lewy (CFL) condition, where CFL ≤ 0.5. Our model with the HLLC, Roe, or CU scheme satisfies the well-balanced property; also, the HLLC and CU solvers employed are positivity-preserving. We note that the Roe scheme may in some cases produce negative depths, see [29]; however, in all implementations tested here, we did not find any negative depth with the Roe scheme. The Δt used also fulfills the CFL limitation required by the computations of the local one-sided propagation speeds of the CU scheme for positivity-preserving purpose, see [13].

4.1. Case 1: Circular Dam-Break

This case is included to check the capability of our model for symmetry and shock resolution in shallow water flow modeling. We refer to [16,30], among others. A 40 $\times$ 40 m, flat, and frictionless domain is considered. A cylindrical wall with a radius of 2.5 m, which was centered at the domain, separated two regions of still water; the first one inside the cylinder had a depth of 2.5 m and the second one outside consisted of 0.5 m water. The water was assumed to be initially at rest and all boundaries were set to wall boundary. The main features to be investigated in this case are the rarefaction wave and the hydraulic jump (shock wave) including a transition condition from subcritical to supercritical flow. The total simulation time was set to 4.7 s with $\Delta t = 0.005$ s, thus requiring 940 time steps. The domain was discretized into 160,000 rectangular cells (319,200 edges).

The evolutions of the simulated free surface elevation using the CU scheme are visualized in Figure 5. Suddenly after 0.1 s, water started to move in all directions. At 0.4 s, the circular shock wave propagated outwards, whereas the circular rarefaction wave traveled inwards showing that this wave almost reaches the center of the domain. This phenomenon continued until the rarefaction wave has fully plunged into the center of the domain at approximately 0.8 s and this wave was suddenly reflected creating a sharp gradient of water surface elevation. At 1.6 s, the circular shock wave propagated further outwards the from domain center, whereas the reflected rarefaction wave now caused the water to fall below the initial depth of 0.5 m. This produced a secondary circular shock wave, the depth of which was slightly less than 0.5 m. The primary circular shock wave kept propagating outwards the center of the domain at 3.8 s and interestingly, the secondary circular shock wave that had recently been created traveled towards that center. At 4.7 s, it is shown that the primary circular wave almost reached the domain boundary and at this time a very sharp gradient of water surface elevation had been created near that boundary.

We present the comparison between the analytical and numerical results at 4.7 s in Figure 6 showing that all schemes can simulate this highly discontinuous flow properly. To point out the difference between the three schemes more clearly, we present in Figure 7 both the depth and velocity

profiles near the two discontinuous areas: 20–22 m and 38–40 m, where only non-significant differences are shown.

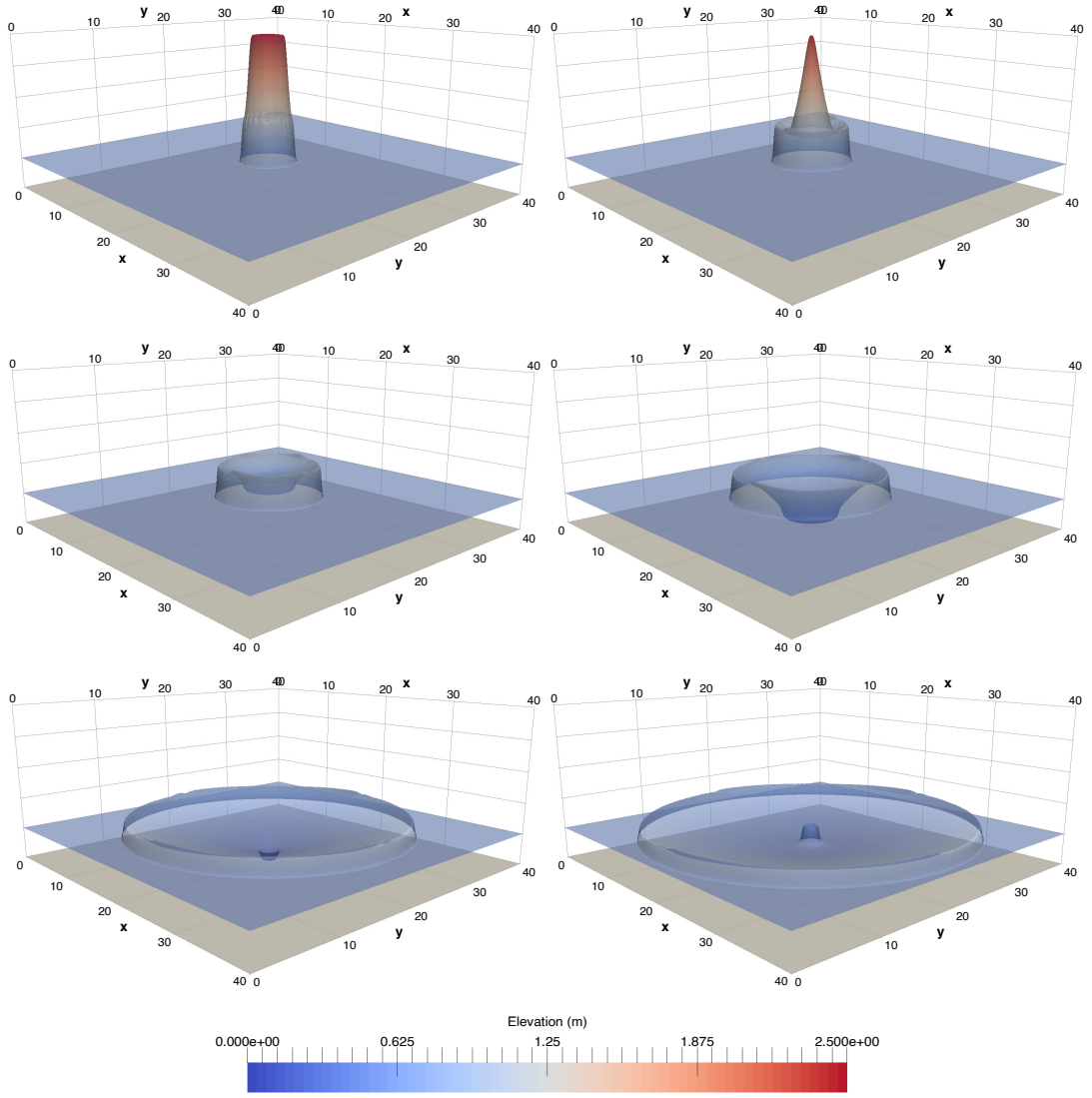


Figure 5. Case 1: results of the central-upwind (CU) scheme at 0.1, 0.4, 0.8, 1.6, 3.8, and 4.7 s.

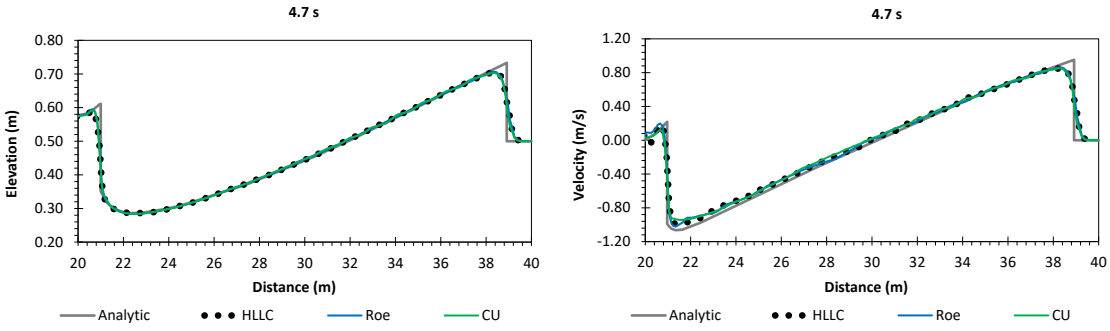


Figure 6. Case 1: comparison between analytical and numerical results at 4.7 s.

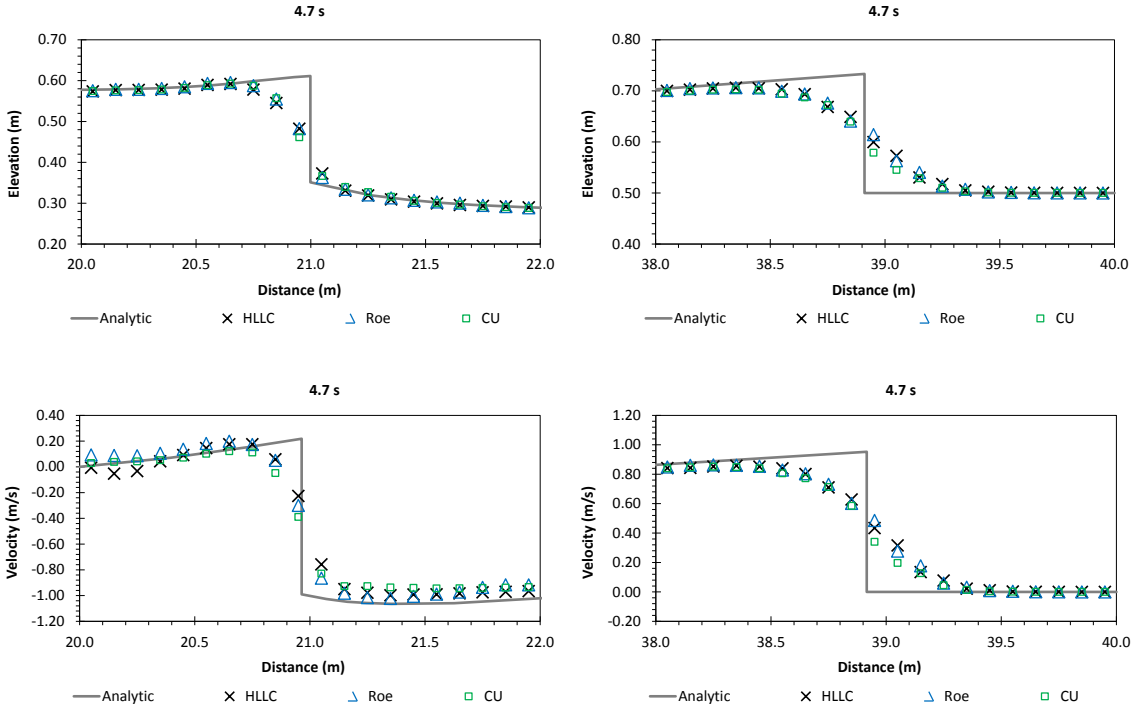


Figure 7. Case 1: comparison between analytical and numerical results at 4.7 s (in detail).

4.2. Case 2: Dam-Break Flow against an Isolated Obstacle

This case was done experimentally in [31]. The channel was trapezoidal; 35.8 m long and 3.6 m wide. A 1 m wide rectangular gate separated the upstream reservoir from the downstream channel, see Figure 8. The Manning coefficient was 0.01 s $m^{-1/3}$. A 0.8 × 0.4 m obstacle was located on the downstream channel with a position that formed an angle of 64^o from the x-axis. The water was set initially to 0.4 m at the reservoir and 0.02 m at the channel, thus the banks at downstream were dry. The upstream end of the reservoir was a closed wall. In this paper, the domain was discretized into 143,280 rectangular cells (285,246 edges). The simulation was set for 30 s with Δt = 0.005 s, thus requiring 6000 time steps.

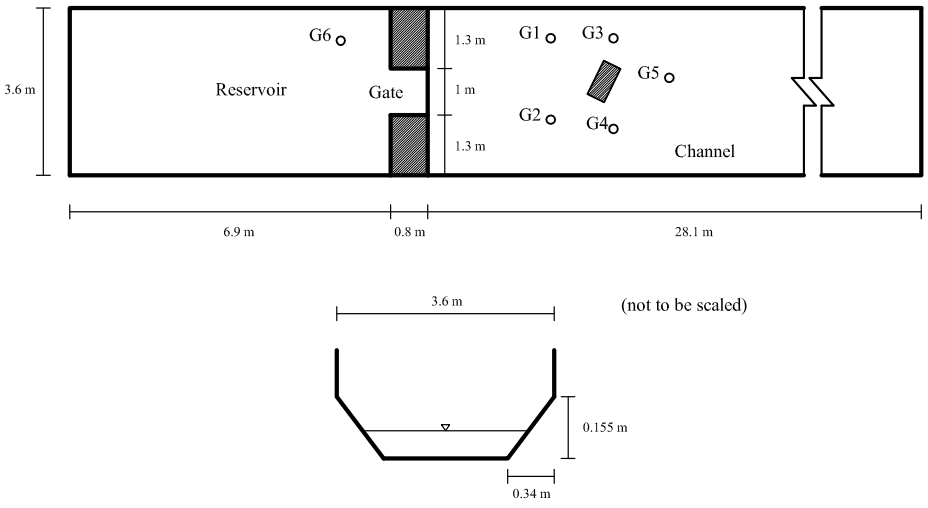


Figure 8. Case 2: sketch of domain and channel shape.

We compared our model at four points: G1 (10.35, 2.95) m, G4 (11.7, 1.0) m, G5 (12.9, 2.1) m, and G6 (5.83, 2.9) m. Our numerical results are given in Figure 9 showing that our model is in general capable of simulating this case properly. At G1, the maximum bore around 2 s was accurately simulated by all schemes, where there were no significant differences shown until 9 s. However, after 9 s, the CU scheme computed the results higher than do the other schemes, where both the HLLC and Roe schemes show almost no different results. At G4, the first bore around 2 s was predicted with a later time of no more than 1 s and a higher depth of no more than 2 cm, where all schemes kept producing the higher values from 2 s to 4.5 s. At G5, no significant differences were again shown between the HLLC and Roe schemes, but the CU scheme showed slightly different values. At G6, highly accurate results were given by all schemes to simulate the water at the reservoir, showing that the schemes can predict the correct incoming discharge from the upstream reservoir to the downstream channel.

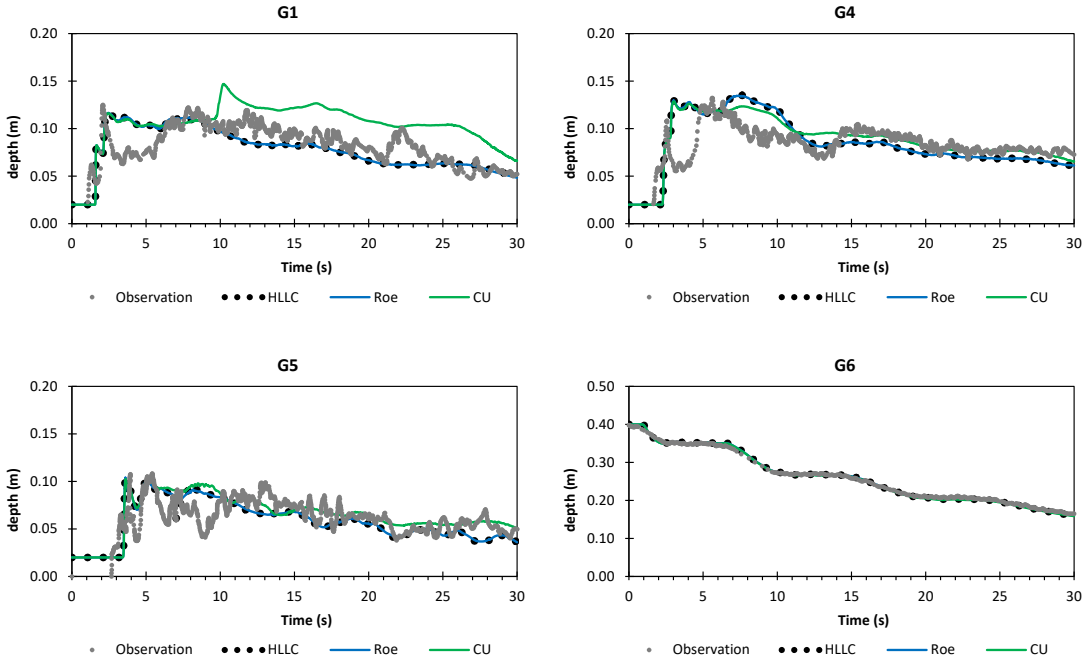


Figure 9. Case 2: comparison of depths between observation and numerical results.

Some errors computed by our model are probably due to the absence of the turbulence terms. Yu and Duan [32] showed the turbulence model was highly important for simulating flow field around the obstacle, where the reflection waves from the obstacle and side walls have superimposed several oblique hydraulic jumps. In Figure 10, we visualize the flood propagation at 1, 3, and 10 s using the CU scheme.

4.3. Case 3: Tsunami Run-Up on a Conical Island

This benchmark case was conducted in a laboratory by [33] to investigate the tsunami run-up on a conical island, the center of which was located near the middle of a 30 × 25 m basin, see Figure 11. To produce planar solitary waves with the specified crest and length, a directional wave maker was used. The left boundary was set as a flow boundary, and the respective water elevation and velocities were defined as

$$\begin{aligned} \eta(0,y,t) &= A_e \,\mathrm{sech}^2\sqrt{\frac{3\,A_e}{4\,H_e}}\,\sqrt{g\,(H_e+A_e)}\,(t-T_e)\,, \\ u(0,y,t) &= \frac{\eta\,\sqrt{g\,(H_e+A_e)}}{\eta+H_e}\,, \qquad v(0,y,t)=0\,, \end{aligned} \tag{7}$$

where A_e, H_e, and T_e are the amplitude of the incident wave, still water depth, and time, at which the wave crest enters the domain—set to 0.032 m, 0.32 m, and 2.45 s, respectively. The other three boundaries were closed boundaries. We compared our results with the values at five gauges located on the domain: P-03, P-06, P-09, P-16, and P-22, whose coordinates were (6.82,13.05) m, (9.36, 13.80) m, (10.36, 13.80) m, (12.96, 11.22) m, and (15.56, 13.80) m, respectively. The Manning coefficient was set to zero as suggested by [34].

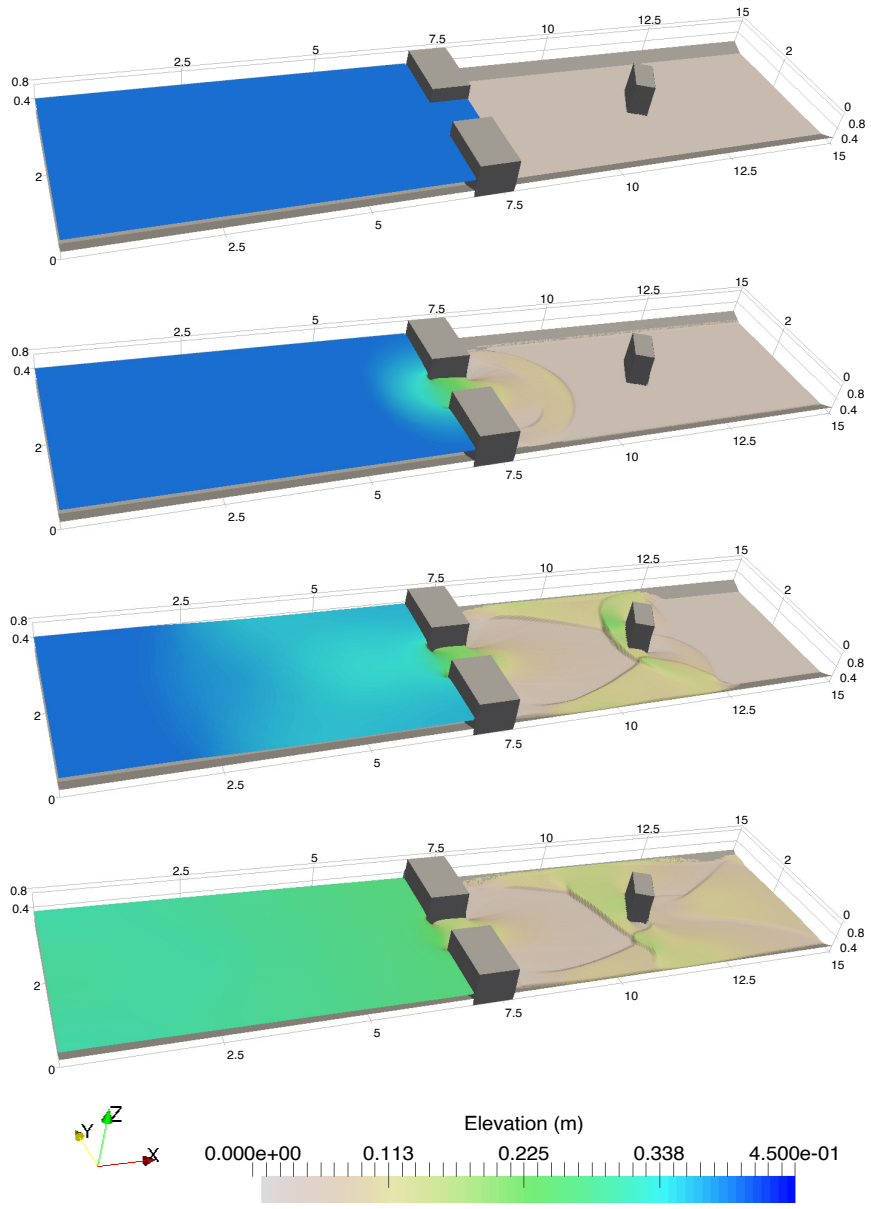


Figure 10. Case 2: visualization of the flood propagation at 0, 1, 3, and 10 s using the CU scheme.

The domain was discretized into 200,704 rectangular cells (402,304 edges). The simulation time was set to 20 s with Δt = 0.002 s leading to 10,000 time steps. One can see in Figure 12, the incident solitary waves in front of the island, which generate a high run-up at about t = 9 s, create wet–dry mechanisms on the conical island. Within this period, the maximum magnitude was reached. After t = 9 s, the waves started to run down the inundated area on the conical island. Some waves were refracted and propagated toward the lee side of the island, where two waves were trapped at each side of the island at around t = 11 s. At t = 13 s, the second wave run-up was generated after these two waves collided. Afterwards, these waves continued to propagate around the island.

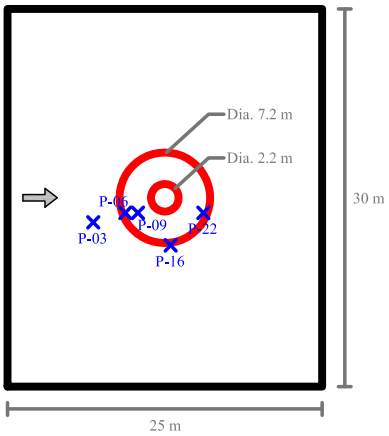


Figure 11. Case 3: computational domain of solitary wave run-up.

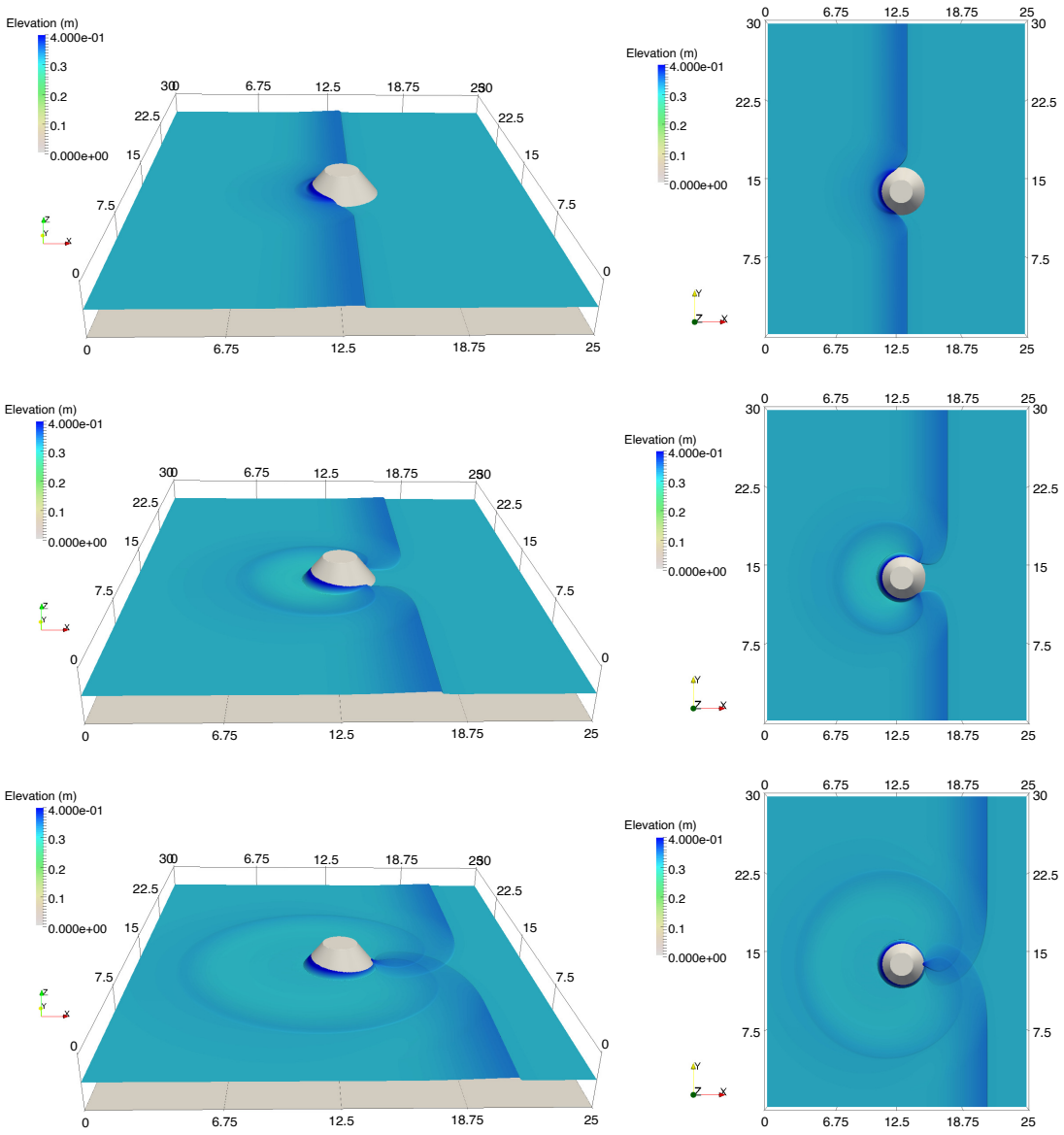


Figure 12. Case 3: numerical results using the CU scheme at 9, 11, and 13 s.

Our numerical results are also compared with laboratory results during 20 s, see Figure 13. Accurate results were produced by all schemes, where no significant differences between them were shown. The arrival times of the highest waves were accurately detected at gauges P-03 and P-22. All schemes rendered later times at gauges P-06, P-09, and P-16 but the differences were no more than 1 s. At gauge P-16, our model computed the wave 1 cm higher than the one mentioned in the laboratory data, and the wave at gauge P-22 was computed 1.3 cm higher. This was probably due to the neglect of the dispersion effects. Note that such discrepancies were also reported in the numerical model of [34].

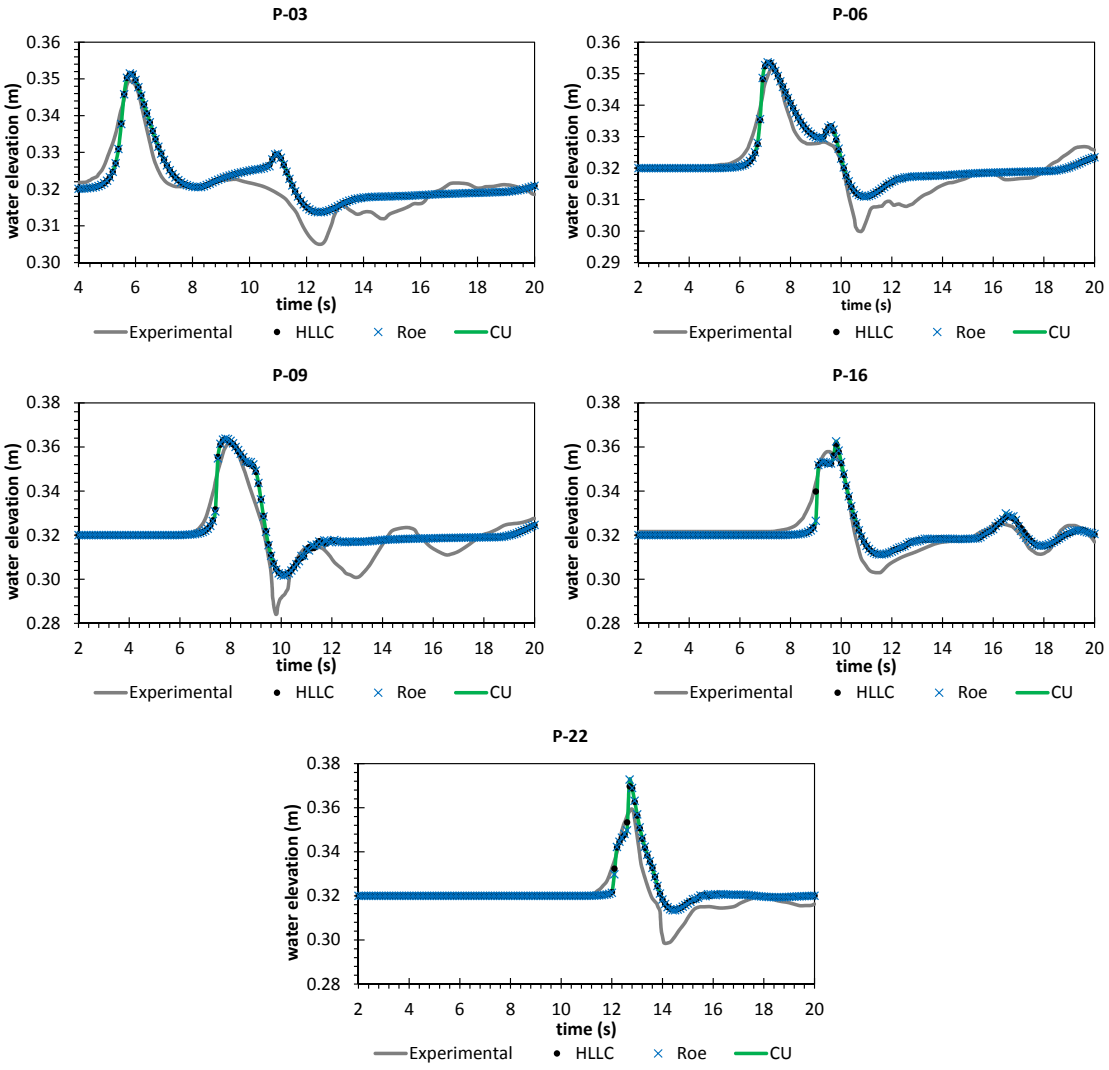


Figure 13. Case 3: comparison between observation and numerical results.

4.4. Case 4: 2011 Japan Tsunami Recorded in Hawaii

This benchmark test is a real tsunami case that occurred in 2011, Japan. The data set was recorded in Hilo Harbor, Hawaii. The raw data can be found in [35]. To avoid the phase differences of the incident wave, the original bathymetry data should be flattened at the depth of 30 m. Interested readers are also referred to [36] for more information. In Figure 14, the sketch of the domain is given as well as the incident wave forcing employed at the northern part as a boundary condition. The Manning coefficient was assumed to be uniform 0.025 s $m^{-1/3}$. The observation points were the Hilo tide station

for elevation (3159, 3472) m, HAI1125/harbor entrance (4686, 2246) m, and HAI1126/inside harbor (1906, 3875) m for velocities. The 1-minute de-tiding of raw data was done for the observed data.

The domain was discretized using 20 m resolution rectangular cells producing 94,600 rectangular cells (189,200 edges). We set the simulation time to 13 h and used Δt = 0.025 s giving 1,872,000 time steps. The results are given in Figure 15 plotted per 150 s. At the Hilo tide Station, each scheme can detect the first incoming wave quite accurately around t = 8.2 h. The lowest water elevation was also predicted properly at approximately t = 8.4 h but with a non-significant difference of about 0.2 m. After that, the water level fluctuations were also computed properly. At the harbor entrance, the velocities were in general accurately computed. Each scheme was able to compute the first incoming wave for the x velocity at t = 8.2 h. The y velocity magnitude at that time was, however, slightly overestimated. Inside the harbor, accurate predictions for x and y velocities were shown, where the first incoming wave was well predicted. After 10 h, each scheme kept exhibiting accurate results at the harbor entrance as well as inside the harbor. One can see that the water current flowed predominantly in North–South direction at the harbor entrance, whereas inside the harbor the water current flowed predominantly in East–West direction. Our results agree with the observed data and those simulated by [36] as well. Although some discrepancies—which are probably due to the neglect of the tidal current effects—still exist, our model shows overall quite accurate results for this hazard event.

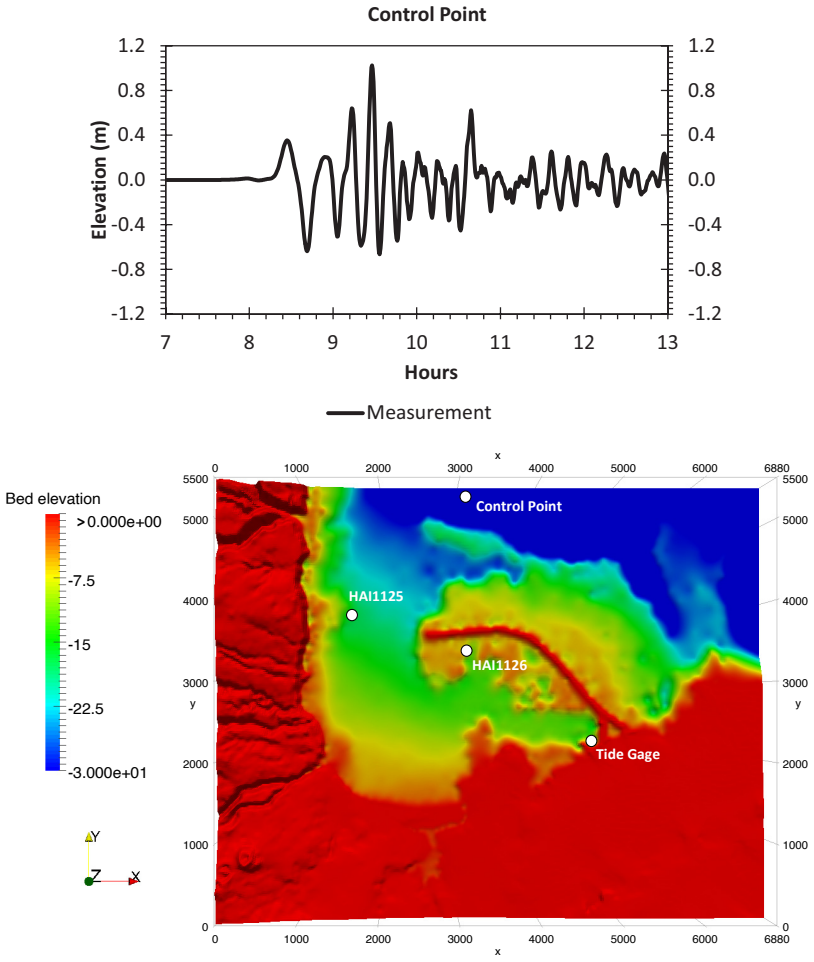


Figure 14. Case 4: bathymetry for simulation and the boundary condition.

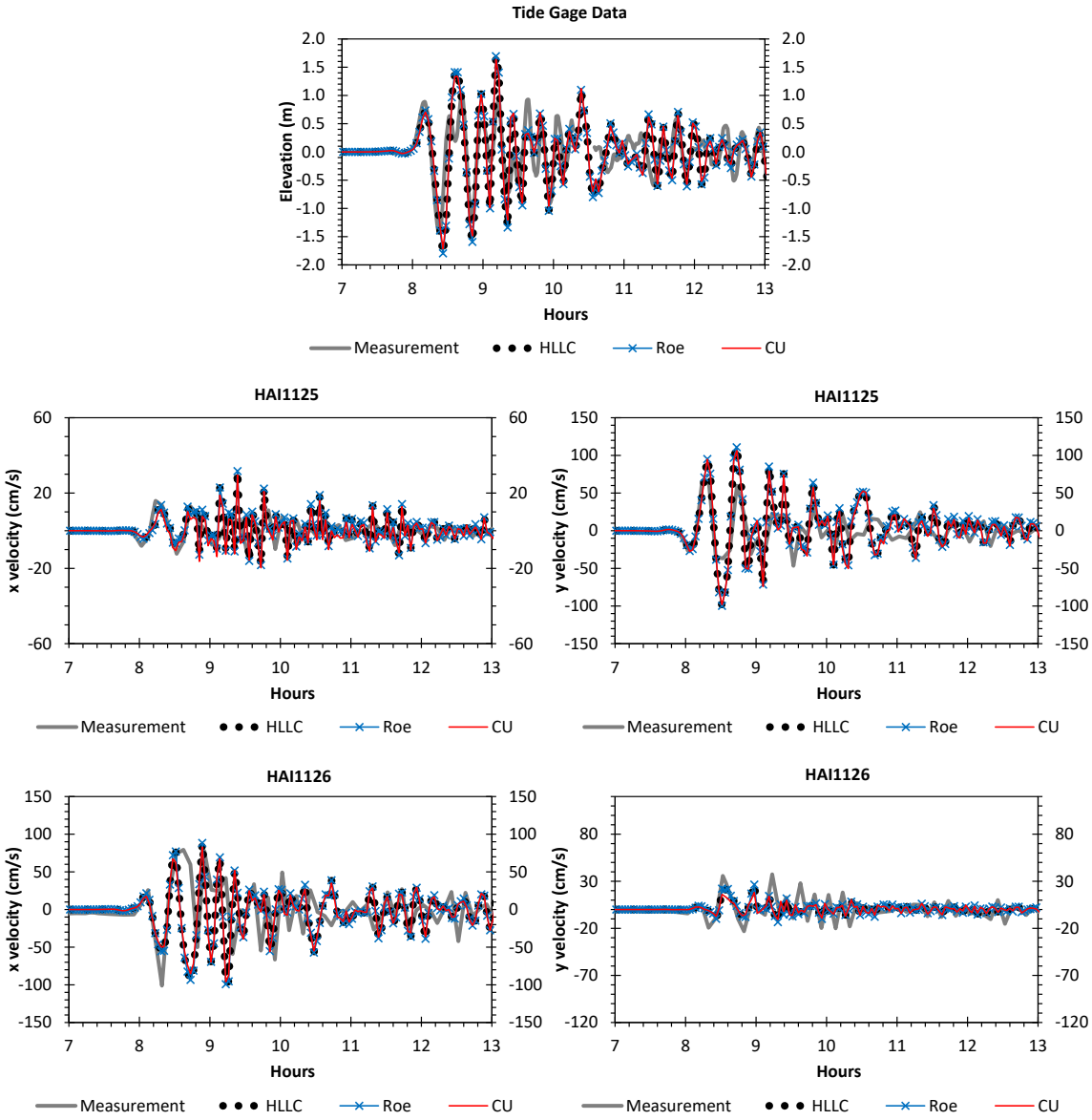


Figure 15. Case 4: comparison between observation and numerical results.

In Figure 16, the visualizations of tsunami inundation are presented using the CU scheme. It is shown at around 9.03 h, the water level reaches approximately 0.5–1 m at the harbor entrance. Meanwhile, the water level is predicted to reach 1–1.5 m inside the harbor. At about 9.53 h, the water level at the harbor entrance remains relatively constant for 0.5–1 m but outside the harbor (near the breakwater) the water level becomes higher up to 2.5 m. After 14 h, the water level near the breakwater (inside and outside the harbor) decreases to approximately −1.25 m. Complex wet–dry phenomena near the coastline as well as the breakwater appear during the simulation time and our model has shown to be robust for modeling such phenomena.

We show in Figure 17 a visualization of the maximum velocity magnitude captured by the HLLC, Roe, and CU schemes during 13 h simulation time. In general, as one can see, no significant differences are shown between all schemes. Along the outer side of the breakwater as well as near the harbor entrance, the velocity magnitudes of more than 4.5 m/s appear. Meanwhile, considerably lower magnitudes are shown inside the harbor. The main difference is only located near the harbor entrance,

where the CU scheme computes the slightly lower magnitudes. The spatial distribution of the velocity magnitude is shown to be extremely sensitive, in agreement with that studied in [36].

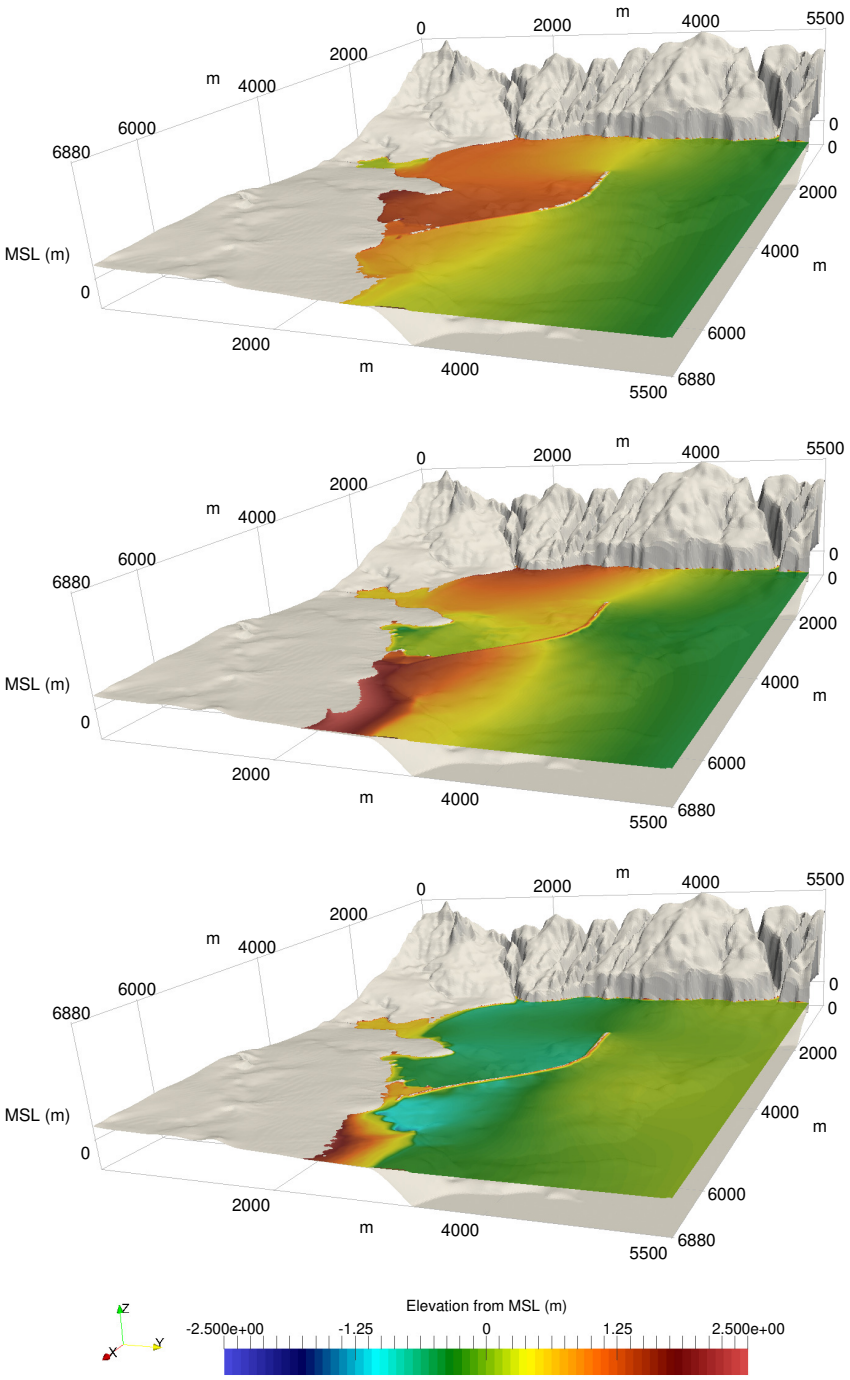


Figure 16. Case 4: visualization of tsunami inundation using the CU scheme at 9.03, 9.53, and 11 h.

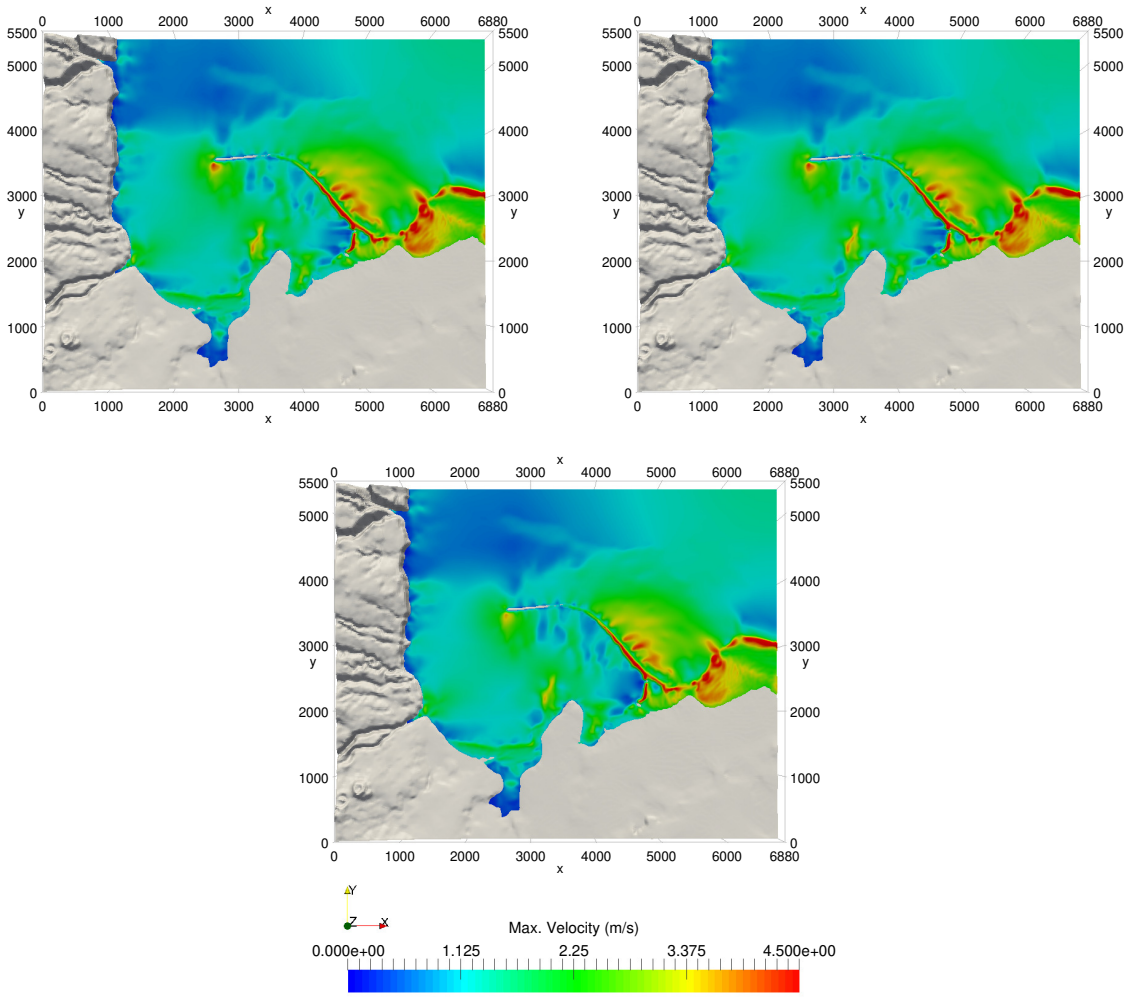


Figure 17. Case 4: numerical result for the maximum velocity captured during the 13 h simulation time using the Harten-Lax-van Leer-Contact (HLLC), Roe, and CU schemes (top left, top right, bottom).

4.5. Performance Comparison

We have shown in the previous sections that the HLLC, Roe, and CU schemes are quite accurate for simulating the test cases, where only non-significant differences are shown between them. In this section we analyze and compare the performance of each scheme. All schemes were written and compiled in the same code NUFSAW2D on three machines—AVX (Intel Xeon E5-2690/ Sandy-Bridge-E), AVX2 (Intel Xeon E5-2697 v3/Haswell), and AVX-512 (Intel Xeon Phi/Knights Landing)—for a Linux operating system using Intel Fortran 19. The first computing resource "Sandstorm" was available at our chair [37] and the last two resources "CoolMUC-2" and "CoolMUC-3" were provided by the Leibniz Supercomputing Centre (LRZ) [38]. Each node of the AVX, AVX2, and AVX-512 machines has a total of eight physical cores (16 logical cores), 14 physical cores (28 logical cores), and 64 physical cores, respectively. Note that AVX-512 is built on many-core architecture that incorporates cores with low-frequency and small memory. Therefore, in order to achieve a notable performance, this machine relies on the vector operations on 512 bit SIMD registers.

We did not use the vectorization directive provided by Intel, e.g., `!dir$ simd`, since we have experienced that this directive was not always able to vectorize the loop. Instead, we implemented the directive `!$omp simd simdlen(VL) aligned(var1,var2,... :Nbyte)` provided by the OpenMP 4.0. The first component (`simdlen`) was aimed to test the benefit of vectorization on our code compared to the theoretical speed-up based on the vector width, while the second one (`aligned`) was employed

to know the benefit of the aligned memory accesses supported by the reordering strategy proposed. Since we would like to emphasize the effect of vector width, we restricted our discussion here to single-precision arithmetic. The variable `VL` was the vector length, set to eight for AVX/AVX2 and 16 for AVX-512—and `Nbyte` was the default alignment of the architecture, set to 32 for AVX/AVX2 and 64 for AVX-512.

Two metrics are used to denote the performance of our code: Medge/s/core (million edges per second per core) and Mcell/s/core (million cells per second per core), which are the comparisons between the total number of simulated edges or cells that can be achieved per unit of time using one core. The former was used to denote the performance of the `SUBROUTINE edge-driven level`, whereas the latter was used to denote the performance of the entire simulation. It is also important to note that since the RKFO method is used, the latter is calculated after four times updating per time-level update, not per calculation-level update. We compiled our code using the flag `-O3 -qopenmp -align 'a' 'b'` for the vectorized version, where `'a' = array32byte` for AVX/AVX2 and `array64byte` for AVX-512—and `'b' = -xAVX` for AVX, `-xCORE-AVX2` for AVX2, and `-xCOMMON-AVX512` for AVX-512. To only emphasize the performance increase by vectorization, we disable all possibilities for auto vectorization by compiling with the flag `-O3 -qopenmp -no-vec -align 'a' 'b'` and by deleting all the SIMD directives in the source code, thus giving a fair benchmark of the non-vectorized version of our code.

As previously explained, we discuss our results using single-core and single-node computations. We observed that for single-node computations, NUFSAW2D with OpenMP gives better performance than MPI because the WDLB technique employed for wet–dry problems requires no communication cost. We only performed strong scaling for all cases, where we achieved averagely 87% efficiency for AVX/AVX2 with 16/28 cores and 88% efficiency for AVX-512 with 64 cores. When using 8/16 cores with AVX/AVX2 or 56 cores with AVX-512, higher efficiency was even achieved by our code being approximately 98%. Although this leads to a better performance, we still use the results with all cores available to show the single-node performance. Note the performance degradation of 12–13% (when using all cores) was not due to inefficient load distribution but probably because of the non-uniform memory access (NUMA) effects, where a processor can access its memory faster than the shared non-local memory, see [21].

4.5.1. Performance of Edge-Driven Level

Figure 18 shows the performance comparison between all solvers, in which we observe a significant performance improvement for each solver. Note the results in Figure 18 represent the average values from the four cases tested. We observed that there are no significant differences of the performance (in the range of 4–5%) achieved in all cases. The worst performance was shown in case 2, whereas the best one was achieved in case 1. This is because case 2 deals with more complex wet–dry problems, for which the WDLB technique in this case works better than in the other cases—thus causing more overheads—in order to balance the load units between wet and dry cells, see [21] for detail. For the edge-driven level, each non-vectorized solver shows performance metrics with a range of 3.42–4.54, 5.03–6.23, and 1.01–1.38 Medge/s/core for the AVX, AVX2, and AVX-512, respectively. This shows the CU scheme was, without vectorization, averagely 1.31× and 1.26× faster than the HLLC and Roe solvers, respectively.

As soon the guided vectorization was activated, the performances of each scheme in the edge-driven level increased significantly. For the AVX machine (1 core), we observed significant improvements being 5.5×, 6.5×, and 6× for the HLLC, Roe, and CU schemes, respectively; this shows the Roe scheme experiences remarkably the benefit of the vectorization, for which the improvement factor is larger than the others. For the AVX2 machine (1 core), the speed-up factors of 4.5×, 4.8×, and 5× were obtained by the HLLC, Roe, and CU schemes, respectively showing that the improvement factor of the CU scheme becomes the largest one among the others. Although significant performance improvements have been shown, our model still cannot fully exploit the theoretical speed-up of 8×

from the vector widths of both the AVX and AVX2 machines used. Nevertheless, we have shown that the data structures of our code are suitable for SIMD instructions as we are able to achieve the efficiency of up to 81.25%. Note in the aforementioned notable works, none of the models could achieve the performance increase of more than 52% from the theoretical speed-up of the machine used. In [20], the average speed-up of 4.1× was achieved on AVX machine (single-precision) for the vectorized augmented Riemann solver; therefore, this leads to the efficiency of 51.25%. In [22], the average speed-up of 1.7× was obtained on an AVX2 machine (double-precision) for the vectorized Riemann solver; this thus gives the efficiency of 42.5%.

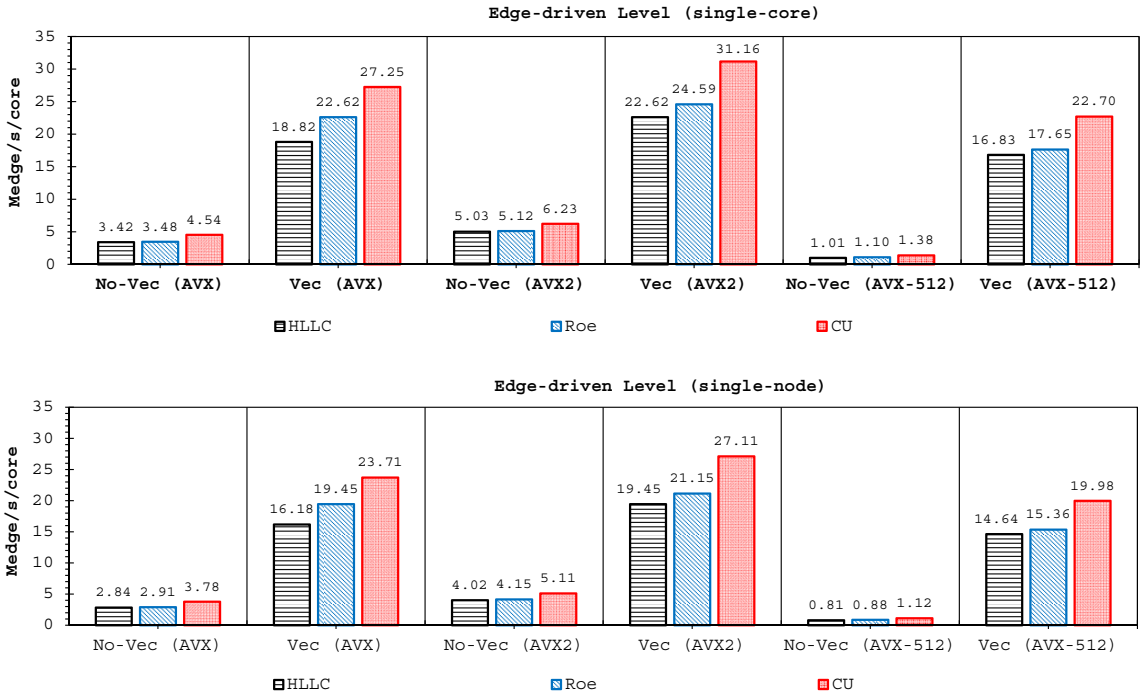


Figure 18. Comparison of performance metrics in the edge-driven level.

For the AVX-512 machine (1 core), the vectorization has tremendously increased the performances of the HLLC, Roe, and CU solvers by the factors of 16.68×, 16.04×, and 16.42×, respectively. This shows that our model can comprehensively exploit the vectorization for the vector width provided, of which the theoretical speed-up is 16×. Also, this represents that the data structures designed in NUFSAW2D efficiently support the vector programming on this vector-computing architecture.

With parallel simulations, each non-vectorized solver exhibits the performance metrics within the ranges of 2.84–3.78, 4.02–5.11, and 0.81–1.12 Medge/s/core for the AVX (16 cores), AVX2 (28 cores), and AVX-512 (64 cores), respectively; compared to the non-vectorized values with 1 core, it gives about 83% efficiency. For the performance analysis of the parallelized-vectorized solvers, the values obtained by the non-vectorized solvers with single-core are used as indicator. For the AVX machine (16 cores), the parallelized-vectorized HLLC, Roe, and CU solvers reached 16.18, 19.45, and 23.71 Medge/s/core, giving speed-ups of 75.7×, 89.4×, and 83.52×, respectively. Similarly, the parallelized-vectorized HLLC, Roe, and CU solvers obtained 19.45, 21.15, and 27.11 Medge/s/core, respectively with the AVX2 machine (28 cores) leading to speed-ups of 108.4×, 115.6×, and 121.8×. The significant performance increase was shown by the AVX-512 machine (64 cores), where the parallelized-vectorized HLLC, Roe, and CU solvers reached 14.64, 15.36, and 19.98 Medge/s/core, respectively; this brings each scheme to achieve speed-ups of 928.9×, 892.9×, and 924.7×.

The results in Figure 18 show an interesting fact, especially for the single-node performance analysis. Without vectorization, the parallelized results of the AVX2 machine can significantly outperform the parallelized results of the AVX-512 machine. For example, see Figure 19, on the AVX2 machine the CU scheme shows a metric of 143.1 Medge/s with 28 cores while on the AVX-512 machine with 64 cores this scheme exhibits a metric of 71.7 Medge/s; the difference is thus almost two-fold. However, with vectorization, the parallelized results of the AVX2 machine (759.1 Medge/s) are now outperformed by those of the AVX-512 machine (1278.6 Medge/s), being approximately 1.7×. This shows the vectorization is non-trivial for increasing the performance.

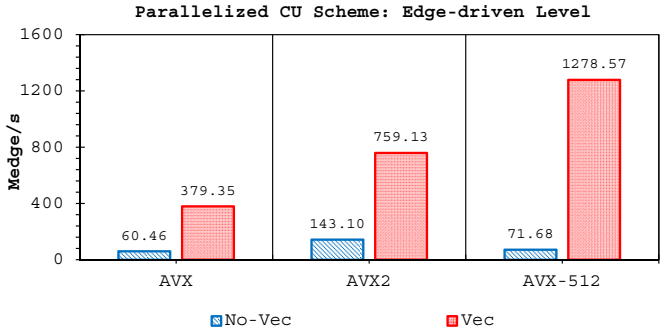


Figure 19. Single-node performance of the non-vectorized and vectorized CU scheme with parallelization (edge-driven level).

Based on these results, one can see although the Roe scheme experiences the largest speed-up on the AVX machine or the HLLC scheme achieves the largest improvement factor on the AVX-512 machine, both of these schemes are still significantly outperformed by the CU scheme with average multiplication factors of 1.4× and 1.25×, respectively. This is not so surprising since the computational procedures of both the HLLC and Roe solvers include complex branch statements (`if-then-else`), thus should theoretically be much more expensive than the CU scheme, see [17]. The HLLC scheme requires the nested branch statements; the first one is to compute the wave speeds, which are later required in the second branch statement for calculating the final convective fluxes. The Roe scheme needs branchings for the intermediate variables and entropy correction computations, the computations of which are quite complex. Such branchings may force the uses of masked operations and assignments, thus significantly decreasing the performance. In contrast to these two solvers, the CU scheme does not experience any branch statement. This is the beauty of this scheme in addition to being quite simple and having no complex procedure, thus can (even) be auto-vectorized by the compiler.

4.5.2. Performance of the Entire Simulation

Prior to investigating the performance of the entire simulation, we firstly show the cost estimation of each level in Algorithm 1 by presenting in Figure 20 a list of cost percentages: initialization, gradient, edge-driven level and cell-driven level. The last three components indicate the same levels to those shown in Algorithm 1, while initialization is a part required for updating the initial value for the RKFO method per time-level update, e.g., to perform $\mathbf{W}^{p=0} = \mathbf{W}^{t}$, see Equation (1). Note for an unbiased representative, the values in Figure 20 are the cost percentage of a vectorized solver relatively to its non-vectorized version. Only the cost percentage of the simulations using single-core is presented in Figure 20; the percentage for single-node is shown to be similar. As expected, we observe that the edge-driven level is the most time-consuming part being 65–75% of the entire simulation for the non-vectorized code. For both AVX and AVX2 machines, the vectorization can decrease the computational cost of the edge-driven level approximately from 71% up to 15%. Meanwhile, for the AVX-512 machine, the vectorization is shown more effective to reduce the cost of the edge-driven level averagely from 72% up to 5%.

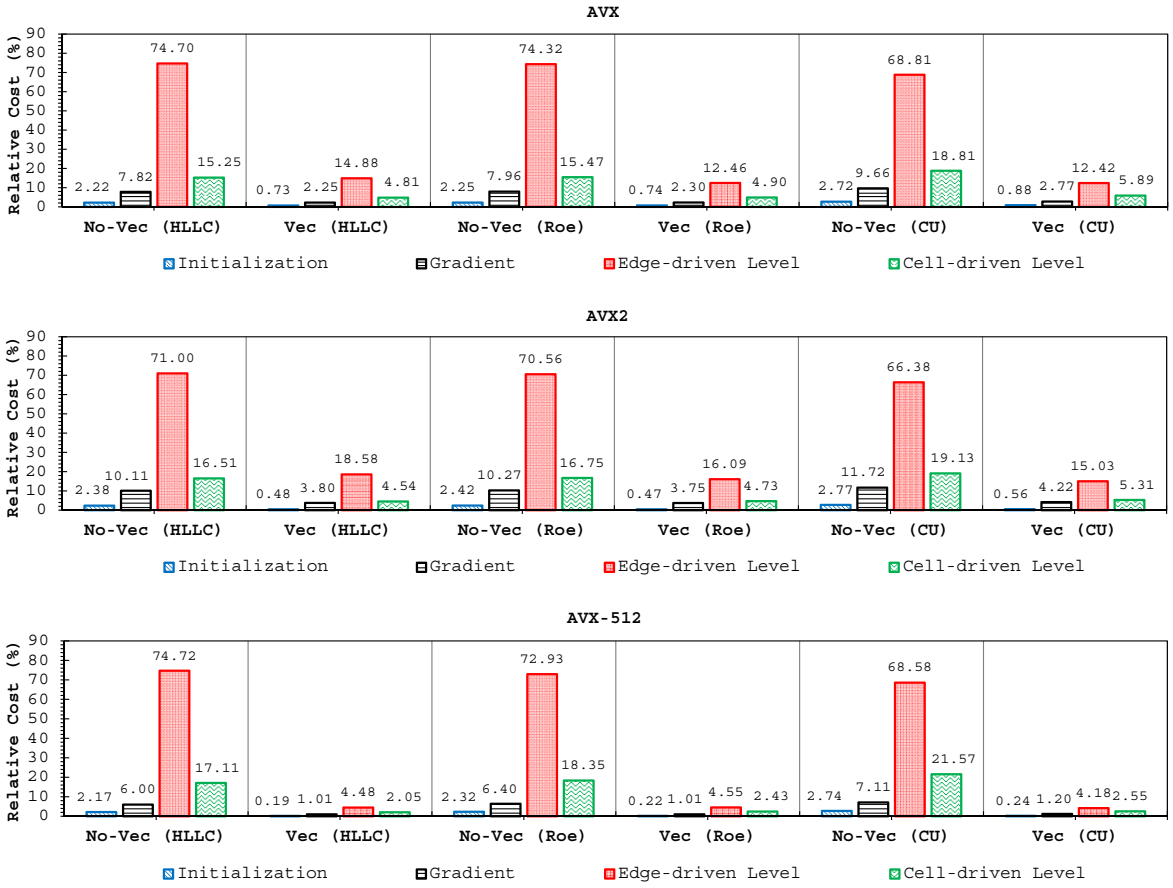


Figure 20. Components of the entire simulations for all schemes.

The second most expensive part is the cell-driven level, which consumes around 16–22% of the total simulation time with the non-vectorized solvers. After vectorization, the cost of the cell-driven level decreases from approximately 17% up to 5% on both the AVX and AVX2 machines. Meanwhile, on the AVX-512 machine, the vectorization has helped by decreasing the computational time of the cell-driven level averagely from 19% up to 3%; this again shows the vectorization works more effectively on this machine.

We now explain the performance of our model for updating the entire simulation. For the AVX, AVX2, and AVX-512 machines with one core, we observed for the non-vectorized solvers the metrics of 1.27–1.56, 1.78–2.06, and 0.38–0.47 Mcell/s/core, respectively—and for the vectorized solvers by 6.37–7.79, 7.12–9.28, and 5.23–6.23 Mcell/s/core, respectively. We achieved the improvements for the AVX machine by 5×, 5.5×, and 5× for the HLLC, Roe, and CU schemes, respectively, while for the AVX2 machine, the speed-up factors of 4×, 4.5×, and 4.5× were obtained by the HLLC, Roe, and CU schemes, respectively. On the AVX-512 machine we observed the speed-up factors of 13.91×, 13.11×, and 13.18× for the HLLC, Roe, and CU schemes, respectively showing that, on this machine, our code can achieve a better performance than those on the other two machines. However, the AVX2 machine still gives the highest metrics among the others.

For parallel simulations with the AVX, AVX2, and AVX-512 machines, the non-vectorized solvers achieved the metrics of 1.05–1.28, 1.42–1.67, and 0.3–0.38 Mcell/s/core, respectively—and for the parallelized-vectorized solvers by 5.48–6.78, 6.12–8.08, and 4.55–5.48 Mcell/s/core, respectively. Similar to the previous analysis, the values obtained by the non-vectorized solvers with single-core are used as indicator here. According to Figure 21, the vectorized HLLC, Roe, and CU solvers on the AVX machine (16 cores) gave the metrics of 5.48, 6.1, and 6.78 Mcell/s/core reaching speed-ups of 68.8×, 75.68×, and 69.6×, respectively. On the AVX2 machine (28 cores) we observed speed-ups of 96.3×,

108.4×, and 109.6× for the vectorized HLLC, Roe, and CU solvers by obtaining the metrics of 6.12, 6.98, and 8.08 Mcell/s/core, respectively. The AVX-512 machine (64 cores) shows again the significant performance increase by allowing the parallelized–vectorized HLLC, Roe, and CU solvers to achieve the metrics of 4.55, 4.57, and 5.48 Mcell/s/core or similar to speed-ups of 774.6×, 729.9×, and 742.1×, respectively. Based on this fact, a similar behavior is noticed for the single-node performance in updating the entire simulation. We take the results of the CU scheme as an example, see Figure 22. Without vectorization, the parallelized results of the AVX2 machine with 28 cores (46.80 Mcell/s) are about 1.93× significantly faster than those of the AVX-512 machine with 64 cores (24.22 Mcell/s). However, the parallelized–vectorized results of the AVX-512 machine (351 Mcell/s) now outperform the parallelized–vectorized results of the AVX2 machine (226.2 Mcell/s) by a factor of 1.55. This again shows the vectorization is highly-important for achieving better performance. For the entire simulation, our code with the vectorized solvers can achieve approximately 31–35% of the theoretical peak performance (TPP) of the AVX/AVX2 machines and 26% of the TPP of the AVX-512 machine.

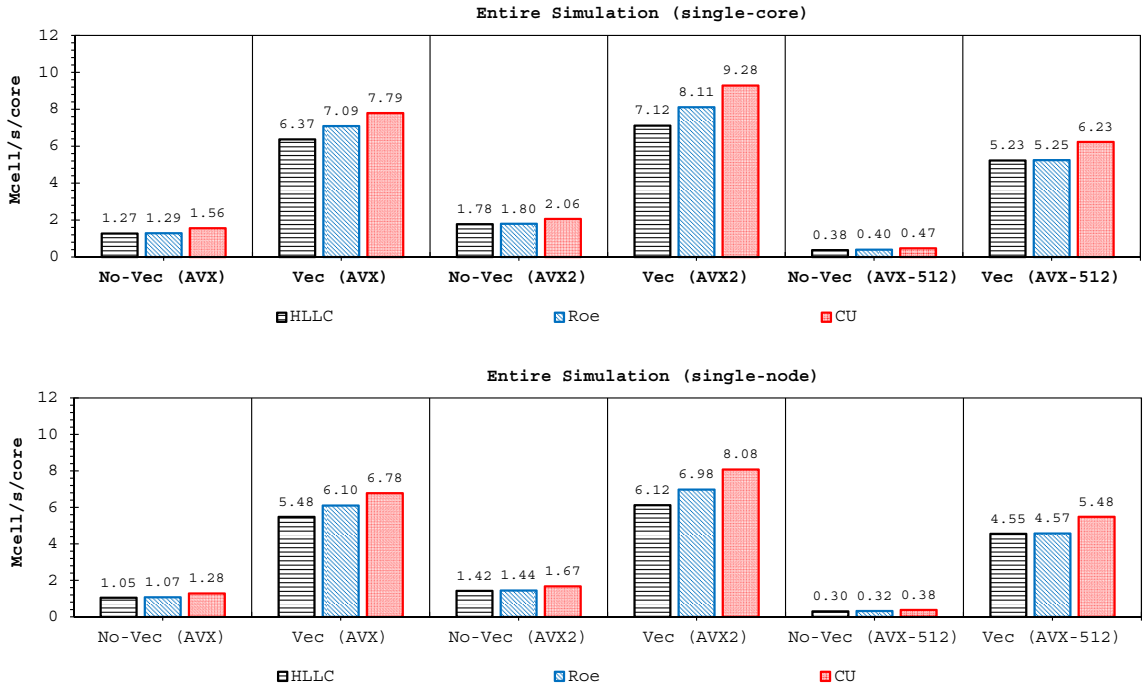


Figure 21. Comparison of performance metrics for the entire simulation.

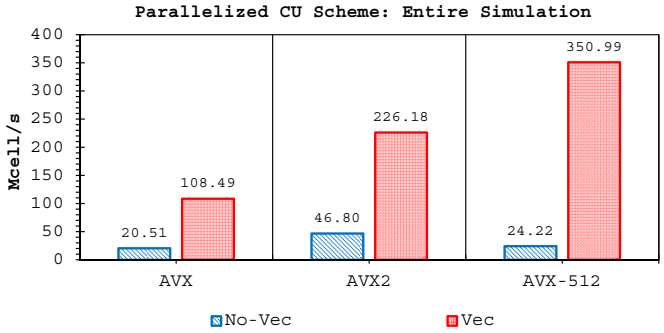


Figure 22. Single-node performance of the non-vectorized and vectorized CU scheme with parallelization (entire simulation).

We also actually studied the effect of the cell-edge reordering strategy on conserving the memory access patterns, where we compared the directive `!$omp simd simdlen(VL) aligned(var1,var2,... :Nbyte)` with the directive `!$omp simd simdlen(VL)`. Using the former, we found on all machines that the HLLC, Roe, and CU schemes averagely benefited from 1.45×, 1.5×, 1.4× more speed-ups in the edge-driven level compared to the results compiled only with the latter. Similarly, for updating the entire simulation, the HLLC, Roe, and CU solvers achieved on all machines approximately 1.41×, 1.42×, and 1.32× more speed-ups, respectively. These results reveal that the cell-edge reordering strategy proposed has helped in easing the aligned memory access pattern, thus enabling a significant performance enhancement. For the sake of brevity, these findings are not presented here.

5. Conclusions

A numerical investigation for studying the accuracy and efficiency of three common shallow water solvers (the HLLC, Roe, and CU schemes) has been presented. Four cases dealing with shock waves and wet–dry phenomenon were selected. All schemes were provided in an in-house code NUFSAW2D, the model of which was of second-order accurate in space wherever the regimes were smooth and robust when dealing with strong shock waves—and of fourth-order accurate in time. To give a fair comparison, all source terms of the 2D SWEs were treated similarly for all schemes, namely the bed-slope terms were computed separately from the convective fluxes using a Riemann-solver-free scheme—and the friction terms were computed semi-implicitly within the framework of the RKFO method.

Two important findings have been shown by our simulations. Firstly, highly-efficient vectorization could be applied to the three solvers on all hardware used. This was achieved by guided vectorization, where a cell-edge reordering strategy was employed to ease the vectorization implementations and to support the aligned memory access patterns. Regarding single-core analysis, the vectorization was shown to be able to speed-up the performance of the edge-driven level up to 4.5–6.5× on the AVX/AVX2 machines for eight data per vector and 16.7× on the AVX-512 machine for 16 data per vector—and to accelerate the entire simulation as well by up to 4–5.5× on the AVX/AVX2 machine and 13.91× on the AVX-512 machine. The superlinear speed-up in the edge-driven level especially using the AVX-512 machine could be achieved probably due to improved cache usage, thus less expensive main memory accesses. Regarding single-node analysis, our code could reach in the edge-driven level the improvements of 75.7–121.8× on the AVX/AVX2 machine while on the AVX-512 machine it achieved up to 928.9× speed-up. For updating the entire simulation, our code was able to reach speed-ups of 68.8–109.6× and 774.6× on the AVX/AVX2 and AVX-512 machines, respectively. We observed an interesting phenomenon, where without vectorization the parallelized results of the AVX2 machine outperformed those of the AVX-512 machine in both the edge-driven level and the entire simulation with a factor of up to 2×; the parallelized-vectorized results of the AVX-512 machine became, however, faster by achieving an average factor of 1.6×. This clearly shows that our reordering strategy could efficiently exploit the vectorization support of such a vector-computing machine. Supporting the aligned memory access patterns, the reordering strategy employed has helped in gaining the performances of the "only" vectorized code by averagely 1.45× and 1.4× for the edge-driven level and updating the entire simulation, respectively.

Secondly, we have shown that for the four cases simulated, strong agreements by all schemes were obtained between the numerical results and observed data, where no significant differences were shown for the accuracy. However, in the term of efficiency, the CU scheme was able to outperform the HLLC and Roe schemes with average factors of 1.4× and 1.25×, respectively. Although the vectorization was successful to significantly gain the performance of all solvers, the CU scheme still became the most efficient one among the others. According to this fact, we could conclude that the CU solver as a Riemann-solver-free scheme would in general be able to outperform the Riemann solvers (HLLC and Roe schemes) even for simulations on the next generation of modern hardware. This is

because the computational procedures of the CU scheme are acceptably simple especially containing no complex branch statements (`if-then-else`) such as required by the HLLC and Roe schemes.

Since simulating shallow water flows—especially complex phenomena that require performing long real-time computations as part of disaster planning such as dam-break or tsunami cases—on modern hardware nowadays and even in the future becomes more and more common, focusing simulations only on numerical accuracy but ignoring the performance efficiency is not an option anymore. Wasting the performance is obviously undesirable due to wasting too much time for such long real-time simulations. Modern hardware offers many features for gaining efficiency, one of which is vectorization that can be regarded as the "easiest" way for benefiting from the vector-level parallelism, is thus non-trivial. However, this is not obtained for free; one should at least understand and support—due to the sophisticated memory access patterns—the vectorization concept. The cell-edge reordering strategy employed here is one of the easiest strategies to utilize the vectorization feature of modern hardware that could easily be applied to any CCFV scheme for shallow flow simulations, together with guided vectorization instead of explicitly by low-level vectorization, which might be error-prone and time-consuming. It is worth pointing out that this strategy is also applicable to any compiler with vectorization support, e.g., Gfortran. We observed that the performance obtained with Intel compiler was typically 2–3$\times$ higher than that obtained with Gfortran, which we believe is due to the correspondence of Intel compiler and Intel hardware.

We have also shown that the edge-driven level, especially the reconstruction technique and solver computations, were the most time-consuming part, which required 65–75% of the entire simulation time. This shows that some more "aggressive" optimization techniques still become a hot topic for future studies to make shallow water simulations more efficient, particularly in the edge-driven level. Finally, we conclude that this study would be useful as a consideration for modelers who are interested in developing shallow water codes.

Author Contributions: B.M.G. developed the numerical code NUFSAW2D, conceived the framework of this work, analyzed the results, and wrote the paper. R.-P.M. contributed to the practical guidance of this work and provided some corrections, especially regarding the theoretical concepts of the vectorization and parallel computing.

Funding: The first author gratefully acknowledges the DAAD (German Academic Exchange Service) for supporting his research in the scope of the Research Grants—Doctoral Programmes in Germany 2015/16 (57129429). This work was supported by the German Research Foundation (DFG) and the Technical University of Munich (TUM) in the framework of the Open Access Publishing Program.

Acknowledgments: The authors appreciate the computational and data resources provided by the Leibniz Supercomputing Centre—and are also grateful to all the anonymous reviewers, who provided many constructive comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cea, L.; Blade, E. A simple and efficient unstructured finite volume scheme for solving the shallow water equations in overland flow applications. *Water Resour. Res.* **2015**, *51*, 5464–5486. [CrossRef]
2. Hou, J.; Liang, Q.; Zhang, H.; Hinkelmann, R. An efficient unstructured MUSCL scheme for solving the 2D shallow water equations. *Environ. Model. Softw.* **2015**, *66*, 131–152. [CrossRef]
3. Duran, A. A robust and well-balanced scheme for the 2D Saint-Venant system on unstructured meshes with friction source term. *Int. J. Numer. Methods Fluids* **2015**, *78*, 89–121. [CrossRef]
4. Özgen, I.; Zhao, J.; Liang, D. Hinkelmann, R. Urban flood modeling using shallow water equations with depth-dependent anisotropic porosity. *J. Hydrol.* **2016**, *541*, 1165–1184. [CrossRef]
5. Xia, X.; Liang, Q.; Ming, X.; Hou, J. An efficient and stable hydrodynamic model with novel source term discretization schemes for overland flow and flood simulations. *Water Resour. Res.* **2017**, *53*, 3730–3759. [CrossRef]
6. Roe, P. Approximate Riemann solvers, parameter vectors and difference schemes. *J. Comput. Phys.* **1981**, *135*, 250–258. [CrossRef]
7. Toro, E. *Shock-Capturing Methods for Free-Surface Shallow Flow*; John Wiley: Chichester, UK, 2001.

8. Harten, A.; Lax, P.; van Leer, B. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Rev.* **1983**, *25*, 35–61. [CrossRef]
9. Davis, S. Simplified second-order Godunov-type methods. *SIAM J. Sci. Stat. Comput.* **1988**, *9*, 445–473. [CrossRef]
10. Einfeldt, B. On Godunov-type methods for gas dynamics. *SIAM J. Numer. Anal.* **1988**, *25*, 294–318. [CrossRef]
11. Toro, E.; Spruce.; M. Speares, W. Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves* **1994**, *4*, 25–34. [CrossRef]
12. Kurganov, A.; Noelle, S.; Petrova, G. Semi-discrete central-upwind schemes for hyperbolic conservation laws and Hamilton-Jacobi equations. *SIAM J. Sci. Comput.* **1994**, *23*, 707–740. [CrossRef]
13. Kurganov, A.; Petrova, G. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. *Commun. Math. Sci.* **2007**, *5*, 133–160. [CrossRef]
14. Horváth, Z.; Waser, J.; Perdigão, R.; Konev, A.; Blöschl, G. A two-dimensional numerical scheme of dry/wet fronts for the Saint-Venant system of shallow water equations. *Int. J. Numer. Methods Fluids* **2015**, *77*, 159–182. [CrossRef]
15. Beljadid, A.; Mohammadian, A.; Kurganov, A. Well-balanced positivity preserving cell-vertex central-upwind scheme for shallow water flows. *Comput. Fluids* **2016**, *136*, 193–206. j.compfluid.2016.06.005. [CrossRef]
16. Delis, A.; Nikolos, I. A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered finite volumes with application to shallow water dynamics. *Int. J. Numer. Methods Fluids* **2013**, *71*, 584–633. [CrossRef]
17. Ginting, B.; Mundani, R.P. Artificial viscosity technique: A Riemann-solver-free method for 2D urban flood modelling on complex topography. In *Advances in Hydroinformatics*; Gourbesville, P., Cunge, J., Caignaert, G., Eds.; Springer Water: Singapore, 2018; pp. 51–74, doi:10.1007/978-981-10-7218-5_4.
18. Bik, A.; Girkar, M.; Grey, P.; Tian, X. Automatic intra-register vectorization for the Intel architecture. *Int. J. Parallel Program.* **2002**, *2*, 65–98.:1014230429447. [CrossRef]
19. Nuzman, D.; Rosen, I.; Zaks, A. Auto-vectorization of interleaved data for SIMD. In Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation, Ottawa, ON, Canada, 10–16 June 2006; pp. 132–143. [CrossRef]
20. Bader, M.; Breuer, A.; Hölzl, W.; Rettenberger, S. Vectorization of an augmented Riemann solver for the shallow water equations. In Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS), Bologna, Italy, 21–25 July 2014; pp. 193–201. [CrossRef]
21. Ginting, B.; Mundani, R.P. Parallel flood simulations for wet-dry problems using dynamic load balancing concept. *J. Comput. Civ. Eng. (ASCE)* **2019**, *33*, 1–18. [CrossRef]
22. Ferreira, C.; Mandli, K.; Bader, M. Vectorization of Riemann solvers for the single- and multi-layer shallow water equations. In Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS), Orléans, France, 16–20 July 2018; pp. 415–442. [CrossRef]
23. Liu, J.Y.; Smith, M.; Kuo, F.A.; Wu, J.S. Hybrid OpenMP/AVX acceleration of a Split HLL finite volume method for the shallow water and Euler equations. *Comput. Fluids* **2015**, *110*, 181–188. [CrossRef]
24. Ginting, B.; Mundani, R.P.; Rank, E. Parallel simulations of shallow water solvers for modelling overland flows. In Proceedings of the 13th International Conference on Hydroinformatics (HIC 2018), EPiC Series in Engineering, Palermo, Italy, 1–6 July 2018; La Loggia, G., Freni, G., Puleo, V., De Marchis, M., Eds.; Volume 3, pp. 788–799. [CrossRef]
25. Ginting, B. Central-upwind scheme for 2D turbulent shallow flows using high-resolution meshes with scalable wall functions. *Comput. Fluids* **2019**, *179*, 394–421. [CrossRef]
26. Ginting, B. A two-dimensional artificial viscosity technique for modelling discontinuity in shallow water flows. *Appl. Math. Model.* **2017**, *45*, 653–683. [CrossRef]
27. Audusse, E.; Bouchut, F.; Bristeau, M.O.; Klein, R.; Perthame, B. A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. *SIAM J. Sci. Comput.* **2004**, *25*, 2050–2065. [CrossRef]
28. Gallouët, T.; Hérard, J.M.; Seguin, N. Some approximate Godunov schemes to compute shallow-water equations with topography. *Comput. Fluids* **2003**, *32*, 479–513. [CrossRef]
29. Castro, M.; Gonzales-Vida, J.; Pares, C. Numerical treatment of wet/dry fronts in shallow flows with a modified Roe scheme. *Math. Model. Methods Appl. Sci.* **2006**, *16*, 897–931. [CrossRef]
30. Yu, H.; Huang, G.; Wu, C. Efficient finite-volume model for shallow-water flows using an implicit dual time-stepping method. *J. Hydraul. Eng. (ASCE)* **2015**, *141*, 1–12. [CrossRef]

31. Soarez-Frazão.; S. Zech, Y. Experimental study of dam-break flows against an isolated obstacle. *J. Hydraul. Res.* **2007**, *45*, 27–36. [CrossRef]
32. Yu, C.; Duan, J. Two-dimensional depth-averaged finite volume model for unsteady turbulent flow. *J. Hydraul. Res.* **2012**, *50*, 599–611. [CrossRef]
33. Briggs, M.; Synolakis, C.; Harkins, G.; Green, D. Laboratory experiments of tsunami runup on a circular island. *Pure Appl. Geophys.* **1995**, *144*, 569–593. [CrossRef]
34. Nikolos, I.; Delis, A. An unstructured node-centered finite volume scheme for shallow water flows with wet/dry fronts over complex topography. *Comput. Methods Appl. Mech. Eng.* **2009**, *198*, 3723–3750. [CrossRef]
35. Available online: https://coastal.usc.edu/currents_workshop/index.html (accessed on 25 September 2018).
36. Arcos, M.; LeVeque, R. Validating velocities in the GeoClaw tsunami model using observations near Hawaii from the 2011 Tohoku tsunami. *Pure Appl. Geophys.* **2014**, *17*, 849–867. [CrossRef]
37. Available online: https://sandstorm.cie.bgu.tum.de/wiki/index.php/Main_Page (accessed on 25 September 2018).
38. Available online: https://www.lrz.de (accessed on 25 September 2018).

Paper 4

B.M. Ginting and H. Ginting. Hybrid Artificial Viscosity–Central-Upwind Scheme for Recirculating Turbulent Shallow Water Flows. *Journal of Hydraulic Engineering (ASCE)*, 145(12): 04019041, 2019.
https://dx.doi.org/10.1061/(ASCE)HY.1943-7900.0001639

Hybrid Artificial Viscosity–Central-Upwind Scheme for Recirculating Turbulent Shallow Water Flows

Bobby Minola Ginting[1] and Herli Ginting[2]

Abstract: In this paper, a hybrid artificial viscosity–central-upwind (AV-CU) scheme is proposed for simulating recirculating turbulent shallow water flows by combining the artificial viscosity (AV) technique with the central-upwind (CU) scheme. Two-dimensional (2D) depth-averaged Reynolds-averaged Navier–Stokes (DA-RANS) equations are solved using the AV technique, whereas the CU scheme is employed to compute the κ–ϵ model. The model is spatially and temporally second-order accurate. Scalable wall functions (ScWFs) are employed, thus becoming flexible in generating meshes without having to estimate the wall friction velocity at the initial time step as if the standard wall functions (StWFs) were used. The results benefit strongly from this hybrid approach being more accurate than the CU and Harten-Lax-van Leer-contact (HLLC) schemes—and 1.52× cheaper than the HLLC scheme for modeling recirculating turbulent flows. As such, the proposed approach could become a promising method for practical engineering purposes to simulate turbulent shallow water flows more efficiently and accurately. **DOI: 10.1061/(ASCE)HY.1943-7900.0001639.**

Introduction

The importance of turbulence, especially in hydraulics, has been investigated a long time ago, see Levi (1995) for a historical review. In reality, almost all flows in hydraulics are turbulent (Rodi 2017). Both inviscid and viscous models based on shallow water equations (SWEs) have been used extensively in hydraulics (e.g., Mohammadian 2010; Kanayama and Dan 2013; Horvath et al. 2015; Shirkhani et al. 2016). The former neglects viscous terms, while the latter includes such terms using a constant kinematic viscosity, which can only change following depths and velocities. Both models, however, still cannot represent the mass, momentum, and heat transfers existing in turbulent conditions, e.g., flows past structures. To estimate such processes, Reynolds stresses are introduced as the eddy viscosity to characterize the relationships between mean velocity and turbulent fluctuations (Boussinesq 1877) employed in RANS models. Despite assuming an isotropic condition, RANS models are still competitive with large-eddy simulation or direct numerical simulation models due to their efficiency (Rodi 2017).

For two-dimensional (2D) depth-averaged Reynolds-averaged Navier–Stokes (DA-RANS) models with finite-volume methods, Riemann solvers are probably the most well-known scheme, e.g., Roe in Cea et al. (2007) and Cea (2005), Harten-Lax-van Leer (HLL) in Kim and Lynett (2011), and HLL-contact (HLLC) in Yu and Duan (2012), Kim et al. (2009), and Cao et al. (2015). Another solver (Riemann-solver-free scheme) is the central-upwind (CU) method proposed by Kurganov et al. (2001) and Kurganov and Petrova (2007). Recently, this scheme was employed for 2D hyperbolic systems in Kurganov et al. (2017) and coupled water–sediment models in Liu and Beljadid (2017). No splitting approach was required for low-dissipation discontinuous solutions (Beljadid and LeFloch 2017). The CU scheme satisfied both well-balanced and positivity-preserving properties (Beljadid et al. 2016; Cheng and Kurganov 2016). To simulate recirculating inviscid flows, the CU scheme was, however, worse than the Roe and HLLC schemes (Mohamadian et al. 2005). Therefore, the use of the CU scheme alone to simulate recirculating turbulent flows will lead to inaccurate results; otherwise, high-resolution meshes can be used, such as those investigated by the first author in Ginting (2019). However, the CU scheme in Ginting and Mundani (2018) could benefit from compilers' optimizations, e.g., vectorization, on modern hardware that are 1.3× cheaper than the HLLC scheme for 2D SWEs.

Recently, an AV technique was proposed in Ginting (2017) and Ginting and Mundani (2018) and proven to be competitive with the HLLC and CU schemes for efficiency and accuracy. This technique was originally proposed by Jameson et al. (1981)—the Jameson-Schmidt-Turkel (JST) scheme—and extensively used in aeronautics (Jameson and Mavriplis 1986; Mavriplis 1988; van Der Burg et al. 1992; Swanson et al. 1998; Jameson 2017, and references therein). The AV technique in Ginting (2017) employed a spectral radius of 3×3 system Jacobian matrix of 2D SWEs, and the dimensionless depth-discontinuity (DDD) sensor was devised based on depth values. Therefore, this configuration fits the nonlinearity of the convective fluxes of 2D SWEs. However, in its current formula, the AV technique is not suitable for turbulence applications, in this case the κ–ϵ model; for instance, see Eqs. (22) and (23). Therefore, a novel approach is proposed here, where the AV technique and the CU scheme are used to solve the 2D DA-RANS and κ–ϵ models, respectively. This approach benefits from two points: (1) the hybrid AV-CU scheme resolves the inability/inaccuracy of the CU scheme in simulating recirculating turbulent flows and (2) it can still take advantage of compilers' optimizations on modern hardware and is, thus, efficient. Such lower computational costs are obviously beneficial for practical engineering purposes. Both the AV and CU schemes are free from complex branch conditions (*if-then-else*), which are difficult to vectorize. Such conditions are required by the HLLC scheme for the speed wave and convective flux computations or by the Roe solver for the entropy correction. To the best of the authors' knowledge, no work in the literature used just the AV

[1]Chair for Computation in Engineering, Dept. of Civil, Geo and Environmental Engineering, Technical Univ. of Munich, Arcisstr. 21, Munich D-80333, Germany (corresponding author). ORCID: https://orcid.org/0000-0001-7825-5052. Email: bobbyminola.ginting@tum.de

[2]Dept. of Physics, Univ. of Sumatera Utara, Jl. Bioteknologi No. 1, Medan 20155, Indonesia. Email: herliginting55@gmail.com

Note. This manuscript was submitted on May 24, 2018; approved on March 25, 2019; published online on September 27, 2019. Discussion period open until February 27, 2020; separate discussions must be submitted for individual papers. This paper is part of the ***Journal of Hydraulic Engineering***, © ASCE, ISSN 0733-9429.

technique with a second-order finite-volume model for shallow flow applications. The AV technique developed here is different from that of Chen et al. (2013) and Hernandez-Duenas and Beljadid (2016), where an adaptive AV was applied to enforce the nonlinear stability of the reconstruction process, whereas the convective fluxes were still calculated using the CU scheme; in other words, the AV method replaced the role of common limiters, e.g., MinMod and Superbee, but not the role of solvers. Here the AV technique is tied specifically to convective fluxes and so plays the role of solvers, where the MinMod limiter is utilized for the reconstruction process. Also, the AV formulas developed here differ from the aforementioned works. All computations in this work were performed using the first author's in-house code NUFSAW2D (*Numerical simUlation of Free surface ShAllow Water 2D*).

Governing Equations

2D DA-RANS Equations

The 2D DA-RANS equations are written

$$\frac{\partial Q}{\partial t}+\frac{\partial C_x}{\partial x}+\frac{\partial C_y}{\partial y}=\frac{\partial D_x}{\partial x}+\frac{\partial D_y}{\partial y}+S_b+S_f \tag{1}$$

where Q = conservative variables; C_x and C_y = convective fluxes (convective terms); D_x and D_y = diffusive fluxes (viscous terms) according to Boussinesq's assumption; S_b = bed-slope terms; S_f = bed friction terms; and U = primitive variables. All these vectors are given as (Rodi 1993; Wu 2004)

$$Q=\begin{bmatrix}h\\ hu\\ hv\end{bmatrix},\quad U=\begin{bmatrix}h\\ u\\ v\end{bmatrix},$$

$$C_x=\begin{bmatrix}hu\\ huu+\dfrac{gh^2}{2}\\ hvu\end{bmatrix},\quad C_y=\begin{bmatrix}hv\\ huv\\ hvv+\dfrac{gh^2}{2}\end{bmatrix},$$

$$D_x=\begin{bmatrix}0\\ 2h(\nu_e+\nu_t)\dfrac{\partial u}{\partial x}-\dfrac{2}{3}h\kappa\\ h(\nu_e+\nu_t)\left(\dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\right)\end{bmatrix},\quad D_y=\begin{bmatrix}0\\ h(\nu_e+\nu_t)\left(\dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\right)\\ 2h(\nu_e+\nu_t)\dfrac{\partial v}{\partial y}-\dfrac{2}{3}h\kappa\end{bmatrix},$$

$$S_b=\begin{bmatrix}0\\ -gh\dfrac{\partial z}{\partial x}\\ -gh\dfrac{\partial z}{\partial y}\end{bmatrix},\quad S_f=\begin{bmatrix}0\\ -c_f u\sqrt{u^2+v^2}\\ -c_f v\sqrt{u^2+v^2}\end{bmatrix} \tag{2}$$

where h, u, and v = flow depth and velocities in x and y directions, respectively; g = gravity acceleration; ν_e and ν_t = kinematic and eddy viscosities, respectively; z = bed contour; κ = turbulent kinetic energy; n_m = Manning coefficient; and $c_f=gn_m^2h^{-\frac{1}{3}}$. The water elevation is computed by $\eta=h+z$.

κ–ϵ Turbulence Equations

The turbulence model is solved to compute ν_t and κ in Eq. (2) and expressed as

$$\frac{\partial \Phi}{\partial t}+\frac{\partial C_{\Phi,x}}{\partial x}+\frac{\partial C_{\Phi,y}}{\partial y}=\frac{\partial D_{\Phi,x}}{\partial x}+\frac{\partial D_{\Phi,y}}{\partial y}+S_{\kappa-\epsilon} \tag{3}$$

where Φ = conservative variables; $C_{\Phi,x}$ and $C_{\Phi,y}$ = convective fluxes; $D_{\Phi,x}$ and $D_{\Phi,y}$ = diffusive fluxes; $S_{\kappa-\epsilon}$ = turbulence source terms; and Υ = primitive variables. All these vectors are given by

$$\Phi=\begin{bmatrix}h\kappa\\ h\epsilon\end{bmatrix},\quad \Upsilon=\begin{bmatrix}\kappa\\ \epsilon\end{bmatrix},\quad C_{\Phi,x}=\begin{bmatrix}h\kappa u\\ h\epsilon u\end{bmatrix},\quad C_{\Phi,y}=\begin{bmatrix}h\kappa v\\ h\epsilon v\end{bmatrix}$$

$$D_{\Phi,x}=\begin{bmatrix}\sigma_\kappa^{-1}h\nu_t\dfrac{\partial \kappa}{\partial x}\\ \sigma_\epsilon^{-1}h\nu_t\dfrac{\partial \epsilon}{\partial x}\end{bmatrix},\quad D_{\Phi,y}=\begin{bmatrix}\sigma_\kappa^{-1}h\nu_t\dfrac{\partial \kappa}{\partial y}\\ \sigma_\epsilon^{-1}h\nu_t\dfrac{\partial \epsilon}{\partial y}\end{bmatrix},$$

$$S_{\kappa-\epsilon}=\begin{bmatrix}P_h+P_{\kappa b}-h\epsilon\\ c_{\epsilon 1}\dfrac{\epsilon}{\kappa}P_h+P_{\epsilon b}-c_{\epsilon 2}h\dfrac{\epsilon^2}{\kappa}\end{bmatrix} \tag{4}$$

ϵ is the energy dissipation rate, and P_h accounts for the turbulent energy production due to the horizontal velocity gradient, whereas $P_{\kappa b}$ and $P_{\epsilon b}$ include the turbulent energy production due to bed friction. These terms are defined as

$$P_h=h\nu_t\left[2\left(\frac{\partial u}{\partial x}\right)^2+2\left(\frac{\partial v}{\partial y}\right)^2+\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)^2\right]$$

$$P_{\kappa b}=c_f^{-0.5}U_*^3,\qquad P_{\epsilon b}=\frac{c_{\epsilon\Gamma}c_{\epsilon 2}c_\mu^{0.5}c_f^{-0.75}}{h} \tag{5}$$

where $U_*=\sqrt{c_f(u^2+v^2)}$ = bed friction velocity. ν_t is computed by

$$\nu_t=c_\mu\left(\frac{\kappa^2}{\epsilon}\right) \tag{6}$$

All coefficients in Eqs. (4)–(6) are given in Rodi (1993) as

$$c_\mu=0.09,\quad c_{\epsilon 1}=1.44,\quad c_{\epsilon 2}=1.92,$$
$$\sigma_\kappa=1.0,\quad \sigma_\epsilon=1.3,\quad c_{\epsilon\Gamma}=[1.8,3.6] \tag{7}$$

Numerical Model

Spatial Discretization

Integrating over a control cell Ω and applying the Gauss divergence theorem, both Eqs. (1) and (3) are respectively written

$$\int\int_\Omega \frac{\partial Q}{\partial t}d\Omega+\oint_\Gamma (C_x+C_y-D_x-D_y)\cdot\vec{n}d\Gamma=\int\int_\Omega (S_b+S_f)d\Omega \tag{8}$$

$$\int\int_\Omega \frac{\partial \Phi}{\partial t}d\Omega+\oint_\Gamma (C_{\Phi,x}+C_{\Phi,y}-D_{\Phi,x}-D_{\Phi,y})\cdot\vec{n}d\Gamma=\int\int_\Omega S_{\kappa-\epsilon}d\Omega \tag{9}$$

where Γ = line boundary of control cell Ω; and $\vec{n}$ = unit normal vector pointing outward of boundary. Using the notation $F_x=[C_x,C_{\Phi,x}]^{\mathrm{T}}$ and $F_y=[C_y,C_{\Phi,y}]^{\mathrm{T}}$, the line integrals in Eqs. (8) and (9) are estimated by

$$\oint_\Gamma (F_x+F_y)\cdot\vec{n}d\Gamma\approx\sum_{k=1}^{N}(F_xn_x+F_yn_y)_k\Delta L_k \tag{10}$$

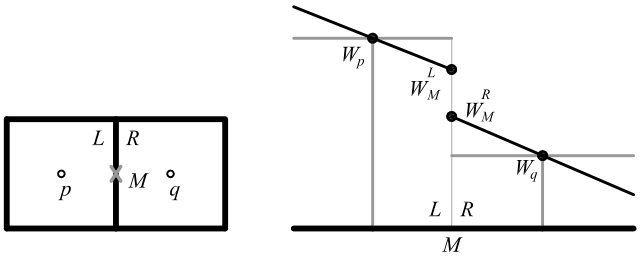


Fig. 1. Representation of L and R Riemann states at midpoint M of edge k.

where N = total number of edges for cell p ($N = 4$ for rectangular grids); and ΔL_k = edge length.

Monotonic Upwind Scheme for Conservation Laws Linear Reconstruction

The Monotonic Upwind Scheme for Conservation Laws (MUSCL) linear reconstruction procedures in van Leer (1979) are employed with the edge-based MinMod limiter (Venkatakrishnan 1993; van Albada et al. 1997; Delis and Nikolos 2013) for preserving monotonicity. These approaches are briefly discussed here. The values of the left (L) and right (R) states at the midpoint M of edge i in Fig. 1 are computed as

$$W_M^L = W_p + \frac{\|r_{pM}\|}{\|r_{pq}\|} LIM[\nabla W_p^{upw} \cdot \mathbf{r}_{pq}, \nabla W^{cent} \cdot \mathbf{r}_{pq}]$$
$$W_M^R = W_q - \frac{\|r_{Mq}\|}{\|r_{pq}\|} LIM[\nabla W_q^{upw} \cdot \mathbf{r}_{pq}, \nabla W^{cent} \cdot \mathbf{r}_{pq}] \quad (11)$$

where $W = [\eta, U, \Upsilon]$; $\|r_{pM}\|$, $\|r_{Mq}\|$, and $\|r_{pq}\|$ = scalar lengths between center points p and q and the midpoint M at the edge, shown in Fig. 1; ∇W_p^{upw}, ∇W_q^{upw}, and ∇W^{cent} = delta gradients corresponding to the two adjacent cells p and q; LIM = edge-based MinMod limiter function used to preserve the monotonicity of W_M^L and W_M^R.

The MUSCL reconstruction process is depicted in Fig. 2. The real condition is shown by Fig. 2(a). Without reconstruction, the values of W are piecewise constant with only first-order spatial accuracy, shown in Fig. 2(b). For this, the values at the edges are extrapolated from the cell centers, e.g., $h_M^L = h_p$. The bed elevation is thus computed by $z_M^L = \eta_M^L - h_M^L = \eta_p - h_p = z_p$. As shown in Fig. 2(c), using the MUSCL method with the limiter in Eq. (11), the values of W achieve spatially second-order accuracy by guaranteeing the monotonicity as

$$\min(W_p, W_q) \le W_M^L \le \max(W_p, W_q),$$
$$\min(W_p, W_q) \le W_M^R \le \max(W_p, W_q) \quad (12)$$

If the conservative variables are reconstructed, the velocities may not be monotonic because the values are not reconstructed directly from the MUSCL method but computed as, e.g., $u_M^R = hu_M^R/h_M^R$, so it may not satisfy $\min(u_p, u_q) \le u_M^R \le \max(u_p, u_q)$. Hou et al. (2013b) found this to be a typical problem for almost-dry cell simulations, which may give local extreme values at edges that can cause negative depths; a criterion was thus proposed to detect such extreme values using a threshold, where the second-order scheme turns to a first-order one when such values are detected. Because the primitive variables are reconstructed here, no local extreme values will appear due to such almost-dry cells. To ensure the monotonicity of the conservative variables, it is employed here:

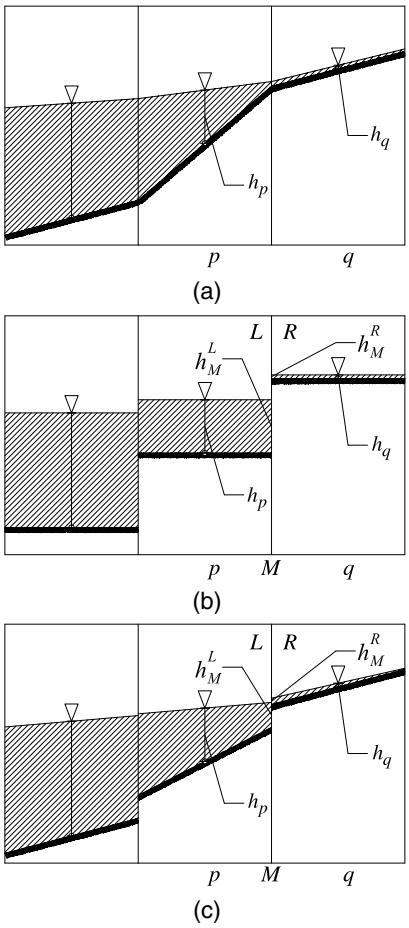


Fig. 2. Concept of reconstruction process: (a) real condition; (b) without reconstruction; and (c) with MUSCL reconstruction.

$$\min(hu_p, hu_q) \le hu_M^L \le \max(hu_p, hu_q),$$
$$\min(hu_p, hu_q) \le hu_M^R \le \max(hu_p, hu_q)$$
$$\min(hv_p, hv_q) \le hv_M^L \le \max(hv_p, hv_q),$$
$$\min(hv_p, hv_q) \le hv_M^R \le \max(hv_p, hv_q)$$
$$\min(\Phi_p, \Phi_q) \le \Phi_M^L \le \max(\Phi_p, \Phi_q),$$
$$\min(\Phi_p, \Phi_q) \le \Phi_M^R \le \max(\Phi_p, \Phi_q) \quad (13)$$

If one of the criteria in Eq. (13) is violated, the model turns to a first-order scheme. This simple approach is able to maintain the stability of all simulations here.

Nonnegative Depth Reconstruction

First, the bed elevation at each edge is calculated following Audusse et al. (2004) and Audusse and Bristeau (2005) as

$$z_M = \max(z_M^L, z_M^R) = \max(\eta_M^L - h_M^L, \eta_M^R - h_M^R) \quad (14)$$

To obtain nonnegative values, the depth of each edge is corrected as

$$h_M^L = \max(\eta_M^L - z_M, 0), \qquad h_M^R = \max(\eta_M^R - z_M, 0) \quad (15)$$

The values in Eq. (15) are then used for the convective flux calculations.

J. Hydraul. Eng., 2019, 145(12): 04019041

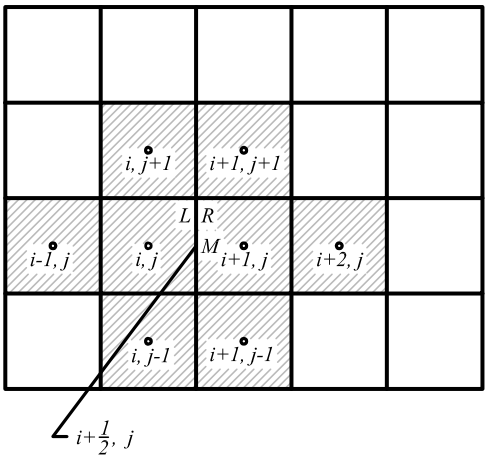


Fig. 3. Representation of domain for AV technique.

Convective Flux Calculations

Averaging the reconstructed L and R Riemann states minus the artificial diffusive terms, C_x and C_y at Point M of each edge are computed as (Ginting 2017; Jameson et al. 1981)

$$[C_x(Q_M)n_x + C_y(Q_M)n_y]_k \Delta L_k = \frac{1}{2}[(C_x(Q_M^R) + C_x(Q_M^L))n_x + (C_y(Q_M^R) + C_y(Q_M^L))n_y]_k \Delta L_k - \mathbf{D}(Q_M)_k \tag{16}$$

$\mathbf{D}$ denotes the artificial diffusive terms, which are devised based on a Laplacian and a biharmonic operator (Fig. 3), expressed respectively as

$$\begin{aligned}\nabla^2 Q_{i,j} &= (Q_{i+1,j} - Q_{i,j}) + (Q_{i,j+1} - Q_{i,j}) \\ &\quad + (Q_{i-1,j} - Q_{i,j}) + (Q_{i,j-1} - Q_{i,j}) \\ &= Q_{i+1,j} + Q_{i,j+1} + Q_{i-1,j} + Q_{i,j-1} - 4Q_{i,j}\end{aligned} \tag{17}$$

$$\begin{aligned}\nabla^4 Q_{i,j} &= (\nabla^2 Q_{i+1,j} - \nabla^2 Q_{i,j}) + (\nabla^2 Q_{i,j+1} - \nabla^2 Q_{i,j}) \\ &\quad + (\nabla^2 Q_{i-1,j} - \nabla^2 Q_{i,j}) + (\nabla^2 Q_{i,j-1} - \nabla^2 Q_{i,j}) \\ &= \nabla^2 Q_{i+1,j} + \nabla^2 Q_{i,j+1} + \nabla^2 Q_{i-1,j} + \nabla^2 Q_{i,j-1} - 4\nabla^2 Q_{i,j}\end{aligned} \tag{18}$$

Eqs. (17) and (18) are manipulated in an edge-based way as

$$\Delta Q_{M_{i+\frac{1}{2}}} = Q_{i+1,j} - Q_{i,j} \tag{19}$$

$$\begin{aligned}\Delta^4 Q_{M_{i+\frac{1}{2},j}} &= \nabla^2 Q_{i+1,j} - \nabla^2 Q_{i,j} \\ &= (Q_{i+1,j-1} - Q_{i,j-1}) + (Q_{i+1,j+1} - Q_{i,j+1}) \\ &\quad + (Q_{i+2,j} - Q_{i-1,j}) - 5(Q_{i+1,j} - Q_{i,j})\end{aligned} \tag{20}$$

$\mathbf{D}(Q_M)_{i+\frac{1}{2},j}$ are thus calculated by

$$\mathbf{D}(Q_M)_{i+\frac{1}{2},j} = \Lambda^{\max}_{i+\frac{1}{2},j}\left(\varepsilon^{(2)}_{i+\frac{1}{2},j}\Delta Q_{M_{i+\frac{1}{2},j}} - \varepsilon^{(4)}_{i+\frac{1}{2},j}\Delta^4 Q_{M_{i+\frac{1}{2},j}}\right)\Delta L_{i+\frac{1}{2},j} \tag{21}$$

where $\varepsilon^{(2)}_{i+\frac{1}{2},j}$ and $\varepsilon^{(4)}_{i+\frac{1}{2},j}$ = Laplacian and biharmonic coefficients, respectively; and $\Lambda^{\max}_{i+\frac{1}{2},j}$ = spectral radius of Jacobian matrix for Eq. (1)—see the first 3×3 system in Eq. (28)—computed by

$$\Lambda^{\max}_{i+\frac{1}{2},j} = \frac{1}{2}\left[|u_{i+1,j} + u_{i,j}|n_{x_{i+\frac{1}{2},j}} + |v_{i+1,j} + v_{i,j}|n_{y_{i+\frac{1}{2},j}} + \sqrt{g(h_{i+1,j} + h_{i,j})}\right] \tag{22}$$

Because $\Lambda^{\max}$ does not need the eigen-decomposition of the Jacobian matrix as required by Riemann solvers, the AV technique is regarded as a Riemann-solver-free scheme.

To calculate $\varepsilon^{(2)}$ and $\varepsilon^{(4)}$, a DDD sensor based on h is required. Slightly changing the sensor in Ginting (2017) based on Jameson (2017), a small value $h_{lim} = 10^{-15}$ m is added, and the sensor is now expressed as

$$\begin{aligned}\varphi_{i,j} &= \frac{|h_{i+1,j} + h_{i,j+1} + h_{i-1,j} + h_{i,j-1} - 4h_{i,j}|}{\max[(1-\omega)P_{i,j} + \omega R_{i,j}, h_{lim}]} \\ P_{i,j} &= |h_{i+1,j} - h_{i,j}| + |h_{i,j+1} - h_{i,j}| + |h_{i-1,j} - h_{i,j}| + |h_{i,j-1} - h_{i,j}| \\ R_{i,j} &= h_{i+1,j} + h_{i,j+1} + h_{i-1,j} + h_{i,j-1} + 4h_{i,j}\end{aligned} \tag{23}$$

where $0 \le \omega \le 1$. h_{lim} works as a tolerance when h is constant and $\omega = 0$ is chosen. For each edge, φ is calculated as

$$\varphi_{i+\frac{1}{2},j} = \max(\varphi_{i+2,j}, \varphi_{i+1,j}, \varphi_{i,j}, \varphi_{i-1,j}) \tag{24}$$

The best forms of $\varepsilon^{(2)}$ and $\varepsilon^{(4)}$ for shallow flows were shown in Ginting (2017) to be

$$\varepsilon^{(2)}_{i+\frac{1}{2},j} = \kappa^{(2)}\varphi_{i+\frac{1}{2},j}, \qquad \varepsilon^{(4)}_{i+\frac{1}{2},j} = \max\left(0, \kappa^{(4)} - \varphi_{i+\frac{1}{2},j}\right) \tag{25}$$

where $\kappa^{(2)}$ and $\kappa^{(4)}$ = problem-dependent coefficients. After data fitting, $1.5 \le \kappa^{(2)} \le 5$ and $1/250 \le \kappa^{(4)} \le 1/32$ were found, where $\kappa^{(2)} = 5$, $\kappa^{(4)} = 1/32$, and $\omega = 0.5$ are recommended (Ginting 2017).

It is clear that $\Lambda^{\max}$ and φ in Eqs. (22) and (23) fit the nonlinearities of hu, huu, hvv, and huv in C_x and C_y. For the κ–ϵ model, $h\kappa u$, $h\epsilon u$, $h\kappa v$, and $h\epsilon v$ appear in $C_{\Phi,x}$ and $C_{\Phi,y}$, creating new nonlinear forms. Eqs. (22) and (23) are therefore no longer suitable, hypothetically because $\Lambda^{\max}$ and φ do not match such new nonlinearities. This may be anticipated by considering Eq. (22) based on the full 5×5 global system in Eq. (28) and by adding two new sensors similar to Eq. (23) but based on κ and ϵ or $h\kappa$ and $h\epsilon$. Although this works, CPU time will significantly increase due to huge computational-memory requirements. This has motivated the authors to employ the CU scheme for the κ–ϵ model.

Using the CU scheme, both $C_{\Phi,x}$ and $C_{\Phi,y}$ are computed as

$$\begin{aligned}&[C_{\Phi,x}(\Phi_M)n_x + C_{\Phi,y}(\Phi_M)n_y]_k \Delta L_k \\ &= \frac{\Delta L_k}{(a^{in} + a^{out})_k}[(a^{in}C_{\Phi,x}(\Phi_M^R) + a^{out}C_{\Phi,x}(\Phi_M^L))n_x \\ &\quad + (a^{in}C_{\Phi,y}(\Phi_M^R) + a^{out}C_{\Phi,y}(\Phi_M^L))n_y - a^{in}a^{out}(\Phi_M^R - \Phi_M^L)]_k\end{aligned} \tag{26}$$

where a^{in} and a^{out} are the local one-sided propagation speeds calculated by

$$\begin{aligned}a^{in} &= -\min(\Lambda_{1M}^L, \Lambda_{1M}^R, 0) \\ a^{out} &= \max(\Lambda_{5M}^L, \Lambda_{5M}^R, 0)\end{aligned} \tag{27}$$

$\Lambda_1 \le \ldots \le \Lambda_5$ are the eigenvalues of the 5×5 system Jacobian matrix for Eqs. (1) and (3). This matrix reads

$$\begin{bmatrix} 0 & n_x & n_y & 0 & 0 \\ (-u^2+gh)n_x - uvn_y & 2un_x + vn_y & un_y & 0 & 0 \\ -uvn_x + (-v^2+gh)n_y & vn_x & un_x + 2vn_y & 0 & 0 \\ -(\kappa un_x + \kappa vn_y) & \kappa n_x & \kappa n_y & un_x + vn_y & 0 \\ -(\epsilon un_x + \epsilon vn_y) & \epsilon n_x & \epsilon n_y & 0 & un_x + vn_y \end{bmatrix} \tag{28}$$

Note that there is no direct correlation between the AV and CU schemes. For instance, the coefficient for Eq. (16) is constant (1/2), whereas Eq. (26) depends on a^{in} and a^{out}. If $a^{in} = a^{out}$, Eqs. (16) and (26) look similar. However, they are different even if $\varepsilon^{(4)} = 0$, since $\varepsilon^{(2)}$ is not included in the last term of Eq. (26).

Diffusive Flux Calculations

The gradients of W at cell p computed using the primitive variables in Eq. (11) are used for the diffusive fluxes and expressed in the x-direction using the compact stencil method:

$$\left(\frac{\partial W}{\partial x}\right)_p \approx \frac{1}{A_{q\leftarrow 1-4}} \sum_{q=1}^{4} \left[\frac{1}{2}(W_q + W_{q+1}) n_{x_{q-q+1}}\right] \Delta L_{q-q+1} \tag{29}$$

where $A_{q\leftarrow 1-4}$ = area formed by surrounding cells $q = 1-4$ shown in Fig. 4. A similar method is applied to the y-direction. Using the notation $G_x = [D_x, D_{\Phi,x}]^{\mathrm{T}}$ and $G_y = [D_y, D_{\Phi,y}]^{\mathrm{T}}$, these vectors are computed by

$$-\oint_{\Gamma} (G_x + G_y) \cdot \vec{n} d\Gamma \approx -\sum_{k=1}^{4} (G_x n_x + G_y n_y)_k \Delta L_k \tag{30}$$

To calculate G_x and G_y at edges, the centered method is used:

$$(G_x n_x + G_y n_y)_k \Delta L_k = \frac{\Delta L_k}{2} [(G_x(W_p) + G_x(W_q))n_x + (G_y(W_p) + G_y(W_q))n_y]_k \tag{31}$$

Other methods are also possible for Eq. (31) (Castanedo et al. 2005).

Bed-Slope Term Calculations

Before computing the bed-slope terms, z_M in Eq. (14) must be reconstructed to ensure a flux balance, expressed as

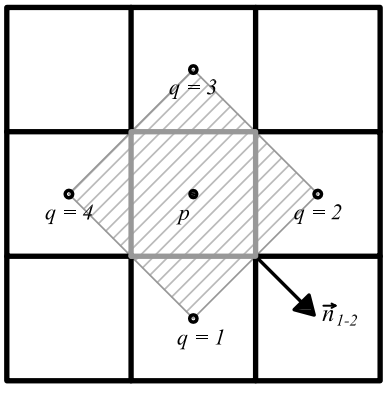


Fig. 4. Compact stencil method for gradient of W.

$$z_M \leftarrow z_M - \max(0, z_M - \eta_M^L, z_M - \eta_M^R) \tag{32}$$

The bed-slope terms in the x-direction (S_{bx}) for a cell p are discretized as (Mohammadian and Le Roux 2006; Ginting and Mundani 2018)

$$\begin{aligned} \int\int_{\Omega} S_{bx} d\Omega &= -\int\int_{\Omega} \left(gh\frac{\partial z}{\partial x}\right) d\Omega \\ &\approx -\frac{g}{2}\sum_{k=1}^{4}\left[\left(\frac{(h_{Mk}^L + h_{Mk}^R)}{2} + h_p\right) z_{Mk} \right. \\ &\quad \left. - \frac{(h_{Mk}^L + h_{Mk}^R)}{2} z_p\right] n_{xk} \Delta L_k \end{aligned} \tag{33}$$

A similar method is applied to S_{by}. To explain the procedures of the nonnegative depth reconstruction and bed-slope calculation, Figs. 5(a and d) are given for wet–wet and wet–dry interfaces, respectively. For the former, Eqs. (14) and (15) are applied to cell p to preserve the flux conservativeness, sketched in Fig. 5(b), yielding $h_M^L < h_p$; no change applies to h_p. Consequently, to have a correct bed slope for h_M^L, the value of z_M must be recalculated, where z_p does not change [Fig. 5(c)]. For this wet–wet interface, Eq. (14) gives the same value to that given by Eq. (32). For the wet–dry interface in Fig. 5(d), no flux must be transferred between cells p and q. To accomplish this, the hydrostatic reconstruction in Eq. (14) is employed, and Eq. (15) corrects the depth [Fig. 5(e)], so $h_M^L = 0$; again, h_p does not change. Accordingly, the bed slope must be corrected using Eq. (32), so $z_M = \eta_M^L$, shown by Fig. 5(f). Thus, Eq. (32) affects only wet–dry interfaces but plays no role in wet–wet interfaces. Therefore, Eq. (32) is used for all interface conditions; it also applies to dry–dry interfaces to ensure a no-flux-transfer condition (Ginting 2019). Eq. (32) is used only for edges, so the real topography at the centers never changes. Eq. (33) satisfies the well-balanced property (Appendix) and is a Riemann-solver-free technique computed separately from Eq. (16) and is thus also applicable to schemes with Riemann solvers.

Friction Term Calculations

Semi-implicit treatments were developed to compute friction terms (Brufau et al. 2004; Delis et al. 2011; Cea and Vazquez-Cendon 2012). Here the friction terms are treated semi-implicitly following Xilin and Liang (2018), where in the x-direction for a cell p they are expressed as

$$hu_p^{t*} = hu_p^t + \Delta t \mathbf{M}_{xp}^t - \Delta t g \mathbf{N}_{xp}^t \tag{34}$$

where t = current time level; t^* = calculation level of Runge-Kutta second-order (RKSO) method; and Δt = time step. $\mathbf{M}_{xp}^t$ and $\mathbf{N}_{xp}^t$ are written

$$\begin{aligned} \mathbf{M}_{xp}^t &= -\frac{1}{A_p}\left[\sum_{k=1}^{4} ((C_x - D_x - S_{b_x})^t n_x)_k \Delta L_k\right] \\ \mathbf{N}_{xp}^t &= \left[n_m^2 (h^t)^{-7/3} hu^{t*} \sqrt{(hu^{t*})^2 + (hv^{t*})^2}\right]_p \end{aligned} \tag{35}$$

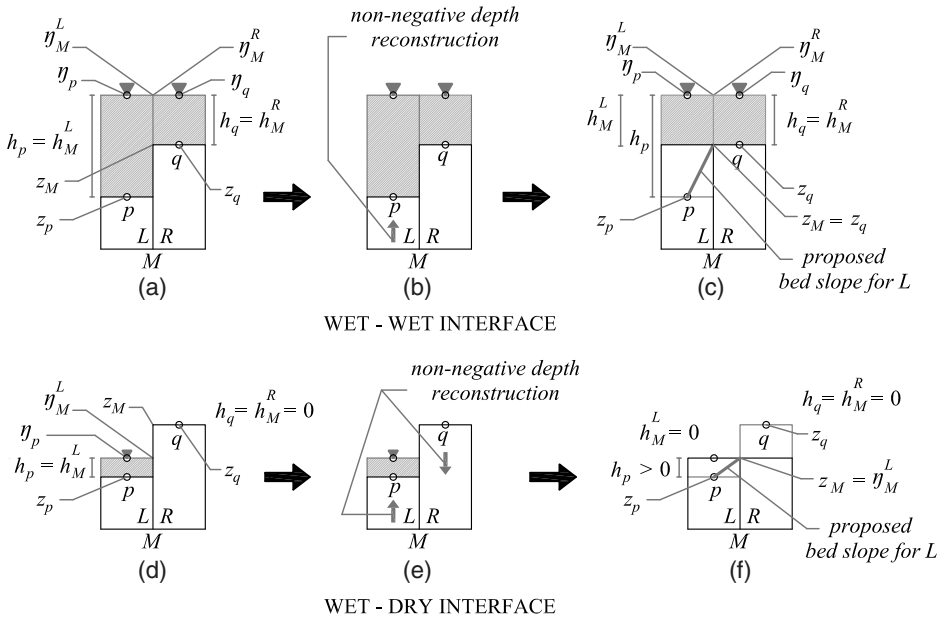


Fig. 5. Procedures of nonnegative depth reconstruction and bed-slope term calculation for wet–wet and wet–dry interfaces.

where A_p = area of cell p. Similarly, $\mathbf{M}_{yp}^t$ and $\mathbf{N}_{yp}^t$ can easily be computed. Both $\mathbf{N}_{xp}^t$ and $\mathbf{N}_{yp}^t$ are the nonlinear functions of hu_p^{t*} and hv_p^{t*}. The effective methods to find the roots of such functions were described well in Xilin and Liang (2018) and so are not presented here.

Turbulence Source Term Calculations

The turbulent source terms are calculated much like in Cea et al. (2007) and Davidson (1993). $D_{\Phi,x}$ and $D_{\Phi,y}$ are treated semi-implicitly as source terms and computed together with $S_{\kappa-\epsilon}$, which is calculated explicitly. Here some new symbols are used according to Eqs. (4) and (5):

$$\mathbf{H}_1 = \begin{bmatrix} \oint_\Gamma \left[\sigma_\kappa^{-1} h\nu_t \left(\frac{\partial \kappa}{\partial x} + \frac{\partial \kappa}{\partial y}\right)\right] \cdot \vec{n} d\Gamma \\ \oint_\Gamma \left[\sigma_\epsilon^{-1} h\nu_t \left(\frac{\partial \epsilon}{\partial x} + \frac{\partial \epsilon}{\partial y}\right)\right] \cdot \vec{n} d\Gamma \end{bmatrix}, \quad \mathbf{H}_2 = \begin{bmatrix} P_h^* \\ c_{\epsilon 1} \dfrac{\epsilon}{\kappa} P_h^* \end{bmatrix}$$

$$\mathbf{H}_3 = \begin{bmatrix} P_{\kappa b} \\ P_{\epsilon b} \end{bmatrix}, \quad \mathbf{H}_4 = \begin{bmatrix} -\dfrac{\epsilon}{\kappa} \\ -c_{\epsilon 2} \dfrac{\epsilon}{\kappa} \end{bmatrix} \tag{36}$$

The total of $D_{\Phi,x}$, $D_{\Phi,y}$, and $S_{\kappa-\epsilon}$ at cell p is now denoted by $\mathbf{H}_p$ and is computed semi-implicitly using

$$\mathbf{H}_p^t = \left[\max\left(\frac{\mathbf{H}_1^t}{A_p}, 0\right) + \mathbf{H}_2^t + \mathbf{H}_3^t\right] + \Phi^{t*}\left[\min\left(\frac{\mathbf{H}_1^t}{(A_p \Phi^t)}, 0\right) + \mathbf{H}_4^t\right] \tag{37}$$

Some Limitations for Turbulence Properties

To ensure computational stability, the turbulence properties must be limited so that negative values of the Reynolds stresses can be avoided. These limitations include the limiter values for both P_h and $P_{\kappa b}$—and a realizability condition for ν_t. For the details, see Cea et al. (2007), Cea (2005), and Durbin (1996).

Temporal Discretization

Following Liang and Borthwick (2009), the RKSO method is written for the 2D DA-RANS equations as

$$Q_p^{t+1} = \tfrac{1}{2}(Q_p^t + Q_p^{t*} + \mathbf{K}(Q_p^{t*})) \tag{38}$$

where $Q_p^{t*} = Q_p^t + \mathbf{K}(Q_p^t)$ and $\mathbf{K}(Q_p^t) = \Delta t \mathbf{M}(Q_p)^t - \Delta t g \mathbf{N}(Q_p)^t$, with $\mathbf{M} = [\mathbf{M}_x, \mathbf{M}_y]^{\mathrm{T}}$ and $\mathbf{N} = [\mathbf{N}_x, \mathbf{N}_y]^{\mathrm{T}}$. The κ–ϵ model is calculated as

$$\Phi_p^{t+1} = \tfrac{1}{2}(\Phi_p^t + \Phi_p^{t*} + \mathbf{K}_\Phi(\Phi_p^{t*})) \tag{39}$$

where $\Phi_p^{t*} = \Phi_p^t + \mathbf{K}_\Phi(\Phi_p^t)$ and $\mathbf{K}_\Phi(\Phi_p^t) = -\frac{\Delta t}{A_p}[\sum_{k=1}^4 (C_{\Phi,x} n_x + C_{\Phi,y} n_y)_k^t \Delta L_k] + \Delta t \mathbf{H}_p^t$. Since $\mathbf{H}_p^t$ includes Φ^{t*} in Eq. (37), one must first calculate Φ by summing all corresponding fluxes (the first term of $\mathbf{K}_\Phi$) without including any turbulent source term. Afterwards, Eq. (37) is used to recalculate $\mathbf{H}$. Since the CU scheme is used to solve Eq. (39), the hybrid AV-CU scheme follows the Courant-Friedrichs-Lewy (CFL) criteria in Kurganov and Petrova (2007). Also, the Peclet (Pe) number $\mathrm{Pe} \leq 2/\mathrm{CFL}$ is considered (Hirsch 2007).

Boundary Conditions

Treatments similar to those of Ginting (2017) are employed for inflow/outflow boundaries and the AV technique. For the turbulence model, treatments similar to those in Cea (2005) are applied.

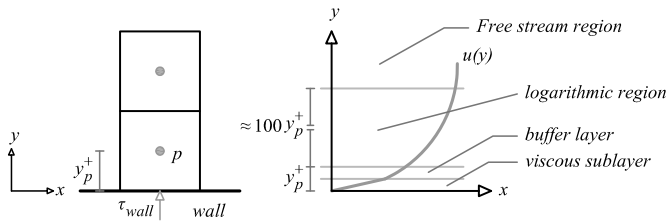


Fig. 6. Near-wall mesh and scaling of turbulent boundary layer: τ_{wall} is wall shear stress.

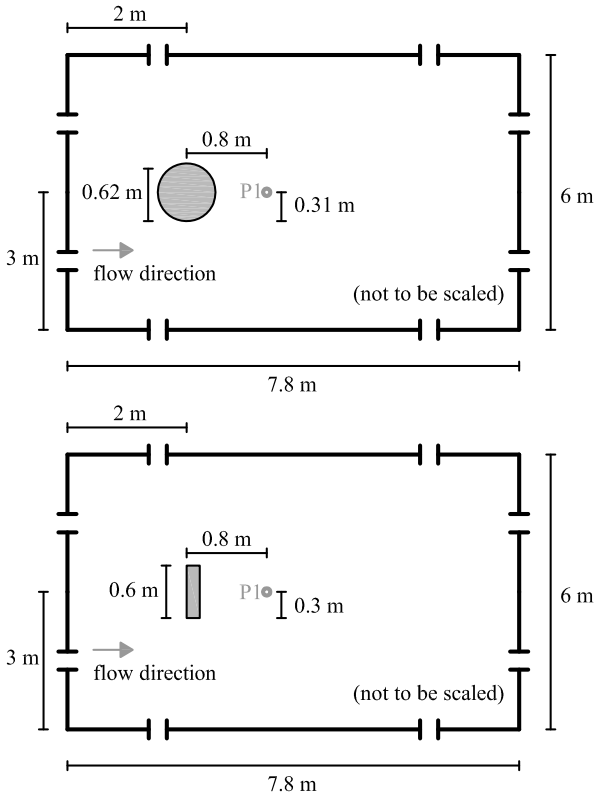


Fig. 7. Case 1: Domain of circular and square cylinders.

No flux through a wall is allowed, and the turbulence values at the wall are computed using the scalable wall function (ScWF) as (Menter and Esch 2001)

$$w_p^{tg} = \frac{w_p^*}{\upsilon}\ln(y_p^+ E), \qquad y_p^+ = \max\left(11.067, \frac{\Delta y_p^{nor} w_p^*}{\nu_e}\right) \tag{40}$$

where υ = von Karman coefficient (0.41); w_p^{tg} = velocity component of cell p parallel to wall; w_p^* = wall friction velocity; y_p^+ = wall distance unit; Δy_p^{nor} = normal distance from cell center p to wall; and E = roughness parameter [here $E = 8.432$ for smooth wall; for another formula see Duan (2004)]. The Newton-Raphson iteration is used to solve Eq. (40) and expressed in Algorithm 1. w_p^* is then used to calculate κ and ϵ at wall boundary cells for the subsequent time level without solving the κ–ϵ model.

Algorithm 1. Algorithm for calculating w_p^*

1: **if** $\left[w_p^{tg} \le 11.067\dfrac{\ln(11.067E)\nu_e}{\Delta y_p^{nor}\upsilon}\right]$ **then**

2: $\quad w_p^* = \dfrac{\upsilon w_p^{tg}}{\ln(11.067E)}$

3: **else**

4: $\quad$ solve the Newton-Raphson iteration as follows:

5: $\quad w_p^{*\text{trial1}} = w_p^{*\text{trial0}} - \dfrac{w_p^{*\text{trial0}}\ln\left(\frac{\Delta y_p^{nor}E}{\nu_e}w_p^{*\text{trial0}}\right) - \upsilon w_p^{tg}}{\ln\left(\frac{\Delta y_p^{nor}E}{\nu_e}w_p^{*\text{trial0}}\right)-1}$,

6: $\quad w_p^{*\text{trial2}} = w_p^{*\text{trial1}} - \dfrac{w_p^{*\text{trial1}}\ln\left(\frac{\Delta y_p^{nor}E}{\nu_e}w_p^{*\text{trial1}}\right) - \upsilon w_p^{tg}}{\ln\left(\frac{\Delta y_p^{nor}E}{\nu_e}w_p^{*\text{trial1}}\right)-1}$,

7: $\quad w_p^{*\text{trial3}} = \ldots$

8: **end if**

The difference between the ScWF and standard wall function (StWF) lies in the computation of y^+. Typically, $y^+ \ge 11.067$ is applied in the StWF for near-wall meshes (Wu 2004; Yu and Duan 2012), and one must consequently place the first boundary cells in the logarithmic region (Fig. 6). Although using fine meshes increases the accuracy of RANS models (Coroneo et al. 2011), such meshes may contravene this criterion. A grid study and the enhanced wall treatments were performed in Karimi et al. (2012) to determine the proper size of near-wall meshes. Using the ScWF, such grid studies are not required even for high-resolution meshes, giving users flexibility to generate meshes without imposing a

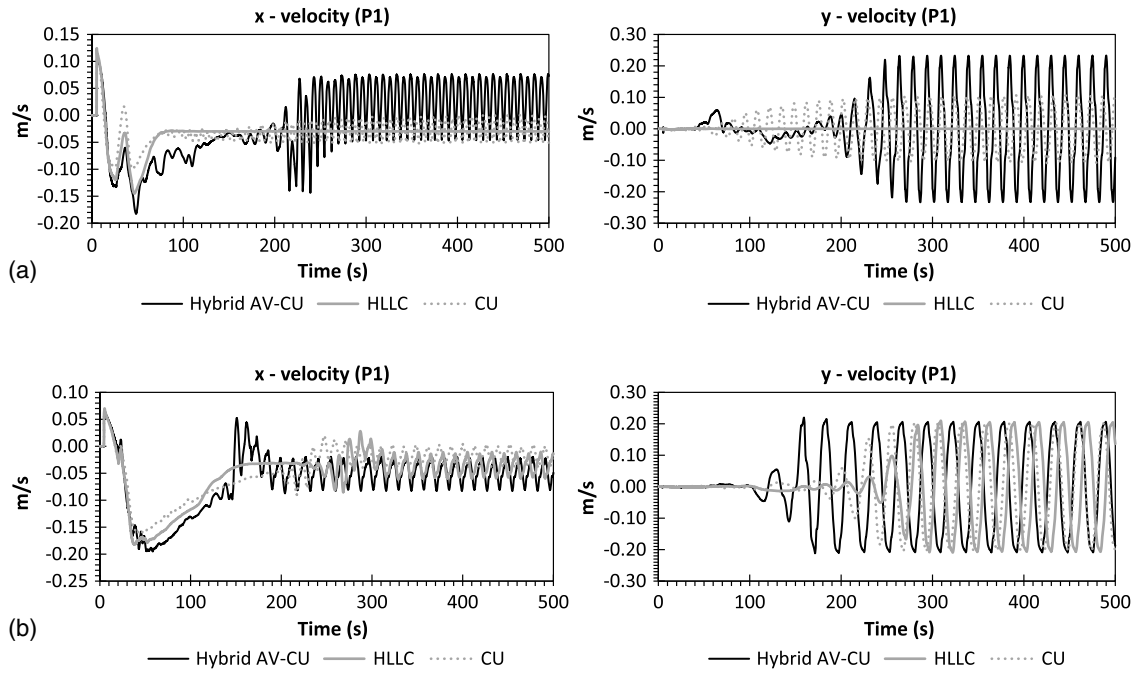


Fig. 8. Case 1: Convergence histories at P1 for (a) circular; and (b) square cylinders.

J. Hydraul. Eng., 2019, 145(12): 04019041

lower limit from the StWF (Ginting 2019). Algorithm 1 shows that the ScWF establish a linear relationship between w_p^{tg} and w_p^*. This becomes a shortcoming, where the displacement effect of the viscous sublayer may no longer be considered. Nevertheless, this approach is still acceptable because no wall formulation can accurately resolve the sublayer (Menter and Esch 2001).

Results and Discussion

NUFSAW2D, written in Fortran, was compiled using Intel Fortran 17 for Linux. The target machine is a Linux cluster "CoolMUC2" provided by the Leibniz Supercomputing Centre (Garching bei München, Germany). OpenMP was used for parallelization on 28 cores with "–O3–xHost" optimization and double-precision arithmetic (64-bit). $\nu_e = 10^{-6}\,\mathrm{m^2\,s^{-1}}$, $\rho = 998.2\,\mathrm{kg\,m^{-\frac{1}{3}}}$, and $g = 9.81\,\mathrm{m\,s^{-2}}$ were applied. All the turbulence coefficients were set similarly to those given in Eq. (7), where $c_{\epsilon\Gamma} = 3.6$. $\kappa^{(2)} = 5$, $\kappa^{(4)} = 1/32$, and $\omega = 0.5$ were used. A performance metric (PM) of Mcell/s/core (million cells per second per core), being a comparison between the number of cells for a total number of calculation steps that can be processed per unit of time using one core, is used for a performance comparison among all schemes.

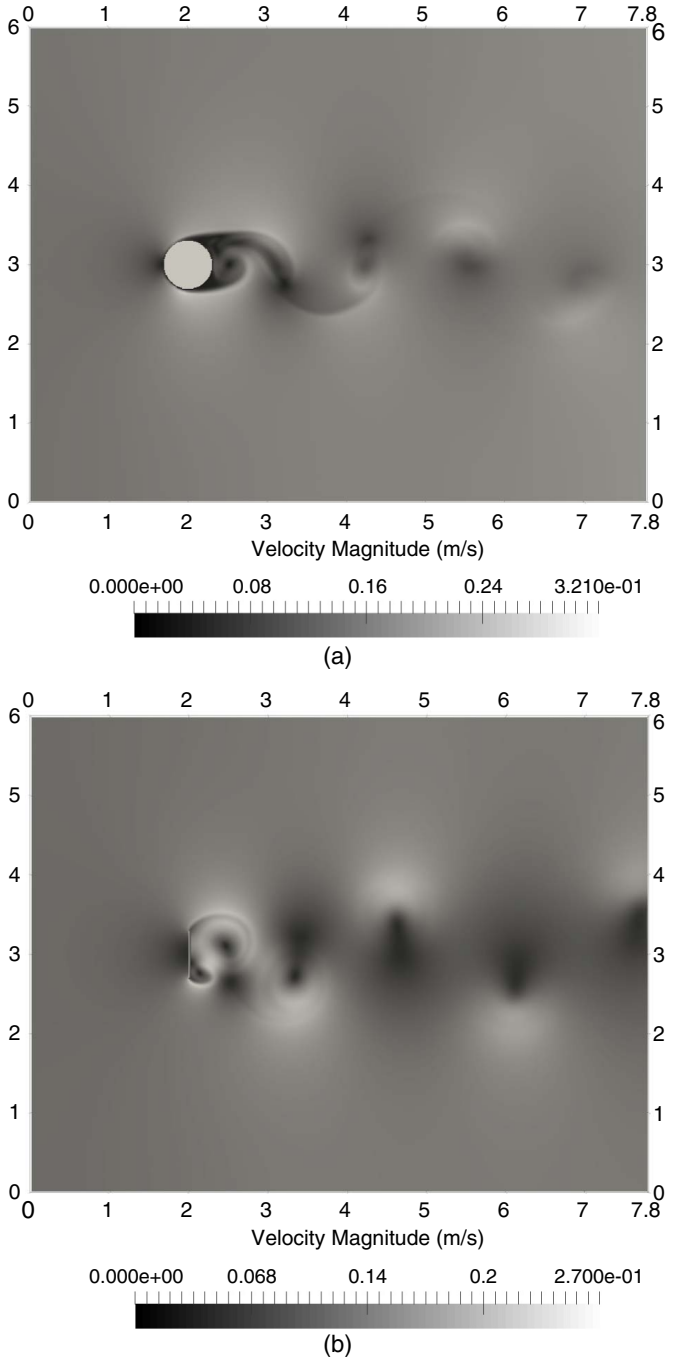


Fig. 9. Case 1: Visualization of wake flows along channel at (a) 488.5 s; and (b) 481 s.

Case 1: Flow Past Circular and Square Cylinders

This case is adopted from Chen and Jirka (1995) (Fig. 7). A depth of 0.02 m and a velocity of 0.295 m/s were constant at upstream for the circular cylinder, whereas 0.035 m and 0.157 m/s were given for the square one. Free outflow was set downstream, where $n_m = 0.01\ \mathrm{s\,m^{-\frac{1}{3}}}$. The Reynolds numbers (R) were R = 5,900 (circular) and R = 5,500 (square). No measurement data were given at a specific point. However, the Strouhal numbers (S) were presented experimentally as an indicator for the vortex-shedding frequency: S = 0.215 (circular) and S = 0.17 (square). Points P1 are used only to compare the numerical results.

NUFSAW2D was run for 500 s with a mesh size of 0.01 m. Fig. 8 shows that both the hybrid AV-CU and CU schemes achieve a similar convergence rate at P1 for the circular case approximately after 250 s; the latter, however, computes the lower magnitudes. Interestingly, the HLLC scheme fails to capture the recirculating flow, shown by the constant u while $v = 0$. For the square case, all schemes can detect the recirculating flows but with different convergence rates. The CU, HLLC, and hybrid AV-CU schemes achieve convergence after approximately 240, 270, and 180 s, respectively, showing the proposed scheme becomes the fastest one. To approximate S, the vortex-shedding frequency must be known. This was done in Chen and Jirka (1995) by visualizing the oscillation periods of the experimental data averaged over 10 cycles. Therefore, the visualizations are observed here for 20 s. For brevity, only the visualizations of the hybrid AV-CU scheme are presented in Figs. 9 and 10. For the circular case, the hybrid AV-CU scheme calculates a period of 9.3 s (488.5–497.8 s), whereas the CU scheme computes 8.5 s, giving S = 0.226 and 0.247, respectively. This shows that the latter is less accurate. For the square case, the hybrid AV-CU, CU, and HLLC schemes produce periods of 19.8 (479.4–499.2 s), 18.1, and 17.3 s, giving S = 0.193, 0.211, and 0.221, respectively, showing that the proposed scheme is better than the others. The CU, HLLC, and hybrid AV-CU schemes achieve PMs of 2.5, 1.61, and 2.45 Mcell/s/core, respectively.

Case 2: Recirculating Wakes around a Submerged Conical Island

This case (Fig. 11) was investigated experimentally and 2D/3D numerically with 0.0152-m meshes in Lloyd and Stansby (1997) (L–S). A depth of 0.054 m and a velocity of 0.115 m/s were constant at the upstream and free outflow at the downstream, giving R = 6,210, where $n_m = 0.01\ \mathrm{s\,m^{-\frac{1}{3}}}$. Some publications are also noted: 2D Boussinesq model with 0.01-m meshes (Kim et al. 2009), several 2D/3D models, e.g., Funwave-TVD (fully nonlinear Boussinesq wave–total variation diminishing), GeoClaw (Geo Conservation Laws), BOSZ (Boussinesq ocean and surf zone), with 0.01- to 0.03-m meshes (NTHMP 2018), and 2D/3D hydrostatic RANS models, e.g., SCHISM-2D/3D (Semi-implicit Cross-scale Hydroscience Integrated System Model-2D/3D), with 0.012-m

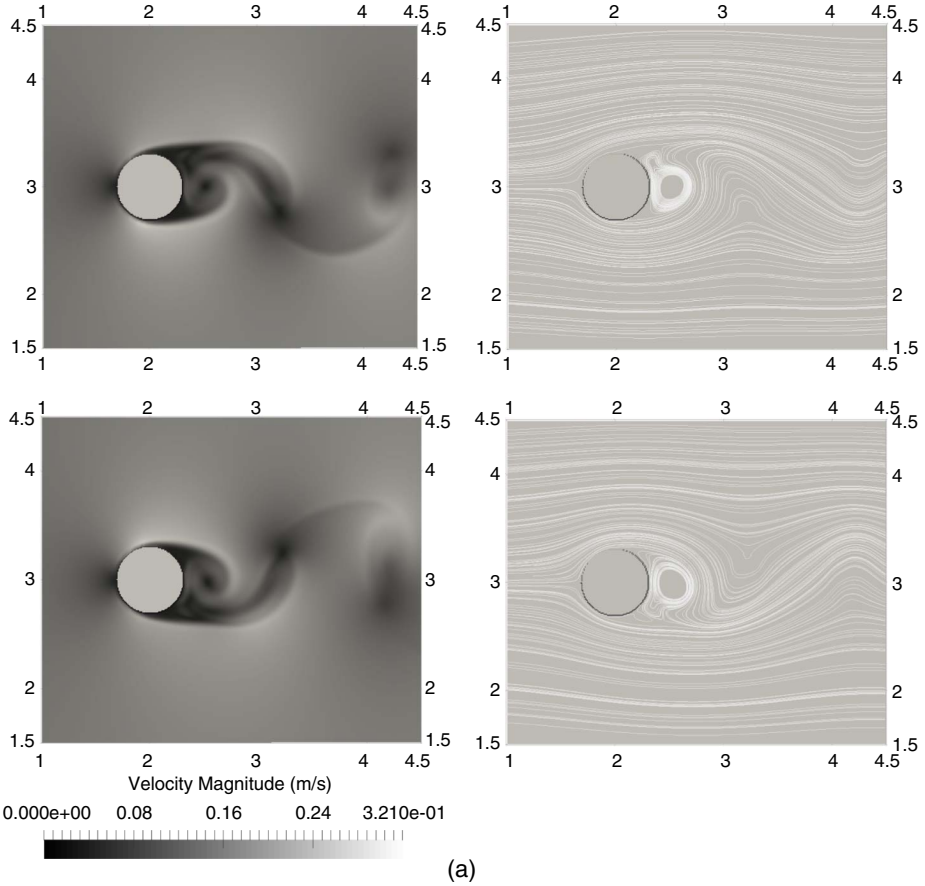


Fig. 10. Case 1: Visualization of wake flows near cylinders at (a) 488.5–497.8 s; and (b) 479.4–499.2 s.

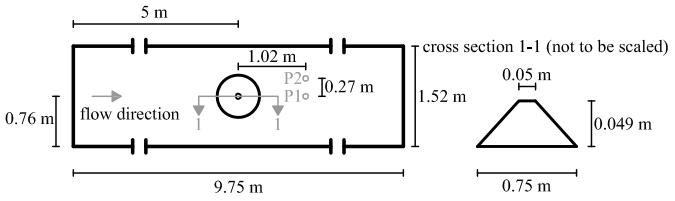


Fig. 11. Case 2: Domain of conical island.

meshes (Zhang et al. 2016). Since 2D simulation is of interest, only 2D models of L–S and SCHISM-2D are considered here for comparison.

NUFSAW2D was run with a 0.015-m mesh size for 500 s (although the convergence was achieved at around 100 s) to show that the results are not chaotic and do not deteriorate using the ScWF. The results obtained during the last 100 s are thus compared (after slightly shifting the time series due to different transient phases). Surprisingly, the CU scheme captures no recirculation

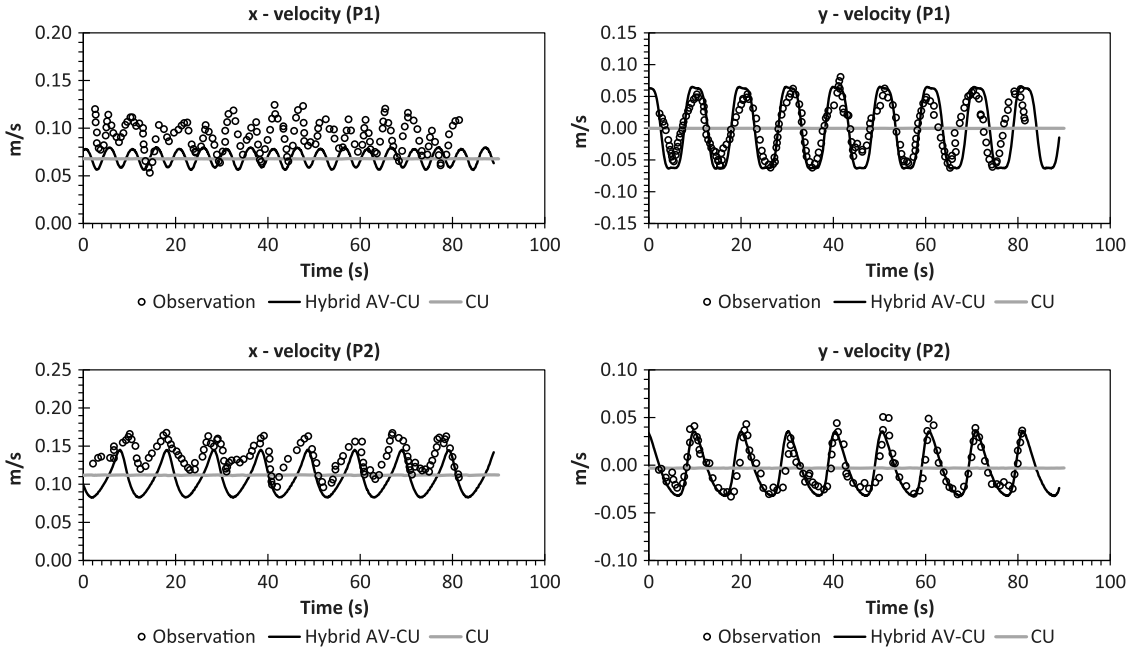


Fig. 12. Case 2: Comparison at P1 and P2 between hybrid AV-CU and CU schemes.

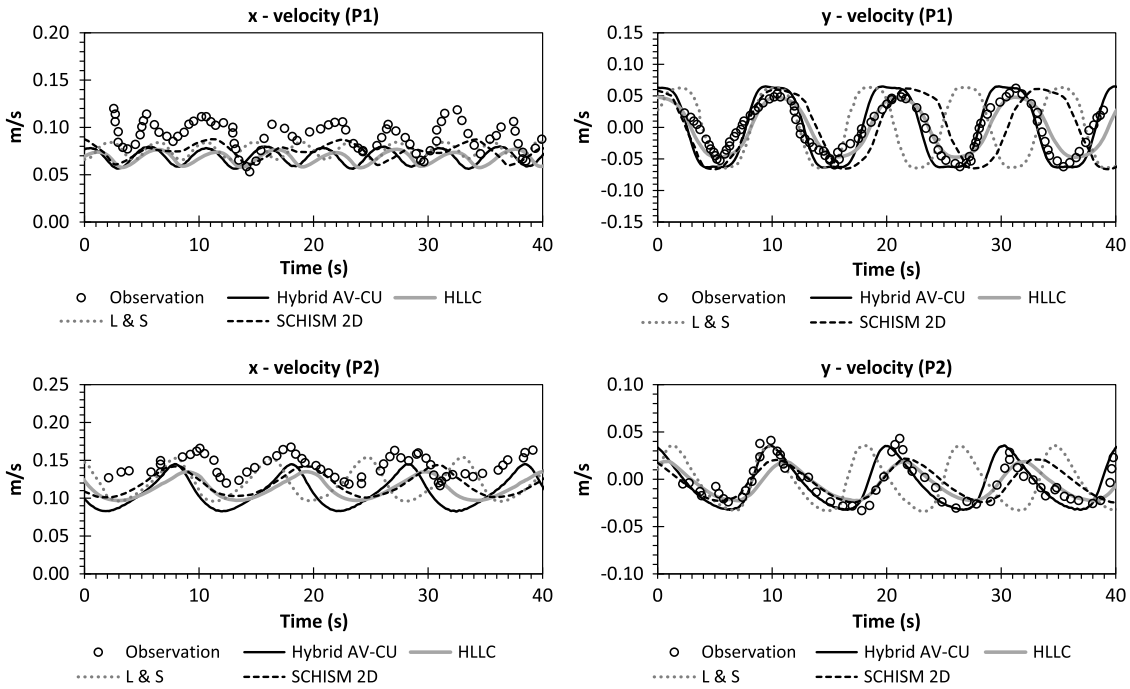


Fig. 13. Case 2: Comparison at P1 and P2 between some models.

(Fig. 12) by only detecting $u = 0.07$ m/s at P1 and $u = 0.11$ m/s at P2, while $v = 0$. The hybrid AV-CU scheme yields sufficiently accurate results for both wakes' magnitudes and periods; however, u at P1 is underestimated. In the experiment, a strong vertical mixing at P1 was observed, so 3D properties existed and 2D assumptions failed. Nevertheless, this hybrid approach successfully shows the recirculation at P1 and P2, where the CU scheme cannot. Note that no 2D model from the aforementioned publications could properly simulate the magnitudes or the wakes' periods at P1.

Fig. 13 shows comparisons only for 40 s to give a clear representation. The HLLC scheme computes the wake periods properly but still underestimates the magnitudes. Both L–S and SCHISM-2D predict the magnitudes properly but significantly overestimate the periods after 15 s. This shows that the hybrid AV-CU scheme becomes the most accurate one. The HLLC and hybrid AV-CU schemes achieve PMs of 1.62 and 2.46 Mcell/s/core, respectively. Figs. 14 and 15 visualize the recirculating near-wake bubble flows behind the conical island produced by the hybrid AV-CU scheme, representing good qualitative agreement with the experimental visualizations. Fig. 16 shows the stable wake patterns after 100 s computed by the hybrid AV-CU and HLLC schemes, giving similar convergence rates.

Case 3: Turbulent Recirculating Flows due to Channel Expansion

The case in Fig. 17 is adopted from Han (2015) and Han et al. (2017), who experimentally and numerically with Simulation of Turbulent Flow in Arbitrary Regions–Computational Continuum Mechanics+ (StarCCM+) investigated the recirculation zone length (L_{circ}) at a sudden expansion in a PVC-made channel with a streamwise slope of 0.18%. A constant discharge of 20 L/s was set at the upstream and free outflow at the downstream. A roughness value of 3.4×10^{-5} m was given and converted using the Colebrook–White equation and the Darcy–Weisbach coefficient and assumed to be constant, resulting in $n_m = 0.0094$ s m$^{-\frac{1}{3}}$. During the experiment, the recirculation zone was unsteady, so L_{circ} always changed with respect to time. On average, $L_{circ} = 1.29$ m with the recirculation center at (0.735, 0.104) m, and $h = 0.156$ m, $u = 0.23$ m/s, $v = 0$ were observed at P1. A grid study was done using StarCCM+ with 12.5-, 6.25-, and 3.125-mm mesh sizes.

To approximate a proper steady-state condition for the numerical simulation, two criteria are used here: (1) convergence history for h and u at P1 and (2) the slight difference between the inflow and outflow discharges. A 10-mm mesh size is employed for 400-s simulation, and with the ScWF no grid study is required. The convergence is achieved after 200 s (Fig. 18) (where the convergence rates are almost similar), and an error on the order of 10^{-4} m^3/s at 400 s is obtained, so comparing L_{circ} at 400 s is acceptable. The results for u along Section 1-1 (at the centers of the first boundary cells) are plotted; as u changes from positive to negative, the reattachment point can be determined. Fig. 19 shows the CU, HLLC, and hybrid AV-CU schemes calculate $L_{circ} = 0.8, 0.81$, and 1.28 m, respectively. This shows that the hybrid AV-CU scheme yields sufficiently accurate results, whereas the others are more diffusive. Along $x = 0.675$ m and $x = 1.32$ m, the hybrid AV-CU scheme becomes again the most accurate one near the wall ($y \leq 0.2$ m). At $y > 0.2$ m, all schemes, however, exhibit some errors, but they are not significant.

Fig. 20 shows that mixing layers exist along the separating streamlines. The CU and HLLC schemes exhibit almost similar characteristics, where the separating streamlines end approximately at (0.85, 0) m, but the recirculation centers slightly differ, (0.41, 0.12) m and (0.3, 0.12) m, respectively. The separating streamline

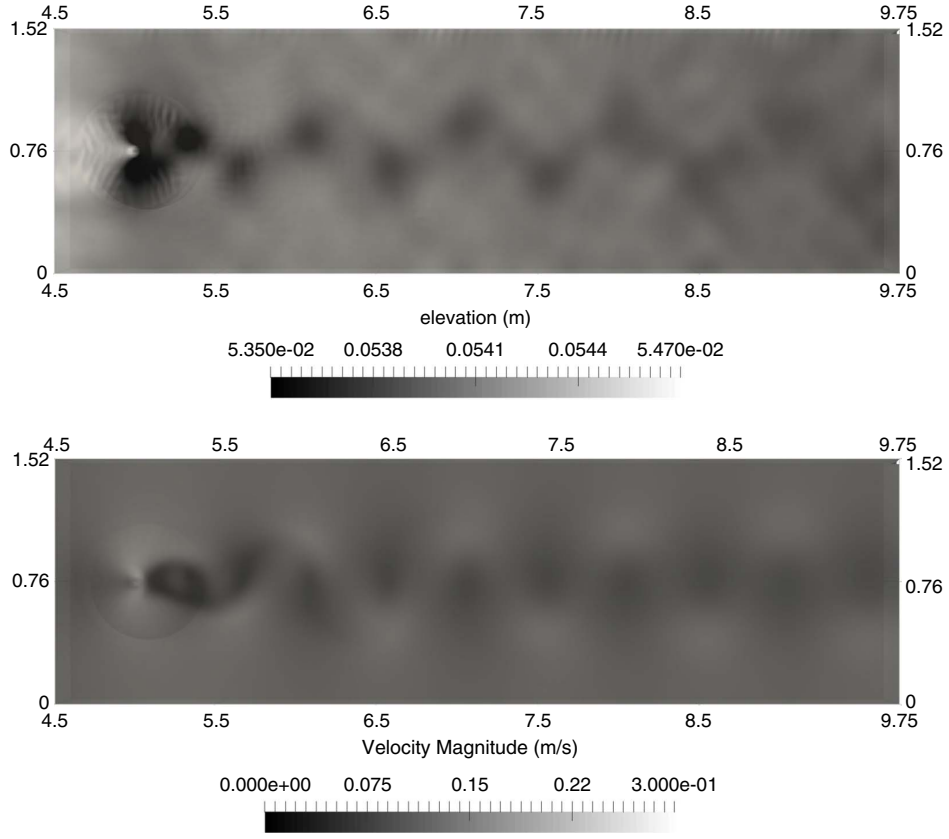


Fig. 14. Case 2: Visualization of wake flows along channel at 40 s.

J. Hydraul. Eng., 2019, 145(12): 04019041

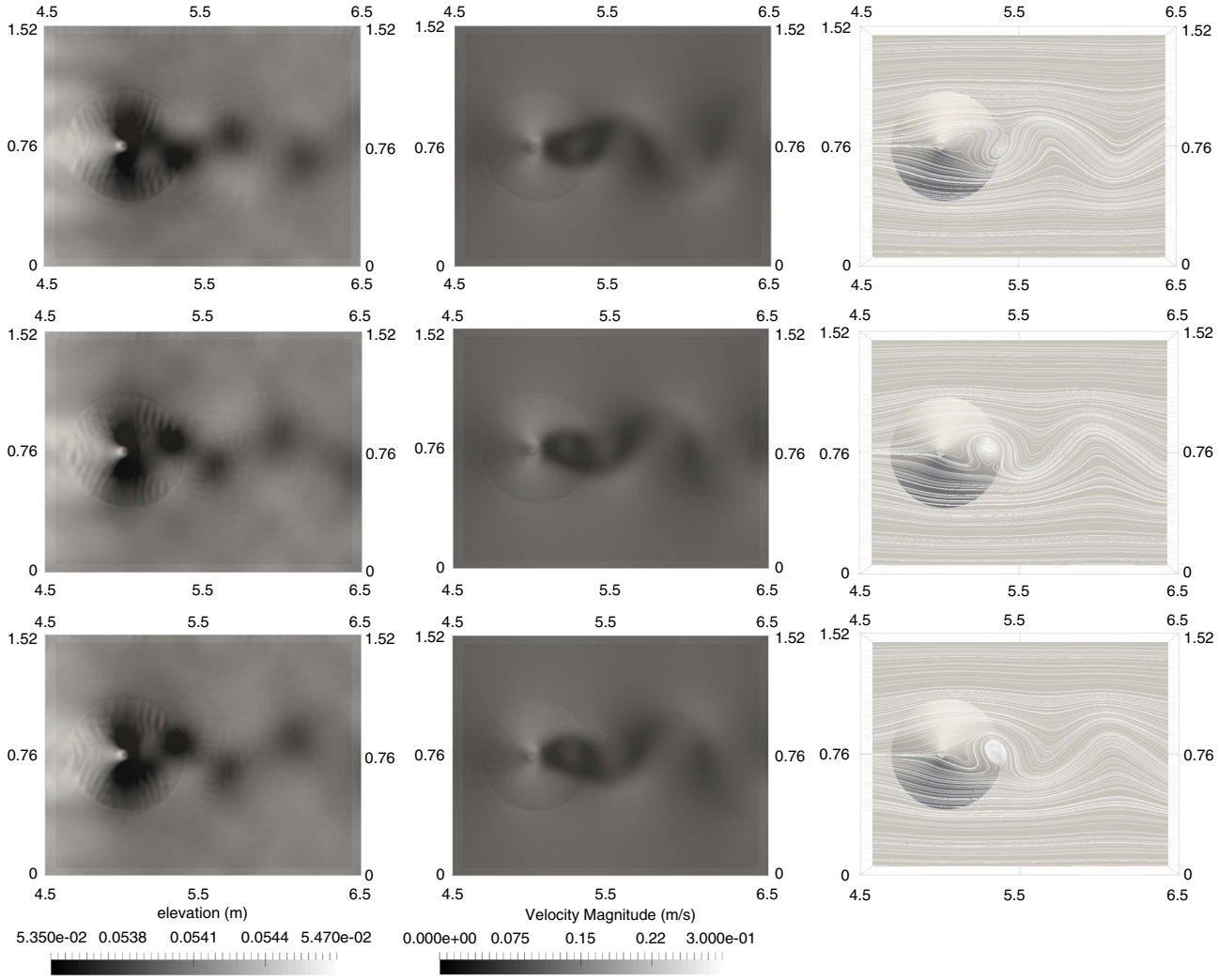


Fig. 15. Case 2: Visualization of wake flows near conical island at 35, 38, and 40 s.

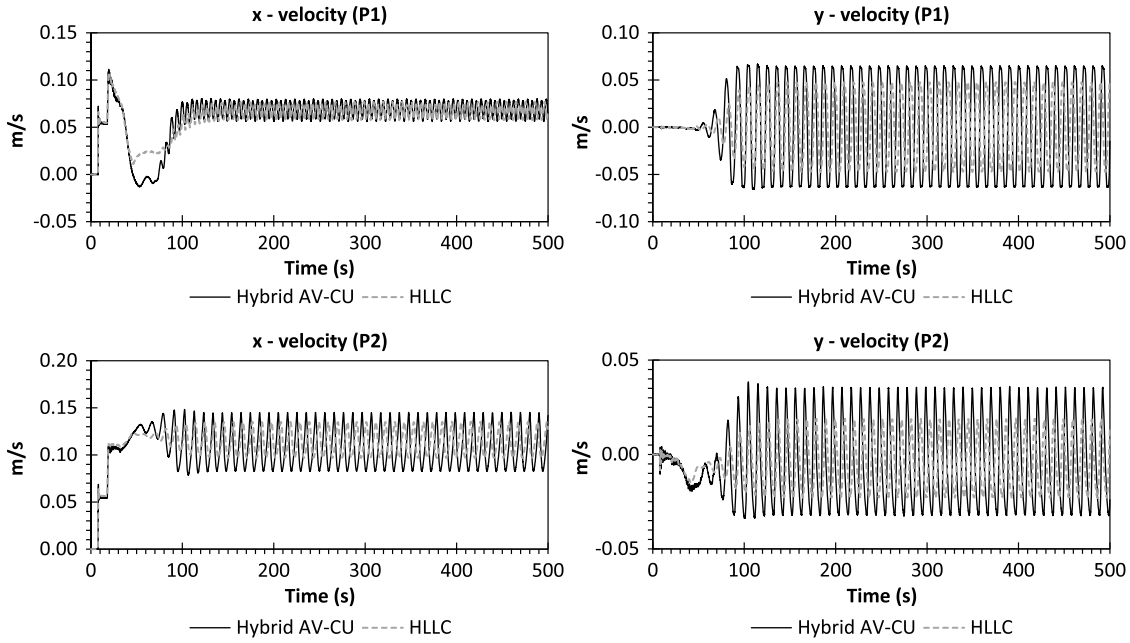


Fig. 16. Case 2: Convergence histories at P1 and P2.

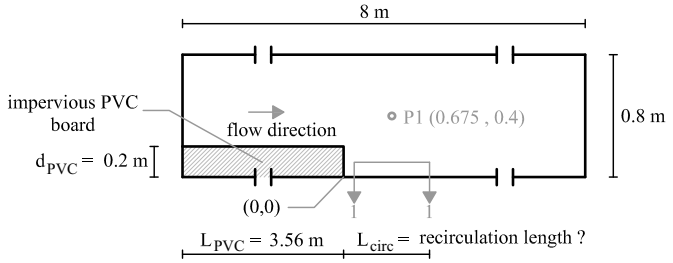


Fig. 17. Case 3: Domain of channel.

of the hybrid AV-CU scheme ends approximately at (1.31, 0) m with the recirculation center of (0.63, 0.11) m. This shows that the hybrid AV-CU scheme is more accurate and in accordance with the experimental data. The CU, HLLC, and hybrid AV-CU schemes achieve PMs of 2.51, 1.61, and 2.46 Mcell/s/core, respectively.

Conclusion

Simulations of recirculating turbulent shallow flows were presented using a hybrid AV-CU scheme for solving convective fluxes of 2D

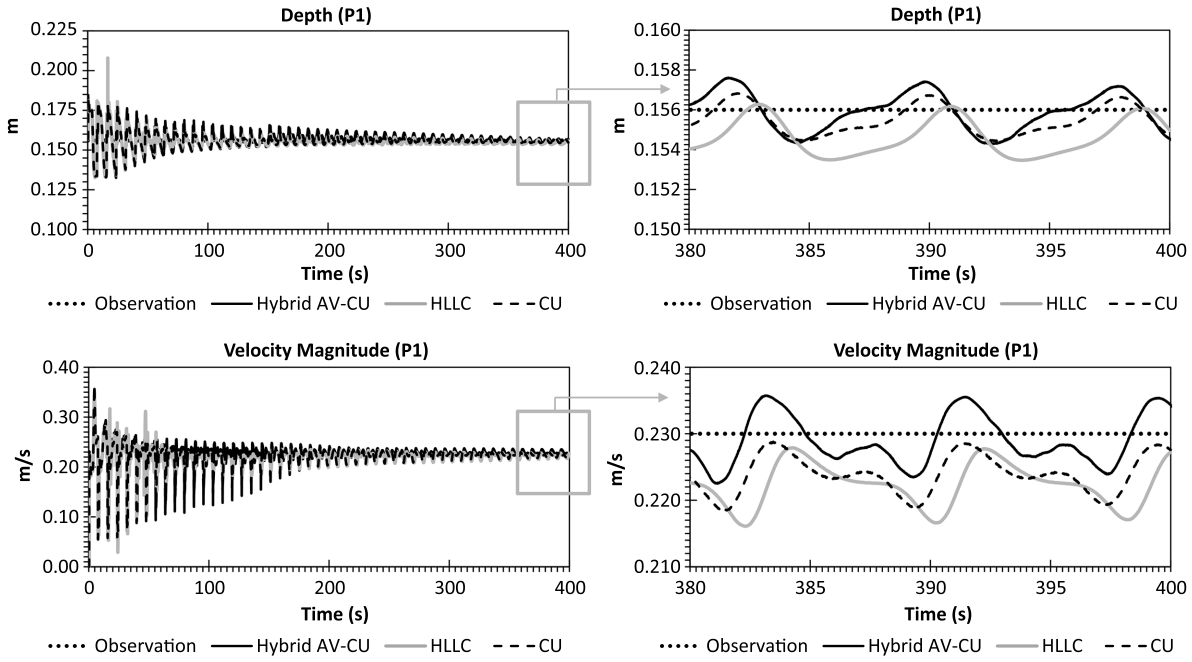


Fig. 18. Case 3: Convergence histories at P1.

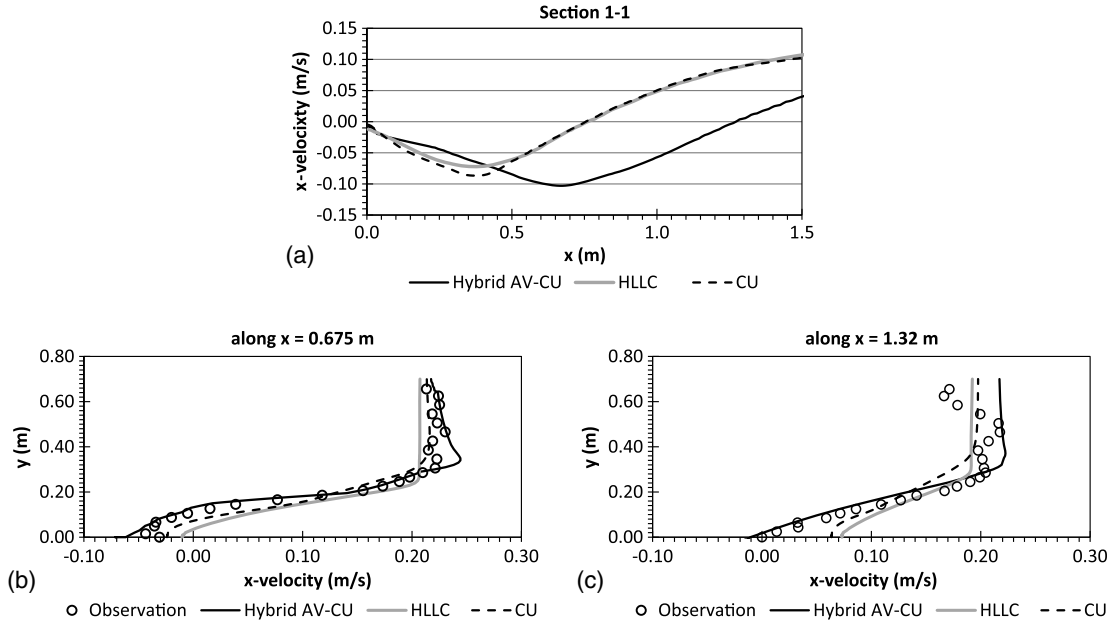


Fig. 19. Case 3: Comparisons along (a) Section 1-1; (b) $x = 0.675$ m; and (c) $x = 1.32$ m.

J. Hydraul. Eng., 2019, 145(12): 04019041

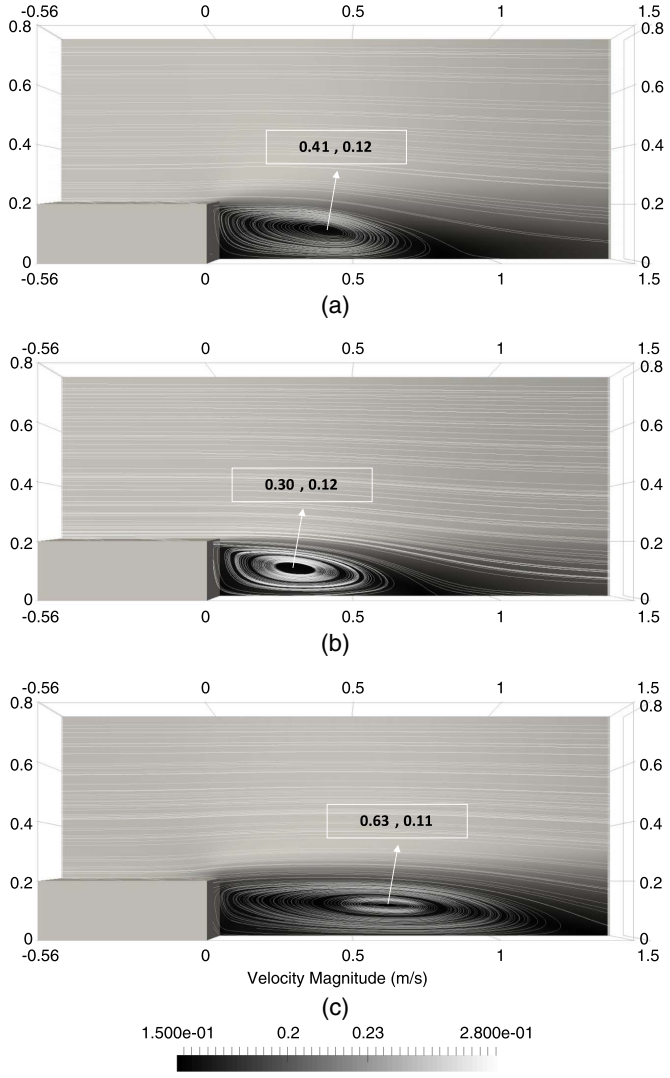


Fig. 20. Case 3: Visualization of velocity magnitude near recirculation zone at 200 s of (a) CU; (b) HLLC; and (c) hybrid AV-CU schemes.

DA-RANS and κ–ϵ models. Second-order spatial and temporal accuracies were achieved using a MUSCL scheme with a MinMod limiter and a RKSO scheme, respectively. ScWFs were employed, so no grid study was required, and additionally this gave users flexibility to generate meshes. The proposed scheme proved to be well balanced.

Using similar mesh sizes, the hybrid AV-CU scheme was significantly more accurate than the other models (CU, HLLC, L–S, and SCHISM-2D). The proposed scheme could resolve the inability of the CU scheme to simulate recirculating turbulent shallow flows. It can significantly improve results since the biharmonic operator $\Delta^4 Q$ is third-order accurate and activated in smooth regions of the flow field, as in Case 2. Note that the AV technique combines the Laplacian operator (first-order accurate) to work for discontinuities and the biharmonic operator to work for smooth flows. These operators are switched on/off automatically by the DDD sensor during runtime.

The hybrid AV-CU scheme remained as efficient as the CU scheme but was 1.52× cheaper than the HLLC solver. Regarding its accuracy and performance, the proposed scheme could be a promising method for practical engineering purposes, especially when it comes to simulating recirculating turbulent shallow flows. Finally, a study of the new dimensionless discontinuity sensors for the turbulence properties according to the authors' hypothesis would be worth exploring in the future.

Appendix. Proof of Well-Balanced Property

For simplicity, all analyses are given for one dimension here; a similar approach can easily be extended to two dimensions. The steady-state condition is in general expressed as

$$\frac{\partial hu}{\partial x} = 0$$
$$\frac{\partial}{\partial x}\left(\frac{(hu)^2}{h} + \frac{1}{2}gh^2\right) = \frac{\partial D_x}{\partial x} - gh\frac{\partial z}{\partial x} - gn_m^2 h^{-\frac{1}{3}}|u|u \quad (41)$$

where $D_x = 2h(\nu_e + \nu_t)\partial u/\partial x - 2/3h\kappa$. The first term in Eq. (41) indicates $hu = hu_0 = \text{constant}$, so

$$\frac{\partial}{\partial x}\left(\frac{(hu_0)^2}{h} + \frac{1}{2}gh^2\right) = \frac{\partial D_x}{\partial x} - gh\frac{\partial z}{\partial x} - gn_m^2 h^{-\frac{1}{3}}|u|u \quad (42)$$

Two conditions are considered: a lake-at-rest condition (Audusse et al. 2004; Hou et al. 2013a), expressed as $u = 0$ and $h + z = \text{constant}$, and a smooth steady-state solution (Chertock et al. 2015; Michel-Dansac et al. 2016), which follows $h = h_0 = \text{constant}$, $hu = hu_0 = \text{constant}$, and $\partial z/\partial x = -z_0 = \text{constant}$.

Lake-at-Rest Condition

Applying Eq. (11), η and h are reconstructed for the edge $i + \frac{1}{2}$ as

$$\eta^L_{i+\frac{1}{2}} = \eta_i + \frac{1}{2}\theta^\eta_i(\eta_{i+1} - \eta_i),$$
$$\eta^R_{i+\frac{1}{2}} = \eta_{i+1} - \frac{1}{2}\theta^\eta_{i+1}(\eta_{i+1} - \eta_i)$$
$$h^L_{i+\frac{1}{2}} = h_i + \frac{1}{2}\theta^h_i(h_{i+1} - h_i),$$
$$h^R_{i+\frac{1}{2}} = h_{i+1} - \frac{1}{2}\theta^h_{i+1}(h_{i+1} - h_i) \quad (43)$$

where θ^η and θ^h are the MinMod functions for η and h, respectively, satisfying $0 \le \theta^\eta \le 1$ and $0 \le \theta^h \le 1$. Considering Fig. 21(a) gives

$$\eta_{i-1} = \eta_i = \eta_{i+1} = \eta^L_{i-\frac{1}{2}} = \eta^R_{i-\frac{1}{2}} = \eta^L_{i+\frac{1}{2}} = \eta^R_{i+\frac{1}{2}} = \eta$$
$$h^L_{i+\frac{1}{2}} = h_i, h^R_{i+\frac{1}{2}} = h_{i+1}, h_i > h_{i+1} \quad (44)$$

Applying Eq. (14) gives

$$z^M_{i+\frac{1}{2}} = \max(\eta^L_{i+\frac{1}{2}} - h^L_{i+\frac{1}{2}}, \eta^R_{i+\frac{1}{2}} - h^R_{i+\frac{1}{2}})$$
$$= \max(\eta - h_i, \eta - h_{i+1}) = \eta - h_{i+1} = z_{i+1} \quad (45)$$

Eq. (15) is used to compute the nonnegative depths as

$$h^L_{i+\frac{1}{2}} = \max(\eta^L_{i+\frac{1}{2}} - z^M_{i+\frac{1}{2}}, 0) = \max(\eta - z_{i+1}, 0) = \eta - z_{i+1} = h_{i+1}$$
$$h^R_{i+\frac{1}{2}} = \max(\eta^R_{i+\frac{1}{2}} - z^M_{i+\frac{1}{2}}, 0) = \max(\eta - z_{i+1}, 0) = \eta - z_{i+1} = h_{i+1} \quad (46)$$

Since $u = 0$, thus no velocity fluctuation ($\kappa = 0$), the flux-velocity in the convective fluxes, the diffusive fluxes, the friction terms, and the artificial diffusive terms vanish, so

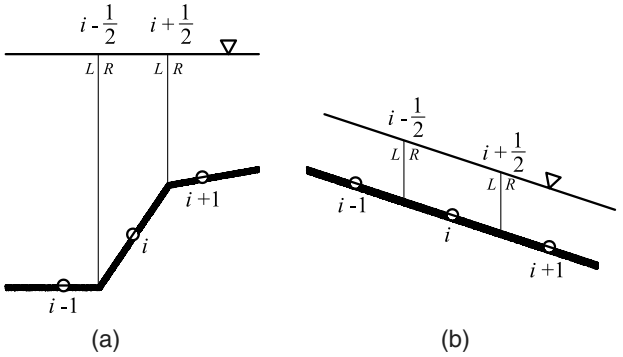


Fig. 21. (a) Lake-at-rest condition; and (b) smooth steady state.

$$[C_x n_x]_{i+\frac{1}{2}} = \frac{1}{2}[(C_x(Q^R) + C_x(Q^L))n_x)]_{i+\frac{1}{2}} = \frac{g}{2}\left[\frac{1}{2}(h^R_{i+\frac{1}{2}} + h^L_{i+\frac{1}{2}})\right]^2 = \frac{g}{2}h^2_{i+1} \tag{47}$$

Eq. (32) corrects the bed elevation by

$$\begin{aligned} z^M_{i+\frac{1}{2}} \leftarrow z^M_{i+\frac{1}{2}} - \max(0, z^M_{i+\frac{1}{2}} - \eta^L_{i+\frac{1}{2}}, z^M_{i+\frac{1}{2}} - \eta^R_M) \\ = z_{i+1} - \max(0, z_{i+1} - \eta, z_{i+1} - \eta) \\ = z_{i+1} - \max(0, -h_{i+1}, -h_{i+1}) = z_{i+1} \end{aligned} \tag{48}$$

The bed-slope fluxes are now computed based on Eq. (33) as

$$\begin{aligned} [S_{bx}n_x]_{i+\frac{1}{2}} &= -\frac{g}{2}\left[\left(\frac{h^L_{i+\frac{1}{2}} + h^R_{i+\frac{1}{2}}}{2} + h_i\right) z^M_{i+\frac{1}{2}} - \frac{h^L_{i+\frac{1}{2}} + h^R_{i+\frac{1}{2}}}{2} z_i\right] \\ &= -\frac{g}{2}[h_{i+1}(z_{i+1} - z_i) + h_i z_{i+1}] \end{aligned} \tag{49}$$

An approach similar to Eqs. (45)–(49) is applied to the edge $i - \frac{1}{2}$; thus,

$$[C_x n_x]_{i-\frac{1}{2}} = -\tfrac{g}{2}h^2_i, \qquad [S_{bx}n_x]_{i-\frac{1}{2}} = \tfrac{g}{2}h_i z_i \tag{50}$$

Now, showing that

$$\begin{aligned} hu^{t+1}_i &= hu^t_i - \frac{\Delta t}{\Delta x}[((C_x n_x)_{i+\frac{1}{2}} + (C_x n_x)_{i-\frac{1}{2}}) \\ &\quad - ((S_{bx}n_x)_{i+\frac{1}{2}} + (S_{bx}n_x)_{i-\frac{1}{2}})] \\ &= hu^t_i - \frac{\Delta t}{\Delta x}\frac{g}{2}[(h^2_{i+1} - h^2_i) \\ &\quad - (h_i z_i - h_{i+1}z_{i+1} - h_i z_{i+1} + h_{i+1}z_i)] \\ &= hu^t_i - \frac{\Delta t}{\Delta x}\frac{g}{2}[(h_{i+1} + h_i)(h_{i+1} - h_i) \\ &\quad - (h_{i+1} + h_i)(z_i - z_{i+1})] \\ hu^{t+1}_i &= hu^t_i \end{aligned} \tag{51}$$

proves that the proposed scheme satisfies the well-balanced property since $(h_{i+1} - h_i) = (z_i - z_{i+1})$.

Smooth Steady State

See Fig. 21(b) fulfills, as such it fulfills $\partial\eta/\partial x = -z_0$ and $\partial hu/\partial x = 0$, so there is no velocity fluctuation ($\kappa = 0$) and again the diffusive fluxes D_x vanish. Although the 2D problem is more complex, its solution is similar when one assumes a quasi-1D steady state, that is, $h = \text{constant}$, $hu = \text{constant}$, $hv = 0$, $\partial z/\partial x = \text{constant}$, and $\partial z/\partial y = 0$—or $h = \text{constant}$, $hu = 0$, $hv = \text{constant}$, $\partial z/\partial x = 0$, and $\partial z/\partial y = \text{constant}$. A so-called normal depth can thus be assumed as (Chertock et al. 2015)

$$h = h_0 = \left(\frac{n^2_m (hu_0)^2}{z_0}\right)^{\frac{3}{10}}. \tag{52}$$

Since $h_{i-1} = h_{i+1} = h_i = h_0$ and $hu_{i-1} = hu_{i+1} = hu_i = hu_0$, the artificial diffusive terms **D** in Eq. (21) become zero and the convective fluxes at edges in Eq. (42), e.g., $((hu_0)^2/h + 0.5gh^2)_{i+\frac{1}{2}}$, are obtained only by averaging the values of cells i and $i+1$; the fluxes at edges $i + \frac{1}{2}$ and $i - \frac{1}{2}$ thus cancel each other. Now there remains

$$(-gh\partial z/\partial x - gn^2_m h^{-\frac{1}{3}}|u|u)_i = 0 \tag{53}$$

Employing a method similar to Eqs. (44)–(49), it is easy to show $(\partial z/\partial x)_i = -z_0$; therefore, using this value together with Eq. (52) for Eq. (53) one easily proves

$$(gh_0 z_0)_i = \left(gn^2_m h_0^{-\frac{7}{3}}|hu_0|hu_0\right)_i \tag{54}$$

which can be substituted into Eq. (38), so $h^{t+1}_i = h^t_i = h_0$ and $hu^{t+1}_i = hu^t_i = hu_0$, which satisfy the well-balanced property.

Acknowledgments

The first author gratefully acknowledges the German Academic Exchange Service for the Research Grant Doctoral Programs in Germany 2015/16 (57129429) and appreciates the computer and data resources provided by the Leibniz Supercomputing Centre. Fruitful discussions with Prof. Michael Manhart, Prof. Ernst Rank, and Dr. rer. nat. habil. Ralf-Peter Mundani (Technical University of Munich) are appreciated. Constructive comments from all the anonymous reviewers are acknowledged.

References

Audusse, E., F. Bouchut, M.-O. Bristeau, R. Klein, and B. Perthame. 2004. "A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows." *SIAM J. Sci. Comput.* 25 (6): 2050–2065. https://doi.org/10.1137/S1064827503431090.

Audusse, E., and M.-O. Bristeau. 2005. "A well-balanced positivity preserving 'second-order' scheme for shallow water flows on unstructured meshes." *J. Comput. Phys.* 206 (1): 311–333. https://doi.org/10.1016/j.jcp.2004.12.016.

Beljadid, A., and P. LeFloch. 2017. "A central-upwind geometry-preserving method for hyperbolic conservation laws on the sphere." *Commun. Appl. Math. Comput. Sci.* 12 (1): 81–107. https://doi.org/10.2140/camcos.2017.12.81.

Beljadid, A., A. Mohammadian, and A. Kurganov. 2016. "Well-balanced positivity preserving cell-vertex central-upwind scheme for shallow water flows." *Comput. Fluids* 136 (Sept): 193–206. https://doi.org/10.1016/j.compfluid.2016.06.005.

Boussinesq, J. 1877. "Essai sur la théorie des eaux courantes [Essay on the theory of running waters]." [In French.] In Vol. XXIII of *Proc., Memoires presentes par divers savants a l'Academie des Sciences*, 1–680. Paris: Institut Géographique National.

Brufau, P., P. Garcia-Navarro, and M. Vazquez-Cendon. 2004. "Zero mass error using unsteady wetting–drying conditions in shallow flows over dry irregular topography." *Int. J. Numer. Methods Fluids* 45 (10): 1047–1082. https://doi.org/10.1002/fld.729.

Cao, Z., P. Hu, K. Hu, G. Pender, and Q. Liu. 2015. "Modelling roll waves with shallow water equations and turbulent closure." *J. Hydraul. Res.* 53 (2): 161–177. https://doi.org/10.1080/00221686.2014.950350.

Castanedo, S., R. Medina, and F. J. Mendez. 2005. "Models for the turbulent diffusion terms of shallow water equations." *J. Hydraul. Eng.* 131 (3): 217–223. https://doi.org/10.1061/(ASCE)0733-9429(2005)131:3(217).

Cea, L. 2005. "An unstructured finite volume model for unsteady turbulent shallow water flow with wet-dry fronts: Numerical solver and experimental validation." Ph.D. thesis, Departamento de Métodos Matemáticos y de Representación, Universidade da Coruña.

Cea, L., J. Puertas, and M.-E. Vazquez-Cendon. 2007. "Depth averaged modelling of turbulent shallow water flow with wet-dry fronts." *Arch. Comput. Methods Eng.* 14 (3): 303–341. https://doi.org/10.1007/s11831-007-9009-3.

Cea, L., and M.-E. Vazquez-Cendon. 2012. "Unstructured finite volume discretisation of bed friction and convective flux in solute transport models linked to the shallow water equations." *J. Comput. Phys.* 231 (8): 3317–3339. https://doi.org/10.1016/j.jcp.2012.01.007.

Chen, D., and G. Jirka. 1995. "Experimental study of plane turbulent wakes in a shallow water layer." *Fluid Dyn. Res.* 16 (1): 11–41. https://doi.org/10.1016/0169-5983(95)00053-G.

Chen, Y., A. Kurganov, M. Lei, and Y. Liu. 2013. "An adaptive artificial viscosity method for the saint-venant system." In *Recent developments in the numerics of nonlinear hyperbolic conservation laws. Notes on numerical fluid mechanics and multidisciplinary design*, edited by R. Ansorge, H. Bijl, A. Meister, and T. Sonar. Berlin: Springer.

Cheng, Y., and A. Kurganov. 2016. "Well-balanced positivity preserving cell-vertex central-upwind scheme for shallow water flows." *Commun. Math. Sci.* 14 (6): 1643–1663. https://doi.org/10.4310/CMS.2016.v14.n6.a9.

Chertock, A., S. Cui, A. Kurganov, and T. Wu. 2015. "Well-balanced positivity preserving central-upwind scheme for the shallow water system with friction terms." *Int. J. Numer. Methods Fluids* 78 (6): 355–383. https://doi.org/10.1002/fld.4023.

Coroneo, M., G. Montante, A. Paglianti, and F. Magelli. 2011. "CFD prediction of fluid flow and mixing in stirred tanks: Numerical issues about the rans simulations." *Comput. Chem. Eng* 35 (10): 1959–1968. https://doi.org/10.1016/j.compchemeng.2010.12.007.

Davidson, L. 1993. *Implementation of a K-ϵ model and a Reynolds stress model into a multiblock code*. Tech. Rep. CRS4-APPMATH-93-21. Cagliari, Italy: Applied Mathematics and Simulation Group CRS4.

Delis, A. I., and I. K. Nikolos. 2013. "A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered finite volumes with application to shallow water dynamics." *Int. J. Numer. Methods Fluids* 71 (5): 584–633. https://doi.org/10.1002/fld.3674.

Delis, A. I., I. K. Nikolos, and M. Kazolea. 2011. "Performance and comparison of cell-centered and node-centered unstructured finite volume discretizations for shallow water free surface flows." *Arch. Comput. Methods Eng.* 18 (1): 57–118. https://doi.org/10.1007/s11831-011-9057-6.

Duan, J. 2004. "Simulation of flow and mass dispersion in meandering channels." *J. Hydraul. Eng.* 130 (10): 964–976. https://doi.org/10.1061/(ASCE)0733-9429(2004)130:10(964).

Durbin, P. A. 1996. "On the k-3 stagnation point anomaly." *Int. J. Heat Fluid Flow* 17 (1): 89–90. https://doi.org/10.1016/0142-727X(95)00073-Y.

Ginting, B. 2019. "Central-upwind scheme for 2D turbulent shallow flows using high-resolution meshes with scalable wall functions." *Comput. Fluids* 179 (Jan): 394–421. https://doi.org/10.1016/j.compfluid.2018.11.014.

Ginting, B. M. 2017. "A two-dimensional artificial viscosity technique for modelling discontinuity in shallow water flows." *Appl. Math. Modell.* 45 (May): 653–683. https://doi.org/10.1016/j.apm.2017.01.013.

Ginting, B. M., and R.-P. Mundani. 2018. "Artificial viscosity technique: A Riemann-solver-free method for 2D urban flood modelling on complex topography." In *Advances in hydroinformatics*, edited by P. Gourbesville, J. Cunge, and G. Caignaert. Singapore: Springer Water.

Han, L. 2015. "Recirculation zone developing downstream of an expansion in a shallow open-channel flow." Ph.D. thesis, Laboratoire de Mecanique des Fluides et d'Acoustique, INSA de Lyon.

Han, L., E. Mignot, and N. Riviere. 2017. "Shallow mixing layer downstream from a sudden expansion." *J. Hydraul. Eng.* 143 (5): 04016105. https://doi.org/10.1061/(ASCE)HY.1943-7900.0001274.

Hernandez-Duenas, G., and A. Beljadid. 2016. "A central-upwind scheme with artificial viscosity for shallow-water flows in channels." *Adv. Water Resour.* 96 (Oct): 323–338. https://doi.org/10.1016/j.advwatres.2016.07.021.

Hirsch, C. 2007. *Numerical computation of internal and external flows*. Amsterdam, Netherlands: Elsevier.

Horvath, Z., J. Waser, R. A. P. Perdigao, A. Konev, and G. Bloeschl. 2015. "A two-dimensional numerical scheme of dry/wet fronts for the saint-venant system of shallow water equations." *Int. J. Numer. Methods Fluids* 77 (3): 159–182. https://doi.org/10.1002/fld.3983.

Hou, J., Q. Liang, F. Simons, and R. Hinkelmann. 2013a. "A 2D well-balanced shallow flow model for unstructured grids with novel slope source term treatment." *Adv. Water Resour.* 52 (Feb): 107–131. https://doi.org/10.1016/j.advwatres.2012.08.003.

Hou, J., Q. Liang, F. Simons, and R. Hinkelmann. 2013b. "A stable 2D unstructured shallow flow model for simulations for wetting and drying over rough terrains." *Comput. Fluids* 82 (Aug): 132–147. https://doi.org/10.1016/j.compfluid.2013.04.015.

Jameson, A. 2017. "Origins and further development of the Jameson-Schmidt–Turkel scheme." *AIAA J.* 55 (5): 1487–1510. https://doi.org/10.2514/1.J055493.

Jameson, A., and D. Mavriplis. 1986. "Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh." *AIAA J.* 24 (4): 611–618. https://doi.org/10.2514/3.9315.

Jameson, A., W. Schmidt, and E. Turkel. 1981. "Numerical solution of the euler equations by finite volume methods using runge kutta time stepping schemes." In *Proc., 14th Fluid and Plasma Dynamics Conf.* Reston, VA: American Institute of Aeronautics and Astronautics.

Kanayama, H., and H. Dan. 2013. "A tsunami simulation of Hakata Bay using the viscous shallow-water equations." *Jpn. J. Ind. Appl. Math.* 30 (3): 605–624. https://doi.org/10.1007/s13160-013-0111-7.

Karimi, M., G. Akdogan, and S. M. Bradshaw. 2012. "Effects of different mesh schemes and turbulence models in cfd modelling of stirred tanks." *Physicochem. Probl. Miner. Process.* 48 (2): 513–531. https://doi.org/10.5277/ppmp120216.

Kim, D.-H., and P. J. Lynett. 2011. "Turbulent mixing and passive scalar transport in shallow flows." *Phys. Fluids* 23 (1): 016603. https://doi.org/10.1063/1.3531716.

Kim, D.-H., P. J. Lynett, and S. A. Socolofsky. 2009. "A depth-integrated model for weakly dispersive, turbulent, and rotational fluid flows." *Ocean Modell.* 27 (3): 198–214. https://doi.org/10.1016/j.ocemod.2009.01.005.

Kurganov, A., S. Noelle, and G. Petrova. 2001. "Semi-discrete central-upwind schemes for hyperbolic conservation laws and Hamilton-Jacobi equations." *SIAM J. Sci. Comput.* 23 (3): 707–740. https://doi.org/10.1137/S1064827500373413.

Kurganov, A., and G. Petrova. 2007. "A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system." *Commun. Math. Sci.* 5 (1): 133–160. https://doi.org/10.4310/CMS.2007.v5.n1.a6.

Kurganov, A., M. Prugger, and T. Wu. 2017. "Second-order fully discrete central-upwind scheme for two-dimensional hyperbolic systems of conservation laws." *SIAM J. Sci. Comput.* 39 (3): A947–A965. https://doi.org/10.1137/15M1038670.

Levi, E. 1995. *The science of water: The foundation of modern hydraulics*. Reston, VA: ASCE.

Liang, Q., and A. G. L. Borthwick. 2009. "Adaptive quadtree simulation of shallow flows with wet-dry fronts over complex topography." *Comput. Fluids* 38 (2): 221–234. https://doi.org/10.1016/j.compfluid.2008.02.008.

Liu, X., and A. Beljadid. 2017. "A coupled numerical model for water flow, sediment transport and bed erosion." *Comput. Fluids* 154 (Sep): 273–284. https://doi.org/10.1016/j.compfluid.2017.06.013.

Lloyd, P. M., and P. K. Stansby. 1997. "Shallow-water flow around model conical islands of small side slope. II: Submerged." *J. Hydraul. Eng.* 123 (12): 1068–1077. https://doi.org/10.1061/(ASCE)0733-9429(1997)123:12(1068).

Mavriplis, D. 1988. "Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes." *AIAA J.* 26 (7): 824–831. https://doi.org/10.2514/3.9975.

Menter, F., and T. Esch. 2001. "Elements of industrial heat transfer predictions." In *Proc., 16th Brazilian Congress of Mechanical Engineering*. Rio de Janeiro, Brazil: Associação Brasileira de Engenharia e Ciências Mecânicas.

Michel-Dansac, V., C. Berthon, S. Clain, and F. Foucher. 2016. "A well-balanced scheme for the shallow-water equations with topography." *Comput. Math. Appl.* 72 (3): 568–593. https://doi.org/10.1016/j.camwa.2016.05.015.

Mohamadian, A., D. Y. Le Roux, M. Tajrishi, and K. Mazaheri. 2005. "A mass conservative scheme for simulating shallow flows over variable topographies using unstructured grid." *Adv. Water Resour.* 28 (5): 523–539. https://doi.org/10.1016/j.advwatres.2004.10.006.

Mohammadian, A. 2010. "Numerical approximation of viscous terms in finite volume models for shallow waters." *Int. J. Numer. Methods Fluids* 63 (5): 584–599. https://doi.org/10.1002/fld.2097.

Mohammadian, A., and D. Y. Le Roux. 2006. "Simulation of shallow flows over variable topographies using unstructured grids." *Int. J. Numer. Methods Fluids* 52 (5): 473–498. https://doi.org/10.1002/fld.1167.

NTHMP (National Tsunami Hazards Mitigation Program). 2018. "NTHMP mapping and modeling benchmarking workshop: Tsunami currents." Accessed March 23, 2018. http://coastal.usc.edu/currents_workshop/problems.html.

Rodi, W. 1993. *Turbulence models and their application in hydraulics: A state-of-the art review.* Rotterdam, Netherlands: A.A. Balkema.

Rodi, W. 2017. "Turbulence modeling and simulation in hydraulics: A historical review." *J. Hydraul. Eng.* 143 (5): 03117001. https://doi.org/10.1061/(ASCE)HY.1943-7900.0001288.

Shirkhani, H., A. Mohammadian, O. Seidou, and A. Kurganov. 2016. "A well-balanced positivity-preserving central-upwind scheme for shallow water equations on unstructured quadrilateral grids." *Comput. Fluids* 126 (Mar): 25–40. https://doi.org/10.1016/j.compfluid.2015.11.017.

Swanson, R. C., R. Radespiel, and E. Turkel. 1998. "On some numerical dissipation schemes." *J. Comput. Phys.* 147 (2): 518–544. https://doi.org/10.1006/jcph.1998.6100.

van Albada, G., B. van Leer, and W. Roberts. 1997. "A comparative study of computational methods in cosmic gas dynamics." In *Upwind and high-resolution schemes*, edited by M. Hussaini, B. van Leer, and J. Van Rosendale. Berlin: Springer.

Van Der Burg, J. W., J. G. M. Kuerten, and P. J. Zandbergen. 1992. "Improved shock-capturing of jameson's scheme for the euler equations." *Int. J. Numer. Methods Fluids* 15 (6): 649–671. https://doi.org/10.1002/fld.1650150603.

Van Leer, B. 1979. "Towards the ultimate conservative difference scheme. v. A second-order sequel to Godunov's method." *J. Comput. Phys.* 32 (1): 101–136. https://doi.org/10.1016/0021-9991(79)90145-1.

Venkatakrishnan, V. 1993. "On the accuracy of limiters and convergence to steady state solutions." In *Proc., AIAA Paper.* Reno, NV: American Institute of Aeronautics and Astronautics.

Wu, W. 2004. "Depth-averaged two-dimensional numerical modeling of unsteady flow and nonuniform sediment transport in open channels." *J. Hydraul. Eng.* 130 (10): 1013–1024. https://doi.org/10.1061/(ASCE)0733-9429(2004)130:10(1013).

Xilin, X., and Q. Liang. 2018. "A new efficient implicit scheme for discretising the stiff friction terms in the shallow water equations." *Adv. Water Resour.* 117 (Jun): 87–97. https://doi.org/10.1016/j.advwatres.2018.05.004.

Yu, C., and J. Duan. 2012. "Two-dimensional depth-averaged finite volume model for unsteady turbulent flow." *J. Hydraul. Res.* 50 (6): 599–611. https://doi.org/10.1080/00221686.2012.730556.

Zhang, Y. J., G. Priest, J. Allan, and L. Stimely. 2016. "Benchmarking an unstructured-grid model for tsunami current modeling." *Pure Appl. Geophys.* 173 (12): 4075–4087. https://doi.org/10.1007/s00024-016-1328-6.

Paper 5

B.M. Ginting and R.-P. Mundani. Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography. In P. Gourbesville, J. Cunge, and G. Caignaert, editors, *Advances in Hydroinformatics*, chapter 4, pages 51–74. Springer Water, Springer, Singapore, 2018.
https://dx.doi.org/10.1007/978-981-10-7218-5_4

Artificial Viscosity Technique: A Riemann-Solver-Free Method for 2D Urban Flood Modelling on Complex Topography

Bobby Minola Ginting and Ralf-Peter Mundani

1 Introduction

Using the two-dimensional (2D) shallow water equations (SWEs) for urban flood modelling is a well-established approach in computational science and engineering. These SWEs have been successfully used to describe free surface flows on complex topography, in which the horizontal length scale of waves is much greater than the vertical length scale. Since the SWEs can, in general, not be solved analytically, a numerical approach is used instead. Many types of numerical approaches have been developed such as finite differences, finite elements, finite volumes and lattice Boltzmann methods. In this paper, we focus on the finite volume method (FVM), which is probably the most popular scheme for solving the SWEs.

In the past two decades, Riemann solvers such as Roe, Harten-Lax-van Leer (HLL) and Harten-Lax-van Leer-contact (HLLC) schemes for a cell-centred finite volume (CCFV) scheme have been applied to solve the SWEs. These Riemann solvers have been proven to be robust and can simulate complex mixed-flow regimes by capturing sharp gradients with low-level oscillations. For details, we refer to [1–11]. The HLLC scheme, which was developed by Toro [4], is very suitable in modelling flow problems on complex topography that deals with both very shallow water and wet–dry problems such as floods in urban and rural areas. Hou et al. [9] applied the HLLC scheme for simulating some flow cases which involved wet–dry interfaces. Dry beds could be taken directly into account to compute the left and right wave speeds which are required in the HLLC scheme. Delis and Nikolos [10] used the Roe approximate Riemann solver in order to

B. M. Ginting (✉) · R.-P. Mundani
Chair for Computation in Engineering, Technical University of Munich,
Arcisstr. 21, D-80333 Munich, Germany
e-mail: bobbyminola.ginting@tum.de

R.-P. Mundani
e-mail: mundani@tum.de

P. Gourbesville et al. (eds.), *Advances in Hydroinformatics*, Springer Water,
https://doi.org/10.1007/978-981-10-7218-5_4

compute the convective flux in a CCFV scheme, in which the conservation of the flow at rest with dry regions was well satisfied.

Despite its robustness in capturing strong gradients, the use of Riemann solvers may lead to a huge computational overhead particularly when using a high-order temporal discretisation scheme such as the Runge–Kutta fourth-order (RKFO) method. This is because the convective flux, which is computed based on a linearisation consisting of eigenvalues and eigenvectors of the approximate Jacobian matrix in the Roe scheme or based on the computation of the left, middle (contact) and right wave speeds in the HLLC scheme, must be re-evaluated four times per time step. Hence, the complex *if-then-else* statements in the HLLC scheme would entail a computational overhead. In order to avoid this problem, we developed a technique which is free from a computation of the Riemann solver. The main concept of our technique is that we compute the convective flux only by averaging the left and right states of every edge, thus simplifying computations compared to the HLLC scheme or to a calculation of the entropy correction in the Roe scheme and saving computational time. However, this averaging technique will not be stable and produces significant oscillations without a stabilisation scheme, particularly when dealing with discontinuous flows. As a consequence, an artificial viscosity (AV) technique was developed to work as a stabilisation scheme for minimising unphysical oscillations. This AV technique is computed separately from the convective flux. In other words, the convective flux can be re-evaluated only by averaging the left and right states of every edge without depending on the result of the AV technique. This AV technique is then computed in a hybrid manner which is only computed once per time step before computing the RKFO method. We will show in Sect. 2.5 that the computational time can be significantly reduced.

In addition to lower computational times, another aspect we want to emphasise is the capability of the AV scheme in producing non-diffusive results for the use of a first-order scheme. Based on our observation, most of the aforementioned publications used a second-order numerical scheme, in which the main variables of the SWEs such as depth and fluxes are reconstructed using a limited-gradient function. Liang and Marche [11], who also used a Riemann solver in their study, stated that the first-order numerical scheme gives diffusive results that are generally not acceptable in practice. However, in this paper, we will show that the proposed AV technique does not give diffusive results in spite of using the first-order scheme; shock waves and sharp gradients are well captured without an excess of diffusivity.

AV techniques are rarely used for solving the 2D SWEs. Most applications were previously conducted in aeronautics to solve the Euler equations for simulating inviscid transonic flows associated with shock waves (see, e.g. [13–17]). In simulating free surface discontinuous flows, Ginting [18] and Ginting et al. [19, 20] used the AV technique for some flood and dam-break problems. The discontinuous flows were accurately and stably simulated. Ginting [12] then improved the AV technique to achieve higher accuracy and presented arguments why this AV scheme could be much cheaper than a Riemann solver. In this paper, we prove the aforementioned statement by giving a comparison of benchmark tests for the HLLC scheme and the AV technique. Another scheme interestingly pointing out is the

central-upwind (CU) scheme which was first introduced by Kurganov and Levy [21] and then improved by Kurganov and Petrova [22]. The CU scheme is a Riemann-solver-free method which is able to both preserve stationary steady states (lake at rest) and to guarantee the positivity of the computed fluid depth. We also compare our model with the CU scheme.

Other improvements based on Ginting [12] are a proper treatment of the friction term in which a semi-implicit method is combined with a wet–dry technique, and an advanced solely edge-based bed-slope discretisation. By combining these treatments with the AV scheme, our model is able to simulate real flood phenomena on complex topography in which discontinuities due to the rapid change of a flow regime (caused by very shallow water) or due to wet–dry problems must be taken into account. This paper is organised as follows. The governing equations and numerical model are explained in Sect. 2. Some benchmark tests are given in Sect. 3 in order to verify our model. Finally, conclusions are given in Sect. 4.

2 Mathematical Formulation

2.1 Governing Equations

The 2D SWEs in vector form are written as [12, 18–20]

$$\frac{\partial \mathbf{W}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{S}, \tag{1}$$

where **W**, **F**, **G** and **S** are the vectors given by

$$\begin{aligned} \mathbf{W} &= \begin{bmatrix} h \\ uh \\ vh \end{bmatrix}; \quad \mathbf{F} = \begin{bmatrix} uh \\ u^2h + \frac{1}{2}gh^2 \\ uvh \end{bmatrix}; \\ \mathbf{G} &= \begin{bmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 \end{bmatrix}; \quad \mathbf{S} = \begin{bmatrix} R - I \\ gh\,(Sx - Sfx) \\ gh\,(Sy - Sfy) \end{bmatrix}. \end{aligned} \tag{2}$$

The variables h, u, v and g denote, respectively, the water depth, velocity in the x and y directions and acceleration due to gravity. The variables R and I define, respectively, rainfall intensity and infiltration. Terms Sx, Sy, Sfx and Sfy are the bed slopes and friction source terms in the x and y directions and described in Sects. 2.2.4 and 2.2.5. The full mathematical derivation of our model can be read in [12].

2.2 Numerical Model: Spatial Discretisation

By applying the Gauss divergence theorem over a polygonal control volume Ω_k, we get an integration form of (1) as

$$\frac{\partial}{\partial t}\iint_{\Omega_k} \mathbf{W}^{el}{}_k \, d\Omega_k + \sum_{i=1}^{N} \left(\mathbf{F}^{ed}{}_i \, \mathbf{n}_x^{ed}{}_i + \mathbf{G}^{ed}{}_i \; \mathbf{n}_y^{ed}{}_i \right) L^{ed}{}_i = \iint_{\Omega_k} \mathbf{S}^{el}{}_k \; d\Omega_k, \qquad (3)$$

where *el* and *ed,* respectively, define element and edge, $L^{ed}{}_i$ defines the length of edge i and N is the total number of edges that surround element k. The symbols $\mathbf{n}_x^{ed}{}_i$ and $\mathbf{n}_y^{ed}{}_i$ denote the unit outward vectors normal to edge i and are computed in counter-clockwise direction. The second term on the left-hand side of (3) is the convective flux, whose calculation is very important in order to ensure stability. In this paper, we do not treat elements as representing computational cells; rather, we calculate the value of every edge over the entire domain. The computation of the convective flux at edge i is therefore only performed once. Furthermore, the result can simply be multiplied by -1 when we are computing another element that also belongs to edge i. Prior to explaining the difference between the HLLC, CU and AV schemes, we present a 2D domain which is discretised into nine elements as shown in Fig. 1.

As shown in Fig. 1, the domain has a total of 24 edges. Since our model is based on an edge-based computation, we need to re-evaluate the convective flux 24 times for those 24 edges per time step. In other words, the convective flux requires a re-evaluation of 96 times (four times more) when we use the RKFO method. Using the HLLC scheme, which has complex nested *if-then-else* statements, increases the computational overhead significantly. This is because the pattern of true or false condition cannot exactly be predicted by the branch prediction logic of a processor. If the condition is always true or false, the processor can follow the pattern;

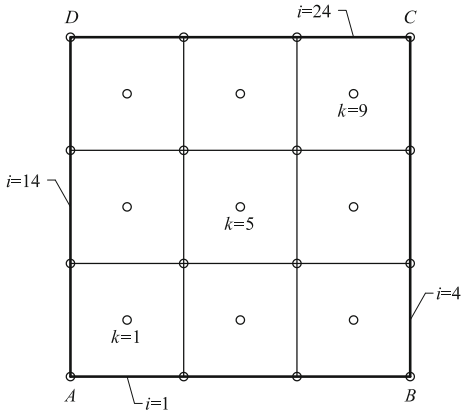


Fig. 1 A 2D domain discretised by 9 elements (denoted by k) with 24 edges (denoted by i)

however, if the pattern is unpredictable, the *if-then-else* statements will be more expensive. Of course, one might get different results after compiling the same *if-then-else* statements with different compilers, since the compilers might make optimisations, such as loop-unrolling, that affects the performance. Also, since the branch prediction logic is encoded in hardware, one might have different results on different processors in spite of compiling the same *if-then-else* statements with the same compiler. Instead of being confused by this problem and of giving extra efforts to investigate it, we propose an AV scheme, which is very easy to be programmed, in order to replace the role of the Riemann solver. Later, we will also show that the AV scheme can produce more accurate results than those of HLLC and CU schemes. Since our focus in this paper is mainly on showing the difference between the HLLC, CU and AV schemes, we do not reconstruct the left and right states of every edge; rather, we simply calculate these schemes using the first-order scheme. In the following sections, we present in general the formulas of the HLLC, CU and AV schemes.

2.2.1 HLLC Scheme

This scheme is an approximate Riemann solver which was developed by Toro [4]. Using this scheme, the calculation of $\left(\mathbf{F}^{ed}{}_i\,\mathbf{n}_x^{ed}{}_i + \mathbf{G}^{ed}{}_i\,\mathbf{n}_y^{ed}{}_i\right)$ in (3) is generally expressed as (4), where $S_{L\,i}^{ed}$, $S_{M\,i}^{ed}$ and $S_{R\,i}^{ed}$ are, respectively, the left, middle (contact) and right wave speeds, which are proper to treat wet–dry bed conditions. The variables $\mathbf{H}_{L\,i}^{ed}$ and $\mathbf{H}_{R\,i}^{ed}$ are the interface fluxes of the left and right states, respectively. $\mathbf{H}_{L*\,i}^{ed}$ and $\mathbf{H}_{R*\,i}^{ed}$ denote the interface fluxes beside the contact wave. It is clearly shown by (4) that in order to compute these four interface fluxes, all wave speeds $S_{L\,i}^{ed}$, $S_{M\,i}^{ed}$ and $S_{R\,i}^{ed}$ must be known first.

$$\left(\mathbf{F}_i^{ed}\,\mathbf{n}_x^{ed}{}_i + \mathbf{G}^{ed}{}_i\,\mathbf{n}_y^{ed}{}_i\right) = \begin{cases} \mathbf{H}_{L\,i}^{ed} & \text{if} \quad 0 \le S_{L\,i}^{ed} \\ \mathbf{H}_{L*i}^{ed} & \text{if} \quad S_{L\,i}^{ed} < 0 \le S_{M\,i}^{ed} \\ \mathbf{H}_{R*i}^{ed} & \text{if} \quad S_{M\,i}^{ed} < 0 \le S_{R\,i}^{ed} \\ \mathbf{H}_{R\,i}^{ed} & \text{if} \quad S_{R\,i}^{ed} \le 0 \end{cases} \tag{4}$$

With regard to this, some expensive mathematical operations are required which are basically the functions of the main variables h, u and v. For a more complete version of the mathematical equations of this scheme, interested readers are referred to [4, 9]. As shown in the aforementioned publications, in addition to (4), another *if-then-else* statement based on the value of h should be performed again to obtain the wave speeds, creating a more complex nested *if-then-else* statement. Therefore, this scheme would suffer from a computational overhead and particularly for the use of the RKFO method, this can even be worse since all *if-then-else* statements must be performed four times more.

2.2.2 Central-Upwind Scheme

This scheme is a Riemann-solver-free method and was developed by [21, 22]. The concept of both the CU and HLLC schemes is similar; the convective flux should be re-evaluated with regard to the wave speed of every edge. However, as we will show later, the computation of the CU scheme is much simpler and cheaper than that of the HLLC scheme. The calculation of $\left(\mathbf{F}^{ed}{}_i\,\mathbf{n}_x^{ed}{}_i + \mathbf{G}_i^{ed}\,\mathbf{n}_y^{ed}{}_i\right)$ in (3) is given by

$$\left(\mathbf{F}_i^{ed}\,\mathbf{n}_x^{ed}{}_i + \mathbf{G}_i^{ed}\,\mathbf{n}_y^{ed}{}_i\right) = \frac{1}{a_{\text{in}\,i}^{ed} + a_{\text{out}i}^{ed}} \begin{bmatrix} \left(a_{\text{in}\,i}^{ed}\mathbf{F}_{R\,i}^{ed} + a_{\text{out}i}^{ed}\mathbf{F}_{L\,i}^{ed}\right)\mathbf{n}_{xi}^{ed} + \left(a_{\text{in}\,i}^{ed}\mathbf{G}_{R\,i}^{ed} + a_{\text{out}i}^{ed}\mathbf{G}_{L\,i}^{ed}\right)\mathbf{n}_{yi}^{ed} \\ -a_{\text{in}\,i}^{ed}a_{\text{out}i}^{ed}\left(\mathbf{W}_{R\,i}^{ed} - \mathbf{W}_{L\,i}^{ed}\right) \end{bmatrix}, \tag{5}$$

where $\mathbf{F}_{L\,i}^{ed}$, $\mathbf{F}_{R\,i}^{ed}$, $\mathbf{G}_{L\,i}^{ed}$, $\mathbf{G}_{R\,i}^{ed}$, $\mathbf{W}_{L\,i}^{ed}$ and $\mathbf{W}_{R\,i}^{ed}$ indicate the variables in the vector of (2) with regard to the left and right states of edge i. The variables $a_{\text{in}\,i}^{ed}$ and $a_{\text{out}i}^{ed}$ define the local one-sided speeds of propagation, given by

$$\begin{aligned} a_{\text{in}\,i}^{ed} &= -\min\left(U_{L\,i}^{ed} - \sqrt{g h_{L\,i}^{ed}}, U_{R\,i}^{ed} - \sqrt{g h_{R\,i}^{ed}}, 0\right) \text{ and} \\ a_{\text{out}i}^{ed} &= \max\left(U_{L\,i}^{ed} + \sqrt{g h_{L\,i}^{ed}}, U_{R\,i}^{ed} + \sqrt{g h_{R\,i}^{ed}}, 0\right). \end{aligned} \tag{6}$$

The variables $U_{L\,i}^{ed}$ and $U_{R\,i}^{ed}$ are expressed as

$$U_{L\,i}^{ed} = u_{L\,i}^{ed}\,\mathbf{n}_x^{ed}{}_i + v_{L\,i}^{ed}\,\mathbf{n}_y^{ed}{}_i \quad \text{and} \quad U_{R\,i}^{ed} = u_{R\,i}^{ed}\,\mathbf{n}_x^{ed}{}_i + v_{R\,i}^{ed}\,\mathbf{n}_y^{ed}{}_i, \tag{7}$$

where $u_{L\,i}^{ed}$, $v_{L\,i}^{ed}$, $u_{R\,i}^{ed}$ and $v_{R\,i}^{ed}$ define the velocities in the x and y directions for the left and right states of edge i. A more complete version of the above equations is also given in Wu et al. [23].

2.2.3 Artificial Viscosity Scheme

In [18–20], the first author of this paper used the AV scheme for solving the 2D SWEs and then improved this scheme in [12] based on the pioneering ideas of [13–17]. A good performance of the AV scheme was shown that stable computations and highly accurate results were always achieved, whereas the computational time remained acceptably low. In order to compare this AV scheme with the two previous schemes, we now write the calculation of $\left(\mathbf{F}_i^{ed}\,\mathbf{n}_x^{ed}{}_i + \mathbf{G}_i^{ed}\,\mathbf{n}_y^{ed}{}_i\right)$ in (3) as

$$\left(\mathbf{F}_i^{ed}\,\mathbf{n}_{x\ i}^{ed} + \mathbf{G}_i^{ed}\,\mathbf{n}_{y\ i}^{ed}\right) = \frac{1}{2}\left[\left(\mathbf{F}_{R\ i}^{ed} + \mathbf{F}_{L\ i}^{ed}\right)\mathbf{n}_{x\ i}^{ed} + \left(\mathbf{G}_{R\ i}^{ed} + \mathbf{G}_{L\ i}^{ed}\right)\mathbf{n}_{y\ i}^{ed}\right]. \tag{8}$$

Obviously, in comparison to (4), the computational effort is significantly reduced in this model. The convective flux is re-evaluated only by averaging the left and right states, even though this re-evaluating process must still be performed four times per time step for the use of the RKFO method. Yet, the computation of (8) is basically unstable without a proper treatment.

In order to minimise unphysical oscillations, an AV scheme is therefore developed and (3) is now expressed as

$$\frac{\partial}{\partial t}\iint_{\Omega_k} \mathbf{W}_k^{el}\, d\Omega_k + \sum_{i=1}^{N}\left(\mathbf{F}_i^{ed}\,\mathbf{n}_{x\ i}^{ed} + \mathbf{G}_i^{ed}\,\mathbf{n}_{y\ i}^{ed}\right)L_i^{ed} - D_k^{el} = \iint_{\Omega_k} \mathbf{S}_k^{el} d\Omega_k, \tag{9}$$

where D_k^{el} denotes an artificial viscosity term, being a function of the vector $\mathbf{W}$. It should be noted that D_k^{el} is not computed for edges, but for elements. According to Fig. 1, D_k^{el} is computed once for each element per time step before starting the RKFO method. These values are then used in all four Runge–Kutta steps. This will be discussed in detail in Sects. 2.3 and 2.5. By considering a rectangular cell with four edges $i = 1, \ldots, 4$ in the counter-clockwise direction as shown in Fig. 2, D_k^{el} for the element $k = 0$ is now expressed as

$$D_{k=0}^{el} = D_{i=2}^{ed} - D_{i=4}^{ed} + D_{i=3}^{ed} - D_{i=1}^{ed}. \tag{10}$$

We present here only a general overview for (10); a complete mathematical derivation can be read in [12]. By combining a Laplacian and a biharmonic operator, we form an AV scheme, in which the variable scaling factor is constructed using the spectral radius of the Jacobian matrix at every edge λ_i^{ed}. We now focus on computing $D_{i=2}^{ed}$, which is expressed as

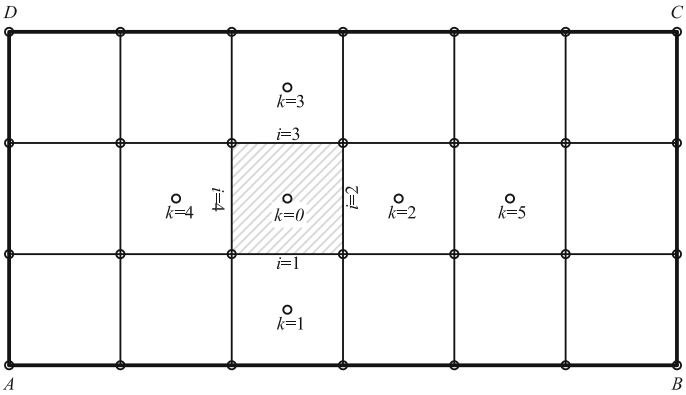


Fig. 2 Definition of the parameter for the AV scheme

$$D_{i=2}^{ed} = L_{i=2}^{ed} \lambda_{i=2}^{ed} \left(\Delta \mathbf{W}_{i=2} \varepsilon_{i=2}^{(2)} - \Delta^3 \mathbf{W}_{i=2} \varepsilon_{i=2}^{(4)} \right), \tag{11}$$

where $\Delta \mathbf{W}_{i=2} = \mathbf{W}_{k=2}^{el} - \mathbf{W}_{k=0}^{el}$ and $\Delta^3 \mathbf{W}_{i=2} = \mathbf{W}_{k=5}^{el} - 3\mathbf{W}_{k=2}^{el} + 3\mathbf{W}_{k=0}^{el} - \mathbf{W}_{k=4}^{el}$. The variable $\lambda_{i=2}^{ed}$ is then computed as

$$\begin{aligned} \lambda_{i=2}^{ed} = & \left| \frac{1}{2} \left(u_{k=0}^{el} + u_{k=2}^{el} \right) \mathbf{n}_{x\ i=2}^{ed} + \frac{1}{2} \left(v_{k=0}^{el} + v_{k=2}^{el} \right) \mathbf{n}_{y\ i=2}^{ed} \right| \\ & + \frac{1}{2} \left(c_{k=0}^{el} + c_{k=2}^{el} \right) \sqrt{\left(\mathbf{n}_{x\ i=2}^{ed} \right)^2 + \left(\mathbf{n}_{y\ i=2}^{ed} \right)^2}, \end{aligned} \tag{12}$$

where c is the wave speed and simply computed by $c = \sqrt{gh}$. In order to control the diffusivity of (11), we construct a dimensionless depth discontinuity sensor for element $k = 0$ $\vartheta_{k=0}^{el}$ as

$$\vartheta_{k=0}^{el} = \frac{\left| h_{k=2}^{el} - 2\, h_{k=0}^{el} + h_{k=4}^{el} \right|}{(1-\omega)\left(\left| h_{k=2}^{el} - h_{k=0}^{el} \right| + \left| h_{k=0}^{el} - h_{k=4}^{el} \right| \right) + \omega \left(\left| h_{k=2}^{el} + 2\, h_{k=0}^{el} + h_{k=4}^{el} \right| \right)}. \tag{13}$$

The value of the depth discontinuity sensor at the edge $i = 2$ $\vartheta_{i=2}^{ed}$ is now computed as

$$\vartheta_{i=2}^{ed} = \max\left(\vartheta_{k=5}^{el}, \vartheta_{k=2}^{el}, \vartheta_{k=0}^{el}, \vartheta_{k=4}^{el} \right). \tag{14}$$

The variables $\varepsilon_i^{(2)}$ and $\varepsilon_i^{(4)}$ in (11) define the adaptive Laplacian and biharmonic coefficients for edge i which are computed by (15). The variables $\kappa^{(2)}$, $\kappa^{(4)}$ and ω denote the coefficients that should be determined empirically. As stated in [12], the values of these coefficients are problem-specific.

$$\begin{aligned} \varepsilon_{i=2}^{(2)} &= \kappa^{(2)} \vartheta_{i=2}^{ed} \\ \varepsilon_{i=2}^{(4)} &= \max\left(0, \kappa^{(4)} - \vartheta_{i=2}^{ed} \right) \end{aligned}. \tag{15}$$

In order to obtain the final value of $D_{k=0}^{el}$ in (10), the same procedure should be applied for the other subscripts (i = 1, 3 and 4).

2.2.4 Hydrostatic and Topography Reconstructions

In general, before computing the convective flux using the three aforementioned schemes and the bed-slope source term, a proper modification for the bed elevation of the considered edge, in which wet–dry interfaces are taken into account, should be made. This is important to ensure both non-negative water depths and the flux

balance particularly for wet–dry interfaces. Following [9, 11], we modify the bed elevation of edge i z_i^{ed} as

$$z_i^{ed} = \max\left(\eta_{L\,i}^{ed} - h_{L\,i}^{ed}, \eta_{R\,i}^{ed} - h_{R\,i}^{ed}\right), \tag{16}$$

where $\eta_{L\,i}^{ed}$, $\eta_{R\,i}^{ed}$, $h_{L\,i}^{ed}$ and $h_{R\,i}^{ed}$ are the water elevations and the depths of the left and right states of edge i. We compute a hydrostatic reconstruction for the depths $h_{L\,i}^{ed}$ and $h_{R\,i}^{ed}$ as

$$h_{L\,i}^{ed} = \max\left(0, \eta_{L\,i}^{ed} - z_i^{ed}\right) \quad \text{and} \quad h_{R\,i}^{ed} = \max\left(0, \eta_{R\,i}^{ed} - z_i^{ed}\right). \tag{17}$$

These modified depths are then employed for the computation of the convective flux using the aforementioned schemes. Prior to calculating both Sx and Sy, we remodify z_i^{ed} to be applied in an edge-based fashion by borrowing the idea of [11] as

$$z_i^{ed} \leftarrow z_i^{ed} - \max\left(0, z_i^{ed} - \eta_{L\,i}^{ed}, z_i^{ed} - \eta_{R\,i}^{ed}\right). \tag{18}$$

2.2.5 Bed-Slope Term Treatment

We now apply a partial derivative computation for the bed-slope source term, for which the integral form of $(ghSx)_k^{el}$ is written as (19), where h_k^{el}, z_k^{el} and A_k^{el} denote, respectively, the depth, bed elevation and area of element k. This technique is similar to that of Mohammdian and Le Roux [6], which is neither an extra upwinding scheme nor a Riemann solution and has been proven to satisfy the compatibility property. The structure of this technique is also applied in an edge-based fashion; it is therefore suitable for our model. A similar manner can also be applied to compute the integral form of $(ghSy)_k^{el}$.

$$\begin{aligned}
\iint_{\Omega_k} (ghSx)_k^{el} d\Omega_k &= -g \iint_{\Omega_k} \left(h\frac{\partial z}{\partial x}\right)_k^{el} d\Omega_k \\
&= -\frac{g}{2} \iint_{\Omega_k} \left(\frac{\partial (hz)}{\partial x} + h\frac{\partial z}{\partial x} - z\frac{\partial h}{\partial x}\right)_k^{el} d\Omega_k \\
&= -\frac{g}{2A_k^{el}} \sum_{i=1}^{N} \left[\left(\frac{h_{L\,i}^{ed} + h_{R\,i}^{ed}}{2} + h_k^{el}\right) z_i^{ed} - \left(\frac{h_{L\,i}^{ed} + h_{R\,i}^{ed}}{2}\right) z_k^{el}\right] \mathbf{n}_{x\,i}^{ed} L_i^{ed}.
\end{aligned} \tag{19}$$

2.2.6 Friction Term Treatment

An explicit treatment of the friction term may produce significant oscillations when the roughness coefficient is very high. In addition, some stability issues with the Courant–Friedrichs–Lewy (CFL) condition would appear when facing wet–dry conditions. This problem, of course, can be avoided either by using a varying time step that is limited by the CFL number or by refining the computational grid. However, both of these possibilities require an unavoidably high computational cost. Therefore, a semi-implicit technique is more suitable for calculating the friction term. Another advantage is that this semi-implicit technique can be integrated in the RKFO method in a straightforward manner. According to [7, 24], we calculate the flux of the x-momentum equation for element k as

$$(hu)_k^{el\,t+1} = (hu)_k^{el*} - (g\,h\,Sfx)_k^{el\,t+1}\Delta t, \tag{20}$$

where $(hu)_k^{el*}$ defines the flux for which the friction force is not taken into account yet. The value of $(g\,h\,Sfx)_k^{el\,t+1}$ in (20) is now expressed as

$$(g\,h\,Sfx)_k^{el\,t+1} = (ghu)_k^{el\,t+1} \left[(1-\theta)\left(n_m^2 h^{-4/3}\sqrt{u^2+v^2}\right)_k^{el\,t+1} + \theta\left(n_m^2 h^{-4/3}\sqrt{u^2+v^2}\right)_k^{el\,t}\right], \tag{21}$$

where θ is the implicitness parameter. When $\theta = 0$ the scheme leads to a full implicit scheme and with $\theta = 1$ the scheme is computed in a full explicit manner. In order to solve (21), the assumption of $t+1 \approx *$ can be made, that means * is incorporated at each step of the RKFO scheme. A similar way is also applied to compute $(hv)_k^{el\,t+1}$. Murillo et al. [24] and Delis et al. [7] proved that (21) does not depend on the time step; therefore, a time step restriction is not required.

In this paper, we propose a wet–dry treatment that is incorporated in the computation of the friction source term, for which the pseudocode (`compute` $\mathbf{W}_k^{el\,p}$) is expressed as follows:

```
if h^{el p}_k > D_min then
    recompute (hu)^{el p}_k and (hv)^{el p}_k based on (20) and (21)
else
    h^{el p}_k = D_min and both (hu)^{el p}_k and (hv)^{el p}_k are set to zero
end if
```

The variable $D_{\min}$ is a limited value to prevent computational instability due to very small water depth. In this case, the value of $D_{\min}$ is set to 1×10^{-6} m. The

coupling between the friction term and the wet–dry treatment as shown in the pseudocode above has two advantages; the first is that this algorithm is capable of both preserving stability for very shallow water on very rough beds and preventing excessive drag forces of those rough beds that can reverse the flow; the second is that this algorithm saves more computational time since the calculations of $(hu)^{elp}_{k}$ and $(hv)^{elp}_{k}$ are only carried out for wet elements. The variable p denotes the calculation step of the RKFO scheme which is explained in Sect. 2.3.

2.3 Numerical Model: Time Discretisation

The time discretisation with the RKFO method is expressed as (22a–e) and applied in its hybrid formulation, in which the AV technique $D^{el}{}_{k}$ is only performed once (denoted by $p = 0$). During the calculation of this RKFO scheme, the value of $D^{el}{}_{k}$ is not reupdated from $p = 1$ to $p = 4$; hence, computational cost will be significantly reduced. It should be noted that for both the HLLC and CU schemes, D^{el}_{k} does not exist. First, one must calculate h^{el*}_{k}, which is not affected by the friction term. Also, $(hu)^{el*}_{k}$ and $(hv)^{el*}_{k}$ are computed, in which the effect of friction is not taken into account yet. Second, both $(hu)^{el*}_{k}$ and $(hv)^{el*}_{k}$ must be converted into velocities u^{el*}_{k} and v^{el*}_{k}. Due to a very low water depth, some errors may exist when converting the fluxes back to the velocities. Thus, the pseudocode `compute` $\mathbf{W}^{elp}_{k}$ must be applied, where the velocities are only computed for wet cells, whose depths are greater than $D_{\min}$. The fluxes for those wet cells are recalculated by including the friction term as given in (20) and (21). Finally, these calculations are repeated until $p = 4$.

$$\mathbf{W}^{el}{}_{k}^{\,p=0} = \mathbf{W}^{el}{}_{k}^{\,t} \tag{22-a}$$

`do p=1, 4`

$$\alpha_{p=1} = 0.25;\quad \alpha_{p=2} = 0.33;\quad \alpha_{p=3} = 0.5;\quad \alpha_{p=4} = 1.0 \tag{22-b}$$

$$\mathbf{W}^{el}{}_{k}^{\,*} = \mathbf{W}^{el}{}_{k}^{\,p=0} + \alpha_{p}\left(\begin{array}{l} -\dfrac{\Delta t}{A^{el}{}_{k}}\left[\displaystyle\sum_{i=1}^{N}\left(\mathbf{F}^{ed}{}_{i}\,\mathbf{n}_{x}{}^{ed}{}_{i} + \mathbf{G}^{ed}{}_{i}\,\mathbf{n}_{y}{}^{ed}{}_{i}\right)^{p-1} L^{ed}{}_{i} - D^{el}{}_{k}^{\,p=0}\right] \\ + \Delta t \displaystyle\iint_{\Omega_k}\left[gh\left(Sx + Sy\right)\right]^{el}{}_{k}^{\,p-1}\, d\Omega_{k} \end{array}\right) \tag{22-c}$$

`call pseudo-code compute` $\mathbf{W}^{el}{}_{k}^{\,p}$ (22-d)

$$\mathbf{W}^{el}{}_{k}^{\,t+1} = \mathbf{W}^{el}{}_{k}^{\,p=4} \tag{22-e}$$

`end do`

2.4 Boundary Conditions

Following [9, 12, 18–20], the boundary conditions are applied based on a flux computation, in which the characteristic method is used for subcritical flow boundaries to determine the velocities in both the x and y directions when the water elevation is specified or conversely. Meanwhile, for supercritical flow boundaries, the depth and velocities in both the x and y directions of the right states are set equal to those of the left states. At solid boundaries, since flow cannot physically emerge through a wall, a ghost cell technique similar to [12, 18–20] is applied.

2.5 Comparison Between the HLLC, CU and AV Schemes

As we previously mentioned, Ginting [12] gave an overview why an AV scheme could be much cheaper than a Riemann solver. In this section, we re-explain this by giving three pseudocodes **`Algorithm 1, 2`** and **`3,`** respectively, for the HLLC, CU and AV schemes.

```
do t = 1,  num_of_time_step

    do k = 1,  num_of_element
        apply eq.(22-a)
        calculate eq.(13) !This applies only to the AV scheme
    end do

    !******** This loop applies only to the AV scheme ********!
    do k = 1,  num_of_element
        calculate eqs.(12),(14),(15),(11)
        apply the similar computation for edge i=1,3,4
        calculate eq.(10)
    end do
    !******************** End of the loop *********************!

    do p = 1,  4   !The RKFO starts
        apply eq.(22-b)

        do i = 1,  num_of_edge
            !see Algorithm 1,2,3 for the HLLC, CU and AV schemes respectively
        end do

        do k = 1,  num_of_element
            calculate eq.(22-c)
                !for both the HLLC and CU schemes, eq.(22-c) is computed without D^el_k
            apply eq.(22-d)
            apply eq.(22-e)
        end do

    end do

end do

Algorithm 1: HLLC scheme
    construct if-then-else based on the value of the depth to obtain S^ed_L i , S^ed_M i and S^ed_R i
      !the formulations of this if-then-else statement are given in [4,9]
    calculate eq.(4)
      !eq.(4) consists of if-then-else (again) that depends on the above condition

Algorithm 2: CU scheme
    calculate eqs.(6),(7),(5)

Algorithm 3: AV scheme
    calculate eq.(8)
```

As shown in the above pseudocodes, the nested *if-then-else* statements required in the HLLC scheme must be performed four times the number of edges (`num_of_edge`) for the use of the RKFO method; the computational overhead will therefore increase significantly. Meanwhile, there are no complex *if-then-else* statements required in both the CU and AV schemes; the computations of these schemes remain simple and acceptably cheap as shown by (5) and (8), respectively. Despite requiring an additional computation of $D^{el}{}_k$, the computational cost of the AV scheme can be decreased since $D^{el}{}_k$ is only computed once before the computation of the RKFO starts. Nevertheless, we would like to emphasise that we are not claiming that the AV scheme is better than a Riemann solver, as also written in [12].

3 Test Cases

3.1 Case 1: Dam Break with a Triangular Bump on a Straight Channel

Prior to validating with a complex topography, we take a dam-break case selected from the CADAM studies to verify the capability of our scheme in dealing with strong shock waves and discontinuous flows. We refer this case to [5, 25]. As shown in Fig. 3, the channel was 38 m long and 0.75 m wide. A gate was located 15.5 m from the upstream part of the channel and the depth was 0.75 m initially along this section. In order to conduct a dam-break flow, this gate was suddenly opened. A triangular bump with a height of 0.4 m and a length of 6 m was located 13 m downstream of the gate. The value of the Manning coefficient was set to 0.0125 $sm^{-1/3}$. In this study, we select the case from the aforementioned publications that both upstream and downstream boundaries are set as wall boundary. We discretise the domain into 45,600 rectangular elements. The total simulation time is set to 40 s and the time step is 0.005 s. For the AV scheme, we set $\kappa^{(2)}$, $\kappa^{(4)}$ and ω, respectively, to 5.0, 0.03125 and 0.5. These coefficients were fitted iteratively. Some suggestions concerning the ranges of these coefficients are given in Ginting [12].

The results of the AV scheme around the bump are shown in Fig. 4. A bore wave is generated suddenly after the gate opening and propagates along the channel. Afterwards, it reaches the bump and starts to run up. When running up the bump, a negative wave appears and flows back upstream since the bore is partially reflected. After overtopping the bump, the bore flows downstream and accelerates over the sloping part. The wave continues to flow until it is reflected when reaching the end wall and then propagates upstream. We also present a comparison between the three schemes and the experimental results in Fig. 5, which shows that all schemes produce sufficiently accurate results. There are no significant differences shown by these three schemes. The arrival time of the bores is accurately predicted. At gauge G4, the water depth increases dramatically at 12 s due to an interaction between the wave from the upstream part and the reflected wave from the bump. All schemes predict the second bore properly at 27 s. However, the third bore is predicted earlier but of no more than 1 s. A similar characteristic to gauge G4 is also

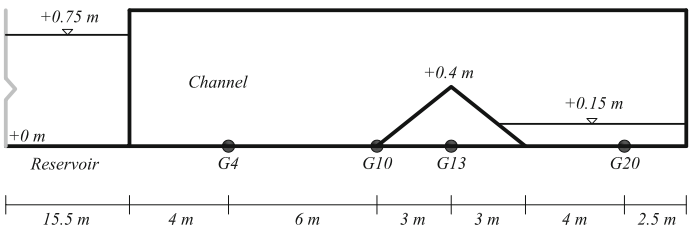


Fig. 3 Case 1: Straight channel with a triangular bump on a straight channel

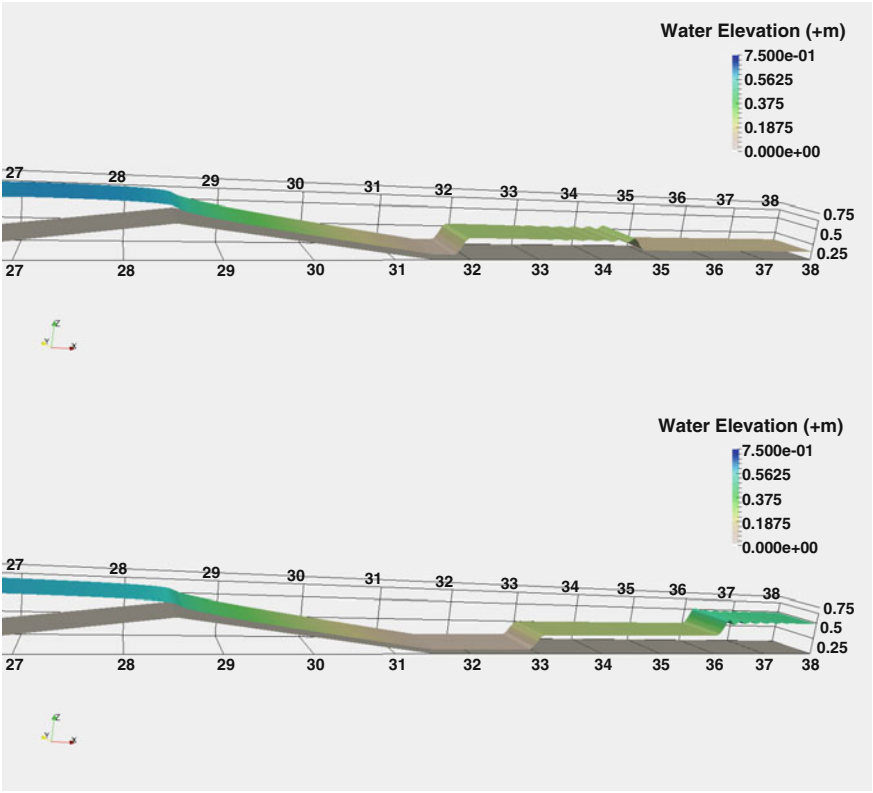


Fig. 4 Case 1: Results of the AV scheme around the bump at 7 s (upper) and at 10 s (bottom)

shown by gauge G10. The proper results are given at gauges G13 and G20, that all schemes simulate the local discontinuity in bed slope accurately, whereas that of [5] cannot. Also, the result at gauge G13 shows that these schemes are highly stable when simulating very shallow water.

3.2 Case 2: L-Shaped Dam Break

While the previous case was to show that our scheme is able to simulate complex wave–wave interactions, we now consider the simulation of discontinuous flows on a discontinuous bed topography, in which the waves propagate both in x and y directions. A laboratory study was conducted by [26] and some numerical simulations of this case were performed by [12, 27–29]. This is a dam-break case, whose domain consists of a square reservoir and an L-shaped channel as shown in Fig. 6. In this case, a square reservoir is connected with an L-shaped channel by a

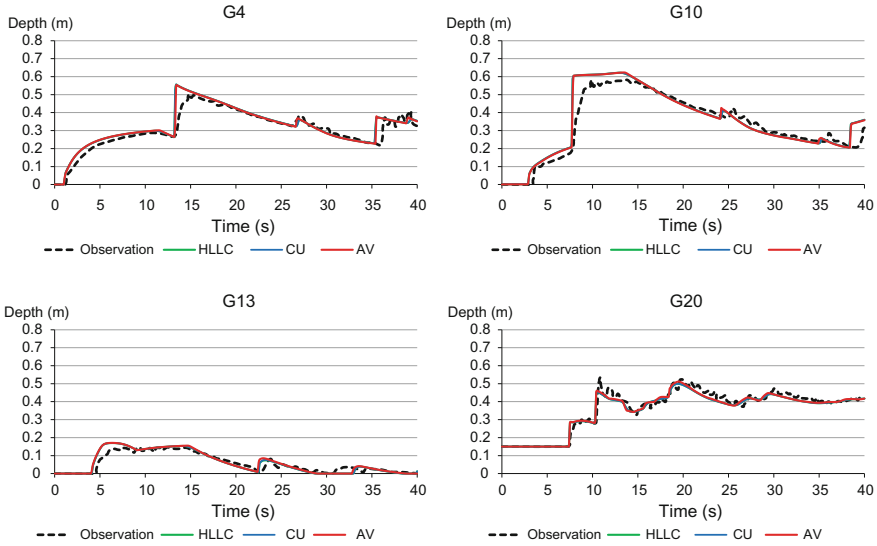


Fig. 5 Case 1: Comparison between experimental and numerical results of all schemes

discontinuous topography, where the bed elevation of the reservoir is 0.33 m lower than that of the channel. We select a dry bed case in this study that the water depth in the channel is initially zero. In the reservoir, the initial water depth is 0.53 m. As suggested by [28, 29], we set the Manning coefficient to 0.0095 $sm^{-1/3}$, both for the reservoir and the channel. The domain is discretised into 24,359 rectangular elements. The total simulation time is then set to 40 s with a time step of 0.0025 s. Only the downstream boundary is set as free outflow, the rest of which are set as wall boundaries.

The coefficients $\kappa^{(2)}$, $\kappa^{(4)}$ and ω are set similar to case 1. Figure 7 shows a comparison of water surface elevation along the downstream channel at 7 s that the AV scheme can capture the strong gradients more accurately than do the HLLC and CU schemes. Neither oscillations nor diffusive results are produced by the AV scheme in spite of using the first-order scheme. The HLLC and CU schemes simulate the discontinuous flows stably; however, both of them are too diffusive that the strong gradients are not well captured. As shown in Fig. 7, the water surface elevations produced by the HLLC and CU schemes tend to be too smooth. The wave–wave interactions that occur in both x and y directions cannot be properly simulated. At 2.5 s, the water reaches the wall at the L-shaped part and the bore is partially reflected. After 7 s, the wave continues to propagate upstream causing a highly discontinuous flow. Meanwhile, some waves still propagate downstream. Along this downstream part, the strong gradients of the very complex wave–wave interactions are accurately captured by the AV scheme, whereas both the HLLC and CU schemes cannot.

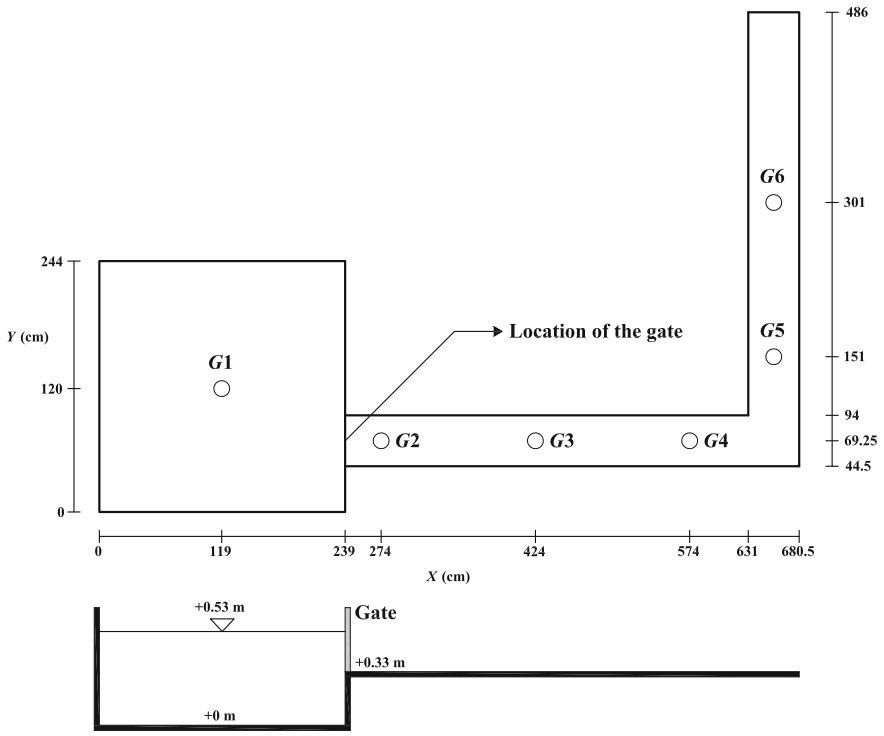


Fig. 6 Case 2: Domain of L-shaped dam break

In Fig. 8, we show a comparison between these three schemes at G2–G5. At gauge G2, the AV scheme detects both the arrival time and the elevation of the first and second bores accurately. Both the HLLC and CU schemes can detect the arrival time of the first bore properly. The elevation of the first bore is, however, underestimated. The HLLC scheme detects both the arrival time and the elevation of the second bore properly; however, the CU scheme computes a delay of 2 s with an underestimated elevation. At gauge G3, all schemes detect both the arrival time and the elevation of the first bore properly. However, only the AV scheme can detect the arrival time of the second bore accurately. The HLLC scheme computes a delay of 1 s and the CU scheme is even worse with a delay of 2 s. At gauge G4, both the AV and HLLC schemes compute the first bore with a delay but of no more than 1 s. The CU scheme gives an underestimated elevation for the first bore with a delay of 2 s. After 10 s, both the HLLC and CU schemes produce similar results, where the elevations are underestimated compared with the observed results, whereas the AV scheme computes more accurately. At gauge G5, all schemes compute the underestimated elevation of the first bore but of no more than 1 cm. However, after 10 s both the HLLC and CU schemes keep producing the overestimated results. Meanwhile, the AV scheme computes the elevation accurately. As shown in Fig. 8,

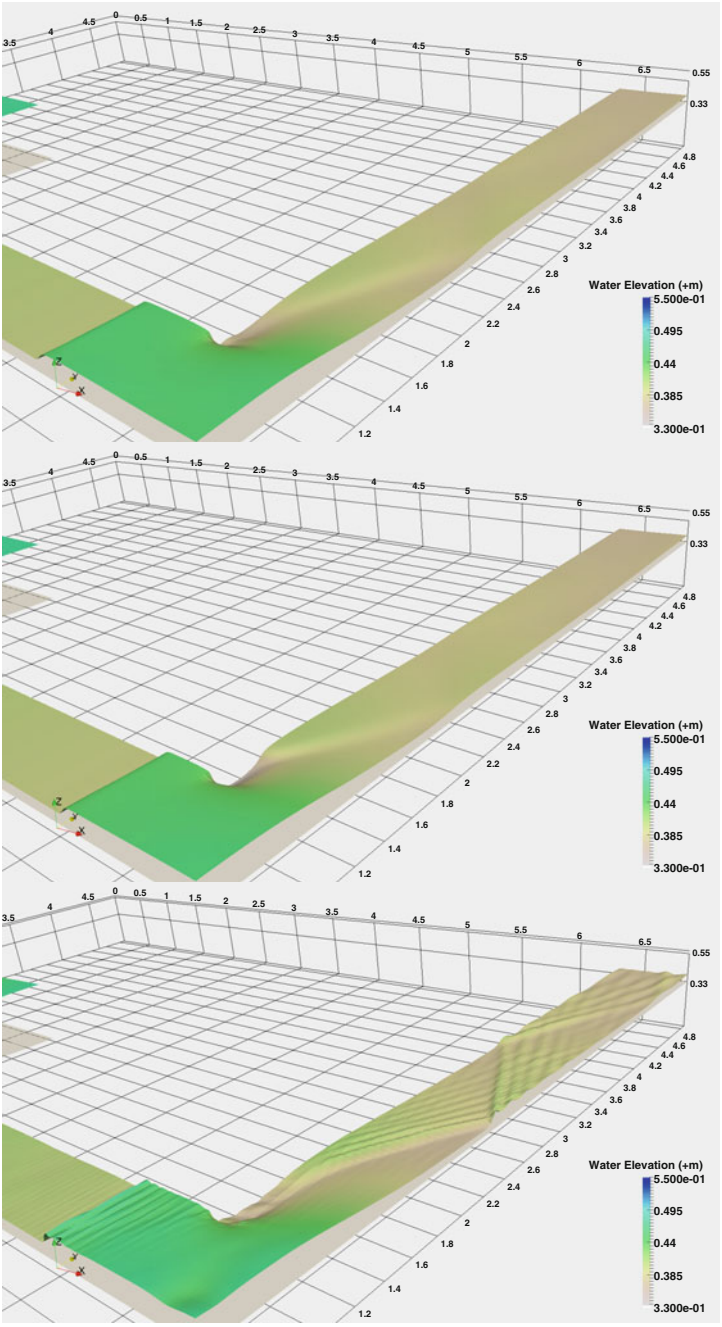


Fig. 7 Case 2: Results of the HLLC (upper), CU (middle) and AV (bottom) schemes at 7 s

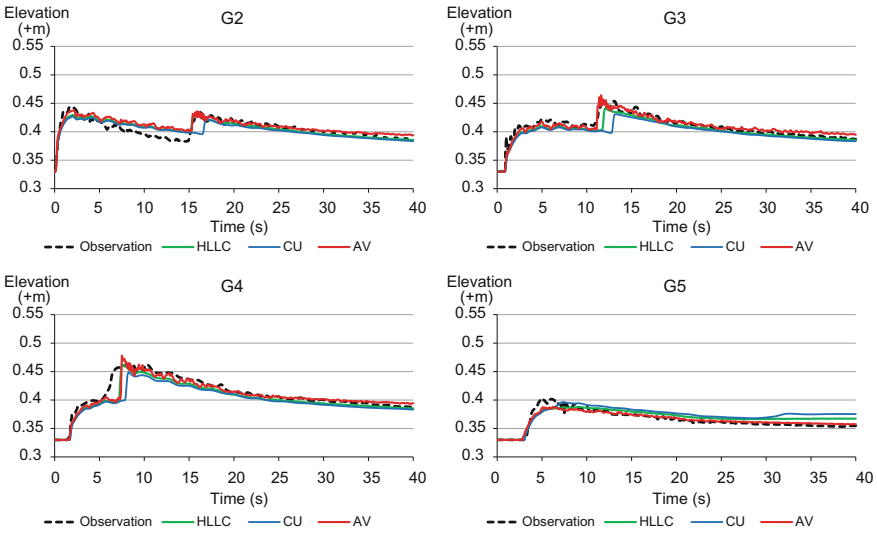


Fig. 8 Case 2: Comparison between experimental and numerical results of all schemes

the results of both the HLLC and CU schemes are too diffusive that the strong gradients are not well captured during the simulation time. The AV scheme shows a better performance which can detect the shock waves accurately.

3.3 Case 3: Rainfall and Point Source Surface Flow in Urban Areas

In this case, a flood propagation on a complex topography is simulated, of which the flood arises from two sources, a uniformly distributed rainfall event and a point source. The input data of this case are given in detail by [30]. The modelled area was taken in Glasgow, UK on 13 August 2009 and approximately 0.4 km 0.96 km provided in a DEM format. The ground elevations are within the range of +21 to +37 m. Any buildings at the real location are ignored in order to simulate the bare earth of the DEM. The initial boundary condition is set as dry bed and all boundaries of the modelled area are set as wall boundaries. As published in [30], there were no observation data provided for this case; instead, several 1D, 2D and 3D numerical models were used to simulate this case and the results were compared at nine points. In this paper, for a comparison purpose, we take the results of the 2D models, such as ISIS 2D, MIKE FLOOD, SOBEK, TUFLOW and XPSTORM; the details of these 2D models are also given in [30]. The grid resolution should be at least 2 m or 97,000 nodes in the 0.388 km^2 modelled area. The Manning

coefficients are 0.02 $sm^{-1/3}$ for the roads and pavements and 0.05 $sm^{-1/3}$ everywhere else.

We discretise the domain into 96,600 rectangular elements (by a 2 m × 2 m rectangular cell). The total simulation time is then set to 18,000 s with a time step of 0.20 s. For the AV scheme, we set the coefficients $\kappa^{(2)}$, $\kappa^{(4)}$ and ω to 1.0, 0.03125 and 0.5, respectively. Figure 9 shows a comparison between the results of the HLLC, CU and AV schemes and the aforementioned 2D numerical models at points 2 and 6. As published in [30], due to the very intense rainfall during 1–4 min, almost all the aforementioned 2D numerical models produced a double-peaked shaped. The similar results are also shown by the HLLC, CU and AV schemes. At point 2 for the first peak flow, both the HLLC and CU schemes exhibit a similar characteristic of the water level which is higher than that of the AV scheme. The HLLC and CU schemes predict a water level of +28.747 m at 7.5 min and of +24.744 m at 8 min, respectively, whereas the AV scheme computes a water level of +28.702 m at 11.5 min. The second peak flow is predicted almost similarly by both the HLLC and CU schemes with a water level of +28.811 m at 43 min. The AV scheme computes, however, a lower depth with a water level of +28.802 m at 44 min. A similar characteristic is also shown for the velocity at point 2 during 0–53 min. However, after 53 min, the CU scheme keeps producing the lower water levels, showing a capability in computing the drying mechanism properly.

At point 6, the first peak flow is predicted almost similarly by the HLLC, CU and AV schemes with a water level of +26.973 m at 4 min. The AV scheme computes the second peak flow at 44.5 min with a water level of +27.029 m, whereas both the HLLC and CU schemes produce a similar result at 46.5 min with a water level of +27.008 m. The maximum velocity of 1.204 m/s is computed by the AV scheme at

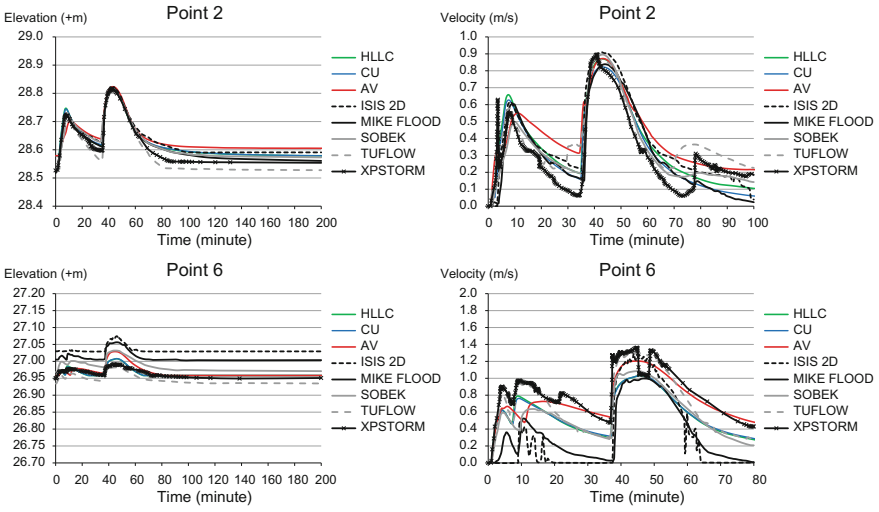


Fig. 9 Case 3: Comparison between numerical results of all schemes and those of [30]

44.5 min. At 46.5 min, both the HLLC and CU schemes produce a similar velocity of 1.033 m/s. We also present an inundation map computed with the AV scheme in Fig. 10, which shows that the AV scheme is stable in simulating a wet–dry mechanism on complex topography. The semi-implicit scheme of the friction term is capable of preventing excessive drag forces of the very rough beds both on the pavements and the bare earth that no negative water depth is produced. Since there is no observed result provided in [30], we cannot determine which scheme is the most proper one for this case. Nevertheless, we have shown that our results are in accordance with those of [30].

3.4 Comparison of the CPU Time

In this section, we show a comparison of the CPU time for all schemes. All codes were executed on a Haswell Dekstop Dell Precision T1700 (single core) and compiled for the Ubuntu 16.4 operating system using an Intel Fortran compiler

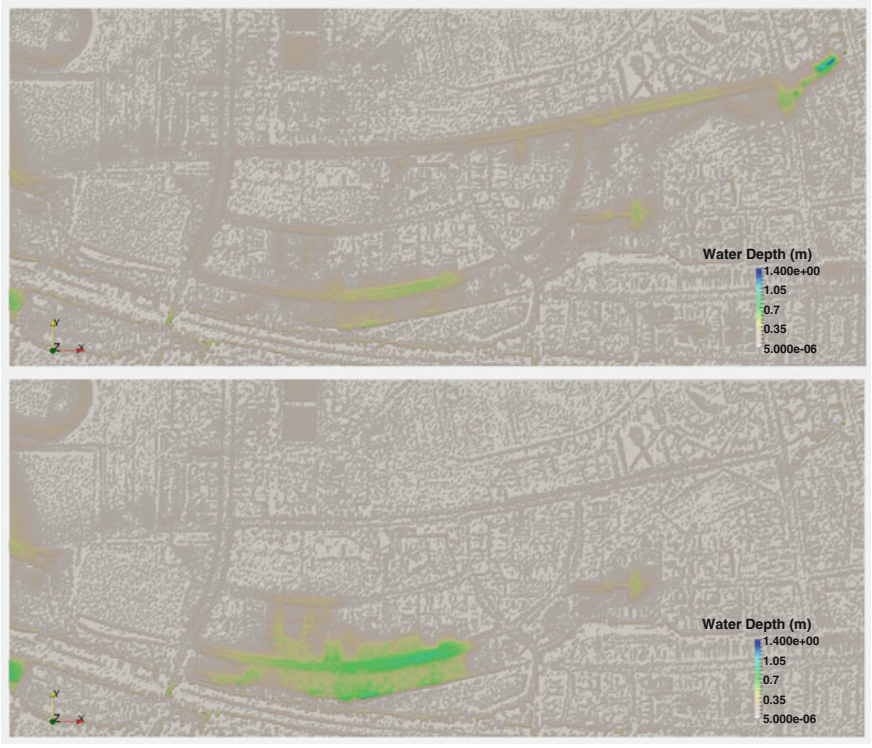


Fig. 10 Case 3: Flood inundation area at 2400 s (upper) and 9000 s (lower) computed using the AV scheme

Table 1 Comparison of the CPU time between the HLLC, CU and AV schemes

	Number of elements	Number of edges	Time step (s)	Number of time step	CPU time (s)			Ratio of CPU time		
					HLLC	CU	AV	HLLC/CU	HLLC/AV	AV/CU
Case 1	45,600	92,750	0.005	8000	82	64	66	1.294	1.241	1.042
Case 2	24,359	49,373	0.0025	16,000	67	53	56	1.263	1.202	1.051
Case 3	96,600	193,883	0.20	90,000	2479	1874	2057	1.323	1.205	1.098
Average								1.293	1.216	1.064

16.0.3. For each case, we performed the simulation of every scheme ten times in order to obtain an average CPU time. The CPU times required by all cases are given in Table 1, which shows that the HLLC scheme requires more CPU time than does the AV scheme by an average multiplicative factor of 1.216. Also, the HLLC is more expensive than the CU scheme by an average factor of 1.293. The ratio of the CPU time between the AV and CU schemes is 1.064 averagely, for which the AV scheme suffers from a little higher CPU time. It should be noted that despite being the cheapest one, the CU method is too diffusive when using the first-order scheme and not accurate in capturing the strong gradients of the very complex wave–wave interactions that occur in the x and y directions as shown in case 2. The AV scheme produces more accurate results than do the other schemes.

4 Conclusions

An AV scheme as a Riemann-solver-free method for 2D urban flood modelling on complex topography has been presented. A combination of a Laplacian and a biharmonic operator, in which the variable scaling factor is formed by using the spectral radius of the Jacobian matrix, has successfully minimised unphysical oscillations. The strong gradients of the very complex wave–wave interactions were captured accurately. The discretisation of the bed-slope source term, which is neither an extra upwinding scheme nor a Riemann solution, has been applied in an edge-based fashion and has been proven to give satisfying results. Also, we have shown that the semi-implicit treatment of the friction term could reduce oscillations significantly when we encountered a very small water depth with a very high roughness coefficient. Our computation ran stably with a minimum limiter value of 1×10^{-6} m. With regard to the CPU time, we have proven that the AV scheme is cheaper than the HLLC scheme, even though an extra computation should be made by constructing ${D^{el}}_k$ in order to minimise unphysical oscillations. This could be achieved since the computations of the Laplacian and the biharmonic operators are simple and also since we required re-evaluating the convective flux of the SWEs only by averaging the left and right states of every edge, instead of evaluating complex *if-then-else* statements as required in the HLLC scheme. The AV scheme requires more CPU time than does the CU scheme by an average multiplicative

factor of 1.064. This is, however, not too significant. The AV scheme is still more accurate in simulating complex discontinuous flows than the CU scheme. A main drawback of the AV scheme is that this scheme requires an extra memory to save both ${\vartheta^{el}}_k$ and ${D^{el}}_k$. A parallelisation version of this AV scheme would be an interesting topic for future study.

Acknowledgements DAAD (*German Academic Exchange Service*), who supports the research of Bobby Minola Ginting in the Research Grants—Doctoral Programmes in Germany 2015/16 (57129429), is gratefully acknowledged.

References

1. Anastasiou, K., & Chan, C. T. (1997). Solution of the 2D shallow water equations using the finite volume method on unstructured triangular meshes. *International Journal for Numerical Methods in Fluids, 24,* 1225–1245.
2. Leveque, R. J. (1998). Balancing source terms and flux gradients in high-resolution godunov-type methods. *Journal of Computational Physics, 146*(1), 346–365.
3. Fujihara, M., & Borthwick, A. G. L. (2000). Godunov-type solution of curvilinear shallow-water equations. *Journal of Hydraulic Engineering ASCE, 126*(11), 827–836.
4. Toro, E. (2001). *Shock-capturing methods for free-surface shallow flow*. London: Wiley.
5. Zhou, J. G., Causon, D. M., Mingham, C. G., & Ingram, D. M. (2004). Numerical prediction of dam-break flows in general geometries with complex bed topography. *Journal of Hydraulic Engineering ASCE, 130*(4), 332–340.
6. Mohammadian, A., & Le Roux, D. Y. (2006). Simulation of shallow flows over variable topographies using unstructured grids. *International Journal for Numerical Methods in Fluids, 52*(5), 473–498.
7. Delis, A. I., Nikolos, I. K., & Kazolea, M. (2011). Performance and comparison of cell-centered and node-centered unstructured finite volume discretizations for shallow water free surface flow. *Archives of Computational Methods in Engineering, 18*(1), 57–118.
8. Murillo, J., & García-Navarro, P. (2012). Augmented versions of the HLL and HLLC riemann solvers including source terms in one and two Dimensions for shallow flow applications. *Journal of Computational Physics, 231*(20), 6861–6906.
9. Hou, J., Simons, F., Mahgoub, M., & Hinkelmann, R. (2013). A robust well-balanced model on unstructured grids for shallow water flows with wetting and drying over complex topography. *Computer Methods in Applied Mechanics and Engineering, 257,* 126–149.
10. Delis, A. I., & Nikolos, I. K. (2013). A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered finite volumes with application to shallow water dynamics. *International Journal for Numerical Methods in Fluids, 71*(5), 584–633.
11. Liang, Q., & Marche, F. (2009). Numerical resolution of well-balanced shallow water equations with complex source terms. *Advances in Water Resources, 32,* 873–884.
12. Ginting, B. M. (2017). A two-dimensional artificial viscosity technique for modelling discontinuity in shallow water flows. *Applied Mathematical Modelling, 45,* 653–683.
13. Jameson, A., Schmidt, W., & E. Turkel. (1981). Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. In *AIAA 14th Fluid and Plasma Dynamic Conference*.
14. Jameson, A., & Mavriplis, D. (1986). Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh. *AIAA Journal, 24*(4), 611–618.

15. Mavriplis, D. (1987). Multigrid solution of the two-dimensional euler equations on unstructured triangular meshes. *AIAA Journal, 26*(7), 824–831.
16. Van der Burg, A., Kuerten, J. G. M., & Zandbergen, P. J. (1992). Improved shock-capturing of Jameson's scheme for the Euler equations. *International Journal for Numerical Methods in Fluids, 15*(6), 649–671.
17. Swanson, R. C., Radespiel, R., & Turkel, E. (1998). On some numerical dissipation schemes. *Journal of Computational Physics, 147*(2), 518–544.
18. Ginting, B. M. (2011). *Two dimensional flood propagation modeling generated by dam break using finite volume method*. Master Thesis, Bandung Institute of Technology, Indonesia.
19. Ginting, B. M., Natakusumah, D. K., Harlan, D., & Ginting, H. (2012). Application of finite volume cell center method with wet and dry treatment in hydrodynamic flow modeling. In *Proceeding of the Second International Conference on Port, Coastal, and Offshore Engineering, Bandung Institute of Technology, Indonesia*. ISBN 9789799616128.
20. Ginting, B. M., Riyanto, B. A., & Ginting, H. (2013). Numerical simulation of dam break using finite volume method case study of Situ Gintung. In *Proceedings of International Seminar on Water Related Disaster Solutions* (vol 1, pp. 209–220). ISBN 978979988550.
21. Kurganov, A., & Levy, D. (2002). Central-upwind schemes for the Saint-Venant system. *M2AN Mathematical Modelling and Numerical Analysis, 36,* 397–425.
22. Kurganov, A., & Petrova, G. (2007). A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system. *Communications in Mathematical Sciences, 5*(1), 133–160.
23. Wu, G., He, Z., & Liu, G. (2014). Development of a cell-centered godunov-type finite volume model for shallow water flow based on unstructured mesh. *Mathematical Problems in Engineering*.
24. Murillo, J., García-Navarro, P., & Burguete, J. (2009). Time step restrictions for well-balanced shallow water solutions in non-zero velocity steady states. *International Journal for Numerical Methods in Fluids, 60*(12), 1351–1377.
25. Morris, M. (2000). *CADAM: Concerted action on dam-break modelling*. Final report no. SR 571, HR Wallingford.
26. Soares-Frazão, S., Sillen, X., & Zech, Y. (1998). Dam-break flow through sharp bends physical model and 2D boltzmann model validation. In *Proceedings of the CADAM Meeting, HR Wallingford, U.K.* (151–169).
27. Liang, Q., Borthwick, A. G. L., & Stelling, G. (2004). Simulation of dam- and dyke-Break hydrodynamics on dynamically adaptive quadtree grids. *International Journal for Numerical Methods in Fluids, 46*(2), 127–162.
28. Tahershamsi, A., Hessaroeyeh, M.G., & Namin, M.M. (2010). Two dimensional modeling of dam-break flows. In Dittrich, Koll, Aberle & Geisenhainer (eds) *River Flow Bundesanstalt für Wasserbau*. ISBN 9783939230007.
29. Prokof'ev, V. A. (2002). State-of-the-art numerical schemes based on the control volume method for modeling turbulent flows and dam-break waves. *Power Technology and Engineering, 36*(4), 235–242.
30. Néelz, S., & Pender, G. (2013). *Benchmarking the latest generation of 2D hydraulic modelling packages*. Environment Agency, Horison House, Deanery Road, Bristol, BS1 9AH.

Paper 6

B.M. Ginting, P.K. Bhola, C. Ertl, R.-P. Mundani, M. Disse, and E. Rank. Hybrid-Parallel Simulations and Visualisations of Real Flood and Tsunami Events using Unstructured Meshes on High-Performance Cluster Systems. In *Advances in Hydroinformatics*, Springer, *accepted*, 2019.

HYBRID PARALLEL SIMULATIONS AND VISUALISATIONS OF REAL FLOOD AND TSUNAMI EVENTS USING UNSTRUCTURED MESHES ON HIGH-PERFORMANCE CLUSTER SYSTEMS

ACCEPTED MANUSCRIPT

Bobby Minola Ginting*
Chair for Computation in Engineering
Technical University of Munich
Arcisstr. 21 D-80333, Munich, Germany
bobbyminola.ginting@tum.de

Punit Kumar Bhola
Chair of Hydrology and River Basin Management
Technical University of Munich
Arcisstr. 21 D-80333, Munich, Germany

Christoph Ertl
Chair for Computation in Engineering
Technical University of Munich
Arcisstr. 21 D-80333, Munich, Germany

Ralf-Peter Mundani
Chair for Computation in Engineering
Technical University of Munich
Arcisstr. 21 D-80333, Munich, Germany

Markus Disse
Chair of Hydrology and River Basin Management
Technical University of Munich
Arcisstr. 21 D-80333, Munich, Germany

Ernst Rank
Chair for Computation in Engineering
Technical University of Munich
Arcisstr. 21 D-80333, Munich, Germany

ABSTRACT

We present simulations of real flood and tsunami events using a hybrid OpenMP-MPI model on high-performance cluster systems. The two-dimensional shallow water equations were solved by means of the in-house code NUFSAW2D, using an edge-based cell-centred finite volume method with the central-upwind scheme for millions of unstructured cells, thus ensuring spatial accuracy, especially near buildings or hydraulic structures. Each node of a cluster system performed simulations using OpenMP and communicated with other nodes using MPI. We explain strategies on reordering the meshes to support contiguous memory access patterns and to minimise communication cost; to this end, a simple criterion was proposed to decide the strategy used. Despite employing static domain decompositions for such unstructured meshes, the computation loads were distributed dynamically based on the complexity level, to each core and node during runtime to ensure computational efficiency. Our model was tested by simulating two real-life cases: the 2011 flood event in Kulmbach (Germany) and the Japan 2011 tsunami recorded in Hilo Harbour, Hawaii (USA). The numerical results show that our model is robust and accurate when simulating such complex flood phenomena, while the hybrid parallelisation concept proposed proves to be quite efficient. We also provide an outlook for an advanced visualisation method employing the Sliding Window technique with an HDF5 data structure. With such a combination of high-performance computing and interactive visualisation, users have a comprehensive predictive tool to take immediate measures and to support decision makers in developing a well-integrated early warning system.

*Corresponding author. This manuscript was accepted in Advances in Hydroinformatics (Springer) in June 2019.

1 Introduction

Many researchers have been developing tools for fast computations of flood events in the past two decades—either in the scope of empirical "black-box" models, e.g. genetic algorithms, neural networks, etc., or physically-based models, e.g. 1D/2D numerical models. The empirical models are quite popular, especially for river applications, see [1, 2, 3], due to a much shorter computational time. For urban flood applications, however, physically-based models still play a more significant role, see [4, 5, 6], because of their ability to predict some flood characteristics that cannot be assessed using empirical models. Rapid technological developments in the field of computing resources have recently ushered in a new era for such physically-based models, enabling flood simulations even on large domains in quite short time.

The probably most common approach to simulate floods on such large domains is to use parallel 2D shallow water equations (SWEs)-based models. In finite volume frameworks, these models have proven to be accurate and robust for modelling flood events, even when confronted with highly-complex phenomena, e.g. transcritical flows, shock-waves, and wet-dry problems, see [7, 8, 9], among others. In the past two decades, parallel 2D SWEs-based models have also become more and more advanced, see TELEMAC-2D [10], LISFLOOD-FP [11], ParBrezo [12], ADCIRC [13], GeoClaw [14], sam(oa)^2 [15], FullSWOF2D [16], and [17], among others. These models were developed based on OpenMP or MPI or a combination of the two running parallel on modern clusters and supercomputers. The performances turned out to be quite promising with regard to exploiting the capabilities of such clusters or supercomputers for reducing computation time, which is beneficial for fast computations of flood events.

Due to their flexibility, unstructured meshes are preferable to the structured ones to simulate complex domains. However, decomposing a domain with unstructured meshes into several processors for a parallel processing is a challenging task. Such a decomposition is not a key issue in OpenMP, because the domain can automatically be assigned to different threads. In MPI the domain decomposition of unstructured meshes can yet be problematic as the domain must be decomposed and allocated to different processors. For this, METIS library [18] can be employed as a domain decomposer. This library was developed based on graph-partitioning tools, with the aim of minimising communication cost and optimising load balance. In [12, 13, 17], METIS was used to decompose the domains with unstructured triangular meshes. Two options are provided by METIS: vertices or cells partition. The first option processes the grid partition for vertices connected by edges, while the latter creates the grid partition for cells linked to neighbouring cells. Typically, the latter is better suited for cell-centred finite volume (CCFV) models, as employed in this work, since cells represent the computational element. Another possibility is to apply space-filling curves (SFCs) for decomposing the domain, see [15].

Two types of data structure commonly employed for CCFV models are cell-based and edge-based. In the former, the solutions at a cell are directly advanced using all the corresponding cell values. When using MPI, communication between cells takes place directly. However, one major drawback of this approach is that values must be computed twice for an edge, thus requiring more computational time. In the latter approach, the solver computations are first carried out for all edges (only once for an edge), which is computationally cheaper. However, the solutions at cells cannot be advanced directly because the cells must access the edges and vice-versa. As the edge-based CCFV model with hybrid OpenMP-MPI is employed here, we consider this matter—which is why we set up the data structure

in our code in such a way that it supports any domain decomposition type. Consequently, a simple criterion is proposed as a quality indicator for inspecting the memory access patterns.

One of several aspects that affects the efficiency of parallel models is the load balancing strategy. Typically, load balancing is required for SWEs simulations with adaptive mesh refinement (AMR), see [15], where structured triangular meshes were used. Due to the use of the AMR, it is necessary to dynamically reallocate the cells to several processors because the total number of such cells changes adaptively with respect to time. For static mesh applications, the domain decomposition is only performed once before the numerical computations, thus requiring no cell-reallocation during runtime. However, as wet-dry phenomena mostly appear during runtime, there are load imbalances among processors, because wet cells demonstrate more computational efforts than dry cells. We thus follow the strategy in [19], which was specifically designed for anticipating the load imbalance due to wet-dry problems on static decomposition with rectangular meshes—and we extend it to the scope of static unstructured meshes, where the cells are dynamically reallocated to processors based on the complexity level of the wet and dry cells. An in-house code of the first author is used here: NUFSAW2D (*Numerical Simulation of Free Surface Shallow Water 2D*). Successful applications of NUFSAW2D can be found in [6, 19, 20, 21, 22, 23].

Finally, we also present a framework of an advanced visualisation technique based on [24, 25], which can be integrated in our code to visualise our numerical results. The Sliding Window technique in [24] enables us to provide an interactive visualisation not only from coarse scales, e.g. the representative water-levels of the entire domain, but also towards fine scales, e.g. the solutions of depths and velocities at the finest resolution. Although the Sliding Window technique is, even for trillions of cells, quite powerful, there can still be huge sets of sequentially-ordered data that are difficult to be accessed efficiently with post-processing tools. To minimise this problem, a Hierarchical Data Format version 5 (HDF5)-based kernel was developed in [25] that supports parallel I/O for speeding-up write operations. Both techniques are quite promising for immediate measures and to support decision makers within a framework of a well-integrated early warning system. Thus, we will provide an outlook on that aspect in this paper.

2 Mathematical Formulations

We spatially integrate the 2D SWEs over a control cell Ω and apply the Gauss divergence theorem to estimate the line integral, so that such equations can be written as

$$\frac{\partial}{\partial t}\iint_{\Omega}\mathbf{W}d\Omega+\sum_{i=1}^{N}\Big(\mathbf{F}\,n_x+\mathbf{G}\,n_y\Big)_i\,\Delta L_i=\iint_{\Omega}\big(\mathbf{S}_w+\mathbf{S}_{bx}+\mathbf{S}_{by}+\mathbf{S}_f\big)d\Omega\,, \tag{1}$$

where N is the total number of edges for a cell ($N=3$ for triangular cells and $N=4$ for quadrilateral cells), ΔL is the length of an edge, n_x and n_y define the normal vector outward from the edge in x and y directions, respectively. The variables $\mathbf{F}$, $\mathbf{G}$, $\mathbf{S}_w$, $\mathbf{S}_{bx}$, $\mathbf{S}_{by}$, and $\mathbf{S}_f$ are the vectors,

defined as

$$\mathbf{W} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} hu \\ huu + \dfrac{gh^2}{2} \\ hvu \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} hv \\ huv \\ hvv + \dfrac{gh^2}{2} \end{bmatrix}, \quad \mathbf{S}_w = \begin{bmatrix} R - I \\ 0 \\ 0 \end{bmatrix},$$
$$\mathbf{S}_{bx} = \begin{bmatrix} 0 \\ -gh\dfrac{\partial z_b}{\partial x} \\ 0 \end{bmatrix}, \quad \mathbf{S}_{by} = \begin{bmatrix} 0 \\ 0 \\ -gh\dfrac{\partial z_b}{\partial y} \end{bmatrix}, \quad \mathbf{S}_f = \begin{bmatrix} 0 \\ -c_f\, u\sqrt{u^2+v^2} \\ -c_f\, v\sqrt{u^2+v^2} \end{bmatrix}, \tag{2}$$

where $c_f = g n_m^2 h^{-\frac{1}{3}}$. The variables h, u, v, z_b, and n_m, R, and I define the depth, velocities in x and y directions, bottom elevation, and Manning coefficient, rainfall, and infiltration, respectively—and the constant g is the gravity acceleration. In order to calculate the convective fluxes **F** and **G**, we employ the central-upwind (CU) scheme, which was originally developed in [26], because it is more efficient on modern hardware compared to other solvers (HLLC and Roe) provided in NUFSAW2D, see [23]. Note that our scheme is well-balanced and positivity-preserving. For simplicity, a first-order-spatial CCFV is applied in this work, so that further reconstruction techniques such as the MUSCL method are not required. For temporal discretisation, we use the Runge-Kutta second-order method with a semi-implicit treatment for $\mathbf{S}_f$, see [19, 22]. This procedure is briefly summarised as

$$\mathbf{K}^t = -\frac{\Delta t}{A}\sum_{i=1}^{N}\Big(\big(\mathbf{F}-\mathbf{S}_{bx}\big)\,n_x + \big(\mathbf{G}-\mathbf{S}_{by}\big)\,n_y\Big)_i^t \Delta L_i + \Delta t\,\mathbf{S}_w{}^t\,, \qquad \mathbf{W}^{t*} = \mathbf{W}^t + \mathbf{K}^t\,,$$
$$\mathbf{W}^{t*} \leftarrow \Pi^{-1}\,\mathbf{W}^{t*}\,, \qquad \Pi = 1 \;+\; g\,\Delta t\left[(1-\theta)\left(\frac{n_m^2\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}\right)_p^{t*} + \theta\left(\frac{n_m^2\sqrt{u^2+v^2}}{h^{\frac{4}{3}}}\right)_p^{(t*-1)}\right],$$
$$\mathbf{W}^{t+1} = \frac{1}{2}\Big(\mathbf{W}^t + \mathbf{W}^{t*} + \mathbf{K}^{t*}\Big)\,, \tag{3}$$

where Δt is the time step, t and $t*$ denote the time level, and θ is an implicitness coefficient set to 0.5. First, we explicitly calculate h^{t*}, hu^{t*}, and hv^{t*} without the friction terms. Dividing hu^{t*} and hv^{t*} by h^{t*}, we then obtain u^{t*} and v^{t*}. Using these two velocities, the variable Π^{-1} can be computed so that the friction terms are now taken into account and later used to update h^{t*}, hu^{t*}, and hv^{t*} until h^{t+1}, hu^{t+1}, and hv^{t+1} are obtained. The detail procedures are provided in [6, 7, 21].

3 Overview of Data Structures

3.1 Mesh Generation

In this work, the meshes are generated in 2DM format using the Surface-water Modeling System (SMS) software [27], containing both triangular and quadrilateral elements. First, the topography or bathymetry data are obtained in Digital Elevation Model (DEM) format, which is converted to shapefile (SHP) using the Geographic Information System (GIS) software. In this way, the domain boundaries as well as the topography data are obtained. The former is transformed into the map data in the SMS software, while the latter is converted to the scatter points. We then create polygons that divide the main domain inside the map data into several sub-domains—for example according to

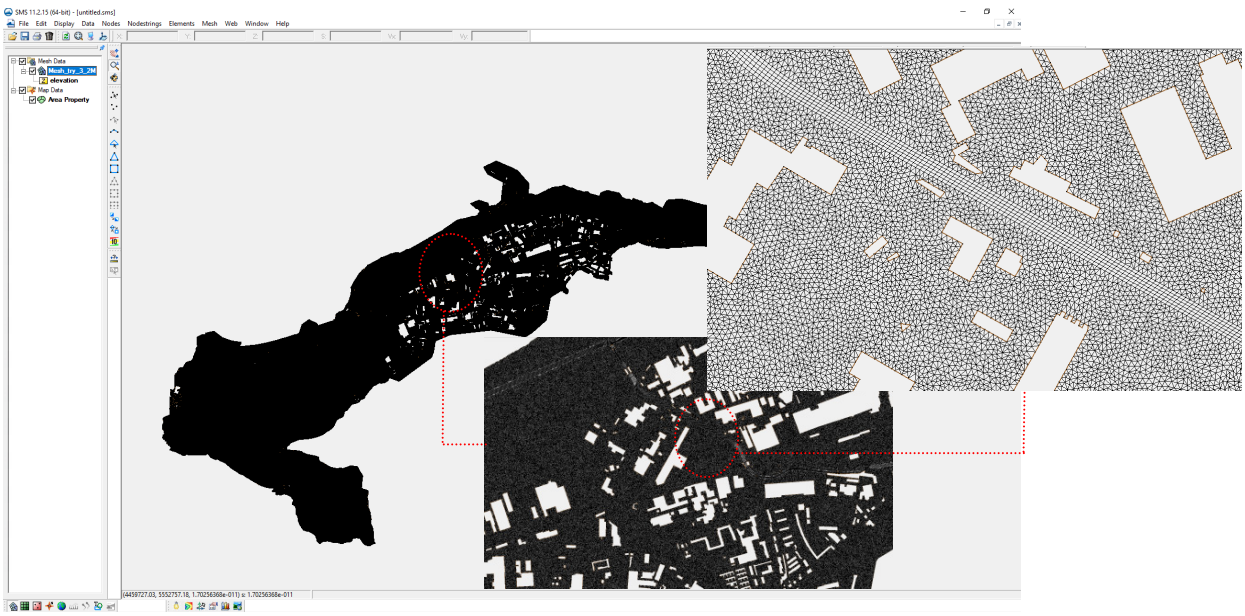


Figure 1: Mesh generation using the SMS software

land-use types such as rivers, lakes, etc.—while no polygon is assigned to buildings. Afterwards, we redistribute the vertices along such polygons and create the meshes, see Fig. 1, where the buildings are shown as voids in the domain representing the wall boundary conditions in our model. Now, the meshes have been generated, but they do not contain any contour values of the topography data yet. Therefore, the corresponding values are interpolated from the scatter points to the meshes using the inverse distance weighted method provided in the SMS software, yielding the final 2DM file. This file is later used as an input for NUFSAW2D. Note that the 2DM format was also used in other works, e.g. by the Flood Forecast Centre of the Federal State of Bavaria with the hydrodynamic model Hydro_AS-2D [28], to provide flood risk maps for 100-year return period and extreme events [29].

3.2 Fundamentals and Concept of Parallelisation

3.2.1 Domain Decomposition with Reordering Strategy

Before explaining the data structure in our code, it is important to give a general overview of how some reordering strategies employed in NUFSAW2D affect the communication patterns for the parallelisation. NUFSAW2D does not employ METIS library for decomposing the domain. Instead, it currently utilises two SFCs (Z-order and Hilbert) and a simple line-by-line (LBL) strategy to renumber the cells. Note that the main objectives are the same—no matter whether METIS, SFCs, or any other kind of reordering technique is used: to improve the data locality inside each node for ensuring better memory access patterns and to minimise the communication cost between nodes.

As an example, we now assume that four nodes are used to simulate the domain in Fig. 2, where the cells are allocated to the nodes using a block-distribution so that Nodes 0–3 receive 3, 3, 3, and 2 cells, respectively. Regarding SFCs, the cells are renumbered before we start detecting the outermost boundary lines of the domain, so that a bounding box is created. A quadtree-based gridding is then applied to the bounding box, creating four smaller squares of the same size. This gridding process continues until each smaller cell either captures a centre of a cell or leaves the

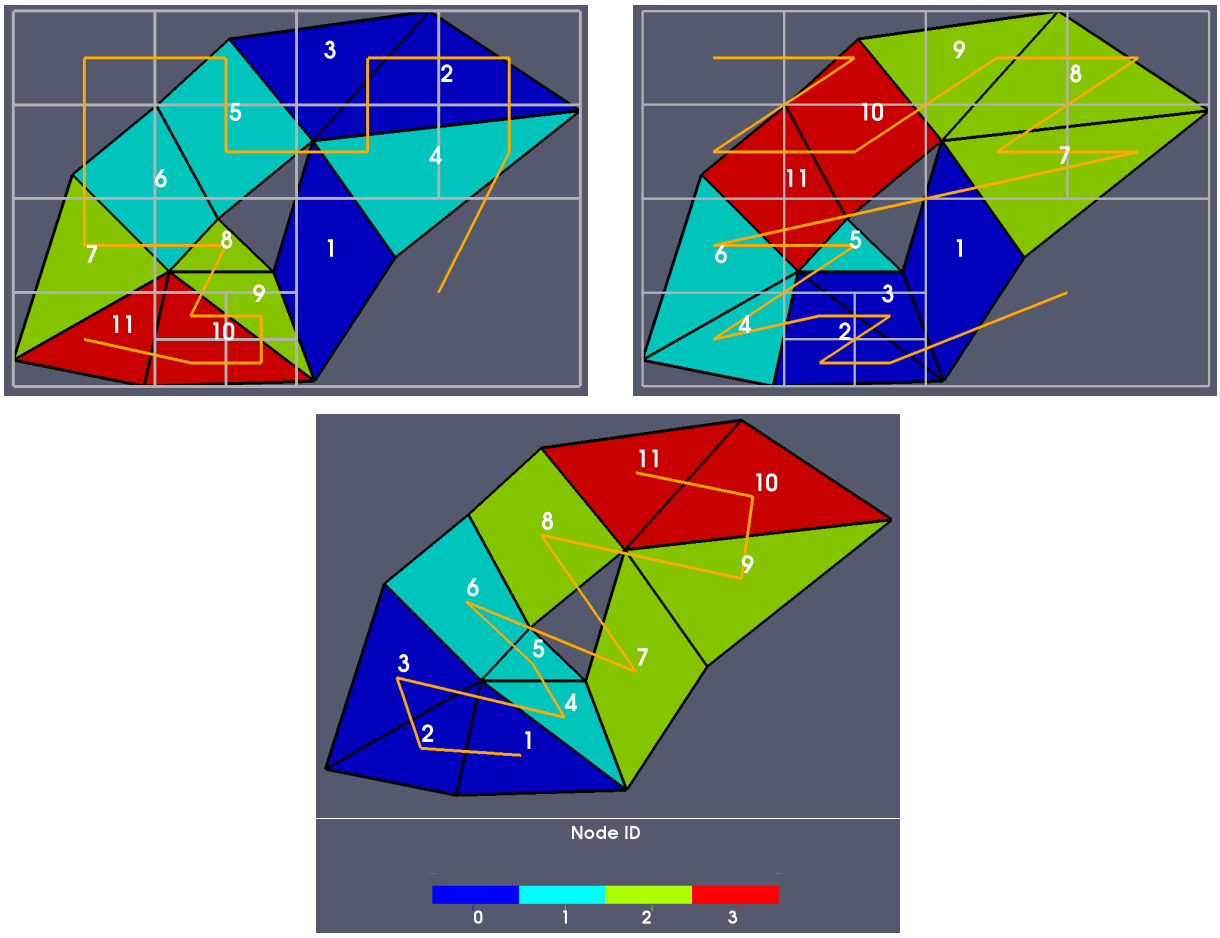


Figure 2: Domain decomposition with the Z-order (top left), Hilbert (top right), and LBL curves (bottom)

smaller cell blank. As this process is finished, the Z-order or Hilbert curve can easily be formed to the quadtrees. For the LBL curve, the idea is simply that the cells are reordered consecutively line-by-line, starting from the first line (selected by user). In this case, for example, the lowest boundary of the domain is selected (in such a way that the line covers three cells). After the first three cells are renumbered, the process continues for all the adjacent cells. They are connected to the same vertices to those of the first three cells, and so on until all cells have been indexed. Now, it can be seen that the LBL strategy produces a contiguous cell partition yielding the contiguous communication patterns among the nodes, whereas the Z-order and Hilbert curves do not. However, this does not necessarily mean that SFCs will always show the worst performance. In some cases, SFCs may give a better decomposition since it can provide the better data locality, thus supporting the contiguous memory access patterns inside each node. We will describe this in detail in the next section.

According to our observations, the suitability of the different strategies depends on the particular problem, which is why we cannot rely only on one strategy for decomposing our domain. Therefore, instead of investigating other reordering strategies that might lead to thousands of different scenarios, we decided to develop a flexible data structure in our code to take various future applications into account. This is how our idea emerged, to be explained in more detail in the next section.

3.2.2 Design of Data Structure

Prior to explaining the data structure in our code, let us first explain the basic concept employed in our parallel model. We employ the similar concept with [19], where a cell-edge reordering strategy was developed for SWEs simulations on rectangular meshes with OpenMP. Our computation is divided into two phases: edge-driven and cell-driven levels, where the former consists of the hydrostatic reconstruction, the convective fluxes computation, and the bed-slope terms calculation—and the latter covers the friction terms computation and the solution update for the next time level. The next step is to categorise the domain (in both levels) based on its complexity level. In the edge-driven level, for example, we distinguish boundary edges (edges corresponding only to one cell) from internal edges (edges corresponding to two cells), see Fig. 3 (showing four cells, three internal edges, and eight boundary edges), as the latter significantly requires more CPU time than the former [19]. In the cell-driven level—as a first-order-spatial CCFV model is used here, thus requiring no gradient computation as in [23]—one does not need to distinguish the internal cells from the boundary one since all cells have similar computational complexities. For simplicity, we consider Fig. 3 to have three types of computation: type A (internal edges), type B (boundary edges), and type C (cells). Note that the vertex in Fig. 3 is never used for computation, only for indexing the edges later on.

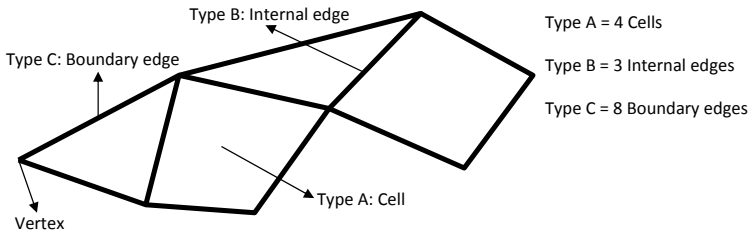


Figure 3: A simple domain formed by triangular and quadrilateral meshes

Regardless of the domain shape and of the domain decomposition, the arrays for the types A, B, and C (independent of one another) are always defined in our code as 1D configuration, see Stage 1 in Fig. 4. This concept can be interpreted as an approach, in which it is sufficient to, before starting a simulation, consider the total numbers of cells, internal edges, and boundary edges to be simulated—without having to think of how the domain is decomposed. This is what NUFSAW2D always assumes at the beginning of simulations. Note that our code will always number the boundary edges consecutively after all the internal edges. Afterwards, users can start to consider how they want to decompose such 1D-shaped arrays for the three types, e.g. by block-distribution, cyclic-distribution, or block-cyclic distribution. In the scope of this work, our code only employs the block-distribution. At this stage, NUFSAW2D requires information about the number of nodes and threads intended by the users. Once the according values are known (say four nodes and two threads for each node, for example), NUFSAW2D allocates (by means of block-distribution) the amount of arrays inside each node for each type A, B, and C, see Stage 2 in Fig. 4. At this stage, the communication patterns among the nodes and the memory access patterns inside each node—e.g. the relationship index between edges and cells as well as cells and edges—are still unknown. Now, we will turn to the challenging task of designing both patterns.

One can employ METIS, SFCs, or any other suitable tool to number the cells on the actual domain so that the real partition shape can be obtained. Note that such tools may also be employed for

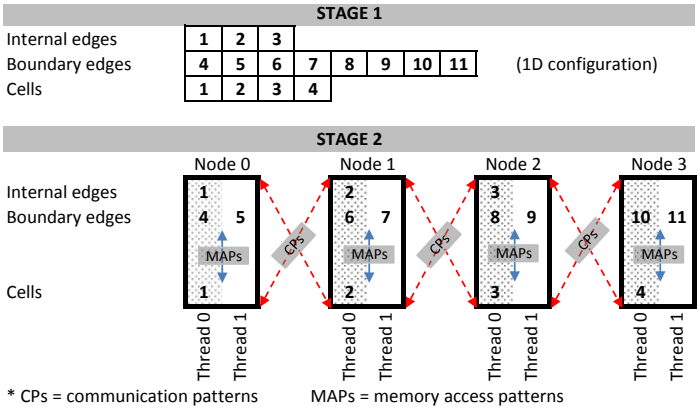
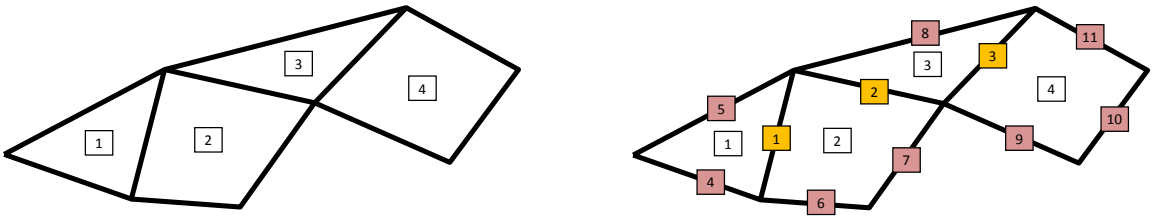


Figure 4: Parallelisation with block-distribution



Edge	MAPs	CPs
	Cell	Cell
1	1	2
2	2	3
3	3	4
4	1	-
5	1	-
6	2	-
7	2	-
8	3	-
9	4	-
10	4	-
11	4	-

* CPs = communication patterns

Cell	MAPs			CPs	
	Edge			Edge	
1	1	4	5	-	-
2	2	6	7	1	-
3	3	8	-	2	-
4	10	11	-	3	9

MAPs = memory access patterns

(see **STAGE 2**)

Edge-driven level

edge 1 needs cell 2 = Node 1 sends to Node 0
edge 2 needs cell 3 = Node 2 sends to Node 1
edge 3 needs cell 4 = Node 3 sends to Node 2

Cell-driven level

cell 2 needs edge 1 = Node 0 sends to Node 1
cell 3 needs edge 2 = Node 1 sends to Node 2
cell 4 needs edge 3 = Node 2 sends to Node 3
cell 4 needs edge 9 = Node 2 sends to Node 3

Figure 5: Cell numbering (top left), edge numbering (top right), and summary of the patterns (bottom)

numbering the edges (instead of the cells) on the actual domain. In this work, we will only explain the cell numbering. Let us now assume that applying a tool for the domain in Fig. 3 leads to a cell partition as shown in Fig. 5 (top left). If this partition is applied to Stage 2 in Fig. 4, the communication patterns are obviously contiguous, e.g. Node 1 communicates only with Nodes 0 and 2. To design the memory access patterns inside each node, two important things must be considered: (1) the edges access the cells, and (2) the cells access the edges. To this end, we provide a consecutive edge numbering, which must be associated to the numbered cells. This process is depicted in Fig. 5 (top right). First, we carry out the loop over the cells consecutively. Each cell records its vertices in counter-clockwise (CCW) direction—and once all corresponding vertices are detected, the edges are created in the order of such vertices. If a boundary edge is detected, the numbering process will be skipped to the next internal edge or to the next cell. For example, if the loop is currently performed at Cell 1, only one edge can be recognised as having two corresponding cells (this edge is then numbered as 1), while the other two edges will only correspond to one cell.

In this manner, we obtain the numbers for all the internal edges. After this process, the boundary edges are numbered through all cells consecutively.

For the sake of completeness, we will briefly summarise the communication and memory access patterns in Fig. 5. In the edge-driven level, the communications occur three times: (a) Node 1 sends Cell 2 to Node 0, (b) Node 2 sends Cell 3 to Node 1, and (c) Node 3 sends Cell 4 to Node 2. In the cell-driven level, the communications exist four times: (a) Node 0 sends Edge 1 to Node 1, (b) Node 1 sends Edge 2 to Node 2, (c) Node 2 sends Edge 3 to Node 3, and again (d) Node 2 sends Edge 9 to Node 3. This shows that the communication cost is not balanced, which becomes a major drawback. Nevertheless, for the data locality inside each node, the contiguous memory access patterns are obtained. In real applications with more complex meshes, the communication patterns may become more irregular—and none of the strategies is generally best suited to obtain the exact contiguous communication patterns. However, in the scope of memory access patterns, the contiguity level of such patterns can still be ensured using our strategy for the 1D array configuration in Fig. 4 and the edge numbering in Fig. 5. We also observed that the effects of the memory access patterns in most of our implementations were more significant than the communication patterns of our results. Therefore, as a practical guide to access the effectiveness of the reordering tools, we employed a simple criterion that can be written as

$$QI = \sum_{j=1}^{J} |2\ E_{ID} - C1_{ID} - C2_{ID}|\ , \tag{4}$$

where QI is the quality indicator, E_{ID} is the edge ID, $C1_{ID}/C2_{ID}$ are the IDs of two cells corresponding to the edge, and J is the total number of internal edges. QI indicates the contiguity level of the memory access patterns: the greater the value of QI, the worse the resulting pattern. Here, we only consider QI for internal edges due to simplicity. In addition, the total number of internal edges is relatively larger than that of boundary edges (for real applications). Lastly, we would like to reiterate that Eq. 4 might not be valid for some specific cases—e.g. pure MPI simulations due to more communication costs—which is why further investigations may be needed.

3.2.3 Load Balancing Strategy for Wet-Dry Problems

Even if an ideal domain decomposition—with which the numbers of edges/cells are uniformly distributed into the nodes—would be obtained, this would not guarantee balanced load-distributions among the processors when wet-dry problems appear, as wet edges/cells exhibit more computational efforts than dry edges/cells. This is a typical load imbalance problem in SWEs simulations, and it cannot be predicted exactly during runtime. To this end, a novel strategy, namely weighted-dynamic load balancing (WDLB), was proposed in [19] for both edge and cell-driven levels. Using a second-order scheme in the edge-driven level, the wet edges were typically found to be 1.7–2 $\times$ more expensive than dry edges. In the cell-driven level, regardless of the first/second-order spatial scheme, wet cells typically turned out to be 1.9–2.1 $\times$ more expensive than dry cells. In this work, we follow the WDLB technique of [19], but only for the cell-driven level, because a first-order spatial scheme is employed here—thus requiring no WDLB technique for the edge-driven level.

To give a clear but simple representation of our load balancing procedure, we consider a domain formed by 24 rectangular cells (note that another cell shape is also applicable), where Cells 1–4, 9–11, and 15 at a certain time level contain water, thus being wet cells, see Fig. 6 (physical domain). We assume to use four nodes (each node with two threads). Following [19], we select a factor of 2 for the weighting of the wet cells in comparison to the dry ones. Without the WDLB technique, the

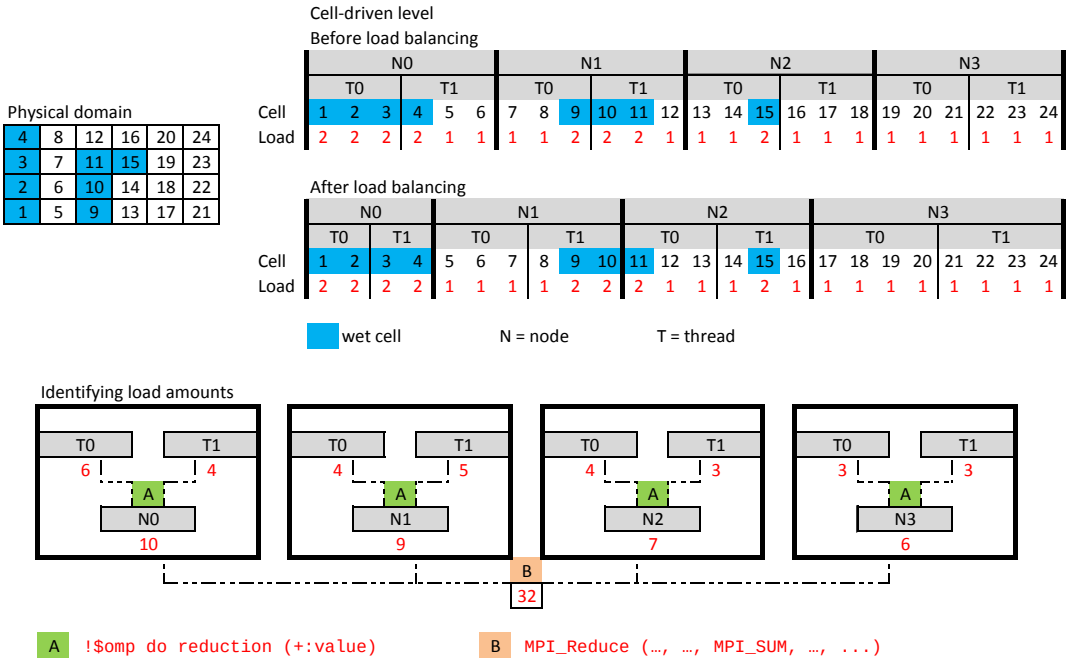


Figure 6: Load balancing procedure

block-distribution applies to the nodes (for MPI), and the common directive `!$omp do` can easily be employed to distribute the loop among the threads (for OpenMP). However, this situation suffers from imbalanced load distributions. For instance, Nodes 0–3 have the load units of 10, 9, 7, and 6, respectively. Furthermore, there appear to be load imbalances among the threads: in Node 0, for example, Thread 0 obtains 6 load units while Thread 1 receives 4 load units. To this end, the WDLB procedures are applied. First, the load amounts among threads inside each node are detected and summed up. This can be done quite simply using the directive `!$omp do reduction(+:value)`. Once each node knows its load amount, it must communicate with the others to determine the total load of the entire domain, yielding 32 load units. This can also be achieved quite simply by employing the directive `MPI_Reduce(...,MPI_SUM,...)`.

Now, the reverse procedures are carried out: the 32 load units must be distributed to four nodes—and after that, the loads assigned to each node are distributed to each thread. Both in MPI and OpenMP levels, procedures like this must be performed consecutively by checking the load attributed to each cell (wet/dry one) sequentially. Therefore, these ways cannot be parallelised, and it may not be efficient to apply the WDLB in every single time level. Also, especially in the MPI level, the dynamic array allocations for each node can lead to a significantly increased overhead if the WDLB is applied in each time level. For this reason, we apply the WDLB for 1/50 step of the total time levels for OpenMP (as also done in [19]) and only perform the WDLB for 1/100 step of the total time levels for MPI. This combination turned out to be the most efficient one in all of our implementations. It is important to point out that we do not, after the WDLB, use the directive `!$omp do` to distribute the loop among threads in the OpenMP level.

4 Test Cases

The domain configurations are summarised in Table 1. The quadrilateral elements are employed along the river for Case 1 and at the northern part (as the boundary condition) for Case 2. Meanwhile, the other parts are covered with triangular meshes.

	Model area	Tot. numb. vertices	Tot. numb. edges (intern./ bound.)	Tot. numb. cells (quad./ triang.)	Average cell area
Case 1	11.5 km^2	1,677,120	4,911,114 (4,852,550/ 58,564)	3,233,010 (64,634/ 3,168,376)	3.56 m^2
Case 2	37.84 km^2	1,738,554	5,136,020 (5,131,069/ 4,951)	3,397,467 (74,688/ 3,322,779)	11.14 m^2

Table 1: Domain configurations for Cases 1 and 2

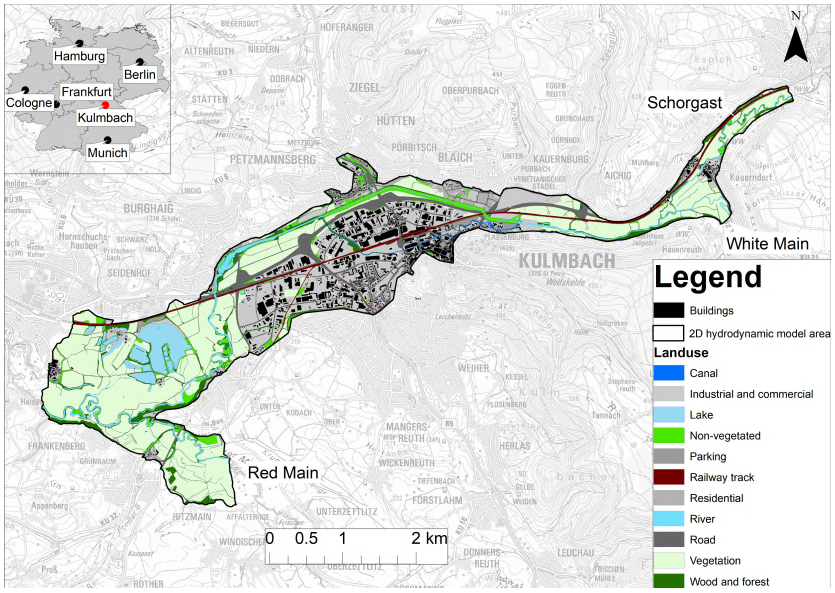


Figure 7: Land use map (Source: Bavarian Environment Agency and Water Management Authority, Hof)

4.1 Case 1: 2011 Flood Case in Kulmbach (Germany)

We aim to simulate a real flood case in Kulmbach (Germany), which occurred on 13–14 January 2011. Two rivers cross the city at the upstream part (White Main and Schorgast). At the downstream part, the river Red Main merges with the White Main, forming the river Main—which is the longest tributary of the river Rhine. An according sketch can be seen in the land use map in Fig. 7. In general, the land use map covers 62% of agricultural land (floodplains and grassland), 7% of water bodies (rivers and lakes), 26% of urban area, and 5% of forests. The topography contour was obtained in DEM format from the Water Management Authority (Hof). The historical discharges were collected by the Bavarian Hydrological Services for some flood events in 2005, 2006, 2011,

and 2013—and we selected the data for 2011 as inputs for boundary conditions of our model, representing the three aforementioned rivers and some flow point sources around the urban area. All information about the discharges, roughness coefficient, etc. were presented in [5, 30, 31].

No information was given about the initial conditions along the river domain. Therefore, to achieve a proper representation of the initial conditions, we first set the constant boundary conditions for the three rivers to the lowest discharges recorded (as the base flows) and ran our simulations until a steady state condition was reached. Then, we used the results as an initial condition for the actual simulation, to obtain the results during the event on 13–14 January 2011. In the scope of this work, the total simulation time investigated is 40 hours.

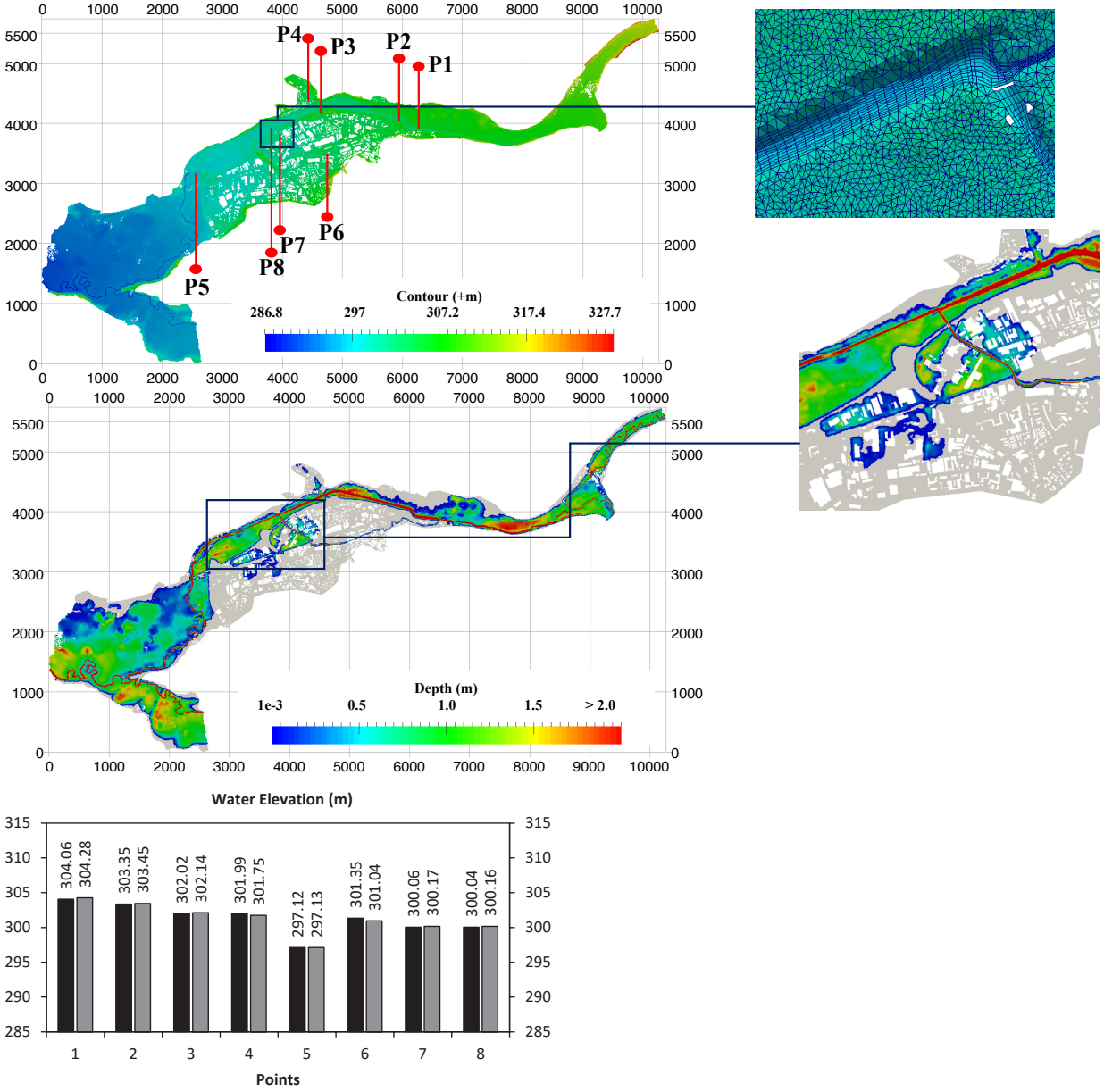


Figure 8: Case 1: Meshes of domain (top) and numerical results (bottom). Elevation units are shown to be above the mean sea level (Measurement source: Water Management Authority, Hof)

Our numerical results are shown in Fig. 8. Note that we plotted our results with a maximum colour range of 2 m in order to provide a clear representation of the inundation depth near the urban area, not the river depth. Thus, the river parts can be seen clearly. Our results show that the urban area is inundated by water with depths of up to 1.6 m. This is in accordance with those investigated in [5,30]. We also compared our results with eight sets of observed data. We are again able to show that NUFSAW2D is quite accurate in predicting the inundation areas for the eight locations—with only non-significant differences to the measurement data—thus proving our model is able to accurately simulate wet-dry mechanisms on rough beds and complex topography.

4.2 Case 2: 2011 Tohoku Tsunami recorded in Hawaii (USA)

We consider the real case of the 2011 Tohoku tsunami event, which was recorded in Hawaii (USA). The raw data including the bathymetry contour and the incident wave (as the boundary conditions) are available from [32]. As done in [14], the original bathymetry data greater than 30 m were flattened to avoid the phase different of the incident wave. The bathymetry map covers the hill (western part), the coastal land area (southern part), the ocean, and the breakwater that indicates the harbour part. Three measurement points were given: Hilo tide station, HAI1125 (harbour entrance), and HAI1126 (inside the harbour), see Fig. 9. The uniform Manning coefficient of 0.025 $\mathrm{sm}^{-1/3}$ was used. In the scope of this work, the total simulation time investigated is 13 hours.

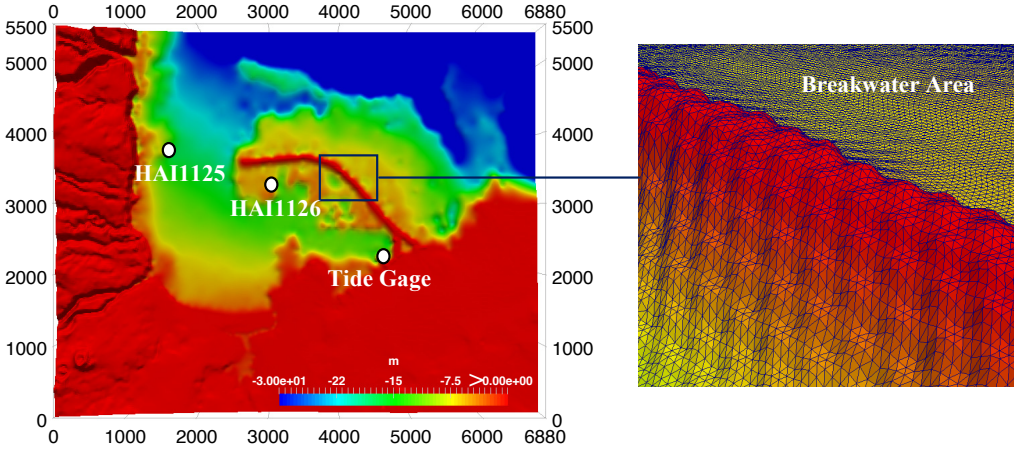


Figure 9: Case 2: Bathymetry of domain

Fig. 10 shows the visualisations of our results. At around 9.33 h, the velocity outside the harbour reaches 0.8 m/s, while it reaches 1 m/s at the harbour entrance. Inside the harbour, the velocity becomes higher—up to 1.3 m/s. Up to now, the overall velocity distribution is still uniform. At 10.67 h, however, the velocity becomes non-uniformly distributed, reaching up to 3.5 m/s outside the harbour and approximately 2.8 m/s inside the harbour. Meanwhile, the velocity at the harbour entrance increases only slightly, to 1.3 m/s. After 13 h, the velocity decreases to 1 m/s inside the harbour and 2 m/s outside the harbour. We also plot the maximum magnitudes for the entire simulation, showing that the velocity reaches up to 5 m/s near the coastline (the eastern part), while the maximum velocity inside the harbour only reaches approximately 2.8 m/s. This accounts for an extremely sensitive spatial distribution of velocity magnitude, which is in line with that shown in [14].

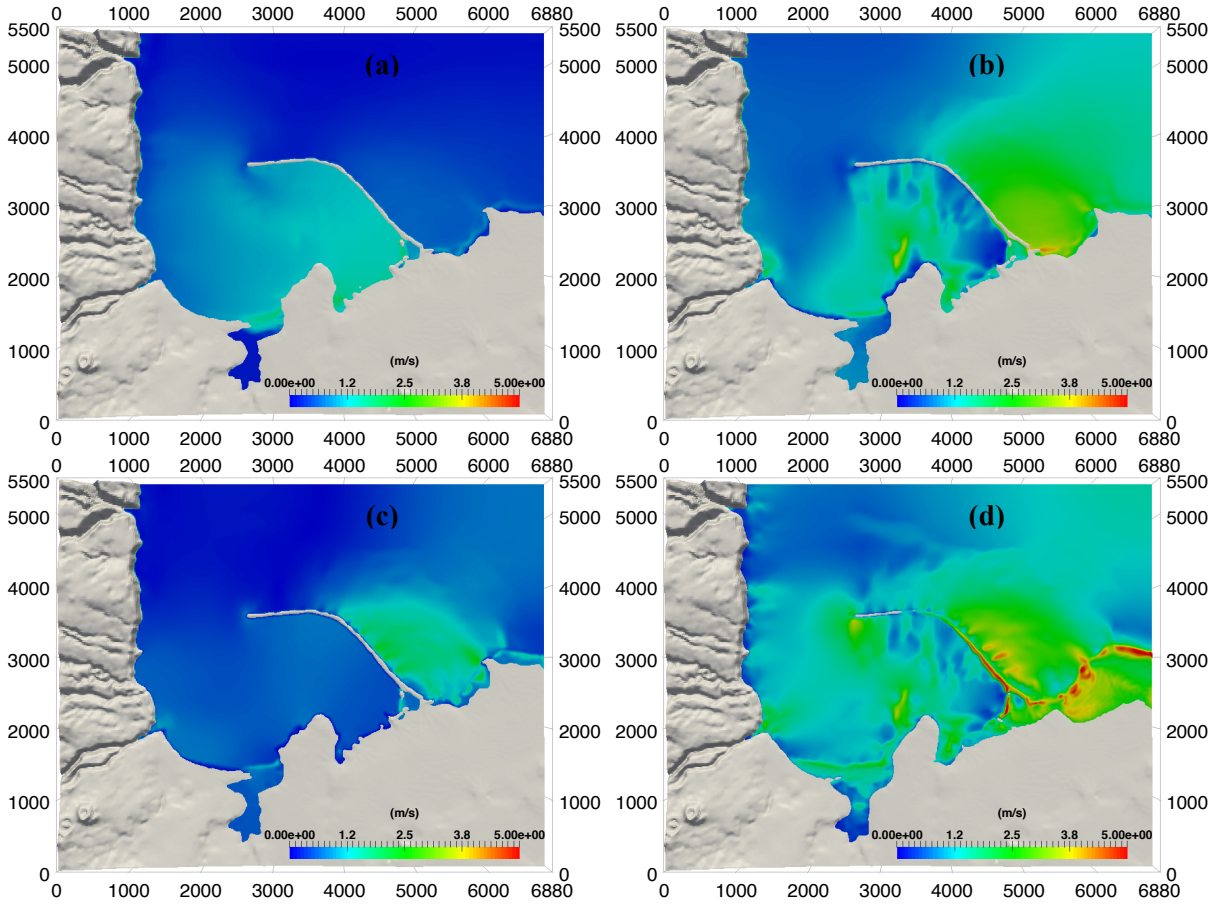


Figure 10: Case 2: Simulation results for velocities (a) 9.33 h (b) 10.67 h (c) 13 h (d) maximum velocity during 13 h

Finally, Fig. 11 shows our results at the three measurement points. At the tide gage, our model can accurately capture the first incoming wave at around 8.2 h. We are also able to capture the lowest elevation approximately at 8.4 h. Afterwards, the fluctuations of water elevation are predicted properly until 13 h. The velocity of the incoming wave is computed accurately inside the harbour and slightly underestimated at the harbour entrance. After 10 h, however, our model becomes sufficiently accurate at both locations. Fig. 11 shows that the current inside the harbour predominantly flows in east-west direction, in contrast to the harbour entrance, where it mostly flows in north-south direction. Overall, our model shows a good agreement with that provided in [32].

4.3 Performance Investigation

NUFSAW2D was written in Fortran and compiled using Intel Fortran 17, where the double-precision arithmetic (64-bit) was used. We investigated the performance of NUFSAW2D on CoolMUC-2 cluster with AVX2 (Intel Xeon E5-2697 v3/Haswell), provided by the Leibniz Supercomputing Centre (LRZ) [33]. In total, this cluster has 10,752 cores (384 nodes each with 28 cores and 64 GB RAM). However, the maximum access for a single job is limited to 60 nodes (1,680 cores). So far, we have tested our code on up to 336 cores with three scenarios: (1) 1–12 nodes$\times$28 threads, (2) 1–24 nodes$\times$14 threads, and (3) 1–48 nodes$\times$7 threads. In this work, we only performed the

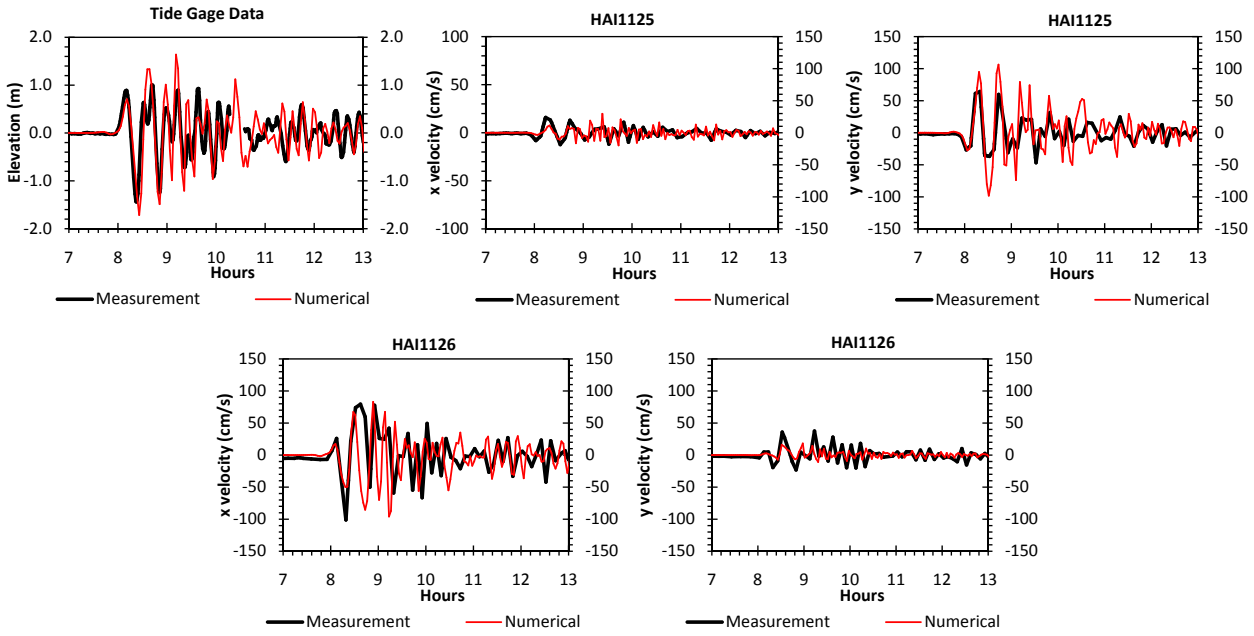


Figure 11: Case 2: Simulation results at the three measurement points during 13 h

strong scaling and observed the speed-up, since only one mesh-configuration was provided for each case (fixed size). The performance metric of our code is given in million cells per second per core (Mcell/s/core), which is a comparison between the numbers of cells for a total number of calculation steps that can be processed per unit of time using one core.

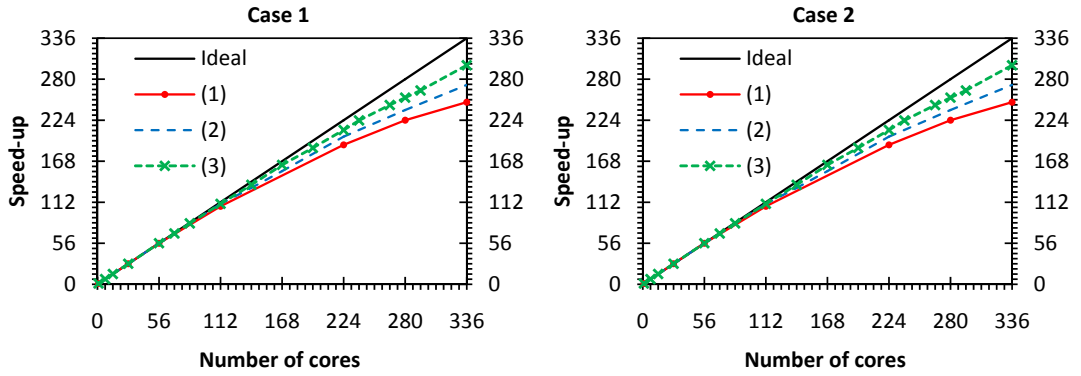


Figure 12: Performance results: (1) 1–12 nodes×28 threads (2) 1–24 nodes×14 threads (3) 1–48 nodes×7 threads

Fig. 12 shows that the results of Cases 1 and 2 are quite similar, although Case 2 obtains slightly higher efficiency. In Case 1, our model yields efficiencies of 74%, 81%, and 89% for the configurations 1–3, respectively, whereas in Case 2 the performances turn out to be 76%, 83%, and 91%. These differences are due to fact that the effects of the wet-dry problems in Case 1 are more dominant than in Case 2. For Case 1, we observed that before reaching the almost-stagnant inundation (as shown in Fig. 8), the wet-dry problems appeared dominantly in the middle of the domain (the urban area); this area, which was previously wet, grew smaller and smaller during

the runtime. The WDLB applied led to more overheads in Case 1 than in Case 2, in which the wet-dry problems exist only near the coastline. Despite these performance losses, we can affirm that our model is still quite efficient in handling such wet-dry phenomena. We observed that the configuration (1) suffers from the performance degradation of 24–26%, which is the least efficient outcome. Meanwhile, the configuration (3) achieves the best performance with a performance degradation of only 9–11%. Finally, we would like to point out that our code can achieve a metric of approximately 5.5 Mcell/s/core (for a single-core computation) on this machine, which corresponds to about 34% of the theoretical peak performance.

4.4 Current Visualisation and Outlook

In order to gain more insight from the occurring phenomena and to help decision makers understand them, the simulation results have to be visualised. One of the suitable options from among the masses of visualisation tools is the Visualisation Toolkit (VTK) format. In this work, NUFSAW2D produces results in VTK files (with legacy format), which can simply be visualised using a front end, e.g. Paraview. With increasing amounts of data, however, it gets more and more difficult to visualise the results. To this end, the Chair for Computation in Engineering (CiE) developed a technique to quickly and efficiently display large scale simulation data, namely the Sliding Window technique [24]. The Sliding Window concept allows to view simulation data with varying granularity while keeping the amount of data visualised constant. This is achieved by determining a subsection of the simulation domain (a window). With a larger window, a view of a coarse representation of the domain selected is obtained, while a smaller window displays the selection in more detail, allowing users to observe local phenomena with better resolution.

This concept was initially employed for the CiE's 3D massive parallel simulation code [24], for which the Sliding Window technique fitted the hierarchical data structure of the code perfectly. The domain is generated starting from a singular grid or mesh of the complete domain, then gradually refined until the simulation domain reaches a suitable resolution. The coarser representations are not discarded but are readily available for the visualisation. Since the amount of data visualised is constant, the Sliding Window is used in an online fashion, i.e. during runtime of the code. For this, users may demand a visualisation window through a front end, which connects to the management instance of the code. This manager then selects the data that fit the window with their respective granularity from the parallel processes, assembles them, and sends them back to the user's front end.

In an overhaul of the code's I/O functionality, the result files in VTK format were replaced by one single HDF5 file in [25]. This single file contained, in addition to the raw data, the hierarchical data structure—and it allowed the Sliding Window technique to work in an offline fashion for all time steps intended. Similar to the online procedure, the amount of data selected is limited, allowing even the largest datasets to be visualised quickly. Currently, we are developing a tool that is able to convert any kind of VTK outputs, even unstructured ones, into a compatible HDF5 file containing representations of the domain in varying resolutions. The general workflow involves a resampling of the raw data into a structured grid, followed by gradually building up coarser representations to set up the hierarchy from the ground up, see the concept in Fig. 13. This makes the sliding window technique easily available for codes with VTK outputs, by providing a standalone tool for high-performance visualisation.

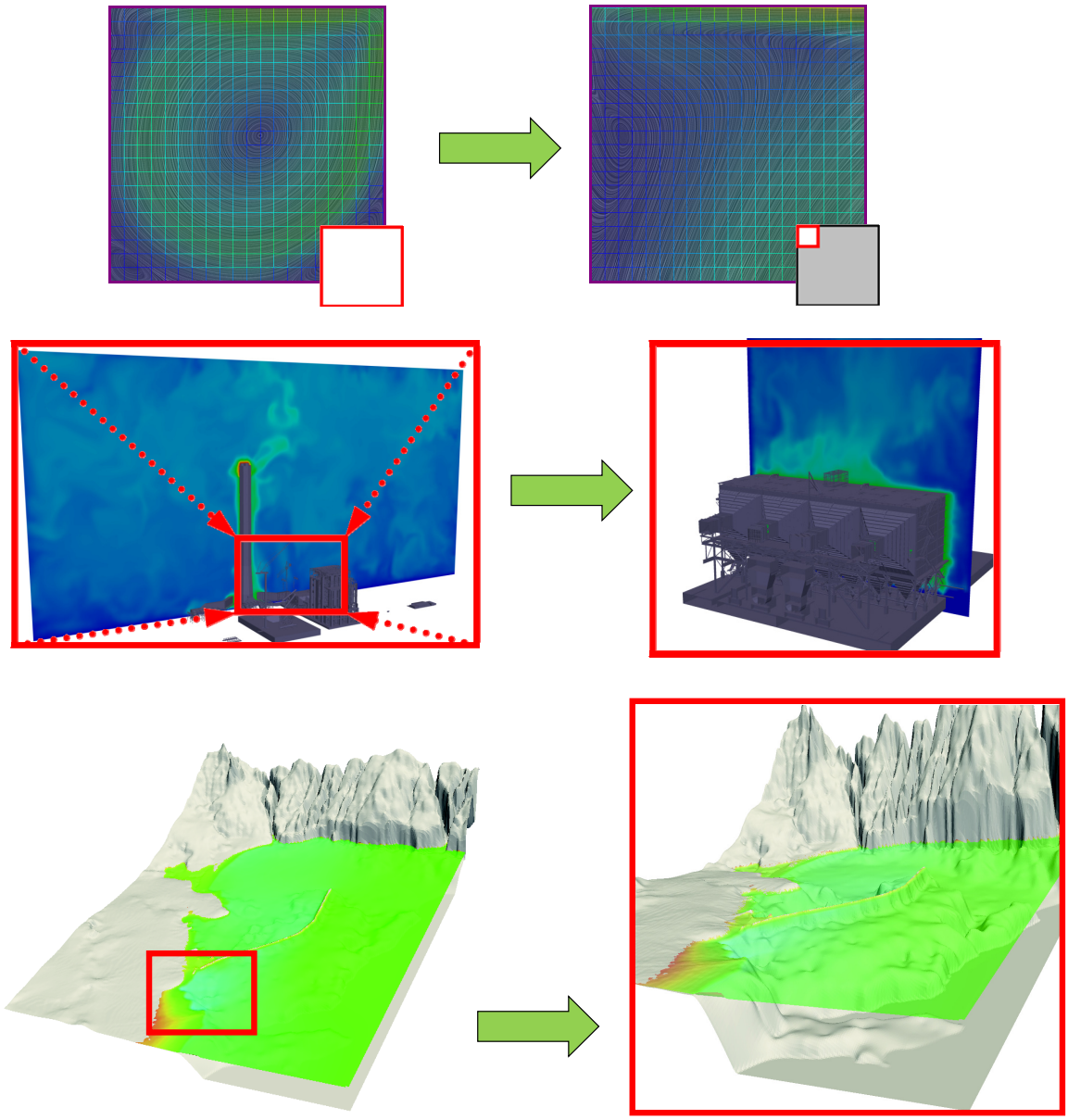


Figure 13: As high-resolutions hinder interactive visual data exploration, the Sliding Window technique allows users to move through and zoom in/out the computational domains, where the amount of details increases/decreases seamlessly while the number of data points visualised stays constant; the first two figures were taken based on the works of [24, 25] for thermal simulations—and the last figure shows an outlook of the current implementation with NUFSAW2D for tsunami simulation (Case 2)

5 Conclusion

A hybrid OpenMP-MPI parallel simulation for solving the 2D SWEs was performed using NUFSAW2D. The code was tested against two benchmark cases dealing with wet-dry simulations on complex topographies, and accurate results were obtained. We presented a general overview of some reordering strategies supported by our code, such as Z-order curve, Hilbert curve, and the

LBL technique and we showed that such strategies may end up with numerous possibilities of decomposing and assigning domains into nodes for a parallel simulation. Therefore, we designed the data structure in our code in such a way that it flexibly supports any kind of reordering tool, which is essentially required to be able to provide the communication patterns among the nodes and the memory access patterns inside each node. Our data structure in both edge-driven and cell-driven levels consisted of contiguous 1D arrays, where a block-distribution was utilised to decompose such arrays at the beginning of simulations. Later, the decomposition changed during runtime based on the WDLB technique implemented as soon as the wet-dry problems came into existence. Our simple load balancing strategy was proven quite efficient in attempts to anticipate the load imbalance issues due to wet-dry problems—with an efficiency of up to 91% with 336 cores. Currently, we are working on ways to add more features of reordering strategies such as implementing Peano-SFC as well as the result of METIS library, not only for cell numbering but also for edge numbering. We are also investigating the application of the WDLB with both types of numbering strategies in the edge-driven level for the second-order CCFV model.

Acknowledgements

Bobby Minola Ginting gratefully acknowledges the DAAD (German Academic Exchange Service), who supports his research in the scope of Research Grants—Doctoral Programmes in Germany 2015/16 (57129429). Punit Kumar Bhola and Markus Disse thank the Bavarian Water Authority and Bavarian Environment Agency in Hof for providing the data of Case 1—and gratefully acknowledge the German Federal Ministry of Education and Research for providing the funding in the scope of FloodEvac project (FKZ 13N13196). The compute and data resources provided by the Leibniz Supercomputing Centre are acknowledged. The contributions of Ugurcan Sari and Mengjie Zhao as the students in this work are highly appreciated.

References

[1] M. Campolo, A. Soldati, and P. Andreussi, "Artificial Neural Network Approach to Flood Forecasting in the River Arno," *Hydrolog Sci J*, vol. 48, no. 3, pp. 381–398, 2003, https://dx.doi.org/10.1623/hysj.48.3.381.45286.

[2] C.-T. Cheng, W.-C. Wang, D.-M. Xu, and K. W. Chau, "Optimizing Hydropower Reservoir Operation Using Hybrid Genetic Algorithm and Chaos," *Water Resour Manage*, vol. 22, pp. 895–909, 2008, https://dx.doi.org/10.1007/s11269-007-9200-1.

[3] B. M. Ginting, D. Harlan, A. Taufik, and H. Ginting, "Optimization of Reservoir Operation Using Linear Program, Case Study of Riam Jerawi Reservoir, Indonesia," *Int J River Basin Manag*, vol. 15, no. 2, pp. 187–198, 2017, https://dx.doi.org/10.1080/15715124.2017.1298604.

[4] J. Parkinson and M. O., *Urban Stormwater Management in Developing Countries*. Second edn. IWA Publishing: London, 2005.

[5] P. K. Bhola, J. Leandro, and M. Disse, "Framework for Offline Flood Inundation Forecasts for Two-dimensional Hydrodynamic Models," *Geosciences*, vol. 8, no. 9, pp. 1–19, 2018, https://dx.doi.org/10.3390/geosciences8090346.

[6] B. M. Ginting and R.-P. Mundani, "Artificial Viscosity Technique: A Riemann-solver-free Method for 2D Urban Flood Modelling on Complex Topography," in *Advances in Hydroin-*

formatics, G. P., C. J., and C. G., Eds. Singapore: Springer, 2018, ch. 4, pp. 51–74, https://dx.doi.org/10.1007/978-981-10-7218-5_4.

[7] B. M. Ginting, "A Two-dimensional Artificial Viscosity Technique for Modelling Discontinuity in Shallow Water Flows," *Appl Math Model*, vol. 45, pp. 653–683, 2017, https://dx.doi.org/10.1016/j.apm.2017.01.013.

[8] X. Xilin and Q. Liang, "A New Efficient Implicit Scheme for Discretising the Stiff Friction Terms in the Shallow Water Equations," *Adv Water Resour*, vol. 117, pp. 87–97, 2018, https://dx.doi.org/10.1016/j.advwatres.2018.05.004.

[9] J. Zhao, I. Özgen-Xian, D. Liang, T. Wang, and R. Hinkelmann, "An Improved Multislope MUSCL Scheme for Solving Shallow Water Equations on Unstructured Grids," *Comput Math Appl*, vol. 77, no. 2, pp. 576–596, 2019, https://dx.doi.org/10.1016/j.camwa.2018.09.059.

[10] J. M. Hervouet, "A High-Resolution 2-D Dam-break Model Using Parallelization," *Hydrol Processes*, vol. 14, no. 13, pp. 2211–2230, 2000, https://dx.doi.org/10.1002/1099-1085(200009)14:13<2211::AID-HYP24>3.0.CO;2-8.

[11] J. C. Neal, T. J. Fewtrell, and M. Trigg, "Parallelisation of Storage Cell Flood Models Using OpenMP," *Environ Modell Software*, vol. 24, no. 7, pp. 872–877, 2009, https://dx.doi.org/10.1016/j.envsoft.2008.12.004.

[12] B. F. Sanders, J. E. Schubert, and R. L. Detwiler, "Parbrezo: A Parallel, Unstructured Grid, Godunov-type, Shallow-water Code for High-resolution Flood Inundation Modeling at the Regional Scale," *Adv Water Resour*, vol. 33, no. 12, pp. 1456–1467, 2010, https://dx.doi.org/10.1016/j934.advwatres.2010.07.007.

[13] S. Tanaka, S. Bunya, J. J. Westerink, C. Dawson, and R. A. Luettich Jr., "Scalability of an Unstructured Grid Continuous Galerkin based Hurricane Storm Surge Model," *J Sci Comput*, vol. 46, pp. 329–358, 2011, https://dx.doi.org/10.1007/s10915-010-9402-1.

[14] M. E. M. Arcos and R. J. LeVeque, "Validating Velocities in the GeoClaw Tsunami Model Using Observations near Hawaii from the 2011 Tohoku Tsunami," *Pure Appl Geophys*, vol. 17, pp. 849–867, 2014, https://dx.doi.org/10.1007/s00024-014-0980-y.

[15] O. Meister, K. Rahnema, and M. Bader, "Parallel Memory-efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells," *ACM T Math Software*, vol. 43, no. 3, pp. 1–27, 2016, https://dx.doi.org/10.1145/2947668.

[16] R. Wittmann, H.-J. Bungartz, and P. Neumann, "High Performance Shallow Water Kernels for Parallel Overland Flow Simulations based on FullSWOF2D," *Comput Math Appl*, vol. 74, no. 1, pp. 110–125, 2017, https://dx.doi.org/10.1016/j.camwa.2017.01.005.

[17] W. Lai and A. A. Khan, "A Parallel Two-dimensional Discontinuous Galerkin Method for Shallow-water Flows Using High-resolution Unstructured Meshes," *J Comput Civ Eng*, vol. 31, no. 3, pp. 1–10, 2017, https://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000647.

[18] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM J Sci Comput*, vol. 20, no. 1, pp. 359–392, 1998, https://dx.doi.org/10.1137/S1064827595287997.

[19] B. M. Ginting and R.-P. Mundani, "Parallel Flood Simulations for Wet–dry Problems Using Dynamic Load Balancing Concept," *J Comput Civ Eng-ASCE*, vol. 33, no. 3, p. 04019013, 2019, https://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000823.

[20] B. M. Ginting, R.-P. Mundani, and E. Rank, "Parallel Simulations of Shallow Water Solvers for Modelling Overland Flows," in *13th International Conference on Hydroinformatics*, ser. EPiC Series in Engineering, G. La Loggia, G. Freni, V. Puleo, and M. De Marchis, Eds., vol. 3. EasyChair, 2018, pp. 788–799, https://dx.doi.org/10.29007/wdn8.

[21] B. M. Ginting, "Central-upwind Scheme for 2D Turbulent Shallow Flows Using High-resolution Meshes with Scalable Wall Functions," *Comput Fluids*, vol. 179, pp. 394–421, 2019, https://dx.doi.org/10.1016/j.compfluid.2018.11.014.

[22] B. M. Ginting and H. Ginting, "Hybrid Artificial Viscosity–Central-upwind Scheme for Recirculating Turbulent Shallow Water Flows," *J Hydraul Eng-ASCE*, vol. 145, no. 12, p. 04019041, 2019, https://dx.doi.org/10.1061/(ASCE)HY.1943-7900.0001639.

[23] B. M. Ginting and R.-P. Mundani, "Comparison of Shallow Water Solvers: Applications for Dam-break and Tsunami Cases with Reordering Strategy for Efficient Vectorization on Modern Hardware," *Water*, vol. 11, no. 4, p. 639, 2019, https://dx.doi.org/10.3390/w11040639.

[24] R.-P. Mundani, J. Frisch, V. Varduhn, and E. Rank, "A Sliding Window Technique for Interactive High-performance Computing Scenarios," *Adv Eng Softw*, vol. 84, pp. 21–30, 2015, https://dx.doi.org/10.1016/j.advengsoft.2015.02.003.

[25] C. Ertl, J. Frisch, and R.-P. Mundani, "Design and Optimisation of an Efficient HDF5 I/O Kernel for Massive Parallel Fluid Flow Simulations," *Concurrency Computat: Pract Exper*, vol. 29, pp. 1–12, 2017, https://dx.doi.org/10.1002/cpe.4165.

[26] A. Kurganov and G. Petrova, "A Second-order Well-balanced Positivity Preserving Central-upwind Scheme for the Saint-Venant System," *Commun Math Sci*, vol. 5, no. 1, pp. 133–160, 2007, https://dx.doi.org/10.4310/CMS.2007.v5.n1.a6.

[27] https://xmswiki.com/wiki/SMS:2D_Mesh_Files_*.2dm, accessed: February 2019.

[28] M. Nujić, *Benutzerhandbuch, Hydro_AS-2D, 2D-Strömungsmodell für die wasserwirtschaftliche Praxis (in German)*. HydrotecIngenieurgesellschaft für Wasser und Umwelt mbH : Aachen, 2014.

[29] M. Disse, I. Konnerth, P. K. Bhola, and J. Leandro, "Unsicherheitsabschätzung für die Berechnung von Dynamischen Überschwemmungskarten – Fallstudie Kulmbach (in German)," in *Vorsorgender und nachsorgender Hochwasserschutz*, S. Heimerl, Ed. Wiesbaden, Germany: Springer Vieweg, 2018, pp. 350–357, https://dx.doi.org/10.1007/978-3-658-21839-3_50.

[30] P. K. Bhola, N. Bhavna, J. Leandro, S. N. Rao, and M. Disse, "Flood Inundation Forecasts Using Validation Data Generated with the Assistance of Computer Vision," *J Hydroinform*, vol. 21, no. 2, pp. 240–256, 2018, https://dx.doi.org/10.2166/hydro.2018.044.

[31] P. K. Bhola, B. M. Ginting, J. Leandro, K. Broich, R. P. Mundani, and M. Disse, "Model Parameter Uncertainty of a 2D Hydrodynamic Model for the Assessment of Disaster Resilience," in *EnviroInfo*, 2018.

[32] https://coastal.usc.edu/currents_workshop/index.html, accessed: February 2019.

[33] https://lrz.de, accessed: February 2019.