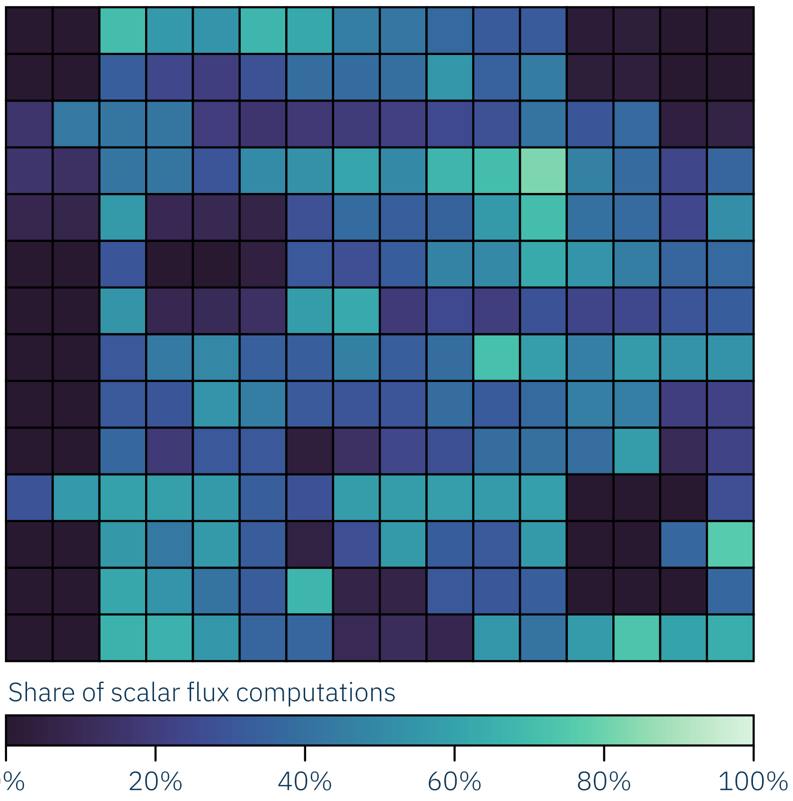
Technische Universität Berlin
Faculty VI – Planning Building Environment | Inst. of Civil Engineering
Chair of Water Resources Management and Modeling of Hydrosystems
Prof. Dr.-Ing. R. Hinkelmann

Robert Egel

robert.egel@campus.tu-berlin.de

A Novel Approach to Local Time Stepping using the Shallow Water Equations Model hms^{++}

Bachelor thesis



First examiner: Prof. Dr.-Ing. R. Hinkelmann
Second examiner: Dr.-Ing. F. Tügel
Advisor: Lennart Steffen, M.Sc.

August 2024

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit eigenständig ohne Hilfe Dritter und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe. Alle Stellen, die den benutzten Quellen und Hilfsmitteln unverändert oder sinngemäß entnommen sind, habe ich als solche kenntlich gemacht.

Sofern generische KI-Tools verwendet wurden, habe ich Produktnamen, Hersteller, die jeweils verwendete Softwareversion und die jeweiligen Einsatzzwecke (z.B. sprachliche Überprüfung und Verbesserung der Texte, systematische Recherche) benannt. Ich verantworte die Auswahl, die Übernahme und sämtliche Ergebnisse des von mir verwendeten KI-generierten Outputs vollumfänglich selbst.

Die Satzung zur Sicherung guter wissenschaftlicher Praxis an der TU Berlin vom 8. März 2017[1] habe ich zur Kenntnis genommen.

Ich erkläre weiterhin, dass ich die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Berlin, den 07. August 2024

[1] `https://www.static.tu.berlin/fileadmin/www/10000060/FSC/Promotion___Habilitation/Dokumente/Grundsaetze_gute_wissenschaftliche_Praxis_2017.pdf`

Abstract

A local time stepping (LTS) scheme for the Hydroinformatics Modeling System (hms^{++}), a shallow water equation solver developed at the Chair of Water Resources Management and Modelling of Hydrosystems, is introduced in this thesis. A central goal of the development of hms^{++} has been to increase computational performance. Thus, the extension of hms^{++} with an LTS scheme aligns well with this goal, as long as it is compatible with existing optimization methods within hms^{++}. This is why the proposed scheme operates on blocks of cells, instead of individual cells. It adopts the approach of reducing the amount of flux computations by *freezing* flux vectors and reusing them according to a local stability criterion. This criterion is based on choosing the minimum local time step inside a block as the block's allowable time step, enabling the block's cells to be computed using parallel methods. Thus, the proposed scheme is named frozen block LTS. Reuse of previously computed fluxes is accomplished by adaptively scaling them according to the current global time step. Initial problems with stability, as well as the synchronization of some adjacent blocks, were mitigated by a revision of the criterion for performing scalar flux computations and the introduction of neighbor propagation. Except for one outlier case, frozen block LTS has shown good stability and exhibits comparable accuracy to the global time stepping (GTS) scheme currently in use. In an analytical benchmark, frozen block LTS was able to reduce simulation runtime by more than 30 %, while in a practical rainfall-runoff scenario, runtime reductions greater than 20 % were achieved.

Kurzzusammenfassung

In dieser Arbeit wird ein lokales Zeitschrittverfahren (*Local Time Stepping* (LTS)) für das Hydroinformatics Modeling System (hms^{++}) vorgestellt. Bei hms^{++} handelt es sich um einen am Fachgebiet für Wasserwirtschaft und Hydrosystemmodellierung entwickelten Flachwassergleichungslöser. Ein Fokuspunkt der kontinuierlichen Entwicklung von hms^{++} ist die Optimierung der Rechenleistung. Die Erweiterung von hms^{++} um ein LTS-Schema folgt diesem Ziel dementsprechend, solange die Kompatibilität mit den bestehenden Optimierungsmethoden von hms^{++} gewährleistet wird. Deshalb arbeitet das vorgeschlagene Verfahren mit Blöcken von Zellen, anstelle einzelner Zellen. Es verfolgt den Ansatz, den Umfang der Flussberechnungen zu reduzieren, indem Flussvektoren *eingefroren* und entsprechend einem lokalen Stabilitätskriterium wiederverwendet werden. Dieses Kriterium basiert darauf, das Minimum des lokalen Zeitschritts aller Zellen innerhalb eines Blocks als dessen zulässigen Zeitschritt zu wählen, sodass die Zellen des Blocks mit Parallelisierungsmethoden berechnet werden können. Daher wird das vorgeschlagene Verfahren als *Frozen Block LTS* bezeichnet. Die Wiederverwendung von zuvor berechneten Flüssen wird durch adaptive Skalierung entsprechend des aktuellen globalen Zeitschritts erreicht. Anfängliche Probleme mit der Stabilität sowie der Synchronisierung einiger benachbarter Blöcke wurden durch eine Überarbeitung des Kriteriums für die Durchführung von skalaren Flussberechnungen und die Einführung der Übertragung von erzwungenen vollen Flussberechnungen auf direkte Nachbarn (*Neighbor Propagation*) behoben. Mit Ausnahme eines Ausreißers hat sich *Frozen Block LTS* als stabil erwiesen und weist eine vergleichbare Genauigkeit zu dem derzeit verwendeten globalen Zeitschrittverfahren auf. In einem analytischen Benchmark konnte *Frozen Block LTS* die Simulationslaufzeit um mehr als 30 % reduzieren, während in einem praktischen Niederschlagsabfluss-Szenario Reduktionen um mehr als 20 % erreicht wurden.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

Symbol	-	Description	Unit
d	-	water depth	[m]
g	-	gravitational acceleration	[m/s^2]
f_x	-	external forces in x direction	[N]
f_y	-	external forces in y direction	[N]
m_w	-	mass sources and sinks	[m/s]
t	-	time	[s]
u	-	depth-averaged flow velocity in x direction	[m]
v	-	depth-averaged flow velocity in y direction	[m]
$v_{i,\max}$	-	maximum propagation velocity	[m/s]
x	-	spatial dimension	[m]
y	-	spatial dimension	[m]
z	-	spatial dimension	[m]
z_B	-	bottom elevation	[m]
A	-	cell area	[m^2]
Cr	-	Courant number	[–]
P	-	cell perimeter	[m]
ν	-	kinematic viscosity	[m^2/s]
τ_B	-	bed shear stress	[N/m^2]
$\mathbf{F}$	-	flux vector normal to cell edge	
$\mathbf{f}$	-	vector of advective and diffusive fluxes in x direction	
$\mathbf{g}$	-	vector of advective and diffusive fluxes in y direction	
$\mathbf{n}$	-	normal vector to cell edge	
$\mathbf{s}$	-	vector of sources and sinks	
$\mathbf{q}$	-	vector of conserved state variables	
$\mathbf{v}$	-	depth-averaged flow velocity	

List of Abbreviations

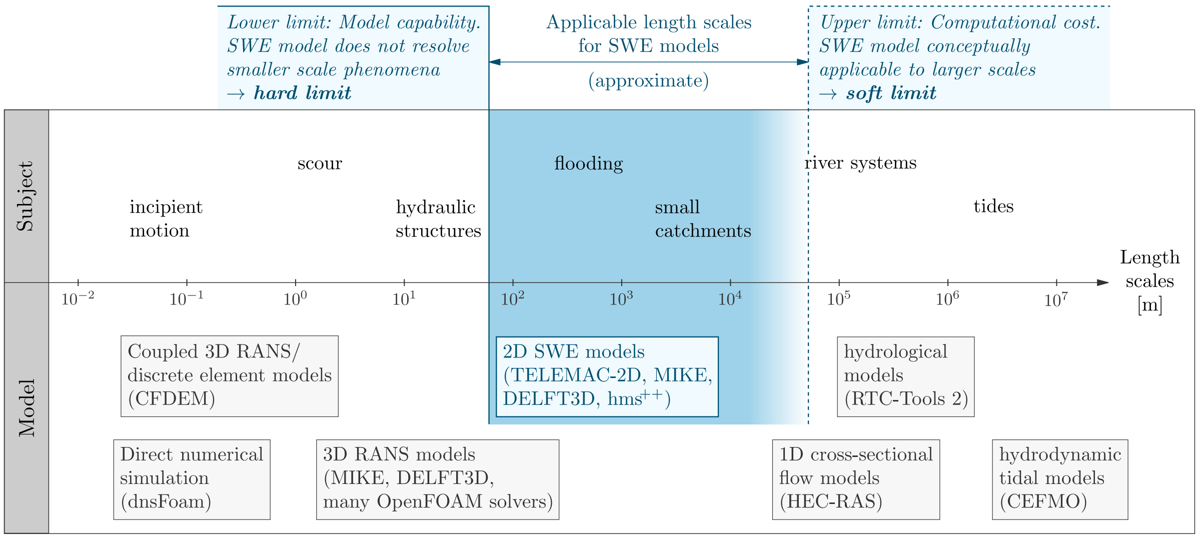
hms^{++}	Hydroinformatics Modeling System
1D	one-dimensional
2D	two-dimensional
ARM	Advanced RISC Machines
BFM	block flux memory
CCFVM	cell-centered Finite Volume Method
CFL	Courant-Friedrichs-Lewy-Condition
fb-LTS	frozen block LTS
FDM	Finite Difference Method
FEM	Finite Element Method
ff-LTS	frozen flux LTS
FOU	first-order upwind
FVM	Finite Volume Method
gcc	GNU Compiler Collection
GPU	Graphics Processing Unit
GTS	global time stepping
HLLC	Harten-Lax-van-Leer Contact
HPC	high-performance computing
LTS	local time stepping
MIMD	multiple instructions, multiple data
MPI	Message Passing Interface
NSE	Nash-Suttcliffe-Efficiency
OpenMP	Open Multi-Processing
pow2-LTS	power-of-two level LTS
RANS	Reynolds-averaged Navier-Stokes
RTR	runtime reduction

SAE	sum of absolute errors
SDK	software development kit
SIMD	single instruction, multiple data
SSC	share of scalar flux computations
SWE	shallow water equation
TVD	total-variation-diminishing
x86	x86 processor architecture

1. Introduction

The Hydroinformatics Modeling System (hms++) is a shallow water equation model and solver, developed in-house at the Chair of Water Resources Management and Modeling of Hydrosystems. The shallow water equations (SWEs) are used to model shallow water flow, e.g., flooding in cases of heavy rainfall events, the behavior of river systems, and urban interventions like green roofs (Fischer, 2024; Sanders, 2008).

A core motivation for developing hms++ has been enabling the use of two-dimensional SWE models in large-scale high-resolution systems by shifting their applicability limit. The lower limit consists of considerations whether a model can represent a certain detail relevant for a specific task, and can be thought of as a *hard limit*. In contrast, the upper limit can be regarded simply as the cost and practicability of running a simulation and therefore represents a *soft limit* (Steffen and Hinkelmann, 2023).



Lower limit: Model capability. SWE model does not resolve smaller scale phenomena → hard limit
Applicable length scales for SWE models
(approximate)
Upper limit: Computational cost. SWE model conceptually applicable to larger scales → soft limit
Subject
scour
incipient motion
hydraulic structures
flooding
small catchments
river systems
tides
Length scales [m]
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}
Model
Coupled 3D RANS/ discrete element models (CFDEM)
2D SWE models (TELEMAC-2D, MIKE, DELFT3D, hms++)
hydrological models (RTC-Tools 2)
Direct numerical simulation (dnsFoam)
3D RANS models (MIKE, DELFT3D, many OpenFOAM solvers)
1D cross-sectional flow models (HEC-RAS)
hydrodynamic tidal models (CEFMO)

Figure 1.1: “Subjects of interest in fluid flow at their characteristic length scales, and models (with software examples in brackets) at their applicable length scales (approximate).” (Steffen and Hinkelmann, 2023)

Because of this, improving computational performance has always been a central goal in the development of hms++. Therefore, parallel processing methods and other optimization techniques have been included in its conceptualization from the very beginning (Steffen et al., 2020).

A method for increasing performance that is currently not implemented, is local time stepping: the idea of locally reducing computations where it is possible while still maintaining stability and accuracy. To accurately solve the SWEs in practical cases, numerical methods are required: the domain is spatially discretized into cells; the simulation duration is explicitly discretized into periods of dynamic length, i.e., time steps. A stability criterion limits these time step lengths according to cell size, maximum propagation velocity, and the Courant number. The global time stepping scheme, which is currently used in hms^{++}, limits the global time step to the minimum value obtained by applying the stability criterion on each cell in the domain. The stability criterion is thus applied globally. Local time stepping schemes use different time steps throughout the domain by applying the stability criterion locally to reduce the number of computations. As these computations are complex and therefore costly, the runtime of a SWE solver decreases with the number of computations. On the other hand, the use of a local time stepping bears the risk of introducing instabilities and hurting accuracy.

The implementation of a local time stepping scheme in hms^{++} is required to build upon optimization techniques currently in use. Primarily, the block-wise traversal is obliged to be maintained, as it has proven to benefit performance immensely by enabling effective parallelization and cache utilization (Steffen et al., 2022).

Thus, a novel local time stepping method is developed and implemented within hms^{++}. Section 2 outlines background resources required for this task, including governing equations, discretization, and existing local time stepping schemes. Section 3 describes specifics of the proposed time stepping scheme, implementation details, as well as test cases. These consist of an analytically solvable one-dimensional dam break case and an example of urban rainfall-runoff. Section 4 covers accuracy and runtime analysis of these test cases. Finally, Section 5 presents a conclusive summary, as well as an outlook into possible future research.

2. Background

2.1. Conservation Laws

Conservation laws form the basis of numerous physical models and are used to describe processes that are stable over time in a given space, e.g., conservation of mass and momentum. Given below is the general form of a conservation law for an infinitesimal two-dimensional (2D) fixed Eulerian control volume (Simons, 2020):

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = \mathbf{s} \tag{2.1}$$

It indicates that in the control volume, conserved state variables $\mathbf{q}$ are preserved over time, unless they are changed by fluxes in x- or y direction ($\mathbf{f}$ and $\mathbf{g}$ respectively), or a source or sink $\mathbf{s}$, e.g., precipitation or infiltration, (ibid.).

2.2. Shallow Water Equations

The SWEs are a simplified form of the Reynolds-averaged Navier-Stokes (RANS) equations for shallow surface water flow, for which wavelength is much larger than water depth. On top of that, we assume (i) a hydrostatic pressure distribution, and (ii) the negligibility of vertical in comparison to horizontal flow, which enables us to average the RANS over the water depth, effectively reducing the equations to 2D space. The variables in the general form of the conservation law (Equation 2.1) are then populated as follows (ibid.):

$$\begin{aligned}
\mathbf{q} &= \begin{bmatrix} d \\ ud \\ vd \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} ud \\ uud + \frac{1}{2}gd^2 - \nu\frac{\partial ud}{\partial x} \\ uvd - \nu\frac{\partial vd}{\partial x} \end{bmatrix}, \\
\mathbf{g} &= \begin{bmatrix} vd \\ vud - \nu\frac{\partial ud}{\partial y} \\ vvd + \frac{1}{2}gd^2 - \nu\frac{\partial vd}{\partial y} \end{bmatrix}, \quad \mathbf{s} = \begin{bmatrix} m_{\mathrm{w}} \\ \frac{\tau_{\mathrm{B}x}}{\rho} + gd\frac{\partial z_{\mathrm{B}}}{\partial x} + f_x \\ \frac{\tau_{\mathrm{B}y}}{\rho} + gd\frac{\partial z_{\mathrm{B}}}{\partial y} + f_y \end{bmatrix}
\end{aligned} \tag{2.2}$$

The terms in these vectors are: (i) the water depth above bottom elevation d, (ii) flow velocities u and v in the x and y direction, respectively, (iii) the kinematic viscosity ν, (iv) mass sources and sinks m_{w}, (v) the bed shear stress τ_{B}, (vi) the bottom elevation z_{B}, (vii) and external forces f_x, f_y.

Mass equilibrium is represented by the first vector components, containing terms for water depth d, specific discharges ud, vd, and mass sources and sinks m_w. The second and third components describe momentum equilibrium in x and y directions, respectively. They include specific discharges ud, vd, advective fluxes uud, uvd, vud, vvd, viscous fluxes $\nu\frac{\partial ud}{\partial x}, \nu\frac{\partial vd}{\partial x}, \nu\frac{\partial ud}{\partial y}, \nu\frac{\partial vd}{\partial y}$, the pressure difference induced momentum transport term $\frac{1}{2}gd^2$, as well as bed friction and bottom slope source terms, $\frac{\tau_B}{\rho}$ and $gd\frac{\partial z_B}{\partial x,y}$, respectively, and, finally, external forces f_x, f_y (Simons, 2020).

The SWEs are partial differential equations, to which no general analytical solution is known to exist. Therefore, numerical solvers are used, which require a discretized version of the SWEs. This is subdivided into spatial discretization, which is described in Section 2.3, and temporal discretization, as discussed in Section 2.4.

2.3. Discretization of Spatial Derivatives

There are various approaches for discretizing the spatial derivates of a conservation law, such as Finite Element Method (FEM), Finite Difference Method (FDM), and Finite Volume Method (FVM). Here, the cell-centered Finite Volume Method (CCFVM) is used. This is a form of the FVM, which, in contrast to FDM and FEM, is inherently conservative regarding state variables (Hinkelmann, 2005).

It discretizes the domain in finitely sized control volumes, also referred to as cells. The specification *cell-centered* indicates that variables at any point in the cell are computed at the center of the cell and can be reconstructed at other points if needed. This approach leads to a loss of detail: non-uniform distributions inside the cell are averaged, as the whole cell is treated as a singular unit. Therefore, a FVM model's accuracy is strongly dependent on the cell size (Simons, 2020).

FVM is based on the principle of "integrating the differential equations in their conservative form over all control volumes" (Hinkelmann, 2005). According to Simons (2020), utilizing the FVM, Equation 2.1 is integrated over the domain of the control volume Ω to:

$$\int_\Omega \frac{\partial \mathbf{q}}{\partial t} \, \mathrm{d}\Omega + \int_\Omega \left(\frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} \right) \mathrm{d}\Omega = \int_\Omega \mathbf{s} \, \mathrm{d}\Omega \tag{2.3}$$

Using the Green Gauss Theorem, the flux term can be reduced from a volume integral to a surface integral over the surface of the control volume Γ, which describes the fluxes through the cell's surfaces (Simons, 2020):

$$\int_\Omega \frac{\partial \mathbf{q}}{\partial t} \, \mathrm{d}\Omega + \oint_\Gamma \left(\frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} \right) \mathrm{d}\Gamma = \int_\Omega \mathbf{s} \, \mathrm{d}\Omega \tag{2.4}$$

These integrals can be solved to gain the spatially discretized form of the conservation law:

$$\frac{\partial \mathbf{q}}{\partial t} A + \sum_{k=1}^{n_b} \mathbf{F}_k \mathbf{n}_k l_k = \mathbf{s} A \tag{2.5}$$

The reduction from three to two dimensions transforms cell surfaces into edges. Accordingly, the integral over the cell circumfence can be replaced by the sum over its edges. In Equation 2.5 the flux vector $\mathbf{F}_k$ represents the flux over an edge. It is perpendicular to that respective edge, thus its scalar product with the edge normal vector $\mathbf{n}$ is required, which resolves to $\mathbf{F}_k \, \mathbf{n}_k = \mathbf{f} n_x + \mathbf{g} n_y$. In addition, Equation 2.5 includes edge index k, length of edge l_k, cell area A, and number of cell edges n_b.

The evaluation of fluxes $\mathbf{F}$ comprises two parts: firstly, the reconstruction scheme and secondly, the flux scheme. The former serves to determine the value of state variables at edges from their stored values at cell centers, and the latter performs the computation of fluxes from edge values. Some methods, like the first-order upwind (FOU) method, combine both parts, while other flux schemes, like the Harten-Lax-van-Leer Contact (HLLC) solver introduced in Toro et al. (1994), require the separate use of a reconstruction scheme. The first-order reconstruction method uses the values computed at the cell-center at each point in a cell, including at its edge. While simple, it causes strong numerical diffusion, leading to a loss in accuracy. Other reconstruction schemes, like the higher order total-variation-diminishing (TVD) methods, are more accurate, but also more complex to implement and computationally intensive. For details on this, the inclined reader is referred to Simons (2020).

2.4. Discretization of Temporal Derivatives

Equation 2.5 still contains the temporal derivative, which can be discretized using the Forward Euler method. It is equivalent to a first-order Taylor series approximation of the temporal function:

$$\frac{\partial \mathbf{q}}{\partial t} \approx \frac{\mathbf{q}^{n+1} - \mathbf{q}^{n}}{\Delta t} \tag{2.6}$$

Applying this to Equation 2.5 results in a fully discretized version of a general conservation law for a finite 2D cell:

$$\mathbf{q}^{n+1} = \mathbf{q}^{n} - \frac{\Delta t}{A} \sum_{k=1}^{n_b} \mathbf{F}_k^n \mathbf{n}_k l_k - \Delta t \, \mathbf{s}^n \tag{2.7}$$

Equation 2.7 describes values at the next point in time t_{n+1} based on values of the current time t_n only, which makes it an explicit one-step discretization. This leads to a low cost per computation at the expense of conditional stability, which is maintained through small time step sizes, as pointed out in the section below.

2.5. Courant-Friedrichs-Lewy Condition

For explicit time stepping to provide a stable solution, it is mandatory to fulfill the Courant-Friedrichs-Lewy-Condition (CFL) condition at any point in space and time (Simons, 2020). It is described by the Courant number Cr, which is required $0 \leq Cr \leq 1$. This dimensionless number originates from cell size Δs_i, maximum propagation velocity $v_{i,\max}$ and time step Δt of cell i:

$$Cr = \frac{v_{i,\max} \, \Delta t}{\Delta s_i} \leq 1, \quad v_{i,\max} = |\mathbf{v}_i| + \sqrt{g d_i}, \quad \Delta s_i = \frac{4 \, A_i}{P_i} \tag{2.8}$$

Δs_i and $v_{i,\max}$ consist of the components absolute flow velocity $|\mathbf{v}_i|$, gravity constant g, water depth d_i, cell area A_i, and cell perimeter P_i

Many explicitly discretized applications use a preset Courant number $Cr \in [0.3, 0.8]$ to set the time step Δt (Dazzi et al., 2018; Hu et al., 2019; Sanders, 2008; Simons, 2020;

Steffen et al., 2023). Thus, Δt is adapted to the current need, reducing computational effort when facing slower flow velocities on a static domain of cells.

$$\Delta t = \min \left(\frac{Cr\,\Delta s_i}{v_{i,\max}} \right) \tag{2.9}$$

To fulfill the CFL condition, the smallest time step of all n_c cells is applied as the time step for the whole domain. Consequently, high velocities in a small portion of the domain impact the whole domain's next time step, which increases the model's overall runtime linearly. This approach forms a scheme called global time stepping (GTS), the basic time stepping scheme for FVM solvers.

2.6. Local Time Stepping

While GTS has proven reliable in many applications involving FVM models, "model efficiency suffers using non-uniform grids and uniform grids with variable [maximum propagation velocity] because the Courant number in many cells is far [below its limit]" (Sanders, 2008). The author refers to non-uniform grids as ones with varying cell sizes, e.g., varying cell length in a one-dimensional (1D) scenario, distortions in a structured 2D grid or an unstructured 2D grid.

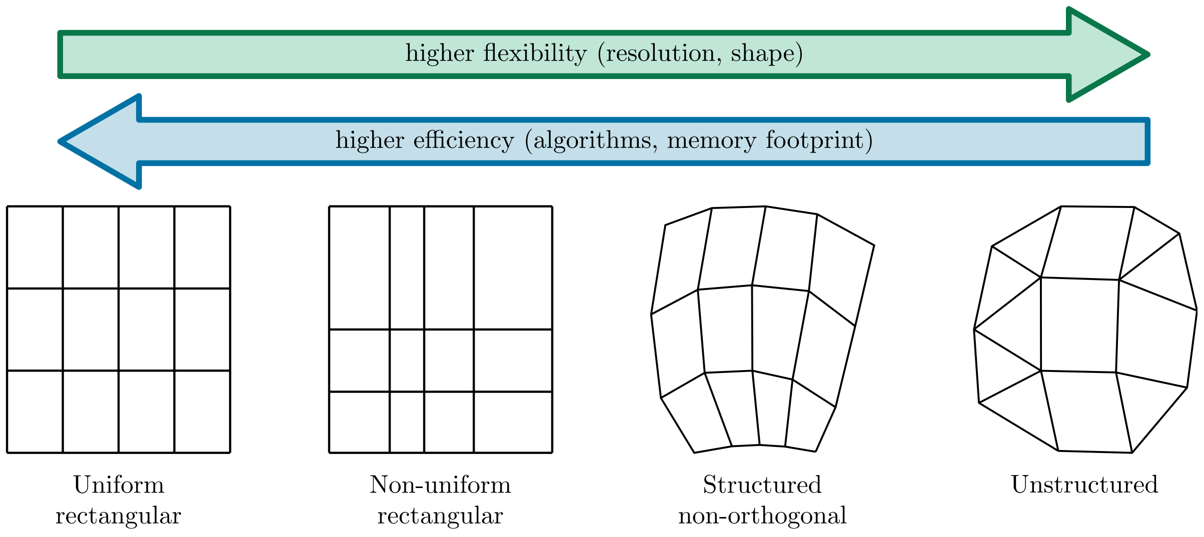

higher flexibility (resolution, shape)
higher efficiency (algorithms, memory footprint)
Uniform rectangular
Non-uniform rectangular
Structured non-orthogonal
Unstructured

Figure 2.1: Typology of 2D grids (Steffen and Hinkelmann, 2023)

To improve efficiency, Osher and Sanders (1983) have developed the concept of local time stepping (LTS). They proved that a local CFL condition is sufficient to provide a correct solution for nonlinear conservation laws in 1D domains. Therefore, given a constant Courant number Cr, the time step can be adapted to each cell i in conformity with maximum propagation velocity $v_{i,\max}$ and cell size Δs_i:

$$\Delta t_i = \frac{Cr\,\Delta s_i}{v_{i,\max}} \tag{2.10}$$

For uniform grids, i.e., fixed-size cells, Δs_i is constant across the whole domain, leaving $v_{i,\max}$ as the only factor impacting the local time step Δt_i.

Wright et al. (1999) were the first to apply LTS in the field of computational hydraulics. They used the Saint-Venant Equations, a 1D version of the SWEs, to model open channel flow and tested two LTS methods on it. In a later publication, that included examinations on non-uniform grids among others, they found that the performance gain provided through LTS increases with the ratio of the largest to the smallest cell lengths (Crossley et al., 2003). They also pointed out that interactions of adjacent cells require attention to guarantee a correct exchange of information about their current state (ibid.). In particular, they compared two approaches to LTS: *frozen flux LTS (ff-LTS)* by Zhang et al. (1994) and *Full Time Integration* by Kleb et al. (1992). The latter of which can be categorized as a power-of-two level LTS (pow2-LTS) and was found to be more effective at reducing runtime on both uniform and non-uniform grids. Both methods were initially developed for the field of aeronautics and astronautics.

2.6.1. Frozen Flux LTS

Frozen flux LTS (ff-LTS) focuses on optimizing flux calculations. Because of their complexity, they account for a large part of a model's runtime. This complexity varies, depending on the chosen reconstruction scheme. When using higher order schemes, like TVD methods, reconstruction can be held accountable for a significant fraction of computational cost.

Zhang et al. (1994) introduced an LTS method based on reducing the total amount of flux updates. They achieved this by *freezing* fluxes and reusing them as long as permitted by their local CFL condition. The following procedure is an adapted version of this scheme. The original authors used a different nomenclature and illustrated it using the 1D unsteady Euler equations instead of the SWEs.

Assuming a constant global time step Δt, we can infer that fluxes of cell i can be reused $j_{\max}$ times, given by:

$$j_{\max} = \left\lfloor \frac{\Delta t}{\Delta t_i} \right\rfloor \tag{2.11}$$

Using this on Equation 2.7, we can deduce the state at all intermediary global time steps $\mathbf{q}^{n+j}$, $j \in [1, j_{\max}]$. To enhance readability, we substitute the flux term with a helper variable $\mathbf{\Phi}^n = \sum_{k=1}^{n_b} \mathbf{F}_k^n \mathbf{n}_k l_k$.

$$\mathbf{q}^{n+j} = \mathbf{q}^n - \frac{j\,\Delta t}{A_i} \mathbf{\Phi}^n - \sum_{j=1}^{j_{\max}} \Delta t\, \mathbf{s}^{n+j-1} \tag{2.12}$$

While the fluxes remain unchanged, state variables are updated each global time step Δt, so exchanging information at cell interfaces is possible at all times.

A more thorough explanation of the scheme can be found in Zhang et al. (1994).

Compared to a GTS, the ff-LTS reduced the number of flux updates by approximately 67% in multiple test cases, deducted by Crossley et al. (2003). Additionally, both Crossley et al. (2003) and Zhang et al. (1994) found that accuracy can improve through the use of this scheme when comparing results with analytical solutions. This effect is explained by the reduction of local truncation error (Crossley et al., 2003; Zhang et al., 1994)

2.6.2. Power-of-two Level based LTS

An alternative approach to LTS consists of skipping a set of time steps at parts of the domain altogether: the power-of-two level LTS (pow2-LTS) scheme. The idea is to cluster cells in proportion to their local time steps Δt_i into groups that share an LTS-level m_i, which determines a shared local time step used for these cells: Δt_m.

$$m_i = \left\lfloor \log_2 \frac{\Delta t_i}{\Delta t_{\text{base}}} \right\rfloor \tag{2.13}$$

These level time steps are defined by an exponential function that scales the base time step: $\Delta t_m = 2^m \Delta t_{\text{base}}$. Many implementations use the global minimum time step as their base (Kleb et al., 1992; Crossley et al., 2003; Dazzi et al., 2018; Yang et al., 2020); others utilize a predefined constant instead (Sanders, 2008).

As such, all cells regularly synchronize to exchange information, as depicted in Figure 2.2. At interfaces of different LTS-levels, intermediate values of the larger level are interpolated on demand. To maintain a stable solution, it is suggested to introduce intermediate regions between directly adjacent cells, which differ by more than one level (Crossley et al., 2003), forming a method called level smoothing and is discussed later on in Section 2.6.4.

The pow2-LTS consists of performing a sequence of level loops, reaching from time t_n to $t_n + \Delta t_{\max}$, where $\Delta t_{\max}$ is the time step of the largest level. In a level loop, the largest level is advanced by a single time step. All other levels are advanced until they synchronize again, i.e., all cells are advanced to $t_n + \Delta t_{\max}$. The base time step t_{base} is required to stay constant within a level loop.

At the start of a level loop, each cell is assigned a level (Equation 2.13). Then, each cell is updated once. As their local time steps vary, they are updated to different points in time t_n. Following this, the lower levels are updated in a way that they catch up with the highest level, so that finally, all regions are synchronized, and level assignment can be re-evaluated for the next loop. Further details of the scheme can be found in Kleb et al. (1992) and Crossley et al. (2003).

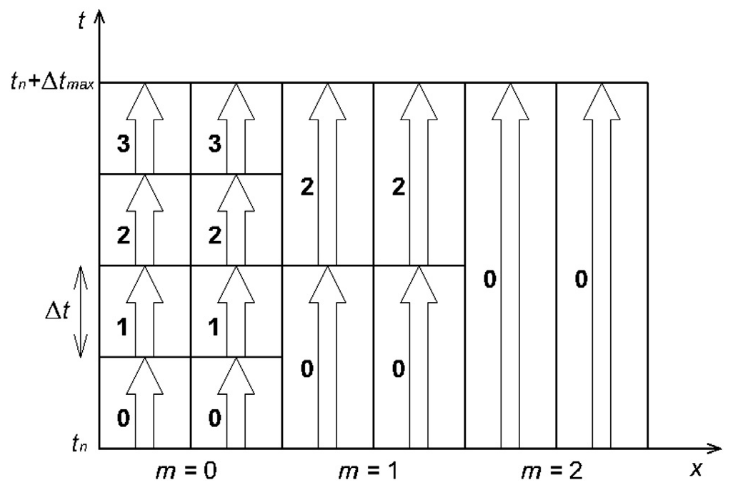


Figure 2.2: Power-of-two level LTS scheme with three levels showing a level loop, i.e., a full synchronization cycle (Dazzi et al., 2018).

Compared to a GTS, the pow2-LTS reduced the number of flux updates by 61%-66% in multiple test cases deducted by Crossley et al. (2003). While pow2-LTS is exceeded by ff-LTS in this metric, it still outperforms the other scheme in terms of runtime: it achieves a mean reduction of 50.8% in contrast to 27.3% for ff-LTS, according to the authors. They state this is due to the need of the latter to update state variables each local time step (Crossley et al., 2003).

2.6.3. 2D LTS

Sanders (2008) has further evolved LTS in the domain of hydraulics simulations by proposing a scheme applicable to 2D models. Similarly to Crossley et al. (2003), the author points out, that models using uniform grids might benefit from LTS, if they have a large spatial heterogeneity in maximum propagation velocity.

His scheme adopts first-order reconstruction and a pow2-LTS method with a predefined base time step, which is constant throughout the simulation (Sanders, 2008). While proving useful in general, he reported stability problems on wet/dry fronts, especially when using more than four LTS-levels, which led him to the conclusion that it should be limited to four in practical applications. He was able to reduce runtime by 50–70% without loss of accuracy for applications of four levels. As an aside, his approach generally showed better performance and stability when bed friction was included in the models.

These stability problems were successfully targeted by Hu et al. (2019), who introduced changes to the calculation of the allowable local time step for almost-dry cells and interface regions, which smoothen borders between different levels.

2.6.4. Interfaces of Divergent Regions

As mentioned above, many LTS schemes utilize techniques to even out interfaces between regions whose time steps diverge (Crossley et al., 2003; Sanders, 2008; Dazzi et al., 2018; Hu et al., 2019; Yang et al., 2020). The main purpose is to limit instabilities and improve accuracy, as waves can be propagated from a cell with a small time step to an adjacent one with a larger time step within a single level loop in a pow2-LTS scheme (Crossley et al., 2003).

There are two popular approaches: (a) neighbor propagation and (b) introducing interface regions.

The former compares a cell with all adjacent cells (neighbors) and chooses the minimum of their local time steps as the cell's allowable time step Δt_i (Sanders, 2008; Yang et al., 2020). In the case of pow2-LTS, adjacent cells may still differ by more than one level, since only direct neighbors are taken into account.

The latter establishes a region in which the local time step Δt_m is gradually increased to provide a smooth transition. For further details on these approaches, the inclined reader may refer to the respective publications (Crossley et al., 2003; Dazzi et al., 2018; Hu et al., 2019; Kleb et al., 1992).

2.7. Scope and Purpose of hms^{++}

The hms^{++} is an open-source numerical solver for the SWEs written in C++ with performance and user-friendliness in mind. It is developed at TU Berlin's Chair of Water Resources Management and Modeling of Hydrosystems and forms the successor to the Hydroinformatics Modeling System (hms) Java framework.

At its core are CCFVM spatial and explicit Euler temporal integration in combination with a first-order and a TVD scheme for flux reconstruction. hms^{++} can handle unstructured and multiple variations of structured meshes. Details are provided in Steffen et al. (2022) and Steffen and Hinkelmann (2023).

hms^{++} aims to shift the SWEs' applicability limit, to enable their use in larger cases, currently restricted by the cost of computation, see Section 1. To achieve this, both single instruction, multiple data (SIMD) and multiple instructions, multiple data (MIMD) parallel processing, as well as methods for optimally utilizing CPU cache are employed. SIMD parallelism is provided through the use of the *Eigen* library for linear algebra, which features clean and easily readable source code while optimizing matrix and vector operations in the background by adapting to specific CPU capabilities. MIMD parallelism is implemented using Open Multi-Processing (OpenMP) for coordinating threads on a single node and Message Passing Interface (MPI) for communication between multiple compute nodes. Each thread is supplied with pre-allocated buffers, which are reused throughout the runtime (Steffen and Hinkelmann, 2023).

When computing simulations with uniform rectangular meshes, hms^{++} can group cells into blocks to facilitate SIMD. It is able to carry out operations efficiently on multiple cells at once through vectorized functions implemented in *Eigen*. Additionally, MIMD is used to distribute blocks across multiple threads or compute nodes. Such block-wise traversal has shown performance gains of 14–25 times in comparison to classic cell-wise traversal of the domain (Steffen and Hinkelmann, 2023).

3. Methodology

3.1. Time Stepping Scheme

The use of block-wise traversal has proven to increase performance by a large magnitude. Therefore, it must be preserved when an LTS scheme is applied to hms^{++}. This requires the development of a novel time stepping scheme, which operates on blocks rather than individual cells.

Benchmark cases have shown that flux computations account for up to 86% of the program's runtime. For that reason, the proposed LTS scheme for hms^{++} focuses on reducing flux computations by adapting the ff-LTS approach. However, partly to maintain block-wise traversal, a few changes are necessary: (i) The time step is uniform throughout the block. It is calculated for each cell and the block-wise minimum is set as its allowable time step Δt_b. (ii) Instead of determining how many times the fluxes can be reused, a block flux expiry time is introduced on the base of the block's allowable time step in Equation 3.1. Accordingly, the scheme is named frozen block LTS (fb-LTS).

$$t_{\text{expiry},\,b} = t + \Delta t_b \tag{3.1}$$

Upon start of the program, runtime-persistent objects are initialized, e.g., a user-defined number of persistent solver threads and a block flux memory (BFM) object for storing frozen block fluxes. Details on the latter can be found in Section 3.3. Subsequently, the time loop is started: each time t_n, a loop traverses the domain on all solver threads in parallel: Each thread is assigned a set of blocks, which it processes one by one, by calling COMPUTE_BLOCK (Algorithm 2) to advance its state to the next time t_{n+1}. This

Algorithm 1 Parallelized time loop including block-wise traversal

```
1: $t = t_0$                                   ▷ get $t_0, t_{end}$ from .ini file
2: INIT_PERSISTENT_SOLVER_THREADS( )
3: BFM = BlockFluxMemory( )
4: while $t < t_{\text{end}}$ do
5:     for each block $b$ do                   ▷ thread-parallel block-wise traversal
           COMPUTE_BLOCK($b$)
6:     end for
7:     $t = t + \Delta t$                      ▷ proceed to next time index
8:     $\Delta t = \min(\Delta t_b)$           ▷ set next global time step
9: end while
```

includes the computation of fluxes, state variables and the allowable time step. Once all blocks have been updated, the time is progressed: $t_{n+1} = t_n + \Delta t$. The next global time step is drawn from the minimum allowable time step $\Delta t = \min(\Delta t_b)$, ending the current time loop iteration. Algorithm 1 sketches the described time loop in pseudocode. As a side note, at each time t_n, the program needs to decide just once for each block, instead of once per cell, whether to perform a full or scalar computation. This leads to a reduced number of conditional statements, which may reduce branching operations.

The time loop calls a function used to advance blocks: COMPUTE_BLOCK (Algorithm 2). This function is used to decide whether to carry out a full or scalar computation on the block fluxes based on expiry time. It delegates the actual block update to COMPUTE_BLOCK_FULL (Algorithm 3) and COMPUTE_BLOCK_SCALAR (Algorithm 4). Criterion 3.2 decides on the type of computation.

$$\text{action for block } b = \begin{cases} \text{full computation,} & \text{if } t_{\text{expiry},\, b} \leq t + \Delta t \\ \text{scalar computation,} & \text{otherwise} \end{cases} \tag{3.2}$$

To automatically initiate a full flux computation at the beginning of the time loop, it is recommended to initialize expiry time to the lower numeric limit of the according datatype.

Algorithm 2 Wrapper function for deducing whether to perform full or scalar computation

```
1: function COMPUTE_BLOCK($b$)
2:     if $t_{\text{expiry},\, b} < t + \Delta t$ then
3:         COMPUTE_BLOCK_FULL($b$)
4:     else
5:         COMPUTE_BLOCK_SCALAR($b$)
6:     end if
7: end function
```

It is necessary for a reuse of precomputed fluxes to save these in a way accessible for all solver threads. This is solved by the BFM, which is introduced below in Section 3.3.

A full block update consists of numerous steps, which have been grouped once more into multiple functions, which perform tasks in the given sequence:

1. Initialize solver thread buffers for state variables with previous or, if time is t_0, initial values

2. Compute fluxes $\Delta\mathbf{q}_b$ using the flux term in Equation 2.7 and TVD or first-order reconstruction
3. Optionally, add mass sources and sinks to the storage buffer $\mathbf{q}_b$
4. Optionally, apply friction source term to the fluxes $\Delta\mathbf{q}_b$
5. Add fluxes to storage $\mathbf{q}_b = \mathbf{q}_b + \Delta\mathbf{q}_b$
6. Compute new values for all state variables based on storage
7. Compute allowable time step Δt_b using Equation 2.9
8. De-scale fluxes by removing the current global time step and store them in BFM for reuse
9. Calculate $t_{\text{expiry},\,b}$ and store it in BFM

Algorithm 3 presents this sequence in the function COMPUTE_BLOCK_FULL.

The two variants of the block update sequence are very similar: their main difference is a full or scalar computation of the fluxes; thus, the corresponding functions are quite similar as well. The latter is shown in Algorithm 4 and described below.

Solver thread buffers are initialized in the same way as in a full update. Subsequently, previously saved fluxes are retrieved from BFM and scaled with the current time step Δt. This is the scalar computation, replacing the more complex full computation. While mass sources and sinks are applied in the same way as before, friction terms are left out because they are already included in the precomputed fluxes. Finally, storage and new state variables are updated as usual. Time step and saved fluxes in BFM are left untouched, since fluxes have not been recomputed fully; therefore the existing expiry time remains.

Algorithm 3 Full block update function

```
1:  function COMPUTE_BLOCK_FULL($b$)
2:      COMPUTE_STORAGE($b$)                                  ▷ initialize solver thread buffers
3:      $\Delta\mathbf{q}_b$ = COMPUTE_FLUXES($b$)                          ▷ full flux calculation
4:      ADD_MASS_SOURCES($b$)                                 ▷ optional, based on settings
5:      COMPUTE_FRICTION($b$)                                 ▷ optional, based on settings
6:      $\mathbf{q}_b = \mathbf{q}_b + \Delta\mathbf{q}_b$                                  ▷ compute new storage state
7:      COMPUTE_NEW_STATE($b$)
8:      $\Delta t_b$ = COMPUTE_NEXT_TIMESTEP($b$)        ▷ compute next allowable block time step
9:      BFM.BLOCK_FLUXES($b$).fluxes = $\Delta\mathbf{q}_b/\Delta t$
10:     BFM.BLOCK_FLUXES($b$).expiryTime = $t + \Delta t_b$
11: end function
```

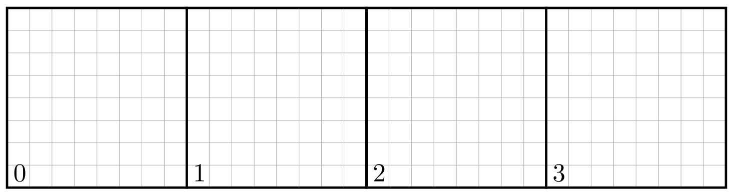
Algorithm 4 Scalar block update function

```
1: function COMPUTE_BLOCK_SCALAR($b$)
2:     COMPUTE_STORAGE($b$)                              ▷ initialize solver thread buffers
3:     $\Delta\mathbf{q}_b$ = BFM.BLOCK_FLUXES($b$).fluxes $*\Delta t$     ▷ scalar flux calculation
4:     ADD_MASS_SOURCES($b$)
5:     $\mathbf{q}_b = \mathbf{q}_b + \Delta\mathbf{q}_b$                          ▷ calculate new storage state
6:     COMPUTE_NEW_STATE($b$)
7: end function
```

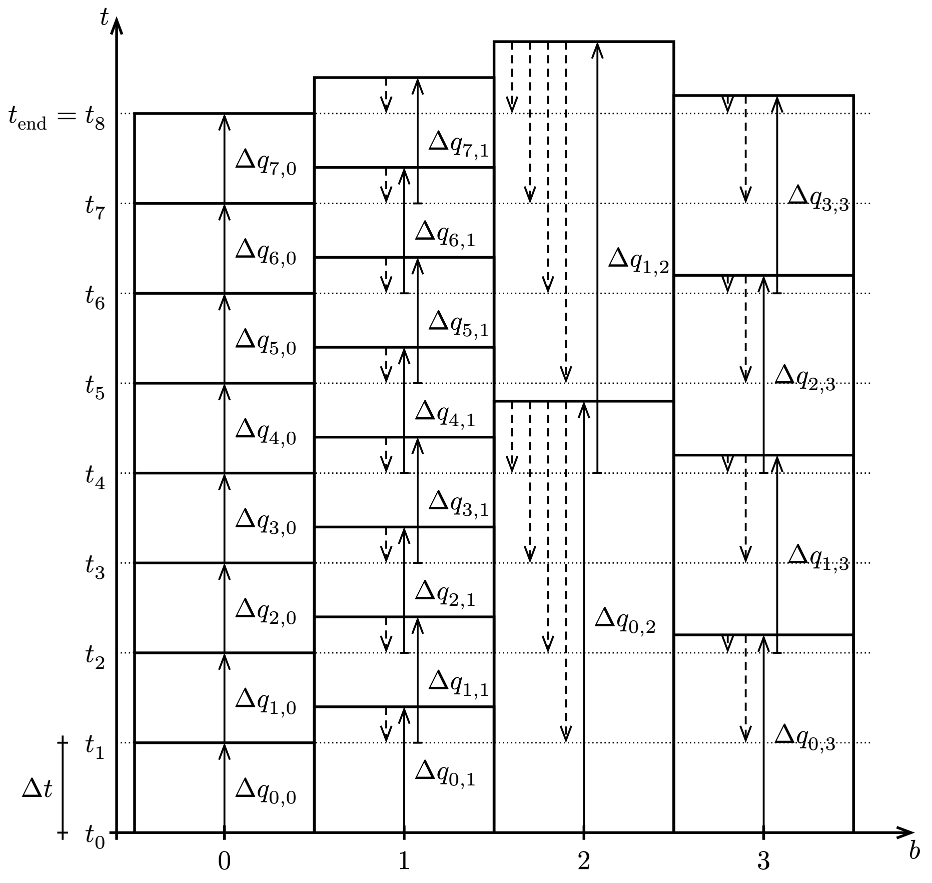
The fb-LTS scheme is characterized by a very flexible approach: each block's fluxes can be used nearly until their expiry time, i.e., until the last point in time t_n before reaching expiry time. This is illustrated in Figure 3.1b, which depicts the scheme similarly to Figure 2.2 for a small uniform mesh with four blocks. To simplify this example, each block's allowable time step remains constant. Block $b = 0$ sets the global time step Δt, as it has the smallest local time step. In contrast, block $b = 2$ features a very large local time step, enabling it to use its fluxes for four times. Block $b = 3$ has a local time step less than half the size of $b = 2$ but more than double the size of $b = 0$, leading to a single reuse of its fluxes. Finally, the local time step of $b = 1$ is less than $2\Delta t$, causing it to perform full computation for each global time step, as fluxes cannot be reused in that case.

Overlaps of flux validity periods exist in all blocks, except for the one with the smallest allowable time step, in this case block ($b = 0$). This is due to the fact that a new full computation always needs to start at a point in time t_n, not at the expiry time of the previously computed fluxes. Thus, blocks with local time steps below $2\Delta t$ need to perform a full flux computation each time.

Figure 2.2 can also be used for comparing fb-LTS to pow2-LTS: we can clearly see it is possible for two neighboring regions to have large differences in their time steps in fb-LTS. In pow2-LTS, it is not recommended for levels to differ by more than a factor of two between directly adjacent regions. Additionally, using an exponential term to set allowable time steps causes large overlaps of the fluxes' validity periods. This behavior increases with large local time steps: e.g., a cell with local step $\Delta t_i \approx 126\Delta t$ is assigned the allowable step of $\Delta t_m = 64\Delta t$. Besides, such a large time step is only admitted, if at least eight LTS-levels are permitted, which can cause instabilities, as literature indicates (Hu et al., 2019; Sanders, 2008).



(a) Uniform mesh with 32×8 cells and block size 8×8. Cell boundaries are shown in gray, while thick black lines indicate block boundaries.



(b) Illustrative frozen block LTS block update procedure. Axes represent time and space (i.e., blocks). Points in time t_n, set apart by a constant Δt, appear as dotted horizontal lines. Blocks boundaries are shown as solid vertical lines; expiry time and therefore local time steps as solid horizontal lines. While solid upward arrows depict full block flux computations, dashed downward arrows symbolize scalar computations.

Figure 3.1: Illustration of the frozen block LTS

3.2. Modifications to Frozen Block LTS

3.2.1. Revised Criterion for Scalar Flux Computations

The initial criterion for deciding on whether to perform full or scalar flux computations, Equation 3.2, has shown to cause instabilities and a violation of mass conservation. Therefore, a modification is applied to counter this behavior. Criterion 3.3 is adopted from Zhang et al. (1994), who developed it to work on similar stability problems of their frozen flux LTS scheme.

$$\text{action for block } b = \begin{cases} \text{full computation,} & \text{if } t_{\text{expiry},\,b} \leq t + 2\Delta t \\ \text{scalar computation,} & \text{otherwise} \end{cases} \tag{3.3}$$

3.2.2. Neighbor Propagation

Initially, fluxes propagated poorly from high-velocity blocks to low-velocity neighbors, as they have larger local time steps. This, again, caused instabilities and loss of mass conservation. An approach to synchronize heterogeneous regions is required to solve this problem.

Since the local time step is uniform throughout a block, it is deemed sufficient to implement a neighbor propagation technique, as described in Section 2.6.4. If a block is adjacent to one that requires a full computation based on Equation 3.3, it is forced to perform a full flux computation as well. Algorithm 5 shows a revision to Algorithm 2, including this neighbor propagation technique in combination with the improved criterion for scalar flux computations.

Algorithm 5 Revised function COMPUTE_BLOCK including neighbor propagation and the improved criterion for scalar flux computations Equation 3.3

```
1: function COMPUTE_BLOCK($b$)
2:     if BFM.BLOCK_FLUXES($b$).expiryTime $< t + 2\Delta t$
           or BFM.NEIGHBORS_REQUIRE_FULL_COMPUTATION($b$, $t + 2\Delta t$) then
3:         COMPUTE_BLOCK_FULL($b$)
4:     else
5:         COMPUTE_BLOCK_SCALAR($b$)
6:     end if
7: end function
```

3.3. Block Flux Memory

In the current implementation of hms^{++}, only state variables are stored in memory, as all other variables are computed from the previous state. Other variables, like fluxes, are temporarily stored in solver thread buffers. Thus, to enable reuse of fluxes, an additional memory structure is needed. A two stage approach was chosen for this task: Firstly, the implementation of a structure `BlockFluxes`, which stores fluxes and expiry time for each block; secondly, the creation of a class `BlockFluxMemory`, which manages `BlockFluxes` objects and implements the concept of a block flux memory. It stores a `std::vector` of `BlockFluxes` and various member variables which help to index the blocks. `BlockFluxMemory` provides access functions for `BlockFluxes` objects and all relevant member variables, as well as a function to determine whether a neighbor block requires a full computation to enable neighbor propagation.

Since hms^{++} supports both first-order and TVD reconstruction, the block flux memory needs to adapt to either option's partitioning scheme.

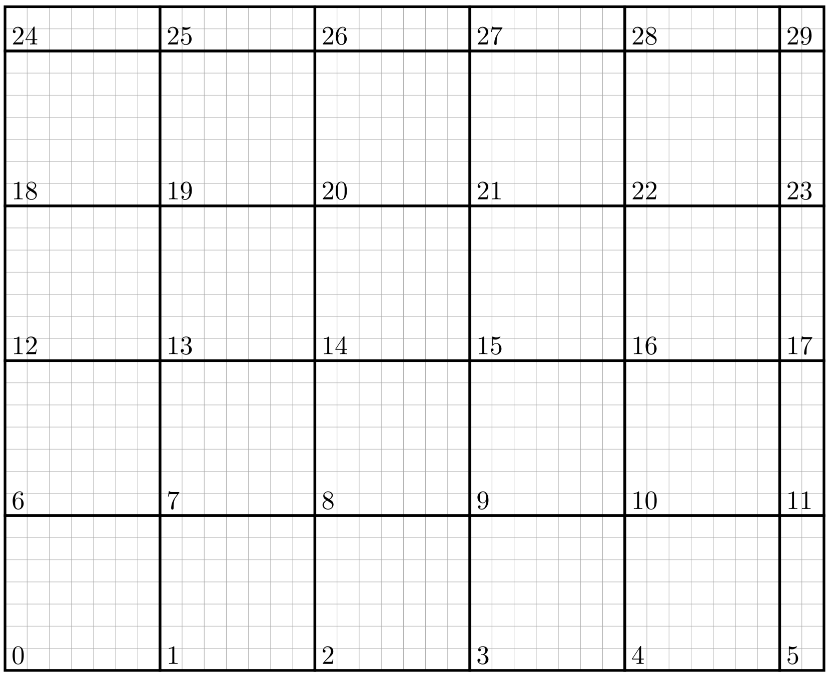


Figure 3.2: Block flux memory: Block map for the use with a first-order scheme. Cell boundaries are drawn as gray lines, whereas the block boundaries are depicted in black. The block index is shown in each block's bottom-left corner. Block size: 7 × 7 cells, mesh size: 37 × 30 cells.

When using first-order reconstruction, the mesh is partitioned into blocks of fixed, user-specifiable sizes and smaller remainder blocks, as shown in Figure 3.2. Starting from the bottom-left corner, the domain is filled with full blocks. Block size is not required to adapt to mesh size; thus, possible remaining cells are covered with smaller remainder blocks.

In contrast, block-wise traversal for the TVD scheme requires a layer of blocks wrapping the mesh's boundary, having a fixed width of two cells. This is due to the nature of second-order schemes, which take derivates of multiple cells' values around an edge into account to compute the fluxes over that edge. Narrow blocks enable correct usage of the TVD scheme at the domains boundaries. A more complex partitioning scheme than for the first-order scheme results of this, as shown in Figure 3.3.

Currently, hms^{++} identifies blocks by the coordinate of the cell in their bottom-left corner. Thus, the block flux memory needs to introduce an indexing mechanism according to both partitioning schemes for accessing `BlockFluxes` based block coordinates. An additional 2D vector stores all block index numbers in a two-dimensional representation

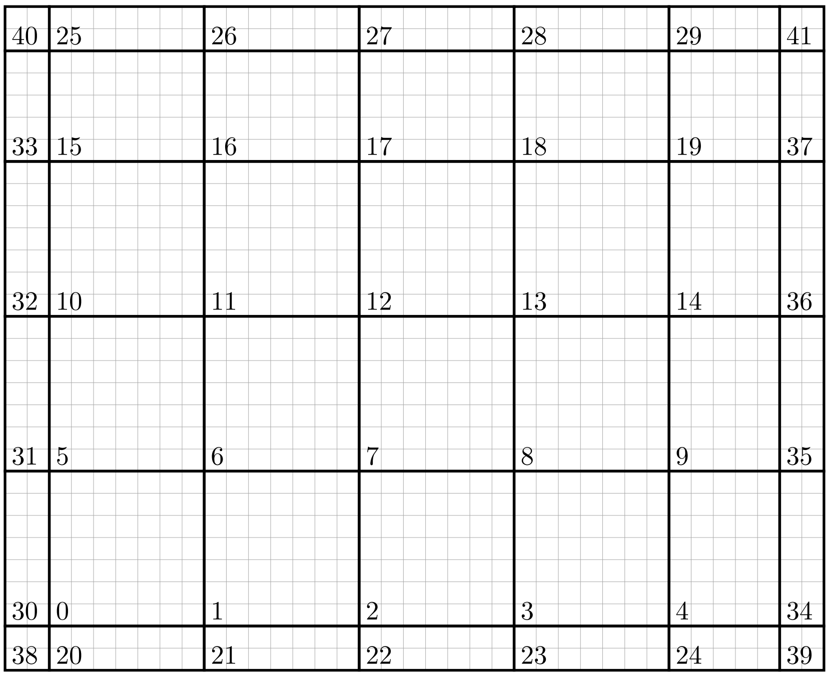


Figure 3.3: Block flux memory: Block map for the use with a TVD scheme. Cell boundaries are drawn as gray lines, whereas the block boundaries are depicted in black. The block index is shown in each block's bottom-left corner. Block size: 7 × 7 cells, mesh size: 37 × 30 cells, boundary layer width: 2 cells.

for efficiently retrieving adjacent blocks.

The BFM is instantiated at the start of each run of a hms^{++} before the time loop starts. Upon initialization, member variables for block indexing are populated, a `BlockFluxes` object for each block is constructed, and stored in a vector, at the position corresponding to its block index.

3.4. Test Cases and Metrics

3.4.1. 1D Dam Break

To validate the proposed LTS model, the 1D schematic dam break case is used, as an analytical solution for this case is available for comparison. It consists of a domain of $20\,\mathrm{m} \times 2\,\mathrm{m}$, discretized using 400×40 cells of $0.05\,\mathrm{m} \times 0.05\,\mathrm{m}$ size, grouped into blocks of 8×8 cells. Upstream and downstream boundaries are set to allow free outflow. The lateral boundaries act as frictionless walls to obtain a 1D flow pattern. Two variants of this scenario are simulated: (i) a wet case, in which the whole domain is covered in water, with a water depth of $d = 4\,\mathrm{m}$ in the left and $d = 1\,\mathrm{m}$ in the right half, and (ii) a dry case, in which only the left half of the domain contains water, with $d = 4\,\mathrm{m}$, whereas the right half is dry, i.e., $d = 0\,\mathrm{m}$. Flow velocity is set to $u = v = 0$ in either case, and bed friction is not included. The analytical solutions to these cases, used as a reference, are taken from Adelmann (2022). Results are interpreted both visually and with the Nash-Suttcliffe-Efficiency (NSE), see Section 3.4.3, as a quantitative metric for accuracy.

As a base configuration for investigating the runtime reduction obtainable through LTS, a scaled-up variant of the wet dam break case with bed friction enabled is chosen. Utilizing a fully wetted domain removes the consideration of load imbalances from the investigation, as hms^{++} contains dry cell optimizations, which may otherwise introduce such effects.

For this base configuration, the domain consists of 2000×2000 cells of $d_x = d_y = 1.0\,\mathrm{m}$, grouped into blocks of 64×64 cells; the Courant number is set to $Cr = 0.3$ and the end time of the simulation to $t_{end} = 0.6\,\mathrm{s}$.

Multiple parameters are examined in terms of their impact on computational cost. These include domain, cell, and block sizes, the Courant number, water depth of the downstream region, and the simulated duration. For domain, cell, and block sizes, downstream water depth, as well as the Courant number, a range of values both above and below the original configuration of $Cr = 0.3$ are investigated. For the simulated duration, the original value is used as a lower limit of the considered range. Finally, the effects of disabling bed friction are examined, as well as of exchanging the scenario to a dry dam break with the same parameters as the base configuration. For each variant, only a single parameter of the base configuration is modified, thus allowing to isolate their effects. All cases are run for each combination of first-order/TVD reconstruction and global/local time stepping. To mitigate statistical variance, every configuration is run 25 times for each combination of reconstruction and time stepping schemes.

3.4.2. Practical Case: Rainfall-Runoff in an Urban Area

In order to assess the runtime reduction obtained with the fb-LTS in a practical application, a heavy precipitation event in an urban area in central Berlin, Germany, is simulated. The domain covers $2048 \times 2000\,\text{m}$ with cells of $d_x = d_y = 1.0\,\text{m}$, grouped into blocks of 64×64 cells; the Courant number is set to $Cr = 0.3$ and the end time of the simulation to $t_{end} = 1200\,\text{s}$. The precipitation data is taken from Fischer (2024). For measuring accuracy in these cases, the sum of absolute errors (SAE) is used, see Section 3.4.3.

3.4.3. Metrics

Equation 3.4 defines the NSE for some variable γ, with γ_{num} as the numerical solution, the analytical or reference solution γ_{ref}, and the mean value of the reference solution throughout time $\gamma_{\text{ref, mean}}$ (Nash and Sutcliffe, 1970):

$$\text{NSE}(\gamma) = 1 - \frac{\sum_{t_0}^{t_{\max}} \left(\gamma_{\text{ref}}^t - \gamma_{\text{num}}^t\right)^2}{\sum_{t_0}^{t_{\max}} \left(\gamma_{\text{ref}}^t - \gamma_{\text{ref, mean}}\right)^2} \tag{3.4}$$

Equation 3.5 defines the SAE for some variable γ, with γ_{LTS} and γ_{GTS} as the numerical solutions obtained with LTS and GTS, respectively:

$$\mathrm{SAE}(\gamma) = \sum |\gamma_{\mathrm{LTS}} - \gamma_{\mathrm{GTS}}| \tag{3.5}$$

4. Results

Results were obtained with hms^{++} using the fb-LTS scheme including neighbor propagation and the revised criterion for scalar flux computations covered in Section 3.2.

4.1. 1D Dam Break

4.1.1. Accuracy

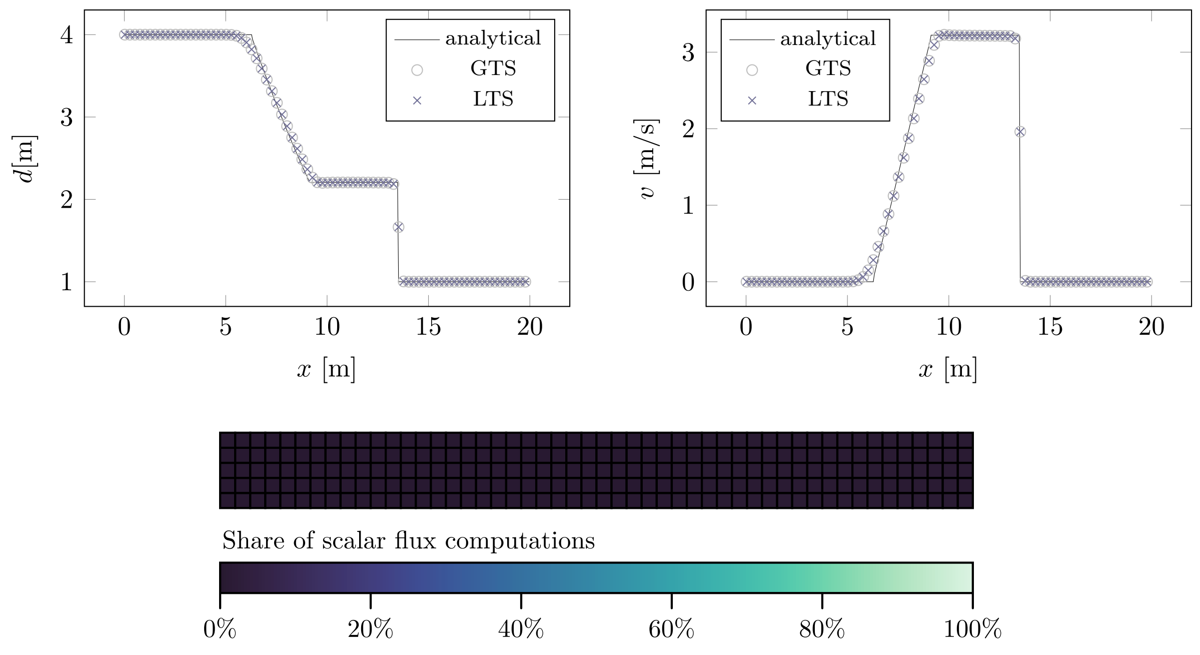
The 1D dam break was used for validating the implemented fb-LTS scheme. Figure 4.1 and Figure 4.2 show water depth and velocity profiles for both wet and dry dam break cases at time $t = 0.6\,\mathrm{s}$ using first-order and TVD reconstruction, respectively. GTS values are represented by circle marks, LTS values are shown as cross marks, and the analytical solution is drawn as a solid black line. Visually, no differences between the numerical results can be identified. This applies to both first-order and TVD reconstruction.

Table 4.1 contains NSE results of the 1D dam break. Similarly to the plots, they do not indicate a loss of accuracy through the use of the LTS scheme. For both water depth and velocity using TVD reconstruction, as well as water depth with first-order reconstruction, the first six decimal places are identical between the GTS and LTS schemes. Only the NSE for the velocity in the case of first-order reconstruction exhibits any difference between the time-stepping schemes: In the wet case, the difference is only 1×10^{-6}, while the largest difference in NSE is observed in the dry case, where it amounts to $< 2 \times 10^{-4}$. Both can be considered negligible.

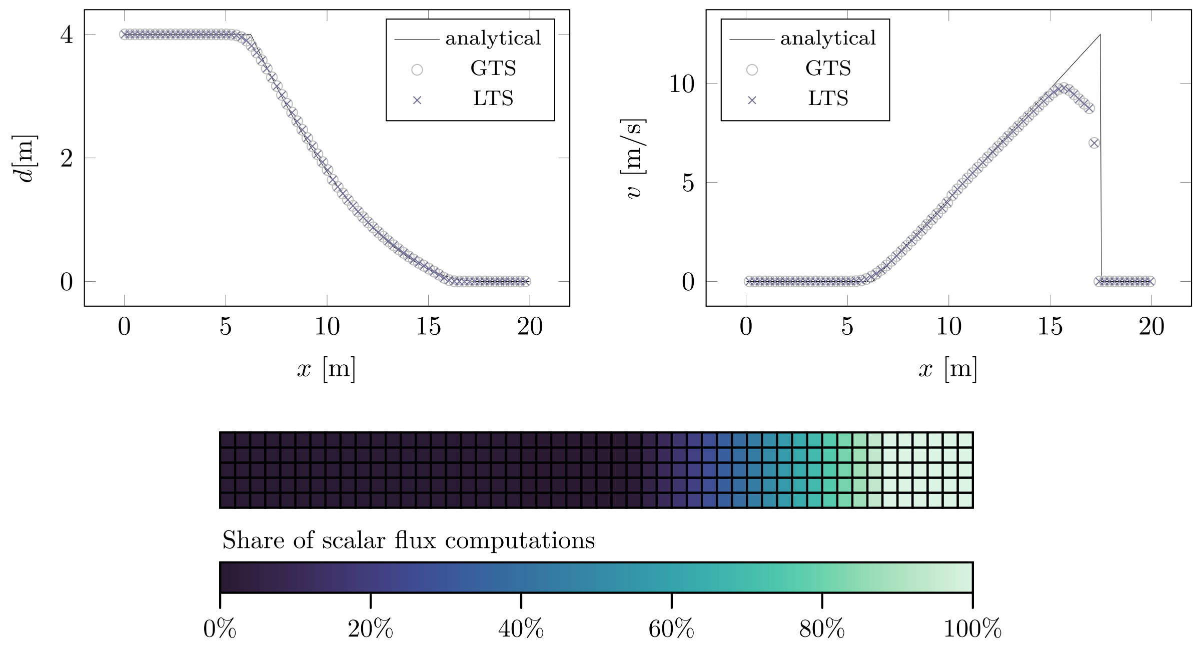
Table 4.1: NSE of water depth d and velocity u at time $t = 0.6\,\mathrm{s}$ for wet and dry 1D dam break cases, using both first-order and TVD reconstruction.

	first-order reconstruction		TVD reconstruction	
	GTS	LTS	GTS	LTS
d wet	0.998 122	0.998 122	0.999 048	0.999 048
v wet	0.989 963	0.989 964	0.993 199	0.993 199
d dry	0.999 629	0.999 629	0.999 975	0.999 975
v dry	0.849 062	0.848 889	0.962 224	0.962 224

However, in the wet dam break cases, only 0.22 % of all flux updates are scalar computations. This is due to the small cell sizes of this scenario, see Section 4.1.2. Therefore, the relevance of this test configuration, as well as its results, may be questioned. The impact of cell size is further analyzed in Section 4.1.2. The heatmaps below the water depth and velocity profiles illustrate the share of scalar flux computations (SSC): lighter colors indicate a lighter load, i.e., a higher share of scalar flux computations, while darker colors denote more full flux computations.

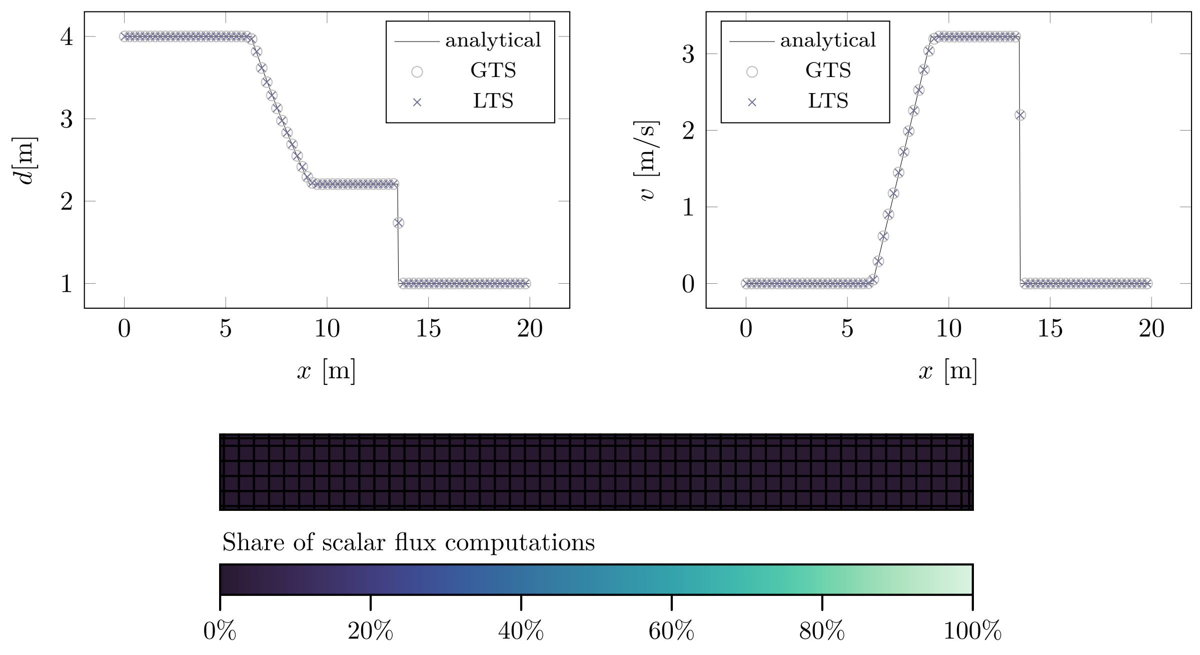


(a) Wet dam break

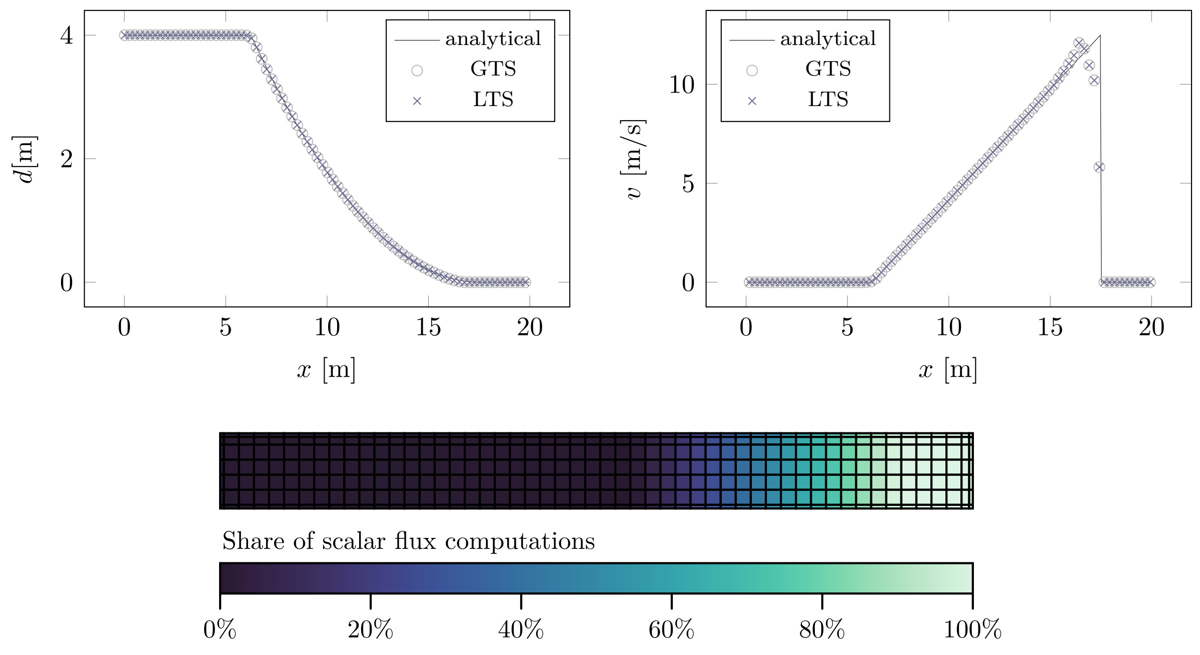


(b) Dry dam break

Figure 4.1: Water depth and velocity profiles at time $t = 0.6\,\mathrm{s}$ for wet and dry 1D dam break cases using first-order reconstruction, with heat maps illustrating the SSC.



(a) Wet dam break



(b) Dry dam break

Figure 4.2: Water depth and velocity profiles at time $t = 0.6\,\text{s}$ for wet and dry 1D dam break cases using TVD reconstruction, with heat maps illustrating the SSC.

4.1.2. Runtime Analysis

In the dam break cases for runtime benchmarks, a collection of variants has been introduced, see Section 3.4.1. Every variant has been run 25 times for each combination of GTS/LTS first-order/TVD reconstruction. Table 4.2 lists the average runtime of these runs, the relative difference between GTS and LTS runtimes as runtime reduction (RTR) and share of scalar flux computations (SSC). Variations were considered negligible, as standard deviations are $< 1\,\%$ of the average values for all configurations, except for the reduced block size of 8×8 cells, where it amounts to $\sim 2\,\%$.

The base configuration shows that runtime can be reduced by 10.09 % through LTS for first-order and by 12.05 % for TVD reconstruction. Other configurations have shown both higher and lower reduction values. While decreasing block size from 64×64 to 32×32 showed an increased reduction, setting it to 8×8 and 128×128 showed a decrease in runtime reduction. This indicates that there is an optimal value for this parameter for the use of fb-LTS somewhere between 8×8 and 64×64, possibly, close to 32×32.

Table 4.2: Average runtime of 25 runs, with runtime reduction (RTR) and share of scalar flux computations (SSC) for the selected dam break cases.

	first-order reconstruction				TVD reconstruction			
Benchmark variant	GTS [s]	LTS [s]	RTR [%]	SSC [%]	GTS [s]	LTS [s]	RTR [%]	SSC [%]
base configuration	1.2	1.1	10.1	15.7	1.8	1.5	12.1	15.7
block size 8×8	2.5	2.3	9.2	16.4	3.3	3.0	9.6	16.5
block size 32×32	1.1	1.0	8.3	14.8	1.6	1.5	10.8	14.9
block size 128×128	1.4	1.2	11.4	15.9	1.9	1.7	12.6	15.9
mesh size 1000×1000	4.9	4.3	10.6	15.9	7.0	6.1	12.2	15.9
mesh size 4000×4000	0.3	0.3	9.8	14.8	0.4	0.4	11.0	15.0
$t_{\text{end}} = 6.0\,\text{s}$	2.4	2.2	10.8	15.7	3.5	3.1	12.2	15.7
$l_x = l_y = 0.5\,\text{m}$	2.3	2.2	4.8	8.2	3.3	3.2	4.8	7.0
$l_x = l_y = 2.0\,\text{m}$	0.8	0.7	16.6	23.5	1.2	1.0	18.5	23.7
$Cr = 0.1$	3.0	3.0	1.6	4.2	4.5	4.4	1.3	3.1
$Cr = 0.5$	0.8	0.7	16.4	23.5	1.2	1.0	18.6	23.6
$d_{\text{downstream}} = 0.1\,\text{m}$	1.3	0.9	31.5	39.1	1.8	1.2	33.7	39.3
$d_{\text{downstream}} = 0.5\,\text{m}$	1.2	1.0	19.1	26.1	1.8	1.4	23.5	28.8
$d_{\text{downstream}} = 2.0\,\text{m}$	1.2	1.2	4.3	7.8	1.8	1.7	2.9	5.3
dry dam break	0.8	0.8	8.5	44.3	1.1	1.0	6.4	44.5
no bed friction	1.1	1.0	10.3	15.7	1.6	1.4	11.9	15.7

Mesh size has a minor impact on the runtime reduction. The slight improvement when using the larger mesh might be due to a longer runtime in general, which reduces the portion of initialization time over the LTS-adjusted time loop. An increase of the simulation's duration has a similar effect on the effectivity of LTS, most likely for the same reason.

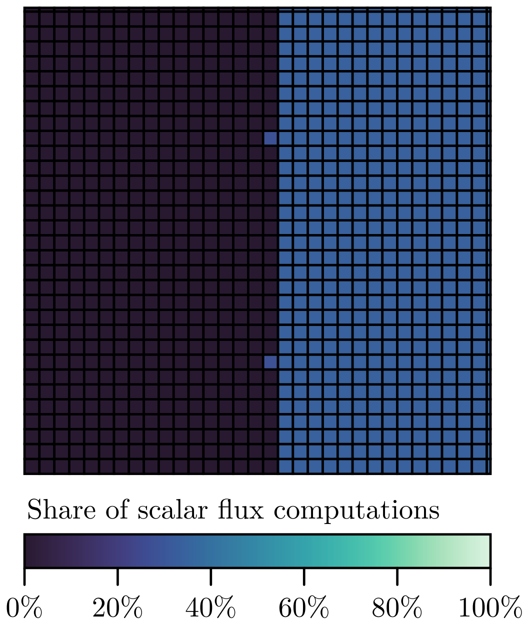
Adjustments to the cell edge lengths l_x and l_y show that RTR increases with cell size. The cell area and perimeter have a direct effect on the local time step (Equation 2.10): With square cells ($l_x = l_y$), Δt_i is proportional to l_x. While this holds true for GTS as well, the RTR results indicate that the cell size disproportionately affects LTS. The same effect can be observed for the Courant number, which is likewise proportional to Δt_i.

Decreasing the downstream water depth from 1 m to 0.1 m had the largest impact on the achievable reduction, which now reached $> 30\,\%$. However, setting $d_{\text{downstream}} = 0.0\,\text{m}$, i.e., changing the simulation case to a dry dam break, leads to a lower reduction. In Equation 2.8, $v_{i,\max}$ decreases with d_i, increasing Δt_i in turn. As only the downstream region's water depth is varied, local time steps of upstream cells remain the same. This causes large differences between those regions, resulting in a large number of scalar computations in the downstream region and therefore large runtime-reduction of the simulations using LTS. However, hms^{++} can recognize dry cells and skip flux computations for these, which explains why LTS-reductions are lower in the dry dam break case.

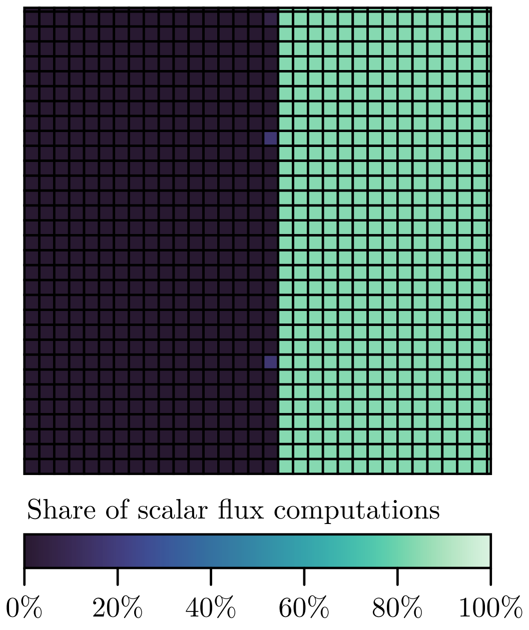
Disabling the bed friction term, despite reducing overall runtime, did not have a notable impact on the RTR.

The effect of decreasing the downstream water depth is shown in Figure 4.3 for first-order reconstruction. In both configurations, cells were updated 18 times. On average, each cell received 2.8 scalar flux updates (15.7 % of all updates) in the base configuration. In the $d_{\text{downstream}} = 0.1\,\text{m}$ variant, this number increased to 7.0 (39.1 %). This behavior is consistent with the runtime reduction described above.

The runtime reduction does not reach the SSC, due to initialization and other processes within a block or cell update sequence, as previously shown in Crossley et al. (2003). This is illustrated in Table 4.3 and Table 4.4. They present the time share of various tasks a solver thread processes within the time loop in absolute and relative terms. In both cases, the absolute time spent on full flux computation decreased significantly, by 15.0 % and 37.1 %, respectively, which is notably close to the percentage of scalar flux updates



(a) Base configuration



(b) $d_{\text{downstream}} = 0.1\,\text{m}$

Figure 4.3: Heatmaps for scalar/full flux computations, showing the SSC, for base configuration and $d_{\text{downstream}} = 0.1\,\text{m}$ variant, computed with first-order reconstruction.

described above. Additionally, time spent on friction and new time step processing were reduced as well, since they are skipped in Algorithm 4. Scalar flux computations and BFM access are disabled when using the GTS scheme, and are thus listed with a time of $0\,\text{s}$. In contrast, when using LTS, access time weighs in at $\sim 1.15\,\%$ in both cases, which is larger than the actual scalar flux computation, at $0.23\,\%$ and $0.72\,\%$, respectively. This indicates that accessing main memory is slower here than the vectorized multiply-add operation of a scalar flux computation. The SSC varies depending on the number of scalar computations, whereas the BFM access time remains constant, as it is accessed in both the full and scalar computations (Algorithm 3 and Algorithm 4, respectively).

Table 4.3: Solver thread timing output for the base configuration, computed using first-order construction.

	GTS		LTS	
	absolute [s]	absolute [%]	absolute [s]	absolute [%]
output	3.10×10^{-4}	0.03	3.90×10^{-4}	0.04
variable initialization	8.83×10^{-2}	7.27	9.38×10^{-2}	8.64
updating source terms	0.00	0.00	2.00×10^{-6}	0.00
full flux computation	8.86×10^{-1}	72.91	7.53×10^{-1}	69.40
scalar flux computation	0.00	0.00	2.54×10^{-3}	0.23
block flux memory access	0.00	0.00	1.27×10^{-2}	1.17
mass sources	9.40×10^{-5}	0.01	8.30×10^{-5}	0.01
friction	1.20×10^{-1}	9.84	1.04×10^{-1}	9.62
new state	7.57×10^{-2}	6.23	7.92×10^{-2}	7.29
time step	4.35×10^{-2}	3.58	3.69×10^{-2}	3.40
updating boundary	7.19×10^{-4}	0.06	8.13×10^{-4}	0.08
plugins	2.00×10^{-6}	0.00	1.00×10^{-6}	0.00
synchronization	3.46×10^{-4}	0.03	4.19×10^{-4}	0.04
other	7.15×10^{-4}	0.06	1.00×10^{-3}	0.09
sum	1.22		1.09	

Table 4.4: Solver thread timing output for the $d_{\text{downstream}} = 0.1\,\text{m}$ variant, computed using first-order construction.

	GTS		LTS	
	absolute [s]	absolute [%]	absolute [s]	absolute [%]
output	3.16×10^{-4}	0.03	3.21×10^{-4}	0.04
variable initialization	8.88×10^{-2}	6.97	9.02×10^{-2}	10.42
updating source terms	1.00×10^{-6}	0.00	1.00×10^{-6}	0.00
full flux computation	8.82×10^{-1}	69.18	5.55×10^{-1}	64.11
scalar flux computation	0.00	0.00	6.24×10^{-3}	0.72
block flux memory access	0.00	0.00	9.96×10^{-3}	1.15
mass sources	8.60×10^{-5}	0.01	8.20×10^{-5}	0.01
friction	1.83×10^{-1}	14.32	9.68×10^{-2}	11.18
new state	7.56×10^{-2}	5.93	7.71×10^{-2}	8.91
time step	4.35×10^{-2}	3.42	2.74×10^{-2}	3.16
updating boundary	7.85×10^{-4}	0.06	7.93×10^{-4}	0.09
plugins	1.00×10^{-6}	0.00	1.00×10^{-6}	0.00
synchronization	3.60×10^{-4}	0.03	3.96×10^{-4}	0.05
other	7.76×10^{-4}	0.06	1.45×10^{-3}	0.17
sum	1.27		8.66×10^{-1}	

4.2. Practical Case: Rainfall-Runoff in an Urban Area

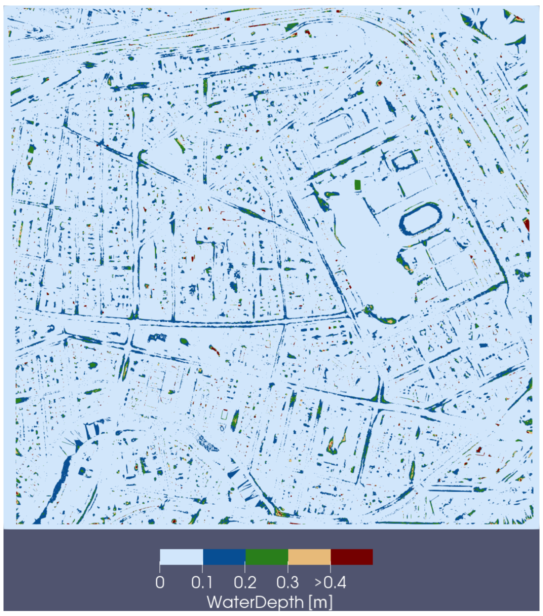
In the practical application case described in Section 3.4.2, the observable runtime reduction obtained by fb-LTS varies significantly over time, as well as by reconstruction scheme. Therefore, it is presented here for three different simulated durations, for both first-order and TVD reconstruction. The durations are 600 s, 1200 s, and 3600 s. However, it seems that even with a low Courant number of $Cr = 0.3$, instabilities occur in this case when using TVD reconstruction, which prevents the simulation from proceeding beyond 1800 s. Hence, values for $t = 3600$ exist for first-order reconstruction only.

4.2.1. Accuracy

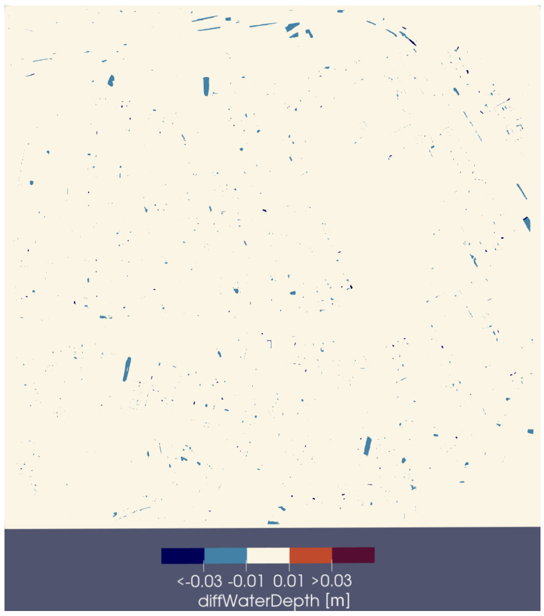
Figure 4.4 shows the resulting water depth of the LTS simulation using first-order reconstruction and a visualization of deviations of the LTS solution from the one obtained with GTS, i.e., errors, at $t = 1200$ s. Here, the SAE of the LTS solution amounts to $\sim$3695 m, or an average of 0.0036 m per cell. Most regions have an absolute error below 0.03 m, i.e., are shown in light blue. 283 of 1024000 (0.028 %) cells show an absolute error greater than 0.05 m, and 50 (0.0049 %) deviate more than 0.10 m. The three cells exhibiting the largest deviations differ by 0.98 m, 0.66 m, and 0.32 m. Qualitatively, similar results are obtained with TVD reconstruction, and thus their description is not repeated here. The respective SAE is $\sim$4260 m, which amounts to $\sim$0.0042 m per cell.

Figure 4.5 shows the resulting water depth of the LTS simulation using first-order reconstruction and a visualization of its error in comparison to the results obtained with GTS, at $t = 3600$ s. Again, only the results with first-order reconstruction are described here, as those obtained with TVD are very similar in their characteristics. Compared to Figure 4.4b, the error regions have become more concentrated: there are fewer, but larger continuous regions. Again, most of them have an absolute error below 0.03 m, just 81 of 1024000 cells (0.0079 %) show an absolute error greater than 0.05 m, and 13 (0.0013 %) deviate more than 0.10 m. The three largest deviations are all $\sim$0.23 m.

Figure 4.6 shows the development of the water depth's SAE of the LTS solution over time, for both first-order and TVD reconstruction. For both schemes, the SAE reaches a maximum value above 4000 m at $t = 100$ s, which is the first time at which results are exported, but stabilizes around 3695 m and 4260 m, respectively, from $t = 300$ s onward.

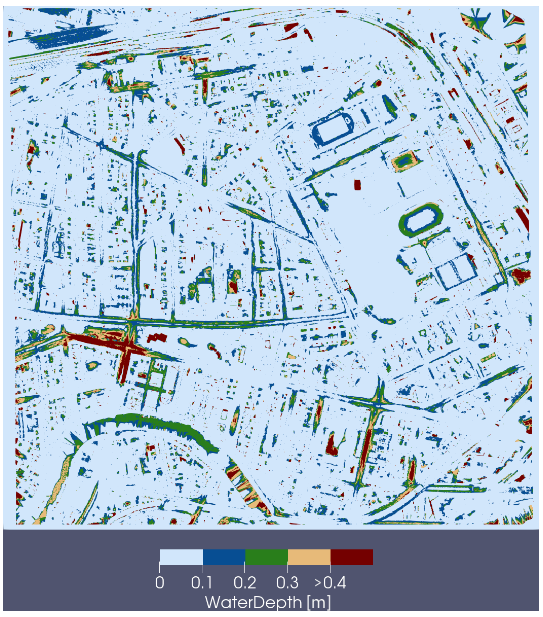


(a) LTS result for water depth

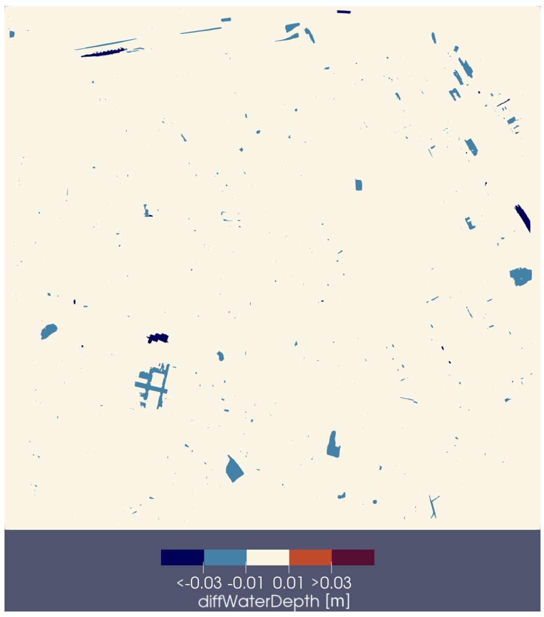


(b) Difference of water depth between results obtained with LTS and GTS schemes

Figure 4.4: Results of the practical case, simulated using first-order reconstruction, $t = 1200\,\mathrm{s}$.



(a) LTS result for water depth



(b) Difference of water depth between results obtained with LTS and GTS schemes.

Figure 4.5: Results of the practical case, simulated using first-order reconstruction, $t = 3600\,\mathrm{s}$.

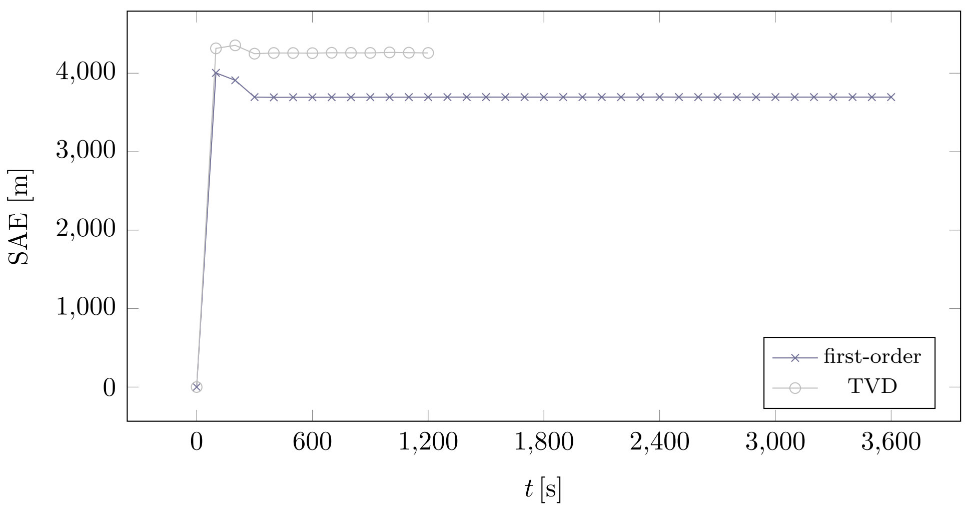


Figure 4.6: Development of the water depth's SAE of the LTS solution for first-order and TVD reconstruction throughout simulation time.

4.2.2. Runtime Analysis

Table 4.5 shows the average runtime of 10 runs each for all three simulation durations, the relative difference between GTS and LTS runtimes and the SSC.

Table 4.5: Average runtime of 10 runs, runtime reduction (RTR) and share of scalar flux computations (SSC) for the urban scenario of each combination of GTS/LTS first-order/TVD reconstruction.

	first-order reconstruction				TVD reconstruction			
	GTS [s]	LTS [s]	RTR [%]	SSC [%]	GTS [s]	LTS [s]	RTR [%]	SSC [%]
$t = 600\,\text{s}$	83.7	81.0	3.3	0.3	179.5	148.0	17.6	24.6
$t = 1200\,\text{s}$	357.3	350.4	1.9	3.9	560.3	521.8	6.9	18.2
$t = 3600\,\text{s}$	2639.3	2057.1	22.1	32.2				

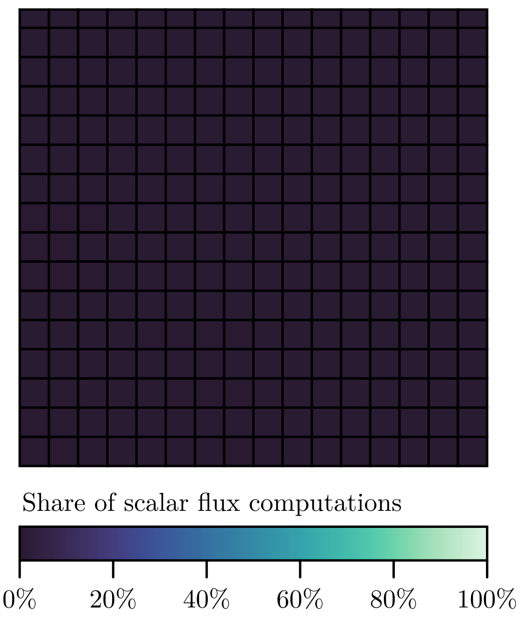
As a first observation, the GTS runtimes in Table 4.5 indicate that the global time step appears to decrease over the duration of the simulation: While it takes $\sim 80\,\text{s}$ ($180\,\text{s}$ with TVD) to simulate the first $600\,\text{s}$, the next interval of the same length takes disproportionately longer, with (rounded) $357 - 84 = 273\,\text{s}$ using first-order and $380\,\text{s}$ using TVD reconstruction. For first-order reconstruction, where such data is available, this trend continues: Here, the latter two thirds of the simulation take up $97\,\%$ of the runtime.

The runtime reductions achieved by LTS over GTS differ greatly between the reconstruction schemes. In the first 600 s, a reduction of just 3.3 % is obtained with first-order reconstruction. However, the SSC amounts to just 0.3 %, which contradicts previous results obtained in Section 4.1.1 and Crossley et al. (2003), as runtime reduction exceeds the SSC. By contrast, a runtime reduction of 17.6 % and a SSC of 24.64 %, is achieved with TVD in the same period. When considering up to 1200 s, the runtime reduction is reduced for both schemes, to $< 2\,\%$ and $< 7\,\%$, respectively. Similarly, the SSC decreased to 18.2 % for TVD, or 7.29 % if only inner blocks are considered. However, the last two thirds of the simulation appear to benefit much more from using LTS, as here, and with first-order reconstruction, a runtime reduction of $> 22\,\%$ was observed.

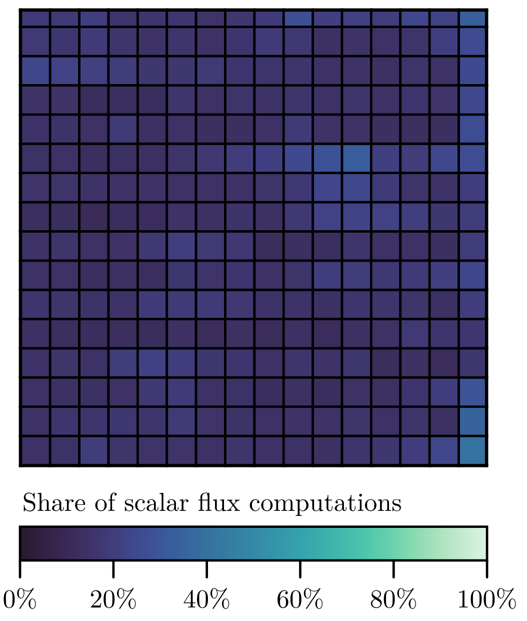
To further investigate this behavior, heatmaps showing the SSC per block, are provided in Figures 4.7–4.9, with first order on the left and TVD on the right. For short simulations of 10 minutes (600 s), the SSC diverges, depending on the utilized reconstruction scheme. Figure 4.7a represents the low overall SSC of the first-order scheme at this time. It does not show any noticeable local SSC either. In contrast, Figure 4.7b shows notably higher values ($\sim 10 - 20\,\%$) in a very uniform distribution of scalar flux computations across the whole domain for the TVD reconstruction, matching the higher overall SSC. However, it has been observed, that TVD boundary blocks receive more scalar flux computations than inner blocks: when looking at all blocks, the SSC amounts to 24.64 %, as it is obtained per block, not per cell. On the other hand, just 16.12 % of inner block updates could be carried out as scalar flux computations. The boundary blocks, which make up 21.6 % (70 out of 324) of all blocks, distort the overall share of scalar flux computations, as their SSC amounts to 55.1 %. Due to their small size, they are not noticeable in Figure 4.7b. Thus far, no explanation could be found for the divergence between reconstruction schemes and the differing behavior of boundary blocks.

In simulations of 20 minutes (1200 s), both reconstruction schemes' runtime reduction decreases in comparison to first 600 s. Both heatmaps in Figure 4.8 show largely similar distributions of SSC values, with small regions exhibiting relatively high SSC values of $\sim 40 - 50\,\%$. These regions are the same for both schemes. Once more, the boundary and inner blocks of the TVD scheme diverge: the overall SSC amounts to 18.2 % for TVD, or 7.3 %, if only inner blocks are considered. In contrast, the outer blocks' value of 57.6 % even exceeds the previous value at $t = 600\,\mathrm{s}$. This, again, cannot be explained by current results, as increases in the SSC should cause runtime reduction to increase in return.

Finally, when considering a full hour of simulated time, the SSC increases further, exceeding all previous values. Figure 4.9 shows a much wider distribution of SSC values between regions: Both higher SSC per block and more blocks with high SSC are achieved. The vast majority of blocks exhibit a SSC $> 10\,\%$ and several clusters of high SSC values can be identified. Blocks with large SSC values at $t = 1200\,\mathrm{s}$ maintain this status, but with increased values of $\sim 80 - 90\,\%$ and several of these form centers of such clusters. The increased range of local SSC is likely due to the much more heterogeneous water depth, shown in Figure 4.5a, as it has a large impact on the size of the local time step.

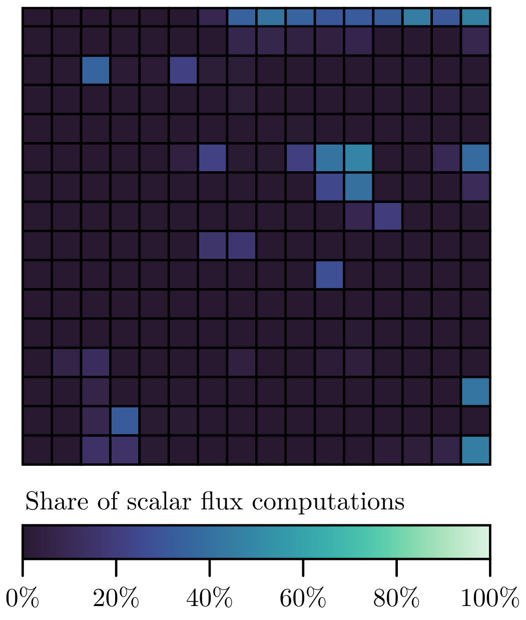


(a) First-order reconstruction

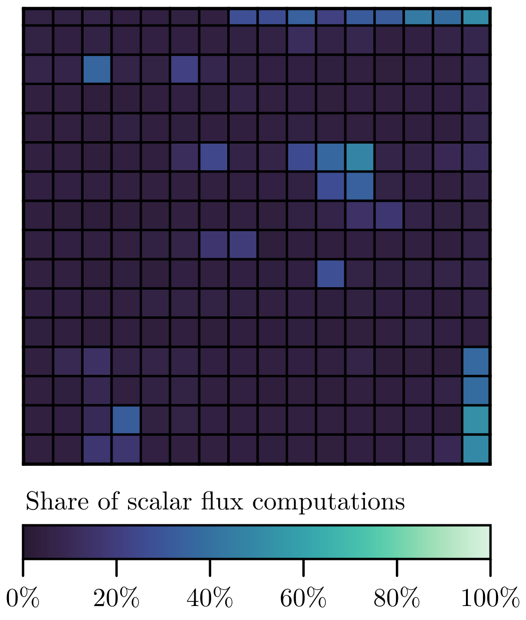


(b) TVD reconstruction

Figure 4.7: Heatmap for scalar/full flux computations, at time $t = 600\,\mathrm{s}$ for the urban scenario.



(a) First-order reconstruction



(b) TVD reconstruction

Figure 4.8: Heatmap for scalar/full flux computations, at time $t = 1200\,\mathrm{s}$ for the urban scenario.

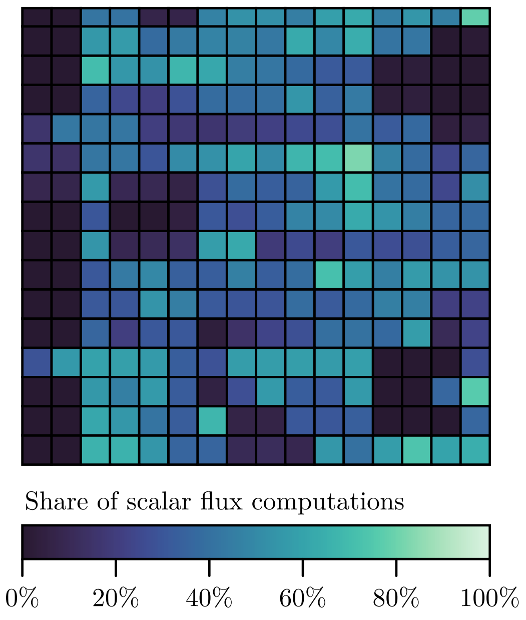


Figure 4.9: Heatmap for scalar/full flux computations, at time $t = 3600\,\mathrm{s}$ for the urban scenario, computed using first-order reconstruction.

4.3. Stability

Within the process of implementing the fb-LTS scheme, additional test cases have been run, e.g., to gain insights into stability and mass conservation of the scheme. One of these is a variation of the 1D wet dam break used above: free outflow was disabled at all boundaries to maintain the total water volume, and the simulation duration was extended to 120 s. Mesh and block sizes were set to 400×150 and 8×8 cells, respectively. Other parameters were adopted from the base configuration used in Section 4.1.2. The extended duration allowed the wave to reach the downstream boundary and be reflected. Figure 4.10 shows a three-dimensional visualization of the water surface at $t = 110$ s – after the reflection of the wave, which occours at $t \sim 35$ s. While the results of the first-order scheme show a smooth surface, the results of the TVD scheme show instabilities in the form of checker-boarding. The modification described in Section 3.2.1 has reduced this behavior, but it could not be fully prevented.

While providing overall good stability, there exist combinations of schemes and boundary conditions, under which the proposed LTS scheme in its current form does not reach the same level of stability as hms^{++}' GTS scheme.

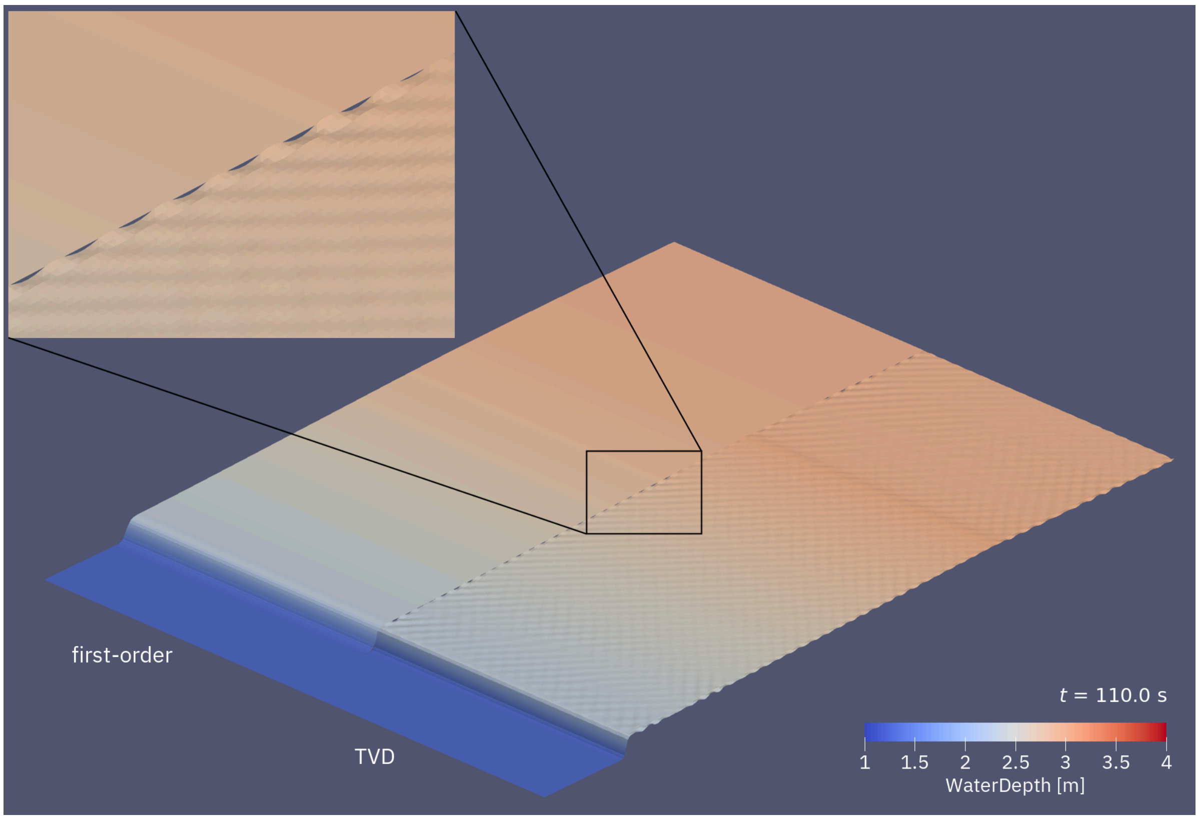


Figure 4.10: Water surfaces of the wet dam break case without outflow at $t = 110$ s. The upper right corner shows a maginified portion of the mesh.

5. Conclusions and Outlook

This thesis covers the development and implementation of the frozen block LTS scheme in the SWE model hms^{++}. To mitigate initial instabilities, modifications have been established regarding the criterion for scalar flux computations and the synchronization of adjacent blocks. Outside of a single outlier case, the scheme provides good stability and maintains comparable accuracy to global time stepping schemes. The results obtained in this thesis further show that an LTS scheme that builds on block-wise traversal can reduce simulation runtimes by upwards of 30 % in some test configurations and $> 20\,\%$ in practical application cases.

Several observations could not yet be fully explained and thus may warrant further investigation. Firstly, it was found that larger cell sizes or Courant number limits – and thus overall larger minimal time steps – disproportionately benefit the proposed LTS scheme's runtime when compared to the reference GTS scheme. Finding the root cause of this behavior could not only close this knowledge gap, but also potentially provide comprehension on how the underlying mechanisms could be exploited to further improve the scheme. Secondly, quantitatively linking the share of scalar computations to the runtime reduction could help to focus any future efforts on improving the scheme. This includes gaining insight into why boundary blocks perform more scalar flux computation than inner blocks. Understanding this might also help to determine the cause of instabilities when combining TVD reconstruction and fb-LTS, manifesting themselves as checkerboarding of the water depth, as shown in Figure 4.10. Furthermore, the development of deviations between LTS and GTS over time in the urban rainfall-runoff simulation (Section 4.2) warrants further analysis. Here, a comparably fast and localized increase in the beginning was followed by deviations remaining constant over the rest of the simulation. Looking into the interplay of load balancing (i.e., the assignment of block updates to solver threads) with the fb-LTS scheme may lead to additional performance gains by distributing scalar and full computations evenly on solver threads. Finally, as LTS schemes typically provide the highest benefits on non-uniform grids, an extension of the scheme to such grids may also be a worthwhile topic for further investigation (Crossley et al., 2003; Sanders, 2008). This, however, would require an extension of hms^{++} and its block-wise traversal as well and as such would be a much larger effort in scope.

References

Adelmann, H. M. (2022). “Validierung eines Simulationsprogramms für die Flachwassergleichungen”. Bachelorarbeit Bauingenieurwesen. Berlin: Technische Universität Berlin. DOI: 10.5281/zenodo.7835444.

Apple Inc. (2024a). *Addressing Architectural Differences in Your macOS Code.* Apple Developer Documentation. URL: https://developer.apple.com/documentation/apple-silicon/addressing-architectural-differences-in-your-macos-code (visited on 05/31/2024).

Apple Inc. (2024b). *Porting Your macOS Apps to Apple Silicon.* Apple Developer Documentation. URL: https://developer.apple.com/documentation/apple-silicon/porting-your-macos-apps-to-apple-silicon (visited on 05/31/2024).

Apple Inc. (2024c). *Writing ARM64 Code for Apple Platforms.* Apple Developer Documentation. URL: https://developer.apple.com/documentation/xcode/writing-arm64-code-for-apple-platforms (visited on 05/31/2024).

Crossley, A. J., Wright, N. G., and Whitlow, C. D. (2003). “Local Time Stepping for Modeling Open Channel Flows”. In: *Journal of Hydraulic Engineering* 129.6, pp. 455–462. DOI: 10.1061/(ASCE)0733-9429(2003)129:6(455).

Dazzi, S., Vacondio, R., Dal Palù, A., and Mignosa, P. (2018). “A Local Time Stepping Algorithm for GPU-accelerated 2D Shallow Water Models”. In: *Advances in Water Resources* 111, pp. 274–288. DOI: 10.1016/j.advwatres.2017.11.023.

Eigen 3.4 (August 18, 2021). URL: https://www.eigen.tuxfamily.org/index.php?title=3.4#New_backends (visited on 08/05/2024).

Fischer, J. C. (2024). “Untersuchung Unterschiedlicher Infiltrationsansätze Und Maßnahmenimplementierungen Für Hydrodynamische Niederschlag-Abfluss-Simulationen an Einer Beispielsenke in Berlin-Tegel”. Bachelorarbeit Bauingenieurwesen. Technische Universität Berlin.

Hinkelmann, R. (2005). *Efficient Numerical Methods and Information-Processing Techniques for Modeling Hydro- and Environmental Systems.* Vol. 21. Lecture Notes in Applied and Computational Mechanics. Berlin/Heidelberg: Springer-Verlag. DOI: 10.1007/3-540-32379-1.

Hu, P., Lei, Y., Han, J., Cao, Z., Liu, H., He, Z., and Yue, Z. (2019). “Improved Local Time Step for 2D Shallow-Water Modeling Based on Unstructured Grids”. In: *Journal of Hydraulic Engineering* 145.12, p. 9. DOI: 10.1061/(ASCE)HY.1943-7900.0001642.

Kleb, W. L., Batina, J. T., and Williams, M. H. (1992). “Temporal Adaptive Euler/Navier-Stokes Algorithm Involving Unstructured Dynamic Meshes”. In: *AIAA Journal* 30.8, pp. 1980–1985. DOI: `10.2514/3.11169`.

Lucy (2007). “Inside the Mac OS X Kernel”. In: 24th Chaos Communication Congress 24C3. Berlin. URL: `https://fahrplan.events.ccc.de/congress/2007/Fahrplan/attachments/986_inside_the_mac_osx_kernel.pdf` (visited on 05/20/2024).

Nash, J. E. and Sutcliffe, J. V. (1970). “River Flow Forecasting through Conceptual Models Part I — A Discussion of Principles”. In: *Journal of Hydrology* 10.3, pp. 282–290. DOI: `10.1016/0022-1694(70)90255-6`.

Osher, S. and Sanders, R. (1983). “Numerical Approximations to Nonlinear Conservation Laws With Locally Varying Time and Space Grids”. In: DOI: `10.2307/2007679`.

Sanders, B. F. (2008). “Integration of a Shallow Water Model with a Local Time Step”. In: *Journal of Hydraulic Research* 46.4, pp. 465–475. DOI: `10.3826/jhr.2008.3243`.

Simons, F. (2020). “A Robust High-Resolution Hydrodynamic Numerical Model for Surface Water Flow and Transport Processes within a Flexible Software Framework”. Faculty VI - Planning Building Environment, Technische Universität Berlin. URL: `https://depositonce.tu-berlin.de/handle/11303/10689`.

StatCounter (2024). *Desktop Operating System Market Share Worldwide*. StatCounter Global Stats. URL: `https://gs.statcounter.com/os-market-share/desktop/worldwide/2023` (visited on 05/20/2024).

Steffen, L., Amann, F., and Hinkelmann, R. (2020). “Concepts for Performance Improvements of Shallow Water Flow Simulations”. In: *Proceedings of the 1st IAHR Young Professionals Congress*. IAHR Young Professionals Congress. Online.

Steffen, L., Amann, F., and Hinkelmann, R. (2022). “Performance Improvement Strategies for an FVM-Based Shallow Water Flow Model on 2D Structured Grids”. In: *Proceedings of the 39th IAHR World Congress*. 39th IAHR World Congress From Snow to Sea. International Association for Hydro-Environment Engineering and Research (IAHR), pp. 3804–3815. DOI: `10.3850/IAHR-39WC252171192022753`.

Steffen, L. and Hinkelmann, R. (2023). “hms++: Open-source Shallow Water Flow Model with Focus on Investigating Computational Performance”. In: *SoftwareX* 22, p. 101397. DOI: `10.1016/j.softx.2023.101397`.

Steffen, L., Zhang, Y., Birke, L., and Hinkelmann, R. (2023). “Decoupling Performance and Flexibility within hms++: A User-Friendly Shallow Water Equations Solver with Advanced Cpu Optimisations and an Extensible Design”. In: *Advances in Hydroinformatics—SimHydro 2023*. SimHydro 2023: New Modelling Paradigms for Water Issues? Chatou: Springer Singapore.

The Linux Foundation (2017). *Linux Runs All of the World's Fastest Supercomputers*. linuxfoundation.org. URL: `https://www.linuxfoundation.org/blog/blog/linux-runs-all-of-the-worlds-fastest-supercomputers` (visited on 06/11/2024).

The Open Brand Register (2024). *macOS Version 14.0 Sonoma UNIX 03*. The Open Brand Register. URL: `https://www.opengroup.org/openbrand/register/brand3700.htm` (visited on 06/14/2024).

Toro, E. F., Spruce, M., and Speares, W. (1994). "Restoration of the Contact Surface in the HLL-Riemann Solver". In: *Shock Waves* 4.1, pp. 25–34. DOI: `10.1007/BF01414629`.

Wright, N. G., Whitlow, C. D., and Crossley, A. J. (1999). "Locally Adapted Time Steps for Open Channel Flow with Transitions". In: *Proceedings of the 28th IAHR World Congress (Graz, 1999)*. 28th IAHR World Congress. Graz. URL: `https://www.iahr.org/library/infor?pid=13930` (visited on 06/20/2024).

Yang, X., An, W., Li, W., and Zhang, S. (2020). "Implementation of a Local Time Stepping Algorithm and Its Acceleration Effect on Two-Dimensional Hydrodynamic Models". In: *Water* 12.4, p. 1148. DOI: `10.3390/w12041148`.

Zhang, X. D., Trepanier, J.-Y., Reggio, M., and Camarero, R. (1994). "Time-Accurate Local Time Stepping Method Based on Flux Updating". In: *AIAA Journal*. DOI: `10.2514/3.12195`.

A. Cross-Platform Portability

As hms^{++} has been developed to run on Linux natively, ports to other operating systems were previously available as Docker containers only. This can lead to reduced performance due to the need of running the container on a Linux kernel, which is virtualized on Windows and macOS machines.

Linux is the de facto standard operating system of scientific computing, especially high-performance computing (HPC) clusters heavily rely on Linux operating systems (The Linux Foundation, 2017). Despite constantly gaining market share in the world of desktop computers, Linux has not yet found its way to average users (StatCounter, 2024). To increase compatibility with more student's personal devices, hms^{++} was ported to macOS.

Since Darwin, macOS's underlying core OS, is POSIX certified (Lucy, 2007; The Open Brand Register, 2024), porting Open-Source software written for Linux (mostly POSIX compliant) is relatively simple. This holds true, especially in comparison to Windows, which is mostly incompatible with many practices in UNIX-like or UNIX systems like Linux and macOS.

hms^{++} is written in C++ (hence the name), a language widely used across numerous operating systems. Although C++ is heavily standardized, compiler behavior can vary intensely. Additionally, operating systems offer optimized system libraries or software development kits (SDKs), which may differ slightly as well. These are usually utilized by higher level libraries like `Eigen` or the program itself. To eliminate the risk of diverging compiler behavior, GNU Compiler Collection (gcc) was chosen over Apple `clang`, the default C/C++ compiler for macOS.

There are differences between Advanced RISC Machines (ARM) and the x86 processor architecture (x86), which, despite having been minimized by SDK developers, should still be addressed when writing portable code (Apple Inc., 2024a; Apple Inc., 2024c). This applies to software that is highly optimized for x86, e.g., because it: "(i) Interacts with third-party libraries you don't own. (ii) Interacts with the kernel or hardware. (iii) Relies on specific [Graphics Processing Unit (GPU)] behaviors. (iv) Contains assembly instructions. (v) Manages threads or optimizes your app's multithreaded behavior. (vi) Contains hardware-specific assumptions or performance optimizations." (Apple Inc., 2024b). Only the first point applies to hms^{++}, since it uses OpenMP for managing parallel execution and relies on `Eigen` for its performance optimized linear algebra operations. OpenMP is a compiler-level feature and readily available in gcc for ARM, while `Eigen` similarly has mature support for ARM since version 3.4 (*Eigen 3.4* 2021). In conclusion, porting hms^{++} from x86 to ARM is rather trivial because it utilizes high-level abstractions provided by well-maintained and widely used third-party frameworks instead of applying low-level optimizations by itself.

Specifics of Porting hms^{++} to macOS

Versions used in the context of this thesis:

- macOS 14.5 for arm64/Apple Silicon
- gcc 13.2.0
- MacOSX14.2 SDK
- Apple XCode Command Line Tools 15.1.0
- Open MPI 5.0.3

Steps undertaken to enable the execution of hms^{++} under macOS:

- `Eigen::Index` is a type alias for `long` under macOS (both gcc and `clang`), not `long long` $\rightarrow$ use of `static_cast` in vtk/xml related functions (conditional changes using compile time flags, therefore there are no type casts on Linux)
- add dependencies in cmake (`mpiHelpers` requires `io_util`)
- use of shared library linkage, because static linkage did not work in some cases, since `gcc` does not install its own linker on macOS, but uses the one provided with XCode CLI Tools instead
- extend cache info functionality to macOS commands
- adapt existing bash scripts
  - use Unix Makefiles for Mac
  - adapt `get_n_threads_build()`
  - set `CXX=g++-13` and `CC=gcc-13` flags (otherwise gcc-wrappers around `clang` is used)
  - set `-ld_classic` flag (for Apple `clang` version $\geq$ 15)
  - add `install_mac.sh` for easy setup including installation of dependencies